A Combinatorial, Primal-Dual approach to Semidefinite Programs

Sanjeev Arora* Satyen Kale*
Computer Science Department, Princeton University
35 Olden Street, Princeton, NJ 08540
{arora, satyen}@cs.princeton.edu

Abstract

Semidefinite programs (SDP) have been used in many recent approximation algorithms. We develop a general primal-dual approach to solve SDPs using a generalization of the well-known multiplicative weights update rule to symmetric matrices. For a number of problems, such as Sparsest Cut and Balanced Separator in undirected and directed weighted graphs, and the Min UnCut problem, this yields combinatorial approximation algorithms that are significantly more efficient than interior point methods. The design of our primal-dual algorithms is guided by a robust analysis of rounding algorithms used to obtain integer solutions from fractional ones.

1 Introduction

Semidefinite programming (SDP) has proved useful in design of approximation algorithms for NP-hard problems, and often (as in case of MaxCut, Sparsest Cut, Min UnCut, Min 2CNF Deletion, etc.) yields better approximation ratios than known LP-based methods.

But in several ways, our understanding of SDPs seriously lags our understanding of LPs. One is running time: though LP and SDP are syntactically similar when viewed as subcases of cone optimization and can theoretically be solved in the same amount of time [4, 23], in practice SDP solvers are slower. Another is conceptual: LP-inspired notions such as duality are ubiquitous in algorithm design whereas corresponding SDP-inspired concepts are rarely used.

Both points come into focus when we consider primal-dual algorithms in the LP world, by which we refer actually to two classes of algorithms. The first compute $(1+\varepsilon)$-approximation to special families of LPs —such as multicommodity flow. They eschew interior point methods in favor of more efficient (and combinatorial) Lagrangian relaxation methods; see Plotkin, Shmoys, Tardos [24], Young [30], Garg, Könemann [13], etc. The second class consists of primal-dual approximation algorithms for NP-hard problems. Though these usually evolve out of (and use the same intuition as) earlier approximation algorithms that used LP as a black box, they do not solve the LP per se. Rather, the algorithm incrementally builds a dual solution together with an *integer* primal solution, updating them at each step using "combinatorial" methods. At the end, the candidate dual solution is feasible and the bound on the approximation ratio is derived by comparing the integer primal solution to the lower bound provided by this feasible dual. Usually the update rule is designed using intuition from the *rounding algorithm* used in the original LP-based algorithm. Some canonical examples are network design problems [3] (or see the survey [14]) and $O(1)$-approximation for k-median (LP-based algorithm in [11]; faster primal-dual algorithm in [18]). Arguably, a primal-dual algorithm gives *more* insight than an algorithm that uses LP as a black box. For instance, the primal-dual algorithm for k-median problem inspired the discovery of algorithms for many related problems, as well as algorithms in the on-line and streaming models.

Since SDPs also satisfy a duality theorem, in principle one should be able to solve them using primal-dual approaches. But several conceptual difficulties arise. First, the basic object in SDPs is a positive semidefinite matrix, whereas it is a halfspace (equivalently, a vector) in LPs, and matrix operations are

*Supported by NSF grants MSPA-MCS 0528414, CCF 0514993, ITR 0205594. Part of this work was done when the authors were visiting Microsoft Research.

just harder to visualize than vector operations. Second, the recent spate of rounding algorithms for SDPs use the global structure of optimum or near-optimum solutions (e.g., the Arora, Rao, Vazirani (ARV)-style rounding depends upon the geometry of ℓ_2^2 spaces), and it is unclear how to use those rounding ideas in context of the grossly infeasible solutions one might encounter during a primal-dual algorithm. Finally, even if one surmounts the previous two difficulties, there is the issue of implementing matrix operations efficiently enough so that the running time is an improvement over interior point methods.

We note that an *ad hoc* primal-dual approach did prove useful for the SPARSEST CUT problem, resulting in a $O(\sqrt{\log n})$-approximation in $\tilde{O}(n^2)$ time [5], improving upon the $\tilde{O}(n^{4.5})$ time using SDPs [8]. A related paper gives an even more efficient $\tilde{O}(m + n^{1.5})$ time algorithm for SPARSEST CUT, albeit with a worse approximation ratio of $O(\log^2 n)$ [19]. But there is no obvious way to generalize these *ad hoc* approaches to other SDPs, especially as both rely upon the connection between eigenvalues and expansion, which does not extend to problems other than SPARSEST CUT.

This paper overcomes the above-mentioned difficulties and presents general techniques that lead to fast primal dual approximation algorithms for a number of problems (the final version will contain a full list).

First, we give a general primal-dual approximation algorithm for *any* SDP that uses an update rule that we call the *Matrix Multiplicative Weights algorithm*. This has the form $\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} \exp(\varepsilon \cdot f(\mathbf{x}))$, where $f(\mathbf{x})$ is some "feedback" function. A similar idea appears in many existing algorithms (see the recent survey [7]), except there $\mathbf{x}$ and $f(\mathbf{x})$ are both numbers whereas in our context they are matrices. The matrix version is useful in an SDP context because $\exp(\mathbf{A})$ is positive semidefinite for all symmetric $\mathbf{A}$. The analysis of the algorithm uses the intuition that symmetric matrices behave in some ways like real numbers, and obey inequalities that are syntactically similar to inequalities such as $e^{-x} \geq 1 - x$. We also need the Golden-Thompson inequality [16, 25] for matrix exponentials.

Second, we describe combinatorial algorithms to compute the "feedback" function for some interesting SDPs. For general SDPs the computation of the feedback function amounts to solving a very simple LP (see Section 3), but the convergence time of the algorithm depends upon the "width" of the problem as in the corresponding LP algorithms [24]. We give very simple and combinatorial implementations of the feedback function for several problems and prove that the width is low, resulting in (very fast) polynomial running times for the algorithms. Sometimes (as in our algorithm for MAXCUT) computing the feedback is as simple as sorting; at other times it may involve multicommodity flows and shortest paths (as in our algorithms for SPARSEST CUT, MIN UNCUT, and all related problems). Since the goal is an approximation algorithm for an **NP**-hard problems, one can terminate the above process far before the primal SDP solution is $(1 + \varepsilon)$-approximate. Instead, one rounds the current primal candidate and proves its goodness by comparing to the dual. This is the basis for all approximation algorithms in this paper. Not surprisingly, the computation of the feedback function is inspired by SDP rounding algorithms from ARV[8] and subsequent papers such as [2] (though we had to modify the rounding algorithms a bit). (This is the SDP analog of what Young [30] called "Randomized Rounding without solving the LP.") We use the following observation about ARV-style rounding techniques: if the rounding fails to yield a good integer solution when applied to a candidate primal solution, then it actually uncovers gross deviations from feasibility in the candidate solution, which can be used as "feedback" (the vector $\mathbf{y}$ in the generic algorithm of Section 3) to improve the primal.

Third, we observe that in our context it suffices to compute matrix exponentials only approximately, for which we give efficient algorithms (based upon known ideas) that can make use of sparsity; see Section 5. (By contrast, exact matrix exponentiation is tricky and inefficient because of accuracy issues; see [22].) This relies upon a subtle use of the Johnson- Lindenstrauss lemma on random projections. The exponentiation is shown to reduce to a primitive available in packages for solving linear differential equations, raising hope that our approach may be practical.

Table 1 lists the running times of our algorithms for approximating various problems on a *weighted* graph with n vertices and m edges, and compares the running time to those of the previously known algorithms from [5, 2]. Throughout the paper, the $\tilde{O}(\cdot)$ notation suppresses polylog(n) and poly$(\frac{1}{\varepsilon})$ factors.

A recent paper [19] suggested that the gold standard for approximation algorithms in this area should be T_{flow}, the time to compute single commodity max flows. Even though methods based upon LP duality can not yet attain this gold standard, that paper gave algorithms that attain it while computing $O(\log^2 n)$-approximation to SPARSEST CUT and BALANCED SEPARATOR in undirected graphs. We can attain this gold standard for four of the problems even with $O(\log n)$-approximation (T_{flow} is $\tilde{O}(m^{1.5})$ for directed graphs and $\tilde{O}(n^{1.5})$ undirected graphs). For the last three problems, we do not know of any published primal-dual

Problem	Previous best $O(\sqrt{\log n})$ approx	This paper: $O(\sqrt{\log n})$ approx	This paper: $O(\log n)$ approx
Undirected Sparsest Cut	$\tilde{O}(n^2)$ [5]	$\tilde{O}(n^2)$	$\tilde{O}(m+n^{1.5})$
Undirected Balanced Separator	$\tilde{O}(n^2)$ [5]	$\tilde{O}(n^2)$	$\tilde{O}(m+n^{1.5})$
Directed Sparsest Cut	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(m^{1.5}+n^{2+\varepsilon})$	$\tilde{O}(m^{1.5})$
Directed Balanced Separator	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(m^{1.5}+n^{2+\varepsilon})$	$\tilde{O}(m^{1.5})$
Min UnCut	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(n^3)$	–

Table 1: Running times obtained for various approximation algorithms

algorithms (even in the LP world). Currently, we have an algorithm for Min 2CNF Deletion that is not better than previous algorithms in the worst case.

Related work. Our primal-dual algorithm should not be confused with earlier *primal-only* methods for approximate solutions to SDPs, such as the authors' previous paper with Hazan [6] and an earlier paper [20]. They depend upon the standard multiplicative weight update framework of [24] and have a significant drawback – they find primal solutions which satisfy every constraint up to an *additive* error ε, and the running time is proportional to $1/\varepsilon^2$. Since the recent wave of SDP-based approximation algorithms for minimization problems require ε to be quite small, it is difficult to get significant running time improvement (though our earlier paper [6] got around this hurdle with "hybrid" approaches). This problem is exacerbated if the graph is weighted, when ε may depend upon the largest weight in the graph. By contrast, our algorithm can round the current primal candidate at each step and stop as soon as the gap with the dual is small enough. Other than a binary search on the optimum value, our algorithms are strongly polynomial so long as the algorithm for max flow is.

A form of the Matrix Multiplicative Weights algorithm for learning problems had been previously discovered in Machine Learning by Tsuda *et al* [26], as the *Matrix Exponentiated Gradient* algorithm. Warmuth and Kuzmin [28] extended these ideas to independently discover the same Matrix Multiplicative Weights algorithm as ours and gave further applications to online learning. They have a different proof of convergence using the quantum relative entropy as a potential function.

We also have some additional applications of the Matrix Multiplicative Weights algorithm that will be the subject of a future paper. Specifically, we can derandomize the Alon-Roichman construction of expanders using Cayley graphs, obtain a deterministic $O(\log n)$ approximation to the quantum hypergraph cover problem (independently discovered by Wigderson and Xiao [29] using the method of pessimistic estimators *à la* Young [30]), and give an alternative proof of Aaronson's result [1] on the fat-shattering dimension of quantum states. We think there may be other applications to quantum computing, since the basic object in both settings is the *density matrix* (see Section 2.1).

2 Preliminaries

All matrices in the paper are symmetric. As usual, $\mathbf{Tr}(\mathbf{A})$ is the sum of the diagonal entries (equivalently, the sum of the eigenvalues) of $\mathbf{A}$. Matrix $\mathbf{A}$ is positive semidefinite, or *PSD*, if there is a matrix $\mathbf{V}$ such that $\mathbf{A} = \mathbf{V}\mathbf{V}^\top$ (equivalently, if every eigenvalue of $\mathbf{A}$ is nonnegative). Such a $\mathbf{V}$ is called the *Cholesky* decomposition of $\mathbf{A}$; note that $\mathbf{A}_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$ where $\mathbf{v}_i$ is the i^{th} row of $\mathbf{V}$, and $\mathbf{A}$ is known as the *Gram matrix* of the vectors $\mathbf{v}_i$. For matrices $\mathbf{A}$ and $\mathbf{B}$, define $\mathbf{A} \bullet \mathbf{B} := \mathbf{Tr}(\mathbf{A}\mathbf{B}) = \sum_{ij} \mathbf{A}_{ij}\mathbf{B}_{ij}$. Notice, this is just the usual inner product if we think of $\mathbf{A}, \mathbf{B}$ as n^2-dimensional vectors. It is easily checked that $\mathbf{A}$ is *PSD* iff $\mathbf{A} \bullet \mathbf{B} \geq 0$ for all *PSD* $\mathbf{B}$. We say $\mathbf{A} \succeq \mathbf{B}$ if $\mathbf{A} - \mathbf{B}$ is *PSD*. We will use the ℓ_2 norm of matrices: $\|\mathbf{A}\|$ is the largest eigenvalue of $\mathbf{A}$ in absolute value, i.e. $\min\{\lambda \geq 0 : -\lambda\mathbf{I} \preceq \mathbf{A} \preceq \lambda\mathbf{I}\}$. Note that $\|\mathbf{A}+\mathbf{B}\| \leq \|\mathbf{A}\|+\|\mathbf{B}\|$.

We often use the following special matrix. If $G = (V, E)$ is a graph with weight $c_{\{i,j\}}$ on edge $\{i, j\}$ then its *combinatorial Laplacian* is a matrix $\mathbf{C}$, with rows and columns indexed by the nodes of G such that $\mathbf{C}_{ii} = \sum_{j \neq i} c_{\{i,j\}}$, i.e. the weighted degree of node i, and $\mathbf{C}_{ij}$ is $-c_{\{i,j\}}$. We will use two important properties of Laplacians. First, if d is the maximum degree in the graph, then $\mathbf{0} \preceq \mathbf{C} \preceq 2d\mathbf{I}$. In particular, $\mathbf{C}$ is *PSD*. Second, for any positive semidefinite matrix $\mathbf{X}$, if $\mathbf{v}_i$ are the vectors obtained from its Cholesky

decomposition, then $\mathbf{C} \bullet \mathbf{X} = \sum_{\{i,j\} \in E} c_{\{i,j\}} \|\mathbf{v}_i - \mathbf{v}_j\|^2$. This final expression should be familiar to readers who have encountered SDP relaxations of problems such as MaxCut and Sparsest Cut.

Finally, we discuss matrix exponentials. If $\mathbf{A}$ is a matrix, then the exponential is $\exp(\mathbf{A}) = \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}$. Notice, since $\mathbf{AB} \neq \mathbf{BA}$ in general, $\exp(\mathbf{A} + \mathbf{B}) \neq \exp(\mathbf{A}) \exp(\mathbf{B})$. Furthermore, $\exp(\mathbf{A})$ is *PSD* for all symmetric $\mathbf{A}$ since $\exp(\mathbf{A}) = \exp(\frac{1}{2}\mathbf{A})^\top \exp(\frac{1}{2}\mathbf{A})$. In particular, this allows us to assume without loss of generality that algorithms for matrix exponentiation can also output the Cholesky decomposition of the output matrix.

A remark on computing matrix exponentials. Matrix exponentiation itself is tricky because of accuracy issues; see the classic article [22]. However, in context of solving SDPs we need the Cholesky decomposition of $\exp(\mathbf{A})$, namely $\exp(\frac{1}{2}\mathbf{A})$. Furthermore, in all the SDPs of interest in this paper, the constraints involve squared lengths of vectors $\|\mathbf{v}_i\|^2$ or $\|\mathbf{v}_i - \mathbf{v}_j\|^2$, rather than inner products $\mathbf{v}_i \cdot \mathbf{v}_j$. Thus it actually suffices to compute the Cholesky decomposition approximately, in a way that these lengths are preserved upon a multiplicative factor $(1 + \varepsilon)$. By the well-known Johnson-Lindenstrauss lemma, the lengths of n vectors in ℓ_2 are essentially determined by their projections on $O(\log n/\varepsilon^2)$ random directions, so it suffices to compute $\exp(\frac{1}{2}\mathbf{A}) \cdot \mathbf{u}$ for a random unit vector $\mathbf{u}$. This operation is extremely efficient (even for arbitrary vector $\mathbf{u}$) since it is at the heart of software packages to solve systems of linear differential equations. The algorithms are also provably efficient and run in $\tilde{O}(m)$ time if $\mathbf{A}$ is "well-conditioned," where m is the number of nonzero entries in $\mathbf{A}$. All our matrices are well-conditioned or can be easily made well-conditioned. Section 5 describes these algorithms in greater detail, as well as a subtlety that needs addressing.

2.1 The Matrix Multiplicative Weights algorithm

The primal-dual method for SDPs relies on something we call the *matrix multiplicative update rule* which is a more general algorithm. It is the matrix analogue of the standard multiplicative update rule framework described in our survey [7], and we hope that it will also apply to many settings. (A few applications will appear in a forthcoming paper.) Throughout this section, for a symmetric matrix $\mathbf{A}$, we let $\lambda_1(\mathbf{A}) \geq \lambda_2(\mathbf{A}) \cdots \geq \lambda_n(\mathbf{A})$ denote its eigenvalues.

Imagine a matrix generalization of the usual 2-player zero-sum game. One player chooses a unit vector $\mathbf{v} \in \mathbb{S}^{n-1}$. The other player chooses a matrix $\mathbf{M}$ such that $\mathbf{0} \preceq \mathbf{M} \preceq \mathbf{I}$. The first player has to pay the second $\mathbf{v}^\top \mathbf{M} \mathbf{v} = \mathbf{M} \bullet \mathbf{v}\mathbf{v}^\top$. We allow the first player to choose his vector from a distribution $\mathcal{D}$ over $\mathbb{S}^{n-1}$. In this case, we are interested in the expected loss to the first player, viz.

$$\mathop{\mathbb{E}}_{\mathcal{D}}[\mathbf{v}^\top \mathbf{M} \mathbf{v}] \;=\; \mathbf{M} \bullet \mathop{\mathbb{E}}_{\mathcal{D}}[\mathbf{v}\mathbf{v}^\top].$$

The matrix $\mathbf{P} = \mathbb{E}_{\mathcal{D}}[\mathbf{v}\mathbf{v}^\top]$ is a *density matrix:* it is positive semidefinite and has trace 1. (Density matrices appear in quantum computation, for example. Indeed, our algorithms have found subsequent use for quantum computing.)

