Lecture 17

Solving LPs/SDPs using Multiplicative Weights*

In the last lecture we saw the Multiplicative Weights (MW) algorithm and how it could be used to effectively solve the experts problem in which we have many experts and wish to make predictions that are approximately as good as the predictions made by the best expert. In this lecture we will see how to apply the MW algorithm to efficiently approximate the optimal solution to LPs and SDPs.

17.1 Multiplicative Weights

Recall the following result from Lecture 16 about the "Hedge" algorithm:

Theorem 17.1. *Suppose the cost vectors are $\bar{m}^{(t)} \in [-1,1]^N$. Then for any $\epsilon \leq 1$, and for any T, the Hedge algorithm guarantees that for all $i \in [m]$,*

$$\sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon + \frac{\ln N}{\epsilon}$$

So the total cost paid by the algorithm is no more than an additive factor of $\epsilon + \frac{\ln N}{\epsilon}$ worse than the cost incurred by any individual component of the cost vector. Theorem 17.1 implies a similar result for the *average* cost incurred per round. (One can get a similar result for the MW algorithm, where instead of the update rule $w_i^{(t)} \leftarrow w_i^{(t)} \cdot \exp(-\epsilon m_i^{(t)})$, we used the rule $w_i^{(t)} \leftarrow w_i^{(t)} \cdot (1 - \epsilon m_i^{(t)})$.)

Corollary 17.2. *Suppose the cost vectors are $\bar{m}^{(t)} \in [-\rho, \rho]^N$. Then for any $\epsilon \leq \frac{1}{2}$, and for any $T \geq \frac{4 \ln N}{\epsilon^2} \rho^2$, the Hedge algorithm guarantees than for all $i \in [m]$*

$$\frac{1}{T} \sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \frac{1}{T} \sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon$$

*Lecturer: Anupam Gupta. Scribe: Tim Wilson.

Note: We did not cover this in lecture, but one can show that if the cost vectors are in $[0, \rho]$, then using the MW algorithm, the setting $T \geq \frac{4 \ln N}{\epsilon^2}\rho$ suffices to get the same guarantee of

Lemma 17.3. *Suppose the cost vectors are $\bar{m}^{(t)} \in [0, \rho]^N$. Then for any $\epsilon \leq \frac{1}{2}$, and for any $T \geq \frac{4 \ln N}{\epsilon^2}\rho$, the MW algorithm guarantees than for all $i \in [m]$*

$$\frac{1}{T}\sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \frac{1}{T}\sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon$$

A proof of this can be found in the Arora, Hazan, and Kale survey [AHK05].

17.2 Solving LPs with Multiplicative Weights

We will use the MW algorithm to help solve LPs with m constraints of the form

$$\begin{aligned} \min\, & c^\top x \\ \text{s.t. } & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Supposing that we know $c^\top x^* = \text{OPT}$ (by binary search), we will aim to find an ϵ-approximate solution $\widetilde{x}$ such that

$$\begin{aligned} c^\top \widetilde{x} &= \text{OPT} \\ A\widetilde{x} &\geq b - \epsilon \mathbf{1} \\ \widetilde{x} &\geq 0 \end{aligned}$$

or output "infeasible" if no solution exists. The runtime for this will be $O\left(\frac{\rho^2 \log m}{\epsilon^2}\right)$ where ρ is the "width" of the LP which will be defined shortly.

17.2.1 Simplifying the Constraints

Instead of searching for solutions $x \in \mathbb{R}^n$, we will package together the "easy" constraints into the simple convex region

$$K = \{x \in \mathbb{R}^n \mid x \geq 0, c^\top x = \text{OPT}\}$$

Now we wish to solve $Ax \geq b$ such that $x \in K$. Note that this is particularly easy to solve if $Ax \geq b$ is only one constraint, i.e., we are trying to determine whether $\exists x \in K$ such that $\alpha^\top x \geq \beta$ for some $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}$. For example, if $c \geq 0$ and

$$\max_i \alpha_i \frac{\text{OPT}}{c_i} \geq \beta$$

we can set $x = \frac{\text{OPT}}{c_i} e_i$ which will satisfy our constraints; else we could output `Infeasible`. For general c we are essentially reduced to solving an LP over two constraints, which while not as trivial as this, is still simple.

We will henceforth assume we have an oracle that given $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}$, and $K \subseteq \mathbb{R}^n$ either returns $x \in \mathbb{R}^n$ such that $\alpha^\top x \geq \beta$, or correctly asserts that there is no such x.

17.2.2 Using Multiplicative Weights

We will use this oracle that allows us to satisfy one constraint ($\alpha x \geq \beta$) for $k \in K$, along with the MW algorithm to get an algorithm satisfy all of the constraints $Ax \geq b$ for $x \in K$.

Each of the constraints $a_i^| topx \geq b_i$ will be viewed as an "expert" for a total of m experts. Each round we will produce a vector $\bar{p}^{(t)}$ that will give us a convex combination of the constraints as follows

$$\underbrace{\bar{p}^{(t)} \cdot A}_{\alpha^{(t)}} x \geq \underbrace{\bar{p}^{(t)} \cdot b}_{\beta^{(t)}}$$

Using our oracle, we can determine whether $\alpha^{(t)} x \geq \beta^{(t)}$ has some solution $x^{(t)} \in K$, or if no such solution exists. Clearly if no solution exists, then $Ax \geq b$ is infeasible over K, so our LP is infeasible. (It's easy to see the contrapositive: if there were a solution to $Ax \geq b, x \in K$, then this vector x would also satisfy $\alpha^{(t)} x \geq \beta^{(t)}$; here we use the fact that $\bar{p}^{(t)} \geq 0$.) Moreover, the vector $\bar{p}^{(t)}$ serves as proof of this infeasibility.

