A Combinatorial, Primal-Dual Approach to Semidefinite Programs

[Extended Abstract]

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.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

General Terms

Algorithms, Theory

Keywords

Matrix Multiplicative Weights, Semidefinite Programming, Sparsest Cut, Balanced Separator, Min UnCut

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.

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

STOC'07, June 11–13, 2007, San Diego, California, USA.

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 multi-commodity 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 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, and resulted in an $O(\sqrt{\log n})$-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. Specifically, we make use of a structure theorem similar to ARV's theorem about well-separated sets to find feedback with bounded "width" parameter, which in turn bounds the running time.

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. (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, while taking care that the computation of the feedback function is "well-conditioned" (c.f. Section 5). 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]. In this 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 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 fur-

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 for approximation algorithms on a weighted graph with n vertices and m edges.

ther 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 6).

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 $\mathbf{A}\mathbf{B} \neq \mathbf{B}\mathbf{A}$ in general, $\exp(\mathbf{A}+\mathbf{B}) \neq \exp(\mathbf{A})\exp(\mathbf{B})$. Since $\exp(\mathbf{A}) = \exp(\frac{1}{2}\mathbf{A})\exp(\frac{1}{2}\mathbf{A})^\top$ for all symmetric $\mathbf{A}$, we conclude that $\exp(\mathbf{A})$ is *PSD*. In particular, we can assume without loss of generality that algorithms for matrix exponentiation can also output the Cholesky decomposition of the output matrix.

Computing exact matrix exponentials is tricky. However, it suffices to compute a "good enough" approximation. In all the SDPs in this paper, the constraints involve squared lengths of vectors obtained from the Cholesky decomposition of the exponential. By the Johnson-Lindenstrauss lemma, these squared lengths can approximated using random projection. The projection can be done very efficiently, and Section 5 briefly discusses the details.

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:

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

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 $\mathbf{X}^{(t+1)}$, and in this it has help from an auxiliary algorithm, called the ORACLE, that tries to certify the validity of the current $\mathbf{X}^{(t)}$ as follows. ORACLE 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

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

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

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

which would contradict (1). Thus $\mathbf{y}$ implicitly contains some useful information to improve the candidate primal $\mathbf{X}^{(t)}$, and we use $\mathbf{y}$ to update $\mathbf{X}^{(t)}$ using a familiar-looking *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 (1), 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.

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 on the right, 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.

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

THEOREM 1. *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* $(1+\delta)\alpha$.

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 1 is a consequence of our more general Matrix Multiplicative Weights algorithm, explored in more detail in Theorem 9 of Section 6.

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.

As a warmup, we illustrate the use of Theorem 1 in the following simple application.

THEOREM 2. *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 2. 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):

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

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 1 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 6 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: $\|\text{diag}(\mathbf{x}) - \mathbf{C}\| \leq O(d)$.

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 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

$$\sum_{j=1}^m (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j^{(t)} - (\mathbf{F}^{(t)} \bullet \mathbf{X}) \ \leq \ 0. \qquad (2)$$

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 1 goes through with these changes.

THEOREM 3. *In the modified Primal-Dual algorithm for a minimization SDP as described above, 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 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.

In this abstract, we only 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 3 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 4.

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 (Sketch). We use the well-known SDP relaxation with triangle inequalities, but to make the algorithm simpler we use an equivalent formulation. We assign vectors $\mathbf{v}_i$ to the nodes in G. Let $p = (i_1, i_2, \ldots, i_k)$ be a generic path of nodes in the complete graph, and let $a = \frac{1}{4}c(1-c)$. The SDP in vector form is:

$$\begin{aligned} \min \sum_{e=\{i,j\}\in E} c_e \|\mathbf{v}_i - \mathbf{v}_j\|^2 \\ \forall i: \quad \|\mathbf{v}_i\|^2 &= 1 \\ \forall p: \quad \textstyle\sum_{j=1}^{k-1} \|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 &\geq \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \\ \textstyle\sum_{i,j} \|\mathbf{v}_i - \mathbf{v}_j\|^2 &\geq an^2 \end{aligned}$$