Now consider an online version of this game where the first player has to react to an external adversary who picks a matrix $\mathbf{M}$ at each step; this is called an *observed event*. An online algorithm for the first player chooses a density matrix $\mathbf{P}^{(t)}$, and observes the event matrix $\mathbf{M}^{(t)}$ in each round $t = 1, 2, \ldots, T$. After T rounds, the best fixed vector for the first player in hindsight is the unit vector $\mathbf{v}$ which minimizes the total loss $\sum_{t=1}^{T} \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v}$. It is easy to see that this is minimized when $\mathbf{v}$ is the unit eigenvector of $\sum_{t=1}^{T} \mathbf{M}^{(t)}$ corresponding to the smallest eigenvalue. Our goal is to design an algorithm whose total expected loss over the T rounds is not much more than the minimum loss $\lambda_n(\sum_{t=1}^{T} \mathbf{M}^{(t)})$.

Matrix Multiplicative Weights algorithm

Fix an $\varepsilon < \frac{1}{2}$, and let $\varepsilon' = -\ln(1 - \varepsilon)$. In every round t, for $t = 1, 2, \ldots$:

1. Compute
$$\mathbf{W}^{(t)} = (1 - \varepsilon)^{\sum_{\tau=1}^{t-1} \mathbf{M}^{(\tau)}} = \exp(-\varepsilon'(\textstyle\sum_{\tau=1}^{t-1} \mathbf{M}^{(\tau)})). \tag{1}$$
2. Use the density matrix $\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\mathbf{Tr}(\mathbf{W}^{(t)})}$ and observe the event $\mathbf{M}^{(t)}$.

Theorem 1 *The Matrix Multiplicative Weights algorithm generates density matrices* $\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(T)}$ *such that:*

$$\textstyle\sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\leq\; (1+\varepsilon)\lambda_n(\sum_{t=1}^{T} \mathbf{M}^{(t)}) + \frac{\ln n}{\varepsilon}. \tag{2}$$

PROOF: The proof is based on a potential function. We track the changes in $\mathbf{Tr}(\mathbf{W}^{(t)})$ over time. The analysis is complicated by the fact that matrix multiplication is non-commutative, so $\exp(\mathbf{A}+\mathbf{B}) \neq \exp(\mathbf{A})\exp(\mathbf{B})$ in general. However, we can use the Golden-Thompson inequality [16, 25]: $\mathbf{Tr}(\exp(\mathbf{A}+\mathbf{B})) \leq \mathbf{Tr}(\exp(\mathbf{A})\exp(\mathbf{B}))$. We have:

$$\begin{aligned}
\mathbf{Tr}(\mathbf{W}^{(t+1)}) &= \mathbf{Tr}(\exp(-\varepsilon'\textstyle\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)})) \\
&\leq \mathbf{Tr}(\exp(-\varepsilon'\textstyle\sum_{\tau=1}^{t-1}\mathbf{M}^{(\tau)})\exp(-\varepsilon'\mathbf{M}^{(t)})) && \because \text{ Golden-Thompson inequality} \\
&= \mathbf{W}^{(t)} \bullet \exp(-\varepsilon'\mathbf{M}^{(t)}) && \because \mathbf{Tr}(\mathbf{AB}) = \mathbf{A}\bullet\mathbf{B} \\
&\leq \mathbf{W}^{(t)} \bullet (\mathbf{I} - \varepsilon\mathbf{M}^{(t)}) && \because (1-\varepsilon)^{\mathbf{A}} \preceq (\mathbf{I}-\varepsilon\mathbf{A}) \text{ if } \mathbf{0}\preceq\mathbf{A}\preceq\mathbf{I} \\
&= \mathbf{Tr}(\mathbf{W}^{(t)}) \cdot (1 - \varepsilon\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}) \\
&\leq \mathbf{Tr}(\mathbf{W}^{(t)}) \cdot \exp(-\varepsilon\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}).
\end{aligned}$$

By induction, since $\mathbf{Tr}(\mathbf{W}^1) = \mathbf{Tr}(\mathbf{I}) = n$, we get that

$$\mathbf{Tr}(\mathbf{W}^{T+1}) \;\leq\; n\exp(-\varepsilon\textstyle\sum_{t=1}^{T}\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}).$$

On the other hand, we have:

$$\mathbf{Tr}(\mathbf{W}^{T+1}) \;=\; \mathbf{Tr}(\exp(-\varepsilon'\textstyle\sum_{t=1}^{T}\mathbf{M}^{(t)})) \;\geq\; \exp(-\varepsilon'\lambda_n(\sum_{t=1}^{T}\mathbf{M}^{(t)})).$$

The last inequality follows because $\mathbf{Tr}(e^{\mathbf{A}}) = \sum_{k=1}^{n} e^{\lambda_k(\mathbf{A})} \geq e^{\lambda_n(\mathbf{A})}$. Thus, we conclude that

$$\exp(-\varepsilon'\lambda_n(\textstyle\sum_{t=1}^{T}\mathbf{M}^{(t)})) \;\leq\; n\exp(-\varepsilon\textstyle\sum_{t=1}^{T}\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}).$$

Taking logarithms and simplifying, we get the required inequality. □

More Intuitive statement: The notation can hide the intuition, so here is a more succinct statement that will be used in the Primal-Dual method for SDPs. Suppose an adversary presents a sequence of matrices $\mathbf{M}^{(1)}$, $\mathbf{M}^{(2)}$, $\mathbf{M}^{(3)}$,... and the algorithm uses these to define matrices $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$, according to $\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\mathbf{Tr}(\mathbf{W}^{(t)})}$, where $\mathbf{W}^{(t)}$ is defined as in (2).

Suppose we require the adversary's matrices to satisfy the seemingly benign condition $\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \geq 0$ for all t. Then in fact this condition is not so benign. It implies a stronger condition on the cumulative sum $S_T = \sum_{t=1}^{T} \mathbf{M}^{(t)}$ after a short number of steps: S_T becomes *almost* positive semi-definite, namely, its smallest eigenvalue is not too negative.

3 Primal-Dual Approach for Approximately Solving SDPs

This section describes a general primal-dual algorithm to compute a near-optimal solution to any SDP (and not just SDPs used in approximation algorithms). As an illustrative example we also describe its use for the SDP relaxation for MAXCUT.

A general SDP with n^2 variables (thought of as an $n \times n$ matrix variable $\mathbf{X}$) and m constraints, and its dual can be written as follows:

$$\begin{array}{rl|rl}
\max & \mathbf{C}\bullet\mathbf{X} & \min & \mathbf{b}\cdot\mathbf{y} \\
\forall j \in [m]: & \mathbf{A}_j \bullet \mathbf{X} \;\leq\; b_j & & \textstyle\sum_{j=1}^{m}\mathbf{A}_j y_j \;\succeq\; \mathbf{C} \\
& \mathbf{X} \;\succeq\; \mathbf{0} & & \mathbf{y} \;\geq\; \mathbf{0}
\end{array}$$

Here, $\mathbf{y} = \langle y_1, y_2, \ldots, y_m\rangle$ are the dual variables and $\mathbf{b} = \langle b_1, b_2, \ldots, b_m\rangle$. Just as in the case of LPs, strong duality holds for SDPs under very mild conditions (always satisfied by the SDPs considered here) and the optima of the two programs coincide.

Note that a linear program is the special case whereby all the matrices involved are diagonal. (Aside: The recipe for writing SDP duals is syntactically similar to the one for LP dual, except instead of vector inequalities such as $\mathbf{a} \geq \mathbf{b}$, one uses matrix inequalities $\mathbf{A} \succeq \mathbf{B}$.)

For notational ease assume that $\mathbf{A}_1 = \mathbf{I}$ and $b_1 = R$. This serves to bound the trace of the solution: $\mathbf{Tr}(\mathbf{X}) \leq R$ and is thus a simple scaling constraint. It is very naturally present in SDP relaxations for combinatorial optimization problems.

We assume that our algorithm uses binary search to reduce optimization to feasibility. Let α be the algorithm's current guess for the optimum value of the SDP. It is trying to either construct a *PSD* matrix that is primal feasible and has value $> \alpha$, or a dual feasible solution whose value is at most $(1+\delta)\alpha$ for some arbitrarily small $\delta > 0$.

As is usual in primal-dual algorithms, the algorithm starts with a trivial candidate for a primal solution, in this case the trivial *PSD* matrix (possibly infeasible) of trace R, viz. $\mathbf{X}^{(1)} = \frac{R}{n}\mathbf{I}$. Then it iteratively generates candidate primal solutions $\mathbf{X}^{(2)}, \mathbf{X}^{(3)}, \ldots$. At every step it tries to improve $\mathbf{X}^{(t)}$ to obtain $\mathbf{X}^{(t+1)}$. In this it has help from an auxiliary algorithm, called the ORACLE, that tries to certify the validity of the current $\mathbf{X}^{(t)}$, and failing which ORACLE produces a *feedback matrix* $\mathbf{M}^{(t)}$. The algorithm will use the matrix multiplicative weight algorithm to generate $\mathbf{X}^{(t+1)}$ from $\mathbf{M}^{(1)}, \mathbf{M}^{(2)}, \ldots, \mathbf{M}^{(t)}$.

The rationale behind this back and forth can be understood by our intuitive explanation of the matrix multiplicative weight rule at the end of Section 2.1. Think of the oracle's task as trying to produce a feasible dual by the end. A feasible dual is one in which the *slack matrix* $\sum_{j=1}^{m} \mathbf{A}_j y_j - \mathbf{C}$ is psd. The matrix multiplicative weight algorithm allows oracle's task to be replaced by a sequence of simpler tasks: focus on finding a slack matrix that has nonnegative matrix inner product with the current $\mathbf{X}^{(t)}$. If it manages to do this for even a short number of steps, then Theorem 1 implies that the average of the slack matrices over these steps will be almost psd! The important saving is that the more difficult condition of producing a semidefinite slack matrix is is replaced by a linear condition, namely, a slack matrix having nonnegative matrix inner product with $\mathbf{X}^{(t)}$.

Now we describe ORACLE in more detail. Given $\mathbf{X}^{(t)}$ it searches for a vector $\mathbf{y}$ from the polytope $\mathcal{D}_\alpha = \{\mathbf{y} : \ \mathbf{y} \geq 0, \ \mathbf{b} \cdot \mathbf{y} \leq \alpha\}$ such that

$$\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \ \geq \ 0. \tag{3}$$

If ORACLE succeeds in finding such a $\mathbf{y}$ then we claim $\mathbf{X}^{(t)}$ is either primal infeasible or has value $\mathbf{C} \bullet \mathbf{X}^{(t)} \leq \alpha$. The reason is that otherwise

$$\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \ \leq \ \sum_{j=1}^{m} b_j y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \ < \ \alpha - \alpha \ = \ 0,$$

which would contradict (3). Thus $\mathbf{y}$ implicitly contains some useful information to improve the candidate primal $\mathbf{X}^{(t)}$, and indeed we use $\mathbf{y}$ to update $\mathbf{X}^{(t)}$ using a *matrix exponential update* rule (step 5 in the algorithm). Our observation about matrix exponentials ensures that the new matrix $\mathbf{X}^{(t+1)}$ is also *PSD*.

On the other hand, if ORACLE declares that there is no vector $\mathbf{y} \in \mathcal{D}_\alpha$ which satisfies (3), then it can be easily checked using linear programming duality that (a suitably scaled version of) $\mathbf{X}^{(t)}$ must be a primal feasible solution of objective value at least α.

As already mentioned, the important point here is that the desired $\mathbf{y}$ is not dual feasible: in fact the ORACLE can ignore the *PSD* constraint and its task consists of solving an LP with *just one non-trivial constraint* (the others are just non-negativity constraints)! Thus, one may hope to implement ORACLE efficiently, and furthermore, even find $\mathbf{y}$ with *nice* properties, so that the algorithm makes fast progress towards feasibility. Now we formalize one aspect of *nice*-ness, the *width*. (Another aspect of niceness concerns a subtle condition that allows quick matrix exponentiation; see the discussion in the Section 5.)

The Primal-Dual SDP algorithm, shown below, depends on the *width parameter* of the ORACLE. This is the smallest $\rho \geq 0$ such that for every primal candidate $\mathbf{X}$, the vector $\mathbf{y} \in \mathcal{D}_\alpha$ returned by the ORACLE satisfies $\|\mathbf{A}_j y_j - \mathbf{C}\| \leq \rho$. The constraint on width may seem like a backdoor way to bring in the semidefiniteness constraint into the oracle but it is more correctly viewed as a measure of the ORACLE's effectiveness in helping the algorithm make progress (high width equals slow progress). In all our applications the vector

$\mathbf{y}$ will be computed using combinatorial ideas, such as simple case analysis or multicommodity flow. Thus our algorithms do adhere to the primal-dual philosophy.

Primal-Dual Algorithm for SDP

Set $\mathbf{X}^{(1)} = \frac{R}{n}\mathbf{I}$. Let $\varepsilon = \frac{\delta\alpha}{2\rho R}$, and let $\varepsilon' = -\ln(1-\varepsilon)$. Let $T = \frac{8\rho^2 R^2 \ln(n)}{\delta^2\alpha^2}$. For $t = 1, 2, \ldots T$:

1. Run the ORACLE with candidate solution $\mathbf{X}^{(t)}$.
2. If the ORACLE fails, stop and output $\mathbf{X}^{(t)}$.
3. Else, let $\mathbf{y}^{(t)}$ be the vector generated by ORACLE.
4. Let $\mathbf{M}^{(t)} = (\sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I})/2\rho$.
5. Compute $\mathbf{W}^{(t+1)} = (1-\varepsilon)^{\sum_{\tau=1}^t \mathbf{M}^{(\tau)}} = \exp(-\varepsilon'(\sum_{\tau=1}^t \mathbf{M}^{(\tau)}))$.
6. Set $\mathbf{X}^{(t+1)} = \frac{R\mathbf{W}^{(t+1)}}{\mathbf{Tr}(\mathbf{W}^{(t+1)})}$ and continue.

The following theorem bounds the number of iterations needed in the algorithm. Let $\mathbf{e}_1 = \langle 1, 0, \ldots, 0\rangle$.

Theorem 2 *In the Primal-Dual SDP algorithm, assume that the* ORACLE *never fails for* $T = \frac{8\rho^2 R^2 \ln(n)}{\delta^2\alpha^2}$ *iterations. Let* $\bar{\mathbf{y}} = \frac{\delta\alpha}{R}\mathbf{e}_1 + \frac{1}{T}\sum_{t=1}^T \mathbf{y}^{(t)}$. *Then* $\bar{\mathbf{y}}$ *is a feasible dual solution with objective value at most* $(1+\delta)\alpha$.

PROOF: This is a simple corollary of Theorem 1. Note that the Primal-Dual SDP algorithm is just simulating the Matrix Multiplicative Weights algorithm with the event matrices $\mathbf{M}^{(t)} = (\sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I})/2\rho$. The candidate solution $\mathbf{X}^{(t)}$ is just $R\mathbf{P}^{(t)}$ where $\mathbf{P}^{(t)}$ is the density matrix generated in the t^{th} round. Now, we have

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;=\; \frac{1}{2\rho}(\sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I}) \bullet \frac{1}{R}\mathbf{X}^{(t)} \;\geq\; \frac{1}{2}$$

since the ORACLE finds a $\mathbf{y}^{(t)}$ such that $(\sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C}) \bullet \mathbf{X}^{(t)} \geq 0$. Now, plugging this into inequality (2) from Theorem 1, dividing by T, and using the values of ε, T, and simplifying we get that $-\frac{\delta\alpha}{R} \leq \lambda_n(\frac{1}{T}\sum_{t=1}^T \sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C})$. Since we assumed that $\mathbf{A}_1 = \mathbf{I}$, by setting $\bar{\mathbf{y}}$ as in the theorem, we get that $\mathbf{0} \preceq \sum_{j=1}^m \mathbf{A}_j \bar{y}_j - \mathbf{C}$, which implies that $\bar{\mathbf{y}}$ is a dual feasible solution. Furthermore, since $b_1 = R$ and since all $\mathbf{y}^{(t)} \in \mathcal{D}_\alpha$, it is easy to check that $\mathbf{b} \cdot \bar{\mathbf{y}} \leq (1+\delta)\alpha$. □

As noted earlier, if all the matrices in question were diagonal, then the SDP reduces to an LP, and the algorithm reduces precisely to the standard Multiplicative Weights algorithm for LPs studied by many authors including Plotkin, Shmoys, Tardos [24] and Young [30]. (See the authors' survey with Hazan [7] for more details on the LP version.) Theorem 2 is a consequence of our more general Matrix Multiplicative Weights algorithm, explored in more detail in Theorem 1 of Section 2.1.

As a warmup, we illustrate the use of Theorem 2 in the following simple application.

Theorem 3 *In a d-regular graph* $G = (V, E)$ *with n nodes and m edges, the* MAXCUT *SDP can be approximated in* $\tilde{O}(m)$ *time.*

Remarks: For comparison, the previous best algorithm for approximating the MAXCUT SDP, by Klein and Lu [20], runs in time $\tilde{O}(mn)$. Our algorithm can be extended to the family of weighted graphs in which all weighted degrees are O(average degree).

PROOF: [of Theorem 3] The MAXCUT SDP in vector and matrix form is as follows (the standard SDP has a factor of $\frac{1}{4}$ in the objective, but we disregard it since the optimum solution is the same):

$$\begin{aligned} \max \sum_{\{i,j\}\in E} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \qquad & \qquad \max\ \mathbf{C} \bullet \mathbf{X} \\ \forall i \in [n]:\ \|\mathbf{v}_i\|^2 \;\leq\; 1 \qquad & \qquad \forall i \in [n]:\ \mathbf{X}_{ii} \;\leq\; 1 \\ & \qquad\qquad \mathbf{X} \;\succeq\; \mathbf{0} \end{aligned}$$

The dual SDP is the following:

$$\begin{aligned} \min \ & \textstyle\sum_{i=1}^{n} x_i \\ \mathrm{diag}(\mathbf{x}) \ & \succeq \ \mathbf{C} \\ \forall i \in [n]: \ x_i \ & \geq \ 0 \end{aligned}$$

Here, $\mathbf{C}$ is the combinatorial Laplacian of the graph, and $\mathrm{diag}(\mathbf{x})$ is the diagonal matrix with the vector $\mathbf{x}$ on the diagonal. Since the maximum degree in the graph is d, we have $\mathbf{0} \preceq \mathbf{C} \preceq 2d\mathbf{I}$.

We use the Primal-Dual SDP algorithm to solve this within a factor of $(1-\delta)$. We may assume that $nd \leq \alpha \leq 3nd$ since the optimum lies in this range. Furthermore, the trace of the desired $\mathbf{X}$ is n. This is the R parameter. We now show how to implement an ORACLE whose width parameter ρ is $O(d)$, which ensures by Theorem 2 that the number of iterations is $O(\log n)$. Each invocation of ORACLE and the matrix exponentiation step will take $\tilde{O}(m)$ time (the latter uses Lemma 7 and the fact that the number of non-zero matrix entries in $\mathbf{C}$ is $O(m)$). This yields the desired running time.

It remains to describe ORACLE. Given a candidate solution $\mathbf{X}$, it needs to find a vector $\mathbf{x} \geq \mathbf{0}$ such that $\sum_i x_i \leq \alpha$ and $\sum_i x_i \mathbf{X}_{ii} - \mathbf{C} \bullet \mathbf{X} \geq 0$. Intuitively, to make $\sum_i x_i \mathbf{X}_{ii}$ as large as possible, we should make x_i large for all i where $\mathbf{X}_{ii}$ is large. However, we always keep $x_i \leq O(\frac{\alpha}{n}) = O(d)$, since this ensures the desired width bound: $\|\mathrm{diag}(\mathbf{x}) - \mathbf{C}\| \leq O(d)$.

1. If $\mathbf{C} \bullet \mathbf{X} \leq \alpha$, then set all $x_i = \frac{\alpha}{n}$ Then since $\sum_i \mathbf{X}_{ii} = \mathbf{Tr}(\mathbf{X}) = n$ we have $\sum_i x_i \mathbf{X}_{ii} - \mathbf{C} \bullet \mathbf{X} \geq \frac{\alpha}{n} \sum_i \mathbf{X}_{ii} - \alpha = 0$.

2. So assume $\mathbf{C} \bullet \mathbf{X} \geq \alpha$. Let $\mathbf{C} \bullet \mathbf{X} = \lambda\alpha$ for some $\lambda \geq 1$. Since $\mathbf{C} \preceq 2d\mathbf{I}$, we have $\lambda\alpha = \mathbf{C} \bullet \mathbf{X} \leq 2nd$. Since $\alpha \geq nd$, we have that $\lambda \leq 2$. Let $S := \{i : \ \mathbf{X}_{ii} \geq \lambda\}$, and let $k := \sum_{i \in S} \mathbf{X}_{ii}$.

   If $k \geq \delta_1 n$ for some constant δ_1 then we set $x_i = \frac{\lambda\alpha}{k}$ for all $i \in S$, and $x_i = 0$ for all $i \notin S$. Then $\sum_i x_i = |S| \cdot \frac{\lambda\alpha}{k} \leq \alpha$ since $k = \sum_{i \in S} \mathbf{X}_{ii} \geq \lambda |S|$. Then $\sum_i x_i \mathbf{X}_{ii} - \mathbf{C} \bullet \mathbf{X} = \frac{\lambda\alpha}{k} \sum_{i \in S} \mathbf{X}_{ii} - \lambda\alpha \geq 0$.

3. If every other case we show that we can easily construct a feasible primal solution from $\mathbf{X}$ with objective value at least $(1-\delta)\alpha$, which is therefore approximately optimum.

   Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the vectors obtained from the Cholesky decomposition of $\mathbf{X}$. Construct new vectors $\mathbf{v}'_i$ such that $\mathbf{v}'_i = \mathbf{v}_i$ for $i \notin S$, and $\mathbf{v}'_i = \mathbf{v}_0$ for $i \in S$, for some fixed unit vector $\mathbf{v}_0$. Let $\tilde{\mathbf{X}}$ be the resulting Gram matrix of the $\mathbf{v}'_i$ vectors. Let E_S be the set of edges with at least one endpoint in S. Now we have $\mathbf{C} \bullet (\tilde{\mathbf{X}} - \mathbf{X}) \geq -\sum_{\{i,j\} \in E_S} \|\mathbf{v}_i - \mathbf{v}_j\|^2$. We can lower bound the RHS as follows:

$$\sum_{\{i,j\} \in E_S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ \leq \ \sum_{\{i,j\} \in E_S} 2[\|\mathbf{v}_i\|^2 + \|\mathbf{v}_j\|^2] \ \leq \ 2d \sum_{i \in S} \|\mathbf{v}_i\|^2 + 2d\lambda|S| \ \leq \ 4dk \ \leq \ 4\delta_1 nd.$$

   The third inequality follows because for all $i \in S$, $\|\mathbf{v}_i\|^2$ appears in at most d edges, and for all $j \notin S$, $\|\mathbf{v}_j\|^2 \leq \lambda$, and there can be at most $d|S|$ edges leading out of S. This implies that $\mathbf{C} \bullet \tilde{\mathbf{X}} \geq \lambda\alpha - 4\delta_1 nd$. Given an error parameter δ, if we choose $\delta_1 \leq \frac{\delta\lambda}{4}$, we can lower bound the RHS by $(1-\delta)\lambda\alpha$. Furthermore, for all i, $\tilde{\mathbf{X}}_{ii} = \|\mathbf{v}'_i\|^2 \leq \lambda$. So the matrix $\mathbf{X}^* = \frac{1}{\lambda}\tilde{\mathbf{X}}$ is a feasible primal solution with objective value at least $(1-\delta)\alpha$.

□

3.1 Extension to minimization problems

Since the rest of the paper concerns minimization problems, we extend the above framework to it. First, given a candidate solution $\mathbf{X}$, the ORACLE needs to find a vector $\mathbf{y}$ from the polytope $\mathcal{D}_\alpha = \{\mathbf{y} : \ \mathbf{y} \geq 0, \ \mathbf{b} \cdot \mathbf{y} \geq \alpha\}$ such that $\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}) y_j - (\mathbf{C} \bullet \mathbf{X}) < 0$. Second, the matrix exponential is computed with base $(1+\varepsilon)$ rather than $(1-\varepsilon)$.

Finally, and most important, we allow the ORACLE to find a matrix $\mathbf{F}^{(t)}$ such that for all primal feasible $\mathbf{X}$, we have $\mathbf{F}^{(t)} \bullet \mathbf{X} \leq \mathbf{C} \bullet \mathbf{X}$, and a vector $\mathbf{y}^{(t)} \in \mathcal{D}_\alpha$ such that

$$\textstyle\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j^{(t)} - (\mathbf{F}^{(t)} \bullet \mathbf{X}) \ \leq \ 0. \tag{4}$$

In this case, we use $\mathbf{M}^{(t)} := (\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{F}^{(t)} + \rho \mathbf{I})/2\rho$. In other words, we can replace $\mathbf{C}$ by $\mathbf{F}^{(t)}$, which is under our control. Note that if $\mathbf{F}^{(t)} \preceq \mathbf{C}$, then because any primal feasible $\mathbf{X}$ is *PSD*, we have $\mathbf{F}^{(t)} \bullet \mathbf{X} \leq \mathbf{C} \bullet \mathbf{X}$. So it suffices to find $\mathbf{F}^{(t)} \preceq \mathbf{C}$. (The reason for allowing $\mathbf{F}$ in the framework is to reduce width; see Section 4.)

The proof of Theorem 2 goes through with all these changes.

Theorem 4 *In the modified Primal-Dual algorithm for a minimization SDP as described above, if the* Oracle *never fails for* $T = \frac{8\rho^2 R^2 \ln(n)}{\delta^2 \alpha^2}$ *iterations, then* $\bar{\mathbf{y}} = \frac{\delta\alpha}{R}\mathbf{e}_1 + \frac{1}{T}\sum_{t=1}^{T} \mathbf{y}^{(t)}$ *is a feasible dual solution with dual objective value at least* $(1-\delta)\alpha$.

4 Primal-dual approximation algorithms via SDPs

In this section, we apply our general framework of Section 3.1 to design faster approximation algorithms for a host of NP-hard problems for which thus far we needed to solve SDPs. We give a few illustrative examples; the rest will appear in the complete paper. The important difference from Section 3 is that we do not try to solve the SDP to near-optimality as that would take too long. Instead, we use the framework to produce a dual solution of a certain value together with an *integer* primal solution whose cost is $O(\log n)$ or $O(\sqrt{\log n})$ factor higher than the value of the dual solution.

We now outline how to implement the Oracle, which, as mentioned in the Introduction, uses the known SDP rounding techniques (stemming from the Arora, Rao, Vazirani paper and subsequent work) for the problem in question. At each step the Oracle starts by applying the rounding algorithm on the current primal solution. Either the rounding succeeds, or else it fails so spectacularly that the Oracle can see a clear way to move the primal closer to feasibility. The main point is that the rounding could potentially succeed even though the primal is quite far from feasibility, which is why the algorithm may end only with a feasible dual solution. (Of course, the general framework of Section 3 could be used to continue the algorithm until it also finds a feasible primal, but the Oracle's width parameter increases, raising the running time a lot.) Thus the running time of the Oracle (and the algorithm) depends upon how efficiently it can compute the "feedback" when the rounding algorithm fails, and often this running time is much less for $O(\log n)$-approximation as compared to a $O(\sqrt{\log n})$-approximation.

The key insights in the implementation of the Oracle are that: (a) the feedback, in the form of dual weights, can be viewed as *Laplacians* of certain weighted graphs, whose spectral behavior is easy to understand, and this allows us to bound the width, and (b) the spectral behavior (in other words, the width bound) is improved by careful choice of these weights, and this was the main reason for allowing $\mathbf{F}$ instead of $\mathbf{C}$ in Theorem 4 in the first place.

4.1 Undirected Balanced Separator

We are given a capacitated graph $G = (V, E)$ with $|V| = n$, $|E| = m$, and capacity c_e on edge $e \in E$. A cut $(S, \bar{S})$ is called *c-balanced* if $|S| \geq cn$, and $|\bar{S}| \geq cn$. The minimum c-Balanced Separator problem is to find the c-balanced cut with minimum capacity. A t pseudo-approximation to the minimum c-Balanced Separator is a c'-balanced cut for some other constant c' whose expansion is within a factor t of that of the minimum c-Balanced Separator.

Theorem 5

1. *An* $O(\log n)$ *pseudo-approximation to the minimum* c-Balanced Separator *can be computed in* $\tilde{O}(m + n^{1.5})$ *time using* $O(\log^2(n))$ *single commodity flow computations.*
2. *An* $O(\sqrt{\log n})$ *pseudo-approximation to the minimum* c-Balanced Separator *can be computed in* $\tilde{O}(n^2)$ *time using* $O(\log n)$ *multicommodity flow computations.*

Proof: First, we consider the well-known c-Balanced Separator SDP. To increase the flexibility in handling the candidate solution $\mathbf{X}$ we throw in additional constraints, which are not part of the standard SDP specification, but which are implied by it. (The reason is that the candidate primal solutions at

intermediate steps are not feasible, so these constraints are actually helpful to ORACLE.) We assign vectors $\mathbf{v}_i$ to the nodes in G. Let $\mathbf{X}$ be the Gram matrix of these vectors. Below, we write the SDP in vector and its corresponding matrix form. Let $\mathbf{C}$ be the combinatorial Laplacian of the graph, and for any subset S of the nodes, let $\mathbf{K}_S$ be the Laplacian of the graph where all nodes in S are connected by edges, and all other edges are absent. We will only consider sets S of size at least $(1-\varepsilon)n$ (for some ε to be fixed later), and we use the notation "$\forall S$" to mean only such sets. Let $p = (i_1, i_2, \ldots, i_k)$ be a generic path of nodes in the complete graph, and let $\mathbf{T}_p$ be the difference of the Laplacian of p and that of a single edge connecting its endpoints.

$$\begin{array}{rcl} \min \sum_{e=\{i,j\}\in E} c_e\|\mathbf{v}_i - \mathbf{v}_j\|^2 & & \\ \forall i: \quad \|\mathbf{v}_i\|^2 & = & 1 \\ \forall p: \quad \sum_{j=1}^{k-1}\|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \\ \forall S: \quad \sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 & \geq & an^2 \end{array} \qquad \begin{array}{rcl} \min \mathbf{C}\bullet\mathbf{X} & & \\ \forall i: \quad \mathbf{X}_{ii} & = & 1 \\ \forall p: \quad \mathbf{T}_p\bullet\mathbf{X} & \geq & 0 \\ \forall S: \quad \mathbf{K}_S\bullet\mathbf{X} & \geq & an^2 \\ \mathbf{X} & \succeq & 0 \end{array}$$

The path inequalities, $\mathbf{T}_p \bullet \mathbf{X} \geq 0$, are usually omitted from the standard c-BALANCED SEPARATOR SDP, which only involves triangle inequalities. However, they are implied by the triangle inequalities, so we retain them. Similarly, the spreading constraints $\mathbf{K}_S \bullet \mathbf{X} \geq an^2$ are also omitted, but they are implied for $a = 4[c(1-c) - \varepsilon]$, so we retain them. The optimum of this SDP divided by 4 is a lower bound on the minimum c-BALANCED SEPARATOR.

The dual SDP is the following. It has variables x_i for every node i, f_p for every path p, and z_S for every set S of size at least $(1-\varepsilon)n$. Let $\mathrm{diag}(\mathbf{x})$ be the diagonal matrix with the vector $\mathbf{x}$ on the diagonal.

$$\begin{array}{rcl} \max \ \sum_i x_i + an^2\sum_S z_S & & \\ \mathrm{diag}(\mathbf{x}) + \sum_p f_p\mathbf{T}_p + \sum_S z_S\mathbf{K}_S & \preceq & \mathbf{C} \\ \forall p, S: \quad f_p, z_S & \geq & 0 \end{array}$$

Let the current guess for the optimum in the Primal-Dual algorithm be α. Let $\mathbf{X}$ be a candidate solution generated by the Primal-Dual algorithm. Note that $\mathbf{Tr}(\mathbf{X}) = n$. The ORACLE needs to find variables x_i, $f_p \geq 0$, $z_S \geq 0$ and a matrix variable $\mathbf{F} \preceq \mathbf{C}$ (c.f. Theorem 4) such that $\sum_i x_i + an^2\sum_S z_S \geq \alpha$ which satisfies

$$\mathrm{diag}(\mathbf{x})\bullet\mathbf{X} + \sum_p f_p(\mathbf{T}_p\bullet\mathbf{X}) + \sum_S z_S(\mathbf{K}_S\bullet\mathbf{X}) - (\mathbf{F}\bullet\mathbf{X}) \ \leq \ 0.$$

If it succeeds in doing this, then the matrix returned as feedback is $\mathrm{diag}(\mathbf{x}) + \sum_p f_p\mathbf{T}_p + \sum_S z_S\mathbf{K}_S - \mathbf{F}$.

Our implementation of ORACLE works as follows. Given a candidate solution $\mathbf{X}$, the ORACLE checks first whether all the $\mathbf{X}_{ii}$ are $O(1)$. If a significant fraction of them aren't, then the ORACLE can punish the solution $\mathbf{X}$ by setting the x_i's appropriately, in a similar way as done for MAXCUT. Next, it checks whether $\mathbf{K}_V \bullet \mathbf{X} \geq \Omega(n^2)$. If it isn't, then by setting some z_S appropriately, it can punish $\mathbf{X}$.

The most complicated case is when almost all $\mathbf{X}_{ii}$ are $O(1)$, and $\mathbf{K}_V \bullet \mathbf{X} \geq \Omega(n^2)$. In this case, we perform a flow computation, and interpret the f_p variables as a multicommodity flow in the graph.