(Note that the standard formulation of triangle inequality only uses paths of length 3, but this formulation is equivalent.) In the following matrix form of the SDP, $\mathbf{X}$ is the Gram matrix of the vectors $\mathbf{v}_i$, $\mathbf{C}$ and $\mathbf{K}$ are the combinatorial Laplacians of the input graph and complete graph respectively, and $\mathbf{T}_p$ is the difference of the Laplacian of p and that of a single edge connecting its endpoints:

$$\begin{aligned} \min \mathbf{C} \bullet \mathbf{X} \\ \forall i: \quad \mathbf{X}_{ii} &= 1 \\ \forall p: \quad \mathbf{T}_p \bullet \mathbf{X} &\geq 0 \\ \mathbf{K} \bullet \mathbf{X} &\geq an^2 \\ \mathbf{X} &\succeq 0 \end{aligned}$$

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. Let $\mathrm{diag}(\mathbf{x})$ be the diagonal matrix with the vector $\mathbf{x}$ on the diagonal.

$$\begin{aligned} \max \textstyle\sum_i x_i + an^2 z \\ \mathrm{diag}(\mathbf{x}) + \textstyle\sum_p f_p \mathbf{T}_p + z\mathbf{K} &\preceq \mathbf{C} \\ \forall p: \ f_p, \ z &\geq 0 \end{aligned}$$

Now, we will briefly describe the implementation of the Oracle. Given a candidate solution $\mathbf{X}$, the Oracle needs to find variables x_i, $f_p \geq 0$, $z \geq 0$ and a matrix $\mathbf{F} \preceq \mathbf{C}$ (c.f. Theorem 3) such that $\sum_i x_i + an^2 z \geq \alpha$ which satisfies

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

First, it is easy to ensure, using techniques similar to the ones used in solving the MaxCut SDP in Section 3, that $\forall i: \ \mathbf{X}_{ii} = O(1)$ and $\mathbf{K} \bullet \mathbf{X} \geq \Omega(n^2)$ (actually, we only check this latter condition for a large subset of nodes). Further, this can be done while ensuring that the width is $O(\frac{\alpha}{n})$.

The novel part is to nudge $\mathbf{X}$ 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 $\forall i: \ f_i \leq d$.

Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}$. Note that for all nodes i, we have $\|\mathbf{v}_i\|^2 = O(1)$. Also, $\mathbf{K} \bullet \mathbf{X} \geq \Omega(n^2)$ implies that $\sum_{i,j} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$. 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:*

1. *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 $O(\log(n)\frac{\alpha}{n})$.*

2. *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 $S = V$, corresponding to the two cases of Theorem 4.

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.

Then we set $\mathbf{F}$ to be the Laplacian of the weighted graph with edge weights f_e. Since $\mathbf{C} - \mathbf{F}$ is the Laplacian of the residual graph, and a valid flow satisfies capacity constraints $f_e \leq c_e$, we conclude that $\mathbf{F} \preceq \mathbf{C}$. Let $\mathbf{D}$ be the Laplacian of the complete weighted graph where edge $\{i, j\}$ has weight f_{ij}.

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

$$\mathrm{diag}(\mathbf{x}) + \mathbf{F} - \mathbf{D} - \mathbf{F} \ = \ \frac{\alpha}{n}\mathbf{I} - \mathbf{D}.$$

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 the algorithms. Using the sparsification algorithm of Benczúr and Karger [10], we may assume the input graph has $\tilde{O}(n)$ edges. The Oracle ensures that the width is bounded by $\tilde{O}(\frac{\alpha}{n})$, and $R = n$, so Theorem 1 implies that the algorithm needs

polylog(n) iterations. Each iteration requires a matrix exponentiation and (possibly) a flow computation. The first algorithm only uses single-commodity max-flow computations in every iteration, which takes $\tilde{O}(n^{1.5})$ time using Goldberg-Rao [15]. The second algorithm uses multicommodity flow, which takes $\tilde{O}(n^2)$ time by Fleischer [12]. Finally, the running time of matrix exponentiation using Lemma 6 is near-linear in the number of entries of the matrix, which is no more than the time taken by the flow computations. □

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, $\frac{E(S,\bar{S})}{\min\{|S|,|S|\}}$.