Now, we describe in detail the implementation of ORACLE, which ensures that the width is $\tilde{O}(\frac{\alpha}{n})$. Given a candidate solution $\mathbf{X}$, the ORACLE runs the following steps (all unspecified variables, including $\mathbf{F}$, are set to 0):

1. Assume, without loss of generality, that $\mathbf{X}_{11} \leq \mathbf{X}_{22} \leq \cdots \leq \mathbf{X}_{nn}$. Let $h = (1-\varepsilon)n + 1$. If $\mathbf{X}_{hh} \geq 2$, then set $x_i = -\frac{\alpha}{\varepsilon n}$ for $i \geq k$, and $x_i = \frac{2\alpha}{(1-\varepsilon)n}$ for $i < k$. Then

$$\mathrm{diag}(\mathbf{x})\bullet\mathbf{X} \ = \ \sum_{i\geq k} -\frac{\alpha}{\varepsilon n}\mathbf{X}_{ii} + \sum_{i\geq k}\frac{2\alpha}{(1-\varepsilon)n}\mathbf{X}_{ii} \ \leq \ -\frac{\alpha}{\varepsilon n}\cdot 2\cdot\varepsilon n + \frac{2\alpha}{(1-\varepsilon)n}\cdot(n-2\varepsilon n) \ \leq \ 0.$$

   Finally, since all $x_i = O(\frac{\alpha}{n})$, $\|\mathrm{diag}(\mathbf{x})\| \leq O(\frac{\alpha}{n})$.

2. Now we assume that for all but εn exceptional nodes i, $\mathbf{X}_{ii} \leq 2$. Let W be the set of all the exceptional nodes, and let $S := V \setminus W$. Note that $|S| \geq (1-\varepsilon)n$, so we have the constraint $\mathbf{K}_S \bullet \mathbf{X} \geq an^2$ in the SDP. If $\mathbf{K}_S \bullet \mathbf{X} \leq \frac{an^2}{2}$, then choose $z_S = \frac{2\alpha}{an^2}$ all $x_i = -\frac{\alpha}{n}$. Then

$$\left(-\frac{\alpha}{n}\mathbf{I} + \frac{2\alpha}{an^2}\mathbf{K}_S\right) \bullet \mathbf{X} \;\leq\; \alpha - \alpha \;=\; 0.$$

Also, since $\mathbf{0} \preceq \mathbf{K}_S \preceq n\mathbf{I}$, we have $\| -\frac{\alpha}{n}\mathbf{I} + \frac{2\alpha}{an^2}\mathbf{K}_S\| \leq O(\frac{\alpha}{n})$.

3. Now assume that $\mathbf{K}_S \bullet \mathbf{X} \geq \frac{an^2}{2}$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}$. Note that for all nodes $i \in S$, we have $\|\mathbf{v}_i\|^2 \leq 2$. Also, $\mathbf{K}_S \bullet \mathbf{X} \geq \frac{an^2}{2}$ implies that $\sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{an^2}{2}$.

   Now, the algorithm nudges $\mathbf{X}$ towards satisfying the path inequality constraints. At first, it is even unclear how to *check* at any time that the path inequalities are satisfied, since there are so many of them. For this we use multicommodity flow. First, some notation. For a flow which assigns value f_p to path p define f_e to be the flow on edge e, i.e. $f_e := \sum_{p \ni e} f_p$. Define f_i to be the total flow from node i, i.e. $f_i = \sum_{p\in\mathcal{P}_i} f_p$ where $\mathcal{P}_i$ is the set of paths starting from i. Finally, define f_{ij} to be the total flow between nodes i, j, i.e. $f_{ij} = \sum_{p\in\mathcal{P}_{ij}} f_p$, where $\mathcal{P}_{ij}$ is the set of paths from i to j. A *valid d-regular flow* is one that satisfies the capacity constraints: $\forall e:\ f_e \leq c_e$, and $\forall i:\ f_i \leq d$.

   Our main tool is the following lemma, which shows that either we can find a nice flow to make progress (i.e., give substantial "feedback"), or a cut with the desired expansion (i.e., a near-optimal integer solution). This lemma can be proved using the techniques of [8, 21]:

   Lemma 1 *Let $S \subseteq V$ be a set of nodes of size $\Omega(n)$. Suppose we are given, for all $i \in S$, vectors $\mathbf{v}_i$ of length $O(1)$, such that $\sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$, and a quantity α. Then:*

   (a) There is an algorithm, which, using a single max-flow computation, either outputs a valid $O(\frac{\log(n)\alpha}{n})$-regular flow f_p such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or a c'-balanced cut of expansion $O(\log(n)\frac{\alpha}{n})$.

   (b) There is an algorithm, which, using a single multicommodity flow computation, either outputs a valid $O(\frac{\alpha}{n})$-regular flow f_p such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or a c'-balanced cut of expansion $O(\sqrt{\log(n)}\frac{\alpha}{n})$.

   We apply the two algorithms of Lemma 1 to the set S, corresponding to the two cases of Theorem 5. In case we find a cut with the desired expansion, then we stop. Otherwise, we get a valid d-regular flow which satisfies $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, where $d = O(\frac{\log(n)\alpha}{n})$ or $O(\frac{\alpha}{n})$ depending on the two cases.

   Now, we set $\mathbf{F}$ to be the Laplacian of the weighted graph with edge weights f_e. The capacity constraints $f_e \leq c_e$ imply that $\mathbf{F} \preceq \mathbf{C}$. Let $\mathbf{D}$ be the Laplacian of the complete weighted graph where only edges $\{i, j\}$ with $i \in S$ and $j \in T$ have weight f_{ij}, and the rest have 0 weight.

   Now, we have that $\mathbf{D} \bullet \mathbf{X} = \sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 2\alpha$. Then we set all $x_i = \frac{\alpha}{n}$, and all $z_S = 0$. It can be checked easily that $\sum_p f_p \mathbf{T}_p = \mathbf{F} - \mathbf{D}$. Thus, the "feedback" matrix becomes

$$\mathrm{diag}(\mathbf{x}) + \mathbf{F} - \mathbf{D} - \mathbf{F} \;=\; \mathrm{diag}(\mathbf{x}) - \mathbf{D}.$$

   Then $\left(\frac{\alpha}{n}\mathbf{I} - \mathbf{D}\right) \bullet \mathbf{X} \leq \alpha - \alpha = 0$. Also, since the flow is d-regular, we have $\mathbf{0} \preceq \mathbf{D} \preceq 2d\mathbf{I}$. Hence, $-2d\mathbf{I} \;\preceq\; \frac{\alpha}{n}\mathbf{I} - \mathbf{D} \;\preceq\; \frac{\alpha}{n}\mathbf{I}$.

We now estimate the running time for each algorithm. We assume the graph has been preprocessed using the algorithm of Benczúr and Karger [10], to leave only $\tilde{O}(n)$ edges.

1. We have $\rho = O(\frac{\log(n)\alpha}{n})$ and $R = n$. Thus, the number of iterations, from Theorem 2 is $O(\log^3(n))$. However, we can take advantage of the fact that the width is "one-sided" (see [7]), to tweak the matrix update rule to have only $O(\log^2(n))$ iterations. Each iteration involves at most one max-flow

computation, which can be done in $\tilde{O}(n^{1.5})$ time since there are $\tilde{O}(n)$ edges, using the algorithm of Goldberg and Rao [15].

In each iteration, we compute an approximation to the Cholesky decomposition of the matrix exponential by projecting on a random $O(\log n)$ dimensional subspace. Furthermore, since there are only $O(\log^2(n))$ iterations and each iteration adds at most $\tilde{O}(n^{1.5})$ demand pairs in the max-flow computation, the matrix to be exponentiated has only $\tilde{O}(n^{1.5})$ non-zero entries (we are disregarding the $\mathbf{K}_S$ matrices here, but it is easy to compute the product $\mathbf{K}_S\mathbf{u}$ for any vector $\mathbf{u}$ in $O(n)$ time). Overall, Lemma 6 shows that the matrix exponentiation step can be done in $\tilde{O}(n^{1.5})$ time.

Overall, the running time, accounting for the initial sparsification, becomes $\tilde{O}(m + n^{1.5})$.

2. We have $\rho = O(\frac{\alpha}{n})$ and $R = n$. Thus, the number of iterations, from Theorem 2 is $O(\log(n))$. Each iteration involves at most one maximum multicommodity flow computation, which can be done in $\tilde{O}(n^{1.5})$ time since there are $\tilde{O}(n)$ edges, using the algorithm of Fleischer [12].

   In each iteration, Fleischer's multicommodity flow algorithm adds at most $\tilde{O}(n)$ demand pairs. Since there are only $O(\log n)$ iterations, the matrix to be exponentiated has only $\tilde{O}(n)$ non-zero entries (again disregarding the $\mathbf{K}_S$ matrices). Overall, Lemma 6 shows that the matrix exponentiation step can be done in $\tilde{O}(n)$ time.

   Overall, the running time becomes $\tilde{O}(n^2)$.

□

Finally, we turn to the proof of Lemma 1.

PROOF:[Lemma 1]

Part 1. We seek a valid d-regular flow f_p for $d := \frac{\beta \log(n)\cdot\alpha}{n}$ where β is a sufficiently large constant to be chosen later. For this, we choose a direction represented by a unit vector $\mathbf{u}$ at random. Since $\mathbf{K}_S \bullet \mathbf{X} \geq \Omega(n^2)$, we have that $\sum_{ij\in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$. Then Lemma 10 in Appendix B shows that we can find sets L and R of size cn each, for some constant $c > 0$, such that for all $i \in L$ and $j \in R$, we have $(\mathbf{v}_j - \mathbf{v}_i)\cdot\mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$ for some constant $\sigma > 0$. Using the Gaussian nature of projections, with very high probability, for any pair of nodes i, j, we have that $|(\mathbf{v}_i - \mathbf{v}_j)\cdot\mathbf{u}| \leq O(\sqrt{\log(n)})\cdot\frac{\|\mathbf{v}_i-\mathbf{v}_j\|}{\sqrt{n}}$. Thus, we conclude that with constant probability, we get sets L and R such that for all nodes $i \in L$ and $j \in R$, we have $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{\gamma}{\log(n)}$ for some constant γ.

Assuming this is the case, we connect all nodes in L to a single source and connect all nodes in R to a single sink with edges of capacity d each. Let f_p be the max flow in this network (which involves the entire graph G, not just nodes in S). Suppose the total flow obtained is at least $\frac{c\beta}{2}\log(n)\cdot\alpha$. We may assume that all the flow originates from some node $i \in L$ and ends at some node $j \in R$. Then we have

$$\sum_{i\in L, j\in R} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\geq\; \frac{c\beta}{2}\log(n)\cdot\alpha\times\frac{\gamma}{\log(n)} \;=\; 2\alpha$$

if we choose $\beta = \frac{4}{c\gamma}$.

Now suppose that the total flow obtained in the previous step is less than $\frac{c\beta}{2}\log(n)\cdot\alpha$. By the max-flow-min-cut theorem, the cut obtained is also at most this size. Note that this cut will be $c/2$-balanced, since at most $\frac{c\beta}{2}\log(n)\cdot\alpha/d = cn/2$ source (and sink) edges can be cut. Thus, the expansion of the cut is $O(\log(n)\cdot\frac{\alpha}{n})$.

Part 2: We seek a valid d-regular flow f_p for $d := \frac{\beta\alpha}{n}$ where β is a sufficiently large constant to be chosen later. Now we consider a maximum multicommodity flow problem, where the demand pairs are nodes $i, j \in S$ such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s to be specified later. Let D be the set of such pairs of nodes. We seek a valid d-regular flow for $d := \frac{\beta\alpha}{n}$, which maximizes the total flow $\sum_{ij\in D} f_{ij}$.

Using Fleischer's algorithm [12], this problem can be solved up to any given constant factor, say $\frac{1}{2}$. Now we check if the flow obtained satisfies $\sum_{ij\in D} f_{ij} \geq \frac{2\alpha}{s}$. If it does, then we have $\sum_{ij\in D} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 2\alpha$.

Assume now that the total flow obtained $\sum_{ij} f_{ij} < \frac{2\alpha}{s}$. Since we chose an approximation factor of $\frac{1}{2}$ in Fleischer's algorithm, this means that the optimum of the multicommodity flow problem is less than $\frac{4\alpha}{s}$. By LP duality, this means that there is a set of edge weights w_e and node weights s_i such that

$$\sum_e c_e w_e + \sum_i \frac{\beta\alpha}{n} s_i \;\leq\; \frac{4\alpha}{s},$$

such that for any demand pair $i, j \in D$ and a path p connecting them, we have

$$s_i + s_j + \sum_{e \in p} w_e \ \geq \ 1.$$

Now we apply the algorithm from the following theorem (with the vectors $\mathbf{v}_i$ scaled down by 2 to ensure their length is at most 1). The proof, which appears in Appendix B, can be derived using the techniques of Arora, Rao and Vazirani [8] and Lee [21].

Theorem 6 *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors of length at most 1, such that $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq an^2$. Let w_e be weights on edges and nodes such that $\sum_e c_e w_e \leq \alpha$. Then there is an algorithm which finds a cut of value C which is c-balanced for some constant c, such that there exists a pair of nodes i, j with the property that the graph distance between i and j is at most $O(\sqrt{\log n} \cdot \frac{\alpha}{C})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ where s is a constant which only depends on a. Furthermore, this is true even if any fixed set of τn nodes are prohibited from being i or j, for some small constant τ.*

At most τn nodes have $s_i \geq \frac{4}{\beta \tau s}$. Let these nodes form the forbidden set in the theorem. We run the algorithm from the theorem to obtain a cut of value C. Let i, j be the pair of nodes whose existence is guaranteed by the theorem. The value s is defined to be the lower bound on $\|\mathbf{v}_i - \mathbf{v}_j\|^2$ from the theorem; thus, i, j is a demand pair in D. Choose $\beta = \frac{16}{\tau s}$. Then $s_i, s_j \leq \frac{1}{4}$, and we have

$$s_i + s_j + O\left(\sqrt{\log n} \frac{\alpha}{C}\right) \ \geq \ 1 \quad \Rightarrow \quad C \ = \ O(\sqrt{\log n} \cdot \alpha).$$

Since the cut is $\Omega(1)$-balanced, its expansion is at most $O(\sqrt{\log n} \frac{\alpha}{n})$. □

4.2 Undirected Sparsest Cut

We have the same setup as in the previous section. The Sparsest Cut in a graph $G = (V, E)$ is the cut $(S, \bar{S})$ with minimum expansion, $\frac{E(S,\bar{S})}{\min\{|S|, |\bar{S}|\}}$.

Theorem 7

1. *An $O(\log n)$ pseudo-approximation to the* Sparsest Cut *can be computed in $\tilde{O}(m + n^{1.5})$ time using $O(\log^2(n))$ single commodity flow computations.*
2. *An $O(\sqrt{\log n})$ pseudo-approximation to the* Sparsest Cut *can be computed in $\tilde{O}(n^2)$ time using $O(\log n)$ multicommodity flow computations.*

Details of the Oracle are omitted. The key lemma in this case is the following:

Lemma 2 *Suppose we are given, for all $i \in V$, vectors $\mathbf{v}_i$, such that for some constant δ_1, $n^2 \geq \sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq (1 - \delta_1)n^2$, and a quantity α. Then there is an algorithm, which, using a single max-flow computation, outputs either*

1. *a valid $O(\frac{\alpha}{n})$-regular flow f_p, such that $\sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or,*
2. *a cut of expansion $O(\frac{\alpha}{n})$, or*
3. *a set of nodes $S \subseteq V$ of size $\Omega(n)$, such that for all $i \in S$, $\|\mathbf{v}_i\|^2 = O(1)$, and $\sum_{i,j \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$.*

Note that in the third case, Lemma 1 applies, and thus the problem is reduced to Balanced Separator.

Connection to KRV's Sparsest Cut algorithm. Khandekar, Rao and Vazirani [19] obtain an $O(\log^2 n)$ approximation to the Balanced Separator and Sparsest Cut in undirected graphs in $\tilde{O}(m + n^{1.5})$ time. We obtain $O(\log n)$-approximation in the same time. Here we note that despite superficial differences, their algorithm is a close cousin of ours. At each iteration, their algorithm maintains a (multi)graph that is a union of perfect matchings. It uses spectral methods (specifically, a random walk) to identify sparse cuts in this graph. Then it computes a single-commodity max flow across this sparse cut, finds a new perfect matching via flow decomposition, and adds it to the current multigraph before proceeding to the next iteration. The

analysis of convergence uses an *ad hoc* potential function, and the main theorem says that the union of matchings converges in $O(\log^2 n)$ iterations to an *expander*.

Our algorithm is somewhat similar except we use matrix exponentiation at each iteration instead of random walks. However, the two are related. If $\mathbf{L}$ is a graph Laplacian, and $\beta < 1/(\text{max degree})$ is any constant, then $\exp(-\beta\mathbf{L})$ is the transition matrix after 1 time unit for the following continuous random walk: in each time interval δt, every node sends out a $\beta\delta t$ fraction of its probability mass to each of its neighbors. The algorithm of [19] simulates such a random walk, where $\mathbf{L}$ is the Laplacian of the union of the perfect matchings found so far. In our case, $\mathbf{L}$ is the Laplacian of the union of the flows found so far. Both algorithms compute a projection of the rows of the transition matrix on a random vector and then compute a max-flow based on the projections.

Of course, their *ad hoc* analysis does not apply to the other problems considered in this paper.

4.3 Directed Balanced Separator

We have a directed graph $G = (V, E)$. For a cut $(S, \bar{S})$ in the graph, define $E(S, \bar{S})$ to be the total capacity of arcs going from S to $\bar{S}$. The minimum c-Balanced Separator is the c-balanced cut $(S, \bar{S})$ with minimum value of $E(S, \bar{S})$.

Theorem 8 *1. An $O(\log n)$ pseudo-approximation to the minimum c-Balanced Separator in directed graphs can be computed in $\tilde{O}(m^{1.5})$ time using polylog(n) single-commodity flow computations.*

2. An $O(\sqrt{\log n})$ pseudo-approximation to the minimum c-Balanced Separator in directed graphs can be computed in $\tilde{O}(m^{1.5} + n^{2+\varepsilon})$ time using polylog(n) single-commodity flow computations, for any specified constant ε. The $\tilde{O}$ notation hides inverse polynomial dependence on $\frac{1}{\varepsilon}$.

Details of Oracle appear in the Appendix A.2. The implementation relies on the following key lemma:

Lemma 3 *Let $S \subseteq V$ be a set of nodes of size $\Omega(n)$. Suppose we are given, for all $i \in S$, vectors $\mathbf{v}_i, \mathbf{w}_i$ of length $O(1)$, such that $\sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$, and $\forall ij, \|\mathbf{w}_i - \mathbf{w}_j\|^2 \leq o(1)$. Define the directed distance $d(i,j) = \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2$. For any given value α,*

1. There is an algorithm, which, using a single max-flow computation, either outputs a valid $O(\frac{\log(n)\alpha}{n})$-regular directed flow f_p such that $\sum_{ij} f_{ij} d(i,j) \geq \alpha$, or a c'-balanced cut of expansion $O(\log(n)\frac{\alpha}{n})$.

2. There is an algorithm, which, using $O(\log n)$ max-flow computations, outputs either:

(a) a c'-balanced cut of expansion $O(\sqrt{\log(n)}\frac{\alpha}{n})$, or

(b) a valid $O(\frac{\alpha}{n})$-regular directed flow f_p flow such that$\sum_{ij} f_{ij} d(i,j) \geq \alpha$, or

(c) $\Omega(\frac{n}{\sqrt{\log n}})$ vertex-disjoint paths such that the path inequality along these paths is violated by $\Omega(1)$.

4.4 Directed Sparsest Cut

We have the same setup as in the previous section. The Sparsest Cut in a directed graph $G = (V, E)$ is the cut $(S, \bar{S})$ with minimum expansion, $\frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}$.

Theorem 9 *1. An $O(\log n)$ pseudo-approximation to the Sparsest Cut in directed graphs can be computed in $\tilde{O}(m^{1.5})$ time using polylog(n) single-commodity flow computations.*

2. An $O(\sqrt{\log n})$ pseudo-approximation to the Sparsest Cut in directed graphs can be computed in $\tilde{O}(m^{1.5} + n^{2+\varepsilon})$ time using polylog(n) single-commodity flow computations, for any specified constant ε. The $\tilde{O}$ notation hides inverse polynomial dependence on $\frac{1}{\varepsilon}$.

This algorithm is exactly analogous to the algorithm for Undirected Sparsest Cut, combined with some ideas from Directed Balanced Separator, so we omit the details of the implementation of the Oracle.

4.5 Min UnCut

The Min UnCut problem has various equivalent forms. The one we will use is the following. We are given a capacitated graph $G = (V, E)$ on the nodes $V = \{-n, \ldots, -2, -1, 1, 2, \ldots, n\}$ such that $\{i, j\} \in E$ iff there $\{-j, -i\} \in E$. Also, we assume that the capacities satisfy $c_{ij} = c_{-j,-i}$. A cut $(S, \bar{S})$ is called symmetric if $\bar{S} = -S$ where $-S = \{-i : i \in S\}$. The Min UnCut problem is to find the minimum symmetric cut in G.

Theorem 10 *An $O(\sqrt{\log n})$ approximation to* Min UnCut *can be computed in $\tilde{O}(n^3)$ time.*

Details of Oracle appear in the Appendix A.3. The implementation relies on the following key lemma:

Lemma 4 *Assume we are given vectors $\mathbf{v}_i$ for every node in G such that for all i, $\|\mathbf{v}_i\|^2 = \Theta(1)$ and $\mathbf{v}_i = -\mathbf{v}_{-i}$ and a value α. Then there is an algorithm, which, using a single multicommodity flow computation, can obtain either:*

1. *a valid $O(\alpha)$-regular flow f_p, such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or*
2. *a symmetric cut $(S, -S)$ of value $O(\sqrt{\log n} \cdot \alpha)$.*

5 Computing the Matrix Exponential

In our general framework of Section 3, the candidate solution $\mathbf{X}^{(t)}$ at each step is $\exp(\mathbf{M})$ for some $\mathbf{M}$. We show how to do this exponentiation faster than the trivial $O(n^3)$ time for the SDPs considered here.

The idea is that approximate computation suffices. Let α be our current estimate of the optimum and $\delta > 0$ be such that we desire a dual solution of cost at most $(1 + \delta)\alpha$. The Oracle's task, when given a candidate solution $\mathbf{X}^{(t)}$, is to find appropriate dual variables $y_1, \ldots, y_m$ such that $\sum_{i=1}^{j} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \geq 0$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}^{(t)}$ such that $\mathbf{X}_{ij}{}^{(t)} = \mathbf{v}_i \cdot \mathbf{v}_j = \frac{1}{2}[\|\mathbf{v}_i\|^2 + \|\mathbf{v}_j\|^2 - \|\mathbf{v}_i - \mathbf{v}_j\|^2]$. Thus Oracle's task is to find appropriate variables s_i and t_{ij} for $i, j \in [n]$ such that $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 0$. (Note that the s_i and t_{ij} variables cannot be set independently, since they are a linear transformation of $y_1, \ldots, y_m$.)

The vectors $\mathbf{v}_i$ obtained from the Cholesky decomposition of $\mathbf{X}^{(t)} = \exp(\mathbf{M})$ are just the row vectors of $\exp(\frac{1}{2}\mathbf{M})$. Since we are only interested in the square lengths of the row vectors and their differences, we can try Johnson-Lindenstrauss dimension reduction. If we project the vectors $\mathbf{v}_i$ on a random $d = O(\frac{\log n}{\delta^2})$ dimensional subspace, and scale the projections up by $\sqrt{\frac{n}{d}}$ to get vectors $\mathbf{v}'_i$, then by the Johnson-Lindenstrauss lemma, with high probability, the squared lengths $\|\mathbf{v}'_i\|^2$ and $\|\mathbf{v}'_i - \mathbf{v}'_j\|^2$ are within $(1 \pm \delta)$ of $\|\mathbf{v}_i\|^2$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2$ respectively, for all $i, j \in [n]$. Thus, we could run the Oracle with the $\mathbf{X}'$ which is the Gram matrix of the vectors $\mathbf{v}'_i$, and hope that the Oracle's "feedback" for $\mathbf{X}'$ would be also valid for $\mathbf{X}^{(t)}$. (Note that the exponential is only used for the Oracle's computation at this step, and never used again in the algorithm.)

Now we mention why Johnson-Lindenstrauss dimension reduction does not suffice in general, and then state conditions under which it does. (All our Oracles satisfy these conditions.) The problem is that the s_i and t_{ij} variables could take both positive and negative values, so $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2$ may no longer be non-negative, even though the corresponding sum for the $\mathbf{v}'_i$'s is. But the difference between the two is at most $\delta \sum_i |s_i| \|\mathbf{v}'_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}'_i - \mathbf{v}'_j\|^2$. So if the Oracle can also ensure that $\sum_i |s_i| \|\mathbf{v}'_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}'_i - \mathbf{v}'_j\|^2 = O(\alpha)$, where α is the current estimate of the optimum, then $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq -O(\delta\alpha)$, and it can be shown that Theorem 2 holds even with this relaxation for the Oracle. The following lemma describes the conditions for this idea to work.

Lemma 5 *In the above setting if the* Oracle *always finds values for the s_i's and t_{ij}'s such that $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 0$ and also $\sum_i |s_i| \|\mathbf{v}_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq O(\alpha)$, then the algorithm works even if instead of $\mathbf{v}_i$, the* Oracle *uses vectors $\mathbf{v}'_i$ obtained by Johnson-Lindenstrauss dimension reduction as above.*

The lemma also works for maximization problems (such as MaxCut) by reversing the inequalities.

Thus, our task is now reduced to finding the projection of the vectors $\mathbf{v}_i$ on a given unit vector $\mathbf{u}$. Note that $\exp(\mathbf{A}) = \exp(\frac{1}{2}\mathbf{A}) \exp(\frac{1}{2}\mathbf{A})$, so it easy to compute the Cholesky decomposition of a matrix exponential. The Johnson-Lindenstrauss lemma requires computing the projection of the vectors $\mathbf{v}_i$ onto

$O(\frac{\log n}{\delta^2})$ random vectors $\mathbf{u}$, which is simply $\exp(\frac{1}{2}\mathbf{M})\mathbf{u}$. This has fast implementations in solvers for linear differential equations, and one can give good complexity bounds.

It suffices to do this approximately: all we have to do is find a vector $\mathbf{v}$ such that $\|\mathbf{v} - \exp(\frac{1}{2}\mathbf{M})\mathbf{u}\| \leq \varepsilon$ for some inverse polynomial ε. One obvious way to do this is to use the Taylor series expansion for $\exp(\frac{1}{2}\mathbf{M})$, which exploits the sparsity of $\mathbf{M}$. A matrix vector product $\mathbf{Mu}$ can be computed in time which is proportional to the number of non-zero entries of $\mathbf{M}$.

Lemma 6 *Let $t_{\mathbf{M}}$ be the time needed to compute the matrix-vector product $\mathbf{Mu}$. Then the vector $\mathbf{v} = \sum_{i=0}^{k} \frac{\mathbf{M}^i}{i!}\mathbf{u}$ can be computed in $O(kt_{\mathbf{M}})$ time and if $k \geq \max\{e^2\|\mathbf{M}\|, \ln(\frac{1}{\varepsilon})\}$, then $\|\exp(\mathbf{M})\mathbf{u} - \mathbf{v}\| \leq \varepsilon$.*

PROOF: It is easy to see that $\mathbf{v}$ can be computed in $O(kt_{\mathbf{M}})$ by k matrix-vector multiplications. It remains to bound $\|\exp(\mathbf{M})\mathbf{u} - \mathbf{v}\|$. This can be bounded as follows. Let $\lambda = \|\mathbf{M}\|$. We have

$$\|\exp(\mathbf{M})\mathbf{u} - \mathbf{v}\| \;=\; \left\|\exp(\mathbf{M})\mathbf{u} - \textstyle\sum_{i=0}^{k} \frac{\mathbf{M}^i}{i!}\mathbf{u}\right\| \;\leq\; \left\|\exp(\mathbf{M}) - \textstyle\sum_{i=0}^{k} \frac{\mathbf{M}^i}{i!}\right\| \;\leq\; \textstyle\sum_{i=k+1}^{\infty} \frac{\lambda^i}{i!}$$

The last inequality follows because the eigenvalues of $\exp(\mathbf{M}) - \sum_{i=0}^{k} \frac{\mathbf{M}^i}{i!}$ are $\sum_{i=k+1}^{\infty} \frac{\mu^i}{i!}$ where μ is a generic eigenvalue of $\mathbf{M}$. Now, using standard approximations for the factorial function, it is easy to see that if $k \geq \max\{e^2\lambda, \ln(\frac{1}{\varepsilon})\}$, then $\sum_{i=k+1}^{\infty} \frac{\lambda^i}{i!} \leq \varepsilon$. □

Since the matrices exponentiated in the Primal-Dual SDP algorithm are $-\varepsilon' \sum_{\tau=1}^{t} \mathbf{M}^{(t)}$, for $t = 1, 2, \ldots, T$, and since for all t, $\mathbf{0} \preceq \mathbf{M}^{(t)} \preceq \mathbf{I}$, we conclude that $\|-\varepsilon' \sum_{\tau=1}^{t} \mathbf{M}^{(t)}\| \leq \varepsilon' T$. Since $\varepsilon' \approx \varepsilon$, this can be bounded by $O(\frac{\rho R \ln(n)}{\delta\alpha})$. So when this is small (such as $O(\log(n))$ as in the case of MAXCUT, BALANCED SEPARATOR and SPARSEST CUT (both directed and undirected), this simple method is good enough.

However, in some situations, this quantity may be large (like in the case of MIN UNCUT, where it is $O(n)$), it may be prohibitively expensive to use this method. We now present a more sophisticated method, the *Shift-and-Invert Lanczos* (SI-Lanczos) method of van den Eshof and Hochbruck [27], adapted by Iyengar, Phillips, and Stein [17] for the very setting we have.

Let $\mathbf{M} := \sum_{\tau=1}^{t} \mathbf{M}^{(t)}$. In the Primal-Dual SDP algorithm, we need to compute $\exp(-\frac{\varepsilon'}{2}\mathbf{M})\mathbf{u}$. Now, the idea is that this product is mostly determined by the smallest eigenvalues of $\mathbf{M}$. Thus, one can apply the Lanczos iteration to $(I + \gamma\mathbf{M})^{-1}$ for some $\gamma > 0$. This emphasizes the eigenvalues of interest.

In each iteration, we need to compute $(I + \gamma\mathbf{M})^{-1}\mathbf{u}$ for some vector $\mathbf{u}$. If we assume that $\mathbf{M}$ is *well-conditioned* (for e.g. if $\mathbf{M} \succeq \mathbf{0}$ and the condition number of $\mathbf{M}$ is bounded by a constant) then we can use the conjugate gradient method to compute this vector to any relative accuracy δ (in the ℓ_2 norm, say) in $O(\log(\frac{1}{\delta}))$ iterations, each of which requires one matrix-vector product.

Now, the key idea is that eventually, we normalize by dividing by the trace. So for any μ, $\frac{\exp(\mathbf{M})}{\mathbf{Tr}(\exp(\mathbf{M}))} = \frac{\exp(\mu\mathbf{I}+\mathbf{M})}{\mathbf{Tr}(\exp(\mu\mathbf{I}+\mathbf{M}))}$. Thus, instead of using $\mathbf{M}$ in the SI-Lanczos algorithm, we can use $\mu\mathbf{I} + \mathbf{M}$. We can now choose μ judiciously, so that $\mathbf{M}$ is well-conditioned. In [27] it is proved that $O(\log^2(\frac{1}{\varepsilon}))$ of iterations of SI-Lanczos are needed to achieve ε accuracy. In all our applications, we need ε, δ to be inverse polynomial. Thus, we have the following lemma:

Lemma 7 *Let $t_{\mathbf{M}}$ be the time needed to compute the matrix-vector product $\mathbf{Mu}$. Using SI-Lanczos, we can compute a vector $\mathbf{v}$ such that $\|\exp(\frac{1}{2}\mathbf{M})\mathbf{u} - \mathbf{v}\| \leq \varepsilon$ in $\tilde{O}(t_{\mathbf{M}})$ time. Thus, if $\mathbf{M}$ has m non-zero entries, then $t_{\mathbf{M}} = O(m)$, and the product $\exp(\mathbf{M})\mathbf{u}$ can be approximated in $\tilde{O}(m)$ time, and $\exp(\mathbf{M})$ itself can be approximated in $\tilde{O}(mn)$ time.*

Acknowledgements

We thank Inderjit Dhillon, Elad Hazan, David Philips, Satish Rao, and Manfred Warmuth for useful discussions. Vijay Vazirani suggested several years ago that primal-dual methods be investigated in the SDP context.

References

[1] S. Aaronson. The learnability of quantum states. *quant-ph/0608142*, 2006.

[2] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

[3] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized steiner problem on networks. In *STOC*, pages 134–144, 1991.

[4] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1):13–51, 1995.

[5] S. Arora, E. Hazan, and S. Kale. $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In *FOCS*, pages 238–247, 2004.

[6] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *FOCS*, pages 339–348, 2005.

[7] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. 2005. Preliminary draft of paper available online at `http://www.cs.princeton.edu/~satyen/publications.php`.

[8] S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC*, pages 222–231, 2004.

[9] S. Baswana. Dynamic algorithms for graph spanners. In *ESA*, 2006.

[10] A. A. Benczúr and D. R. Karger. Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *STOC*, pages 47–55, 1996.

[11] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *STOC*, pages 1–10, 1999.

[12] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[13] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS*, pages 300–309, 1998.

[14] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. pages 144–191, 1997.

[15] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[16] S. Golden. Lower Bounds for the Helmholtz Function. *Physical Review*, 137:1127–1128, February 1965.

[17] G. Iyengar, D. J. Phillips, and C. Stein. Approximation algorithms for semidefinite packing problems with applications to maxcut and graph coloring. In *IPCO*, pages 152–166, 2005.

[18] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[19] R. Khandekar, S. Rao, and U. V. Vazirani. Graph partitioning using single commodity flows. In *STOC*, pages 385–390, 2006.

[20] P. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *STOC*, pages 338–347, 1996.

[21] J. R. Lee. On distance scales, embeddings, and efficient relaxations of the cut cone. In *SODA*, pages 92–101, 2005.

[22] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *J. SIAM Rev.*, 20(4):801–836, October 1978.

[23] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*. SIAM, Philadelphia, 1994.

[24] S. A. Plotkin, D. B. Shmoys, and E. Tardos. Fast approximation algorithm for fractional packing and covering problems. In *FOCS*, pages 495–504, 1991.