Theorem 5.

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

We briefly sketch the Oracle. Given a candidate solutio $\mathbf{X}$ in the form of the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ obtained from its Cholesky decomposition, the Oracle first checks that $n^2 \geq \sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$ just as in Balanced Separator. To check the triangle inequalities, the Oracle runs flow computations as given in the following lemma:

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 the Oracle can either find a cut of small expansion, or a flow to make progress in the Primal-Dual SDP algorithm.

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 6.

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 polynomial dependence on $\frac{1}{\varepsilon}$.*

We briefly sketch the Oracle. The Directed Balanced Separator SDP relies on the directed distance between all nodes pairs (i, j): $\|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{v}_i - \mathbf{v}_0\|^2 + \|\mathbf{v}_j - \mathbf{v}_0\|^2$, where $\mathbf{v}_0$ is an extra unit vector. Since it appears in all directed distances, a simple implementation of the Oracle runs into width issues. So we use the following idea: for each node i, we associate a unit vector $\mathbf{w}_i$, which is supposed to be its own private copy of $\mathbf{v}_0$, and use the directed distance $d(i, j) = \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{v}_i - \mathbf{w}_i\|^2 + \|\mathbf{v}_j - \mathbf{w}_j\|^2$ instead. To enforce all the $\mathbf{w}_i$ to be identical, we thrown in the constraints $\|\mathbf{w}_i - \mathbf{w}_j\|^2 = 0$ for all i, j.

Now, given the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and $\mathbf{w}_1, \ldots, \mathbf{w}_n$ as a candidate solution, the Oracle checks that almost all of them are $O(1)$ in length, that all pairs of nodes i, j satisfy $\|\mathbf{w}_i - \mathbf{w}_j\|^2 = o(1)$, and that $\sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$. Then, to check the triangle inequalities, the Oracle runs flow computations as given in the following 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 inequalities (obtained by combining the triangle inequalities along these paths) are violated by $\Omega(1)$.*

Case 2(c) above is implemented using the dynamic decremental spanners algorithm of Baswana [9]. In this case, the ORACLE generates feedback by setting the dual variables $f_p = O(\frac{\sqrt{\log n}\alpha}{n})$ for all the paths p with violated path inequalities found by the algorithm. Note that this is not a valid flow, but it suffices to make progress.

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 7.

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 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 $G = (V, E)$ where the set of 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 8. *An $O(\sqrt{\log n})$ approximation to* MIN UNCUT *can be computed in $\tilde{O}(n^3)$ time.*

We briefly sketch the ORACLE. The SDP for MINUNCUT assigns vectors $\mathbf{v}_i$ and $\mathbf{v}_{-i}$ for all nodes $i, -i$ respectively, where $\mathbf{v}_{-i} = -\mathbf{v}_i$. By making this substitution for $\mathbf{v}_{-i}$ in the SDP, we may assume that the SDP is written only in terms of $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and the $\mathbf{v}_{-i}$ are obtained by simply negating them.

The ORACLE first checks that all $\mathbf{v}_i = \Theta(1)$. Then, to check the triangle inequalities, it runs a multicommodity flow computation, as given in the following 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 $O(\sqrt{\log n} \cdot \alpha)$.*

In the first case, we make progress in the Primal-Dual SDP algorithm just as in undirected BALANCED SEPARATOR. The second case is implemented by using the dual weights on the edges obtained from the multicommodity flow computation and using them to recursively produce (partial) symmetric cuts in the graph.

5. FAST MATRIX EXPONENTIALS

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 squared lengths of the row vectors and their differences, we can apply 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 on the ORACLE under which it does. 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 α is the current estimate of the optimum, and the ORACLE can also ensure that $\sum_i |s_i| \|\mathbf{v}'_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}'_i - \mathbf{v}'_j\|^2 = O(\alpha)$, 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 1 holds even with this relaxation for the ORACLE. We call such an ORACLE "well-conditioned".