[25] C. J. Thompson. Inequality with applications in statistical mechanics. *Journal of Mathematical Physics*, 6(11):1812–1823, 1965.

[26] K. Tsuda, G. Rätsch, and M. K. Warmuth. Matrix exponentiated gradient updates for on-line learning and bregman projection. *Journal of Machine Learning Research*, 6:995–1018, 2005.

[27] J. van den Eshof and M. Hochbruck. Preconditioning lanczos approximations to the matrix exponential. *SIAM J. Sci. Comput.*, 27(4):1438–1457, 2006.

[28] M. K. Warmuth and D. Kuzmin. Online variance minimization. In *COLT*, pages 514–528, 2006.

[29] A. Wigderson and D. Xiao. Derandomizing the Ahlswede-Winter matrix-valued Chernoff bound using pessimistic estimators and applications. *ECCC TR06-105*.

[30] N. E. Young. Randomized rounding without solving the linear program. In *SODA*, pages 170–178, 1995.

A Implementation of Oracle

A.1 Undirected Sparsest Cut

The Sparsest Cut SDP, in vector and matrix form, is the following:

$$\begin{array}{rrcl} & \min \sum_{e=\{i,j\}\in E} c_e\|\mathbf{v}_i-\mathbf{v}_j\|^2 & & \min \mathbf{C}\bullet\mathbf{X} \\ \forall p: & \sum_{j=1}^{k-1}\|\mathbf{v}_{i_j}-\mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_{i_1}-\mathbf{v}_{i_k}\|^2 \qquad \forall p: \quad \mathbf{T}_p\bullet\mathbf{X} \geq 0 \\ & \|\sum_i \mathbf{v}_i\|^2 & = & 0 \qquad\qquad\qquad\quad \mathbf{J}\bullet\mathbf{X} = 0 \\ & \sum_i\|\mathbf{v}_i\|^2 & = & n \qquad\qquad\qquad\quad \mathbf{Tr}(\mathbf{X}) = n \\ & & & \mathbf{X}\succeq 0 \end{array}$$

Here, $\mathbf{J}$ is the all ones matrix. We note that the last two constraints are not standard. Typically, we only have the constraint $\sum_{ij}\|\mathbf{v}_i-\mathbf{v}_j\|^2 = n^2$. Note that the SDP optimum doesn't change if we throw in the constraint that $\sum_i \mathbf{v}_i = \mathbf{0}$, or equivalently, $\|\sum_i \mathbf{v}_i\|^2 = 0$. With this additional constraint, the constraint $\sum_{ij}\|\mathbf{v}_i-\mathbf{v}_j\|^2 = n^2$ is equivalent to $\sum_i\|\mathbf{v}_i\|^2 = n$, precisely the kind of trace bound we need. As before, the optimum divided by n is a lower bound on the expansion of the Sparsest Cut. The dual SDP is the following:

$$\begin{array}{rcl} \max\ nx & & \\ x\mathbf{I}+\sum_p f_p\mathbf{T}_p + z\mathbf{J} & \preceq & \mathbf{C} \\ \forall p: \quad f_p & \geq & 0 \end{array}$$

Given a candidate solution $\mathbf{X}$, the Oracle always sets $x = \frac{\alpha}{n}$. Since $x\mathbf{I}\bullet\mathbf{X} = \alpha$, it now needs to find f_p, z and $\mathbf{F}\preceq\mathbf{C}$ such that

$$\alpha + \sum_p f_p(\mathbf{T}_p\bullet\mathbf{X}) + z(\mathbf{J}\bullet\mathbf{X}) - (\mathbf{F}\bullet\mathbf{X}) \leq 0.$$

It runs the following steps:

1. If $\mathbf{J}\bullet\mathbf{X} \geq \delta_1 n^2$, for some small constant δ_1, then set $z = -\frac{\alpha}{\delta_1 n^2}$, so that $z(\mathbf{J}\bullet\mathbf{X}) \leq -\alpha$. Furthermore, $\|\frac{\alpha}{n}\mathbf{I} - z\mathbf{J}\| \leq O(\frac{\alpha}{n})$.

2. Assume now that $\mathbf{J}\bullet\mathbf{X} \le \delta_1 n^2$. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}$. Then it is easy to check that the condition $\mathbf{J} \bullet \mathbf{X} \le \delta_1 n^2$ implies that $n^2 \ge \sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge (1-\delta_1)n^2$. Now, we can apply Lemma 2.

   If we get a cut of expansion, $O(\frac{\alpha}{n})$, we output it. If we get a flow f_p such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge \alpha$, then just as in step 3 of the ORACLE for undirected BALANCED SEPARATOR, we can set $\mathbf{F}$ and $\mathbf{D}$ to be the flow and demand graph Laplacians respectively, and make progress. Finally, if we get a set of nodes $S \subseteq V$ of size $\Omega(n)$, such that for all $i \in S$, $\|\mathbf{v}_i\|^2 = O(1)$, and $\sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge \Omega(n^2)$, then we can apply Lemma 1 to S. This will again yield either a cut of small expansion, in which case we stop, or a d-regular flow such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge \alpha$, in which case we again make progress.

Now, we prove Lemma 2.

PROOF:[Lemma 2] Given vectors $\mathbf{v}_i$ such that $n^2 \ge \sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge (1-\delta_1)n^2$, we run the following steps:

1. For a node i, and radius r, let $B(i,r) = \{j : \|\mathbf{v}_i - \mathbf{v}_j\| \le r\}$. If there is a node i such that for some constant δ_2, $|B(i,\delta_2)| \ge n/4$, then any $i_0 \in B(i,\delta_2)$ satisfies $|B(i_0, 2\delta_2)| \ge n/4$. So we can find such an i_0 by simple random sampling. Let $L = B(i_0, 2\delta_2)$, and let $R = V \setminus L$. For $j \in R$, define $d(j,L) = \min_{i\in L}\|\mathbf{v}_i - \mathbf{v}_j\|^2$ (doesn't need to be computed). It is easy to show (see, e.g. [21]) that $\sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge (1-\delta_1)n^2$ implies that $\sum_{j\in R} d(j,L) \ge \frac{n}{10}$, for a suitable choice of δ_1, δ_2. Let $k := \frac{|R|}{|L|}$. Note that $k = \Theta(1)$.

   Now, we connect all nodes in L to a single source with edges of capacity $\frac{10k\alpha}{n}$ and all nodes in R to a single sink with edges of capacity $\frac{10\alpha}{n}$, and compute the max-flow in the graph. If the flow saturates all source and sink nodes, then we have

$$\sum_{i\in L, j\in R} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\ge\; \sum_{j\in R}\frac{10\alpha}{n}\cdot d(j,L) \;\ge\; \alpha.$$

2. If the flow doesn't saturate all source and sink edges, then in the resulting cut, let the number of nodes in L connected to the source be n_s and the number of nodes in R connected to the sink be n_t. Then the capacity of the graph edges cut is at most $\frac{10\alpha}{n}(|R| - kn_s - n_t)$, and the smaller side of the cut has at least $\min\{|L|-n_s, |R|-n_t\}$ nodes. Thus, the expansion of the cut obtained is at most $\frac{10k\alpha}{n} = O(\frac{\alpha}{n})$.

3. Now assume that there for all node i, such that $|B(i,\delta_2)| < n/4$. Then one can check easily that there is a node i such that $|B(i,\sqrt{2})| \ge \frac{(1-\delta_1)}{2}n$. Again, by random sampling, we can find an i_0 such that $|B(i, 2\sqrt{2})| \ge \frac{(1-\delta_1)}{2}n$. Let $S = B(i, 2\sqrt{2})$. Since for every $i \in S$, $|B(i,\delta_2)| < n/4$, it can be checked that $\sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\ge\; \Omega(n^2)$. We output this set S.

□

A.2 Directed BALANCED SEPARATOR

Consider the minimum c-BALANCED SEPARATOR SDP for a directed graph (see [2]). As usual, we have a vector $\mathbf{v}_i$ corresponding to every node. In addition, we also have a vector $\mathbf{w}_i$. For convenience, we will also denote this vector by $\mathbf{v}_{n+i}$. Thus, we now have matrices in $\mathbb{R}^{2n\times 2n}$. For a directed edge (i,j), we define its directed length $d(i,j) := \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2$. With the same notation as before, the SDP, in vector and matrix form, is given below:

$$\begin{aligned}
\min \sum_{e=(i,j)\in E} c_e d(i,j) & \qquad \min \mathbf{C}\bullet\mathbf{X} \\
\forall i \in [2n]: \quad \|\mathbf{v}_i\|^2 &= 1 & \forall i: \quad \mathbf{X}_{ii} &= 1 \\
\forall i,j \in [n]: \quad \|\mathbf{w}_i - \mathbf{w}_j\|^2 &= 0 & \forall i,j\in[n]: \quad \mathbf{E}_{ij}\bullet\mathbf{X} &= 0 \\
\forall p: \quad \textstyle\sum_{j=1}^{k-1}\|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 &\ge \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 & \forall p: \quad \mathbf{T}_p\bullet\mathbf{X} &\ge 0 \\
\forall S: \quad \textstyle\sum_{i,j\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 &\ge an^2 & \forall S: \quad \mathbf{K}_S\bullet\mathbf{X} &\ge an^2 \\
& & \mathbf{X} &\succeq 0
\end{aligned}$$

In the standard SDP for this problem [2], it is intended that all the $\mathbf{w}_i$ vectors are the same, and equal to some unit vector $\mathbf{v}_0$, and this is enforced by the constraint $\|\mathbf{w}_i - \mathbf{w}_j\|^2 = 0$. We introduce these additional vectors for the purpose of keeping the width bounded. The optimum of this SDP divided by n is a lower bound on the expansion of the minimum c-Balanced Separator.

It should be noted that $\mathbf{C}$ is no longer the combinatorial Laplacian of the graph. We will call $\mathbf{C}$ the *directed Laplacian*. Now, the Oracle will find a matrix $\mathbf{F}$ such that for all primal feasible $\mathbf{X}$, we have $\mathbf{F} \bullet \mathbf{X} \leq \mathbf{C} \bullet \mathbf{X}$. This is weaker than requiring that $\mathbf{F} \preceq \mathbf{C}$. The triangle inequalities imply that any primal feasible solution must have the directed distance $d(i,j) \geq 0$. So if we have a flow in the graph which satisfies the capacity constraints $f_e \leq c_e$, then if we set $\mathbf{F}$ to be the directed Laplacian of the flow, then for any primal feasible $\mathbf{X}$, we have that

$$\mathbf{F} \bullet \mathbf{X} \;=\; \sum_{e=(i,j)\in E} f_e d(i,j) \;\leq\; \sum_{e=(i,j)\in E} c_e d(i,j) \;=\; \mathbf{C} \bullet \mathbf{X}.$$

The dual to the SDP is as follows:

$$\begin{aligned} \max\ & \textstyle\sum_i x_i + an^2 \sum_S z_S \\ \operatorname{diag}(\mathbf{x}) + \textstyle\sum_{i,j\in[n]} y_{ij}\mathbf{E}_{ij} + \sum_p f_p \mathbf{T}_p + \sum_S z_S \mathbf{K}_S \;&\preceq\; \mathbf{C} \\ \forall p, S: \quad f_p, z_S \;&\geq\; 0 \end{aligned}$$

The Oracle is implemented as follows:

1. This step is identical to Step 1 in the undirected case. Let $H := \{i \in [2n] :\ \mathbf{X}_{ii} \ \geq 2\}$. If $|H| \geq \delta_1 n$, (for some constant δ_1) then we make progress just as earlier.

2. Now, assume that $|H| < \delta_1 n$. Now we try to find $\Omega(n)$ disjoint pairs $i, j \in [n]$ such that $\|\mathbf{w}_i - \mathbf{w}_j\| \geq \delta_2$ for some $\delta_2 \geq \Omega(\frac{1}{\log n})$ to be fixed later. We restrict attention to vectors $\mathbf{w}_i$ for $i \in W := \{i \in [m] : i, n+i \notin H\}$. Note that $|W| \geq (1-\delta_1)n$.

   For a node $i \in W$, and radius r, let $B(i,r) = \{j \in W :\ \|\mathbf{w}_i - \mathbf{w}_j\| \leq r\}$. Suppose there is no node such that $i \in W$ such that $|B(i,\delta_2)| \geq (1-2\delta_1)n$. Then by randomly sampling, in expected $O(n)$ time, we can greedily remove $k := \frac{\delta_1 n}{4}$ disjoint pairs i, j such that $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta_1^2$. Let these pairs be $(i_1, j_1), \ldots, (i_k, j_k)$. For all these pairs, we set their $y_{ij} = -\frac{4\alpha}{\delta_2^2 \delta_1 n}$. We set all $x_i = \frac{\alpha}{2n}$. Then

$$\frac{\alpha}{2n}(\mathbf{I} \bullet \mathbf{X}) - \frac{4\alpha}{\delta_2^2 \delta_1 n} \sum_{t=1}^{k} (\mathbf{E}_{i_t j_t} \bullet \mathbf{X}) \;\leq\; \alpha - \alpha \;=\; 0.$$

   Furthermore, $-\frac{4\alpha}{\delta_2^2\delta_1 n}\mathbf{I} \;\preceq\; \frac{\alpha}{n}\mathbf{I} - \frac{4\alpha}{\delta_2^2\delta_1 n}\sum_{t=1}^{k}\mathbf{E}_{i_t j_t} \;\preceq\; \frac{\alpha}{n}\mathbf{I}$.

3. Now, assume that there is an $i \in W$ such that for some constant $|B(i,\delta_2)| \geq (1-2\delta_1)n$.

   Then any $i_0 \in B(i,\delta_2)$ satisfies $|B(i_0, 2\delta_2)| \geq (1-2\delta_1)n$. So we can find such an i_0 by simple random sampling. Let $S := B(i_0, 2\delta_2)$. If $2\delta_2 \leq \varepsilon$, then $|S| \geq (1-\varepsilon)n$, and we have the constraint $\mathbf{K}_S \bullet \mathbf{X} \geq an^2$.

   Now we check if $\mathbf{K}_S \bullet \mathbf{X} \leq \frac{an^2}{2}$, and if it is, then we can again make progress just as in step 2 of the undirected case.

4. Finally, assume that $\mathbf{K}_S \bullet \mathbf{X} \geq \frac{an^2}{2}$. We can now apply Lemma 3 to the vectors $\mathbf{v}_i$, $\mathbf{w}_i$ obtained from the Cholesky decomposition of $\mathbf{X}$.

   In either of the two cases, if we get a d-regular directed flow such that $\sum_{ij} f_{ij} d(i,j) \geq \alpha$, then we set $\mathbf{F}$ to be the directed Laplacian of the flow, and $\mathbf{D}$ to be the directed Laplacian of the demand graph. Then $\mathbf{D} \bullet \mathbf{X} = \sum_{i\in L, j\in R} f_{ij} d(i,j) \geq \alpha$. Again, just as in the undirected case, we can set all $x_i = \frac{\alpha}{2n}$, so that $(\frac{\alpha}{2n}\mathbf{I} - \mathbf{D}) \bullet \mathbf{X} \leq \alpha - \alpha \leq 0$, and $-2d\mathbf{I} \;\preceq\; \frac{\alpha}{2n}\mathbf{I} - \mathbf{D} \;\preceq\; \frac{\alpha}{2n}\mathbf{I}$.

   Finally, if we get $k = \Omega(\frac{n}{\sqrt{\log n}})$ vertex-disjoint paths $p_1, \ldots, p_k$, such that the path inequality along these paths is violated by $\Omega(1)$, $\mathbf{T}_p \bullet \mathbf{X} \leq -s$. Then, we set $f_{p_k} = \frac{\alpha}{sk}$ and all $x_i = \frac{\alpha}{2n}$. So, $\big(\frac{a}{2n}\mathbf{I} + \sum_{k=1}^{m} \frac{\alpha}{sk}\mathbf{T}_{p_k}\big) \bullet \mathbf{X} \leq \alpha - \alpha = 0$. We can bound the width as $\|\frac{a}{2n}\mathbf{I} + \sum_{k=1}^{m} \frac{\alpha}{sk}\mathbf{T}_{p_k}\| \leq O(\frac{\sqrt{\log n}\cdot\alpha}{n})$.

The overall running time can be bounded just as in the undirected case. We now prove Lemma 3.

PROOF:[Lemma 3]

We may assume, by scaling down if necessary, that all vectors $\mathbf{v}_i, \mathbf{w}_i$ have length at most 1. Let $\|\mathbf{w}_i - \mathbf{w}_j\| \leq \delta$, where δ is a parameter that we can make as small as needed (but always $\Omega(\frac{1}{\log n})$).

Since we have $\sum_{ij \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$, using Lemma 10 from Appendix B, for a constant fraction of directions $\mathbf{u}$, we can find sets L_0 and R_0, each of size cn for some constant $c > 0$, such that for all $i \in L$ and $j \in R$, we have $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$ for some constant $\sigma > 0$.

Next, let r be the median distance from $\mathbf{w}_{i_0}$ to the vectors $\{\mathbf{v}'_i : i \in L_0\}$. Let $L_0^+ := \{i \in L_0 : \|\mathbf{v}'_i - \mathbf{w}_{i_0}\| \geq r\}$ and $L_0^- := \{i \in L_0 : \|\mathbf{v}'_i - \mathbf{w}_{i_0}\| \leq r\}$. Define R_0^+ and R_0^- analogously. Without loss of generality, assume $|R_0^+| \geq |R_0^-|$. Define $L = L_0^-$, and $R = R_0^+$. Note that both L and R have at least $\frac{cn}{2}$ nodes each.

Now, note that for any pair of nodes $i \in L$ and $j \in R$, we have

$$\begin{aligned} d(i,j) &= \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2 \\ &= [\|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_{i_0} - \mathbf{v}_i\|^2 + \|\mathbf{w}_{j_0} - \mathbf{v}_j\|^2] - (\mathbf{w}_{i_0} - \mathbf{w}_i) \cdot (2\mathbf{v}_i - \mathbf{w}_{i_0} - \mathbf{w}_i) \\ &+ (\mathbf{w}_{i_0} - \mathbf{w}_j) \cdot (2\mathbf{v}_j - \mathbf{w}_{i_0} - \mathbf{w}_j) \\ &\geq \|\mathbf{v}_i - \mathbf{v}_j\|^2 - 8\delta \end{aligned}$$

The last inequality follows because $\|\mathbf{w}_{i_0} - \mathbf{w}_j\| \leq \delta$, and $\|2\mathbf{v}_j - \mathbf{w}_{i_0} - \mathbf{w}_j\| \leq 4$ since all vectors are of length at most 1. Now we have two cases, depending on what approximation we need:

Part 1. To get an $O(\log n)$ approximation, we can proceed exactly as in the case of undirected graphs. Namely, we may assume that all pairs $i \in L$ and $j \in R$ satisfy $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(\frac{1}{\log n})$. We choose $\delta = o(\frac{1}{\log n})$. Next, we connect all nodes in L to a single source with edges of capacity $\frac{\beta \log(n) \alpha}{n}$, and all nodes in R to a single sink with edges of capacity $\frac{\beta \log(n) \alpha}{n}$. We now compute the (directed) max-flow in this network. Just as in the undirected case, if we choose β large enough, either we get a flow such that $\sum_{i \in L, j \in R} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 2\alpha$ or we get a cut of expansion at most $O(\log(n) \cdot \frac{\alpha}{n})$. In the former case, since we chose δ small enough, we conclude that $\sum_{i \in L, j \in R} f_{ij} d(i,j) \geq \alpha$.

Part 2. To get an $O(\sqrt{\log n})$ approximation, we connect all nodes in L to a single source with edges of capacity $\frac{\beta \sqrt{\log(n)} \alpha}{n}$, and all nodes in R to a single sink with edges of capacity $\frac{\beta \sqrt{\log(n)} \alpha}{n}$. We now compute the (directed) max-flow in this network. Now there are three cases:

1. If the total flow obtained is less than $\frac{c\beta}{4} \sqrt{\log(n)} \cdot \alpha$, then by the max-flow-min-cut theorem, the cut obtained is also at most this size. Note that this cut will be $c/4$-balanced, since at most $\frac{c\beta}{4} \sqrt{\log(n)} \cdot \alpha/d = cn/4$ source (and sink) edges can be cut. Thus, the expansion of the cut is $O(\sqrt{\log(n)} \cdot \frac{\alpha}{n})$.

2. Now assume that the total flow obtained is at least $\frac{c\beta}{4} \sqrt{\log(n)} \cdot \alpha$. Then we check if $\sum_{i \in L, j \in R} f_{ij} d(i,j) \geq \alpha$. If it is, then we are done.

3. Otherwise, suppose that the total flow is at least $\frac{c\beta}{4} \sqrt{\log(n)} \cdot \alpha$, but $\sum_{i \in L, j \in R} f_{ij} d(i,j) < \alpha$. We repeat this process for $O(\log(n))$ different random directions. If we always end up in this case, then we try to find many paths for which the path inequality is violated to a large extent.

   For each direction $\mathbf{u}$, at least half the flow is between demand pairs (i,j) such that $d(i,j) \leq \frac{8}{c\beta\sqrt{\log n}}$. Now we choose $\delta = \frac{2}{c\beta\sqrt{\log n}}$. Thus for all such pairs, $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{10}{c\beta\sqrt{\log n}}$. Since at most $\frac{\beta\sqrt{\log(n)}\alpha}{n}$ flow enters or leaves any node in L or R, one can show that in fact there is a matching of size $\Omega(n)$ of pairs $i \in L$ and $j \in R$ such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{10}{c\beta\sqrt{\log n}}$. Note also that such pairs satisfy $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$. We call i, j a "stretched pair along $\mathbf{u}$". Since we found matchings of stretched pairs of $\Omega(n)$ size along $O(\log n)$ random directions, the Chernoff bounds imply that we get such large matchings for a constant fraction of directions. The following lemma follows from Arora, Rao and Vazirani [8] and Lee [21]. It is proved in Appendix B.

Lemma 8 *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots$ be vectors of length at most 1 such for a constant fraction of directions, there is a matching of stretched pairs along the direction of $\Omega(n)$ size. Then there is a pair i, j of nodes such that there is a path p of stretched pairs of length $O(\sqrt{\log n})$ such that each edge in the path is a stretched pair, and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant $s = \Omega(1)$.*

We construct a new graph, M with the same set of nodes, where there is an edge between nodes i and j if $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{10}{c\beta\sqrt{\log n}}$. Now, for some large enough constant C, we greedily find $\frac{n}{C\sqrt{\log n}}$ vertex-disjoint paths of length $O(\sqrt{\log n})$ in M such that the path inequality along each path p is violated by a constant, $\mathbf{T}_p \bullet \mathbf{X} \leq -\Omega(1)$.

We can do this greedily because every time we find a path and remove all nodes on it, we lose at most $\frac{n}{C\sqrt{\log n}} \times O(\sqrt{\log n}) = o(n)$ nodes, if we choose C large enough. Even disregarding these nodes, we still have a matching of size $\Omega(n)$ in a constant fraction of directions, and Lemma 8 tells us that we still have some paths remaining to be found.

To do this greedy path selection efficiently, we can use the dynamic decremental spanners algorithm of Baswana [9], which, for any given constant ε, maintain a spanner of size $O(\frac{1}{\varepsilon}n^{1+\varepsilon})$, such that all distances are stretched by at most a factor of $\frac{2}{\varepsilon}$, which support edge deletion in polylog(n) time. Since the stretch is a constant, we can still greedily find $\Omega(\frac{n}{\sqrt{\log n}})$ vertex-disjoint paths of length $O(\frac{1}{\varepsilon}\sqrt{\log n})$ such that the end points i, j satisfy $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$. This can be done by using BFS on the spanner in a total of $\tilde{O}(n^{2+\varepsilon})$ time.

Now consider the path inequality $\mathbf{T}_p \bullet \mathbf{X} \geq 0$ for all the paths found. Each such path p is of length at most $O(\frac{1}{\varepsilon}\sqrt{\log n})$, and each edge $\{k, \ell\}$ in the path has length $\|\mathbf{v}_k - \mathbf{v}_\ell\|^2$ at most $\frac{10}{c\beta\sqrt{\log n}}$. Thus,

$$\sum_{\{k,\ell\}\in p} \|\mathbf{v}_k - \mathbf{v}_\ell\|^2 \;\leq\; O\left(\frac{1}{\varepsilon}\sqrt{\log n}\right) \times \frac{10}{c\beta\sqrt{\log n}} \;\leq\; \frac{s}{2}$$

if we choose the constant β large enough. If the end points of p are k, ℓ, then we have $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$. This implies that the triangle inequality over p is violated by at least a constant, i.e. $\mathbf{T}_p \bullet \mathbf{X} \leq -\frac{s}{2}$. These are the paths we return.

□

A.3 Min UnCut

As usual, we start by writing the SDP for the problem. The SDP has vectors $\mathbf{v}_i$ for all $i \in V$, and the condition that $\mathbf{v}_i = -\mathbf{v}_{-i}$. So we do not explicitly maintain the vectors for negatively indexed nodes; rather, wherever $\mathbf{v}_{-i}$ appears in the SDP, we replace it by $-\mathbf{v}_i$. Assuming we have made this transformation, the following SDP results:

$$\begin{aligned} \min \sum_{e=\{i,j\}\in E} c_e\|\mathbf{v}_i - \mathbf{v}_j\|^2 \qquad & \qquad \min \mathbf{C} \bullet \mathbf{X} \\ \forall i: \quad \|\mathbf{v}_i\|^2 \;&=\; 1 \qquad & \forall i: \quad \mathbf{X}_{ii} \;&=\; 1 \\ \forall p: \quad \textstyle\sum_{j=1}^{k-1}\|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 \;&\geq\; \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \qquad & \forall p: \quad \mathbf{T}_p \bullet \mathbf{X} \;&\geq\; 0 \\ & & \mathbf{X} \;&\succeq\; 0 \end{aligned}$$

The optimum of this SDP divided by 4 is a lower bound on the value of the minimum symmetric cut. The dual SDP is the following:

$$\begin{aligned} \max \textstyle\sum_i x_i \\ \mathrm{diag}(\mathbf{x}) + \textstyle\sum_p f_p \mathbf{T}_p \;&\preceq\; \mathbf{C} \\ \forall p: \quad f_p \;&\geq\; 0 \end{aligned}$$

Given a candidate solution $\mathbf{X}$, the Oracle works as follows:

1. If there is an i, say $i = 1$, such that $\mathbf{X}_{ii} \geq 2$. Then we set $x_1 = -\alpha$, and $x_i = \frac{2\alpha}{n-1}$ for all $i \geq 2$. Since $\sum_{i \geq 2} \mathbf{X}_{ii} \leq n-2$, it is easy to see that $\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} \leq 0$. Also, $-\alpha \mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) \preceq \frac{2\alpha}{n-1}\mathbf{I}$.

   Similarly, if there is an i, say $i = 1$, such that $\mathbf{X}_{ii} \leq \frac{1}{2}$. Then we set $x_1 = 2\alpha$, and $x_i = -\frac{\alpha}{n-1}$ for all $i \geq 2$. Since $\sum_{i \geq 2} \mathbf{X}_{ii} \geq n - \frac{1}{2}$, it is easy to see that $\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} \leq 0$. Also, $-\frac{\alpha}{n-1}\mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) \preceq 2\alpha \mathbf{I}$.

2. Assume for all i, $\frac{1}{2} \leq \mathbf{X}_{ii} \leq 2$. We can now apply Lemma 4. If we get a $O(\alpha)$-regular flow such that $\sum_{i,j \in D} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, then by setting the flow and demand Laplacians $\mathbf{F}$ and $\mathbf{D}$ appropriately, and all $x_i = \frac{\alpha}{n}$ we again make progress. Note that $-\frac{2\alpha}{s}\mathbf{I} \preceq \frac{\alpha}{n}\mathbf{I} - \mathbf{D} \preceq \frac{\alpha}{n}\mathbf{I}$.

   Otherwise, we get a symmetric cut $(S, -S)$ of value at most $O(\sqrt{\log n} \cdot \alpha)$, and we stop.

Now, we bound the running time. First, since we are working with an undirected graph, we may assume that the graph has been sparsified using the algorithm of Benczúr and Karger [10], to leave only $\tilde{O}(n)$ edges. Each iteration may involve one maximum multicommodity flow problem, which takes $\tilde{O}(n^2)$ using Fleischer's algorithm. To bound the number of iterations, we estimate the relevant parameters. We have $\rho = O(\alpha)$, $R = n$. Thus, we get a bound of $\tilde{O}(n^2)$ iterations from Theorem 2. However, the width is always "one-sided": either the returned matrix lies in $[-O(\alpha)\mathbf{I}, O(\frac{\alpha}{n})\mathbf{I}]$, or in $[-O(\frac{\alpha}{n})\mathbf{I}, O(\alpha)\mathbf{I}]$. Using ideas similar to the ones in the authors' previous work [6], we can bring down the number of iterations to $\tilde{O}(n)$. Finally, Lemma 7 indicates that we can compute an approximation to the matrix exponential in $\tilde{O}(n^2)$ time. Overall, the algorithm takes $\tilde{O}(n^3)$ time.

Now, we prove Lemma 4.

PROOF:[Lemma 4] We consider a maximum multicommodity flow problem where the set of demand pairs D consists of all pairs i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s to be specified later. We only impose the edge capacity constraints $f_e \leq c_e$, and no d-regularity constraints, and solve the maximum multicommodity flow problem $\max \sum_{i,j \in D} f_{ij}$, using Fleischer's algorithm, say to a factor of $\frac{1}{2}$. We have the following cases:

1. If $\max \sum_{i,j \in D} f_{ij} \geq \frac{\alpha}{s}$, then assume that the flow is scaled down so that the total flow is exactly $\frac{\alpha}{s}$. So $\sum_{i,j \in D} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, and we return this flow.

2. If the total flow is less than $\frac{\alpha}{s}$, then the actual max flow is less than $\frac{2\alpha}{s}$, since we have a $\frac{1}{2}$ approximation. Fleischer's algorithm then yields weights w_e on the edges with $\sum_e c_e w_e \leq \frac{2\alpha}{2}$ so that for any demand pair i, j and any path p connecting them, $\sum_{e \in p} w_e \geq 1$.

   Define the volume of the graph G to be $\mathrm{Vol}(G) = \sum_e c_e w_e$, and the volume of any induced subgraph G' analogously. Let s be a constant to be specified later. Then we have the following lemma:

   Lemma 9 *Let G be a graph with nodes $\{-n, \ldots, -1, 1, \ldots, n\}$ and assume we are given vectors $\mathbf{v}_i$ for every node i such that $\|\mathbf{v}_i\|^2 = \Theta(1)$ and for all i, $\mathbf{v}_i = -\mathbf{v}_{-i}$. Also, assume that we are given some weights on edges w_e such that for any node pair i, j with $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$, and for any path p connecting them, $\sum_{e \in p} w_e \geq 1$. Then it is possible to partition the graph into sets $(S, R, -S)$ such that the capacity of edges cut is at most $O(\sqrt{\log n}\, Vol(G))$, and $Vol(R) \leq \gamma\, Vol(G)$ for some $\gamma < 1$.*

   Thus, we can recursively partition the graph to get a symmetric cut of value at most $O(\sqrt{\log n} \cdot \alpha)$, in the exact same way as [2], because the volumes of the remaining graphs decrease geometrically.

   PROOF:[Lemma 9] We repeat each node i $m_i := \lceil \frac{4n \sum_{e \ni i} c_e w_e}{\mathrm{Vol}(G)} \rceil$ times. We get a total of at most $5n$ nodes. Since each node i has a partner node $-i$ with vector $-\mathbf{v}_i$, and all $\|\mathbf{v}_i\|^2 \geq \Omega(1)$, the vectors satisfy $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$. So we can apply Theorem 6 to get a c-balanced cut, of value C such that there is a pair of nodes i, j and a path p connecting them such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ such that $\sum_{e \in p} w_e \leq O(\sqrt{\log n} \frac{\mathrm{Vol}(G)}{C})$. Just as in [2], we may assume that the cut is symmetric. Then since i, j is a demand pair, we get that $O(\sqrt{\log n} \frac{\mathrm{Vol}(G)}{C}) \geq \sum_{e \in p} w_e \geq 1$, which implies that $C \leq O(\sqrt{\log n} \mathrm{Vol}(G))$. Let $(S, R, -S)$ be the partitioning of the graph thus formed. Since we have an c-balanced cut, we have $\sum_{i \in S} m_i \geq \Omega(n)$. So, $\sum_{i \in S} \frac{4n \sum_{e \ni i} c_e w_e}{\mathrm{Vol}(G)} \geq 5cn$, which implies that $\sum_{i \in S} \sum_{e \ni i} c_e w_e \geq \frac{5c}{4} \mathrm{Vol}(G)$. Thus, $\mathrm{Vol}(R) \leq (1 - \frac{5c}{8}) \mathrm{Vol}(G)$. □

□

B Proof of Theorem 6

We restate Theorem 6 here for convenience:

Theorem 11 *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors of length at most 1, such that $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq an^2$. Let w_e be weights on edges and nodes such that $\sum_e c_e w_e \leq O(\alpha)$. Then there is an algorithm which finds a cut of value C which is c-balanced for some constant c, such that there exists a pair of nodes i, j with the property that the graph distance between i and j is at most $O(\sqrt{\log n} \cdot \frac{\alpha}{C})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ where s is a constant which only depends on a. Furthermore, this is true even if any fixed set of τn nodes are prohibited from being i or j, for some small constant τ.*

The Algorithm

0. Let $\sigma = \frac{a}{48}$, and $c = \frac{a}{256}$.
1. Choose a random direction $\mathbf{u}$, and project all $\mathbf{v}_i$ on it. By a linear scan, find sets S and T of size at least $2cn$ each such that for all $i \in S$ and $j \in T$, $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$. If no such sets exist, repeat this step with a new random direction $\mathbf{u}$.
2. Connect all nodes of S to (an artificial) source with edges of capacity 1, and connect all nodes of T to (an artificial) sink with edges of capacity 1. Scale the capacity of the original graph edges by a factor k, and compute the max flow. Let κ be the value of k at which the max flow in the network is cn, and output the min cut found.