LEMMA 5 (WELL-CONDITIONED ORACLE). *In the above setting if the* ORACLE *can always set the s_i's and t_{ij} variables 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.*

All our ORACLEs have the following property: they find s_i and t_{ij} such that the sum $\sum_i s_i\|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2$, when restricted to the positive s_i and t_{ij}, is α; and when restricted to the negative s_i and t_{ij}, is $-\alpha$. Thus, all our ORACLEs are well-conditioned.

Now we turn to details of computing the matrix exponential $\exp(\mathbf{M})$. The Johnson-Lindenstrauss lemma allows us to reduce this to computing $\exp(\frac{1}{2}\mathbf{M})\mathbf{u}$ where $\mathbf{u}$ is one of $O(\frac{\log n}{\delta^2})$ random vectors. This has fast implementations in solvers for linear differential equations, and one can give good complexity bounds. Also, it suffices to do even this computation approximately, namely, find a vector $\mathbf{v}$ such that $\|\exp(\frac{1}{2}\mathbf{M})\mathbf{u} - \mathbf{v}\| \leq \varepsilon$ for some inverse polynomial ε.

One simple method is to approximate $\exp(\frac{1}{2}\mathbf{M})$ by the first $O(\log n)$ terms of its Taylor expansion, which needs $O(\log n)$ matrix-vector products. Each of these products can be computed in time which is proportional to the number of non-zero entries of the matrix.

A more sophisticated algorithm for this task uses the *Shift-and-Invert Lanczos* (SI-Lanczos) method of van den Eshof and Hochbruck [27], as analyzed by Iyengar, Phillips, and Stein [17] for the very setting we have. They prove that $O(\log^2(\frac{1}{\varepsilon}))$ of iterations of SI-Lanczos are needed to achieve ε accuracy, assuming the matrix to be exponentiated is well-conditioned. Since we normalize by dividing by the trace anyway, all our matrices can be easily made well-conditioned by simply adding an appropriate multiple of the identity matrix. We have the following result:

LEMMA 6. *Let $t_{\mathbf{M}}$ denote 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.*

6. THE MATRIX MULTIPLICATIVE WEIGHTS ALGORITHM

Instead of just analyzing the matrix multiplicative update rule in context of SDPs, we analyze a more general algorithm. This algorithm is the matrix analogue of the framework of our survey with Hazan [7]. 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 $\mathbf{v}^\top\mathbf{Mv} = \mathbf{M} \bullet \mathbf{vv}^\top$. We allow the first player to choose his vector from a distribution $\mathcal{D}$ over $\mathbb{S}^{n-1}$. We are interested in the expected loss to the first player, viz.

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

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

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 $\mathbf{W}^{(t)} = (1-\varepsilon)^{\sum_{\tau=1}^{t-1}\mathbf{M}^{(\tau)}} = \exp(-\varepsilon'(\sum_{\tau=1}^{t-1}\mathbf{M}^{(\tau)}))$.
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 9. *The Matrix Multiplicative Weights Update algorithm generates density matrices $\mathbf{P}^1, \mathbf{P}^2, \ldots, \mathbf{P}^T$ such that*

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

Remark: The proof of Theorem 1 follows by thinking of the event matrices as

$$\mathbf{M}^{(t)} \;:=\; (\textstyle\sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I})/2\rho.$$

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.

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]: $\mathbf{Tr}(e^{\mathbf{A}+\mathbf{B}}) \leq \mathbf{Tr}(e^{\mathbf{A}}e^{\mathbf{B}})$. Consider:

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

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:

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

because $\mathbf{Tr}(\exp(\mathbf{A})) = \sum_{k=1}^{n} \exp(\lambda_k(\mathbf{A})) \geq \exp(\lambda_n(\mathbf{A}))$. 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. □

7. FUTURE WORK

We intend to apply the methods of this paper to other problems for which the known SDP-based algorithms are fairly inefficient, such as MIN 2CNF DELETION and MIN LINEAR ARRANGEMENT. To obtain fast primal-dual algorithms for these problems, we would first need to understand the SDP well. Usually, this means devising robust rounding algorithms for obtaining integer solutions from fractional ones.

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 for SDPs.

8. 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.