Let C_{obs} be the capacity of the cut obtained, and let C_{med} be the median value of C_{obs}. We show that C_{med} satisfies the conditions of Theorem 6. To begin, we need the following lemma, which shows that with constant probability, Step 1. of the algorithm succeeds:

Lemma 10 *For at least a $\frac{a}{32}$ fraction of directions $\mathbf{u}$, there are efficiently computable sets S and T, each of size at least $\frac{a}{128}n$, such that for any $i \in S$ and $j \in T$, $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{a}{48\sqrt{n}}$.*

PROOF: Since $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq an^2$, and the maximum value of $\|\mathbf{v}_i - \mathbf{v}_j\|$ is 2, we get that $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\| \geq \frac{a}{2}n^2$. For any node i,

$$\frac{a}{2}n^2 \;\leq\; \sum_{jk} \|\mathbf{v}_j - \mathbf{v}_k\| \;\leq\; \sum_{jk} \|\mathbf{v}_j - \mathbf{v}_i\| + \|\mathbf{v}_i - \mathbf{v}_k\| \;\leq\; n \sum_{j} \|\mathbf{v}_i - \mathbf{v}_j\|,$$

so $\sum_j \|\mathbf{v}_i - \mathbf{v}_j\| \geq \frac{a}{2}n$. Since the maximum value of $\|\mathbf{v}_i - \mathbf{v}_j\|$ is 2, we get that there must be at least $\frac{a}{8}n$ nodes i such that $\|\mathbf{v}_i - \mathbf{v}_j\| \geq \frac{a}{4}$. For any such pair i, j, the Gaussian behavior of projections implies that $|\mathbf{v}_i \cdot \mathbf{u} - \mathbf{v}_j \cdot \mathbf{u}| \geq \frac{a}{24\sqrt{n}}$ with probability at least $\frac{1}{2}$. Call such a pair of nodes a *stretched pair*. The expected number of stretched pairs is at least $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{a}{8}n \cdot n = \frac{a}{32}n^2$. Since there are at most $\frac{1}{2}n^2$ pairs in all, we get that for at least $\frac{a}{32}$ fraction of directions $\mathbf{u}$, we get $\frac{a}{64}n^2$ stretched pairs.

Let $\mathbf{u}$ be such a direction. Let $\delta = \frac{a}{48\sqrt{n}}$. Let m be the median value of $\mathbf{v}_i \cdot \mathbf{u}$. Define the sets $L = \{i : \mathbf{v}_i \cdot \mathbf{u} \leq m - \delta\}$, $M^- = \{i : \mathbf{v}_i \cdot \mathbf{u} \in [m - \delta, m]\}$, $M^+ = \{i : \mathbf{v}_i \cdot \mathbf{u} \in [m, m + \delta]\}$and $R = \{i : \mathbf{v}_i \cdot \mathbf{u} \geq m + \delta\}$. Then any stretched pair has at least one node in $L \cup R$. Now we claim that at least one of L and R has size at least $\frac{a}{128}n$, because otherwise, the number of stretched pairs is less than $2 \cdot \frac{a}{128}n \cdot n = \frac{a}{64}n$, a contradiction. If, say, $|L| \geq \frac{a}{128}n$, we can set $S = L$, and $T = M^+ \cup R$. Note that $|T| \geq \frac{1}{2}n$, since T is the set of all points with projection higher than the median. □

The analysis of the algorithm is on the lines of the ARV algorithm, with Lee's improvement. The first step is to show that in a constant fraction of directions, we get a linear-sized matching of "stretched pairs" of nodes:

Lemma 11 *For at least $4c$ fraction of directions $\mathbf{u}$, there is a matching $M_{\mathbf{u}}$ of node pairs (i, j) such that:*

1. $|M_{\mathbf{u}}| \geq \frac{cn}{16}$,
2. $\forall (i, j) \in M_{\mathbf{u}}$ *we have* $\ell_{ij} \leq \frac{2\alpha}{C_{med}}$, *and*

3. $\forall (i,j) \in M_{\mathbf{u}}$, *we have* $|(\mathbf{v}_i - \mathbf{v}_j) \cdot \mathbf{u}| \geq \frac{\sigma}{\sqrt{n}}$.

PROOF: Lemma 10 shows that with probability at least $8c$, both S and T have size at least $2cn$. Conditioned on this happening, with a probability of $\frac{1}{2}$, the expansion of the cut, $C_{\text{obs}} \geq C_{\text{med}}$. So overall, for c fraction of directions, $C_{\text{obs}} \geq C_{\text{med}}$.

Assume this is the case. Since the max flow is cn, the min cut must attach the source to at least cn nodes to S; and the sink to at least cn nodes of T. So, the min cut is c-balanced.

For a path p in the graph from S to T, let f_p denote the flow on it and ℓ_p its length (under w_e). We have:

$$\sum_p f_p \ell_p \;=\; \sum_p f_p \sum_{e \in p} w_e \;=\; \sum_e w_e \sum_{p \ni e} f_p \;\leq\; \sum_e w_e c_e \cdot \kappa \;\leq\; \kappa\alpha. \tag{5}$$

Let S_{obs} be the set of nodes in G which lie on the same side as the source in the output min cut. We assume, without loss of generality, that S_{obs} is the smaller side on the min cut. By the max-flow min-cut theorem, we have

$$C_{\text{obs}} \cdot \kappa + |S - S_{\text{obs}}| + |T \cap S_{\text{obs}}| = cn$$

This implies that

$$\kappa \;\leq\; \frac{cn}{C_{\text{obs}}}. \tag{6}$$

Inequalities (5) and (6) imply that

$$\sum_p f_p \ell_p \;\leq\; \frac{cn\alpha}{C_{\text{obs}}}.$$

Since $\sum_p f_p = cn$, by Markov's inequality we conclude the following:

$$\text{At least } \frac{cn}{2} \text{ flow is on paths } p \text{ with length } \ell_p \;\leq\; \frac{2\alpha}{C_{\text{med}}}. \tag{7}$$

Let $N_{\mathbf{u}}$ be the set of all pairs (i,j) with $i \in S$ and $j \in T$ such that $\ell_{ij} \leq \frac{2\alpha}{C_{\text{med}}}$. We now use the probabilistic method to show the existence of the desired matching. Let $M'_{\mathbf{u}}$ be the set of pairs (i,j) obtained as follows. For a pair (i,j) in $N_{\mathbf{u}}$, let f_{ij} be the total flow from i to j. For every $i \in S$, note that $d_i := \sum_{j:(i,j)\in N_{\mathbf{u}}} f_{ij} \leq 1$ because the total flow into i from the source is 1. Interpreting the f_{ij}'s as a probability distribution over j's (here, we discard all j's with probability $1 - d_i$), randomly choose a single pair (i,j) for inclusion in $M'_{\mathbf{u}}$. Next, let $M_{\mathbf{u}}$ be the set obtained from $M'_{\mathbf{u}}$ by removing all pairs (i,j) where j occurs again in a different pair.

We now analyze the expected size of $M_{\mathbf{u}}$. For any node $j \in T$, let X_j be the number of times j occurs as a pair $(i,j) \in M'_{\mathbf{u}}$. Then $\mathbb{E}[|M_{\mathbf{u}}|] = \sum_j \mathbf{Pr}[X_j = 1]$. We have that $X_j = \sum_{(i,j)\in N_{\mathbf{u}}} X_{ij}$ where X_{ij} is an indicator random variable which is set to 1 with probability f_{ij}. Thus, $\mathbb{E}[X_j] = d_j := \sum_{i:(i,j)\in N_{\mathbf{u}}} f_{ij}$. Note that $d_j \leq 1$ since the total flow out of j into the sink is at most 1. Now, $\mathbf{Pr}[X_j = 1] = 1 - \mathbf{Pr}[X_j \geq 2] - \mathbf{Pr}[X_j = 0]$. By Markov's inequality, we have that $\mathbf{Pr}[X_j \geq 2] \leq \frac{d_j}{2}$. Also,

$$\mathbf{Pr}[X_j = 0] \;=\; \prod_{(i,j)\in N_{\mathbf{u}}} (1 - f_{ij}) \;\leq\; \prod_{(i,j)\in N_{\mathbf{u}}} \exp(-f_{ij}) \;=\; \exp(-d_j) \;\leq\; 1 - (1 - 1/e)d_j.$$

Here, the last inequality uses the fact that for $x \in [0,1]$, $e^{-x} \leq 1 - (1 - 1/e)x$. Thus,

$$\mathbf{Pr}[X_j = 1] \;\geq\; \left(\frac{1}{2} - \frac{1}{e}\right) d_j \;\geq\; \frac{d_j}{8}.$$

This implies:

$$\mathbb{E}[|M_{\mathbf{u}}|] \;\geq\; \frac{1}{8}\sum_j d_j \;=\; \frac{1}{8} \sum_{(i,j)\in N_{\mathbf{u}}} f_{ij} \;\geq\; \frac{cn}{16}.$$

Here, the last inequality follows from (7). Thus, there must exist a matching $M_{\mathbf{u}}$ of size at least $\frac{cn}{16}$. Also, $M_{\mathbf{u}}$ satisfies conditions 2 and 3 by construction. □

Next, we show the existence of a large *core*, which is a set X of linear size such that each node in X participates in a constant fraction of the matchings found in Lemma 11. For this, consider a complete weighted graph on the same nodes, such that for every pair of nodes i, j the weight is $w_{ij} = \Pr_{\mathbf{u}}[\{i,j\} \in \mathbf{M}_{\mathbf{u}}]$.

Lemma 12 *There is a subset of nodes, X, of size at least $\frac{c^2}{4}n$, such that for every $i \in X$, the degree of i on the subgraph induced by X is at least $\frac{c^2}{8}$.*

PROOF: Lemma 11 implies that the total weight of all edges is at least $4c \cdot \frac{c}{16}n = \frac{c^2}{4}n$. Now, we recursively remove all nodes with (remaining) degree less than $\frac{c^2}{8}$, and the edges incident on them. Altogether, we may remove at most $\frac{c^2}{8} \cdot n$ from the total weight. Thus, at least $\frac{c}{8}n$ weight remains. Since the degree of any node is at most 1, this implies that at least $\frac{c^2}{4}n$ nodes remain, each with degree at least $\frac{c^2}{8}$. □

A node i is said to be (ε, δ) centrally covered by X if for a delta fraction of directions $\mathbf{u}$, there exists a node $j \in X$ such that $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \varepsilon$. Similarly, a set of nodes Y is said to be (ε, δ) centrally covered by X if every node in Y is. Lemma 12 implies that X is (ε, δ) centrally covered by itself, where $\varepsilon = \frac{\sigma}{\sqrt{n}}$ and $\delta = \frac{c^2}{8}$. For a vertex $i \in X$, define $\Gamma(i) = \{j \in X : \exists \mathbf{u} \text{ s.t. } (i,j) \in M_{\mathbf{u}}\} \cup \{i\}$. Extend the definition of Γ to subsets $S \subseteq V$ as $\Gamma(S) = \bigcup_{i \in S} \Gamma(i)$. Define $\Gamma^r(S) = \underbrace{\Gamma(\Gamma(\cdots \Gamma}_{r \text{ times}}(S) \ldots))$. Nodes in $\Gamma^r(i)$ are said to be within r *matching hops* of i.

Lemma 13 *For every $r \geq 0$, one of the following two cases holds:*

1. *There is a non-empty set $S_r \subseteq X$ such that:*
   (a) $|S_r| \geq (\frac{\delta}{4})^r |X|$,
   (b) $|S_r| \geq \delta |\Gamma(S_r)|$, *and*
   (c) S_r *is* $(\frac{r\varepsilon}{2}, 1 - \frac{\delta}{2})$ *centrally covered by* X.
2. *There is a pair of points i, j such that $j \in \Gamma^{3r}(i)$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$, where $s = \frac{\sigma}{24\sqrt{\log(4/\delta)}}$.*

PROOF: We prove this by induction on r. For $r = 0$, we can choose $S_0 = X$. Every point i in X is covered by itself, so X itself is a $(0, 1 - \delta/2)$ cover for i. Further, $\Gamma(X) = X$, so all conditions required for case 1 hold.

For $r > 0$, assume that case 2 doesn't hold, but case 1 holds for $r-1$. Then we use the chaining argument of ARV, and the observation that $|S_{r-1}|/|\Gamma(S_{r-1})| \geq \delta$, to get a set S'_r such that $|S_r| \geq (\frac{\delta}{4})|S_{r-1}| \geq (\frac{\delta}{4})^r |X|$, and S'_r is $(\frac{(r-1)\varepsilon}{2} + \varepsilon, \frac{\delta^2}{4})$ centrally covered by X. Now let k be the first value for which $|\Gamma^k(S'_r)| \geq \delta |\Gamma^{k+1}(S'_r)|$. Let $S_r = \Gamma^k(S'_r)$. Since $|S'_r| \geq (\frac{\delta}{4})^r |X|$, it follows that $k \leq \log_{1/\delta}(\frac{4}{\delta})^r \leq 2r$. Let i be any point in S_r. Then there is a point $i' \in S'_r$ such that $i \in \Gamma^{3r}(i')$. Since we assumed that case 2 doesn't hold, we must have that $\|\mathbf{v}_i - \mathbf{v}_{i'}\| \leq s$. It follows from the ARV lemma of covering close by points, that the central cover X of point i is also a $(\frac{(r-1)\varepsilon}{2} + \varepsilon - \frac{ts}{\sqrt{n}}, \frac{\delta^2}{4} - \exp(-\frac{t^2}{4}))$ central cover for i. Choose $t = 2\sqrt{\log(\frac{8}{\delta^2})} \leq 4\sqrt{\log(\frac{4}{\delta})}$. Thus, $s \leq \frac{\sigma}{6t} = \frac{\sqrt{n}\varepsilon}{6t}$, and so X becomes a $(\frac{(r-1)\varepsilon}{2} + \frac{5\varepsilon}{6}, \frac{\delta^2}{8})$ central cover for i.

Next, if (i,j) is an edge in the central cover for i, then we claim that j is within $3r$ matching hops of i. This is because there is a path from i to j of matching hops which is composed of alternate expansion steps and chaining steps. There are at most r chaining steps, which may reduce the size of the current set by a factor of $\frac{\delta}{4}$, and expansion steps, which increase the size of the current set by a factor of $\frac{1}{\delta}$. Thus, the total number of expansion steps so far is at most $\log_{1/\delta}(\frac{4}{\delta})^r \leq 2r$. In addition to the r chaining steps, we get a total of at most $3r$ matching hops. Again, since we assumed that case 2 doesn't hold, we must have that $\|\mathbf{v}_i - \mathbf{v}_j\| \leq s$. Now we can use concentration of measure to conclude that X forms a $(\frac{(r-1)\varepsilon}{2} + \frac{5\varepsilon}{6} - \frac{2t''s}{\sqrt{n}}, 1 - \exp(-\frac{t'^2}{2}))$ central cover for i, where $t'' = \sqrt{2\log(\frac{16}{\delta^2})} + t'$. If we set $t' = \sqrt{2\log(\frac{2}{\delta})}$, we get that $t'' \leq 4\sqrt{\log(\frac{4}{\delta})}$. Thus, $s \leq \frac{\sigma}{6t''} = \frac{\sqrt{n}\varepsilon}{6t''}$, and so X becomes a $(\frac{r\varepsilon}{2}, 1 - \frac{\delta}{2})$ cover for i.

To complete the induction, note that if case 2 holds for $r-1$, then it trivially holds for r also. □

Finally, we can prove Theorem 6.

PROOF:[Theorem 6] If a vector $\mathbf{v}_i$ is $(\frac{r\varepsilon}{2}, 1 - \frac{\delta}{2})$ covered by vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and all vectors are of length at most 1, then $n \geq (1 - \frac{\delta}{2}) \cdot \exp(\frac{r^2\varepsilon^2 n}{32})$, since $\|\mathbf{v}_j - \mathbf{v}_i\| \leq 2$ and thus any such vector $\mathbf{v}_j - \mathbf{v}_i$ can cover only $\exp(-\frac{r^2\varepsilon^2 n}{32})$ fraction of directions.

Thus, since $\delta < 1$ and $\varepsilon = \frac{\sigma}{\sqrt{n}}$, case 1 of Lemma 13 cannot hold for $r \geq \frac{\sqrt{32 \log(2n)}}{\sigma} = \Theta(\sqrt{\log n})$. So for some $r \leq O(\sqrt{\log n})$, case 2 must hold. Case 2 implies the existence of a pair of points i, j which are within a graph distance of $3r \times \frac{2\alpha}{C_{\text{med}}} = O(\sqrt{\log n} \frac{2\alpha}{C_{\text{med}}})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s = \Omega(1)$. The robustness property follows because we can set aside some τn nodes from participating in the matchings with a small degradation in the constants. □

We can now prove Lemma 8, restated here for convenience. In this setting, i, j is a *stretched pair* if $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\delta}{\sqrt{\log n}}$ but $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$ for constants $\delta, \sigma > 0$.

Lemma 14 *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots$ be vectors of length at most 1 such for a constant fraction of directions, there is a matching of stretched pairs along the direction of $\Omega(n)$ size. Then there is a pair i, j of nodes such that there is a path p of stretched pairs of length $O(\sqrt{\log n})$ such that each edge in the path is a stretched pair, and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant $s = \Omega(1)$.*

Proof: The proof uses Lemmas 12 and 13. Note that neither of these lemmas need any upper bound on the distance between a pair of matched nodes i, j, and so their proofs work as long as we have $\Omega(n)$ sized matchings of nodes for a constant fraction of directions. Thus, for some $r \leq O(\sqrt{\log n})$, case 2 of Lemma 13 must hold, at which point we get a pair of nodes i, j within $3r$ matching hops such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s. Thus, i, j are connected with a path of stretched pairs of length at most $3r = O(\sqrt{\log n})$. □