Otherwise, we will set our cost vector so that $\bar{m}_i^{(t)} = a_i x^{(t)} - b_i$, update our weights and proceed with the next round. If we have not determined the LP to be infeasible after T rounds we will terminate and return the solution

$$\widetilde{x} = \frac{1}{T} \sum_{t \leq T} x^{(t)}$$

Why do we set our cost vectors this way? It almost seems like we should incur no cost when $a_i x^{(t)} - b_i \geq 0$ (i.e., when we satisfy this constraint), whereas we are incurring a higher cost the more we satisfy it. Well, the idea is whenever $a_i^{(t)} x - b_i$ is positive, we have oversatisfied the constraint. Giving a positive cost to this constraint causes us to reduce the weight of this constraint in this next round. This works analogously to the experts problem where an expert who is wrong (has high cost) is given less credence (less weight) in future rounds. Similarly, for any constraint in which $a_i^{(t)} x - b_i$ is negative, we have failed the constraint. Giving a negative cost to this constraint causes us to increase the weight of this constraint in the next round.

Initially we set all of our weights equal to express our ignorance; "all constraints are equally hard". Whenever we update our weights we reduce the weights of constraints we oversatisfied so we'll cover them less in future rounds. We increase the weights of constraints we didn't satisfy so we'll cover them more in future rounds. Our hope is that over time this will converge to a solution where we satisfy all constraints to a roughly equal extent.

17.2.3 Analyzing Multiplicative Weights

Supposing that we do not discover our LP is infeasible, how many rounds should we run and how good will our solution be? If we define

$$\rho = \max\{1, \max_{i, x \in K} \{|a_i^\top x - b_i|\}\}$$

to be the maximum magnitude of any cost assigned to a constraint, then we may immediately apply Corollary 17.2 to find that after $T \geq \frac{4 \ln n}{\epsilon^2} \rho^2$ rounds,

$$\frac{1}{T} \sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \frac{1}{T} \sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon$$

where $\epsilon \leq \frac{1}{2}, \bar{m}^{(t)} = a_i^\top x^{(t)} - b_i \in [-\rho, \rho]^n$ for all $i \in [m]$, and each $x^{(i)} \in K$. Note that we do not actually need to find ρ; it suffices to keep track of $\rho_t = \max\{1, \max_{i,t' \leq t}\{|a_i^\top x^{(t')} - b_i|\}\}$, the maximum cost seen so far, and run until $T \geq \frac{4 \ln n}{\epsilon^2} \rho_T^2$.

What guarantee do we get? On the left hand side of this inequality we have

$$\begin{aligned} \bar{p}^{(t)} \cdot \bar{m}^{(t)} &= \bar{p}^{(t)} \cdot (Ax^{(t)} - b) \\ &= \bar{p}^{(t)} \cdot Ax^{(t)} - \bar{p}^{(t)} \cdot b \\ &\geq 0 \end{aligned}$$

where the final inequality holds due to our oracle's properties. Therefore the left hand side is at least 0. And on the right hand side we have

$$\begin{aligned} \frac{1}{T} \sum_{t \leq T} \bar{m}_i^{(t)} &= \frac{1}{T} \sum_{t \leq T} a_i^\top x^{(t)} - b_i \\ &= a_i^\top \left(\frac{1}{T} \sum_{t \leq T} x^{(t)} \right) - b_i \\ &= a_i^\top \widetilde{x} - b_i \end{aligned}$$

Combining this with our inequality for the right hand side we get

$$\begin{aligned} \forall i : \quad a_i^\top \widetilde{x} - b_i + \epsilon &\geq 0 \\ a_i^\top \widetilde{x} &\geq b_i - \epsilon \end{aligned}$$

Therefore we can obtain an ϵ-feasible solution to $Ax \geq b, x \in K$ in time $O\left(\frac{\log m}{\epsilon^2} \rho^2\right)$ time where $\rho = \max\{1, \max_{i, x \in K}\{|a_i^\top x - b_i|\}\}$ is the width of the LP.

17.2.4 Example: Minimum Set Cover

Recall the minimum fractional set cover problem with m sets $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ and n elements U. The goal is to pick fractions of sets in order to cover each element to an extent of 1: i.e., to solve the following LP—

$$\begin{aligned} \min & \sum_S x_S \\ \text{s.t.} & \sum_{S \ni e} x_S \geq 1 \quad \forall e \\ & x_S \geq 0 \end{aligned}$$

Suppose we know OPT $= L \in [1, m]$, so $K = \{\sum_S x_S = L, x_S \geq 0\}$. We want to find $x \in K$ such that $\sum_{S \ni e} x_S \geq 1$ for all elements e. Our oracle, given some $\bar{p}$, must try to find $x \in K$ such that

$$\begin{aligned} & \sum_e \bar{p}_e \sum_{S \ni e} x_S \geq \sum_e \bar{p}_e \cdot 1 = 1 \\ \iff & \sum_S x_S \sum_{e \in S} \bar{p}_e \geq 1 \\ \iff & \sum_S x_S \cdot p(S) \geq 1 \end{aligned}$$

where $p(S)$ is the total weight of elements in S. This quantity is clearly maximized over K by concentrating on a set with the maximum weight and setting

$$x_S = \begin{cases} L & \text{for some } S \in \mathcal{F} \text{ maximizing } p(S) \\ 0 & \text{for all other } S \end{cases}$$

Note that the width of this LP is at most

$$\max_e \sum_{S \ni e} x_S - 1 \leq L - 1 \leq m - 1$$

How does the weight update step work? Initially we set $w_i^{(1)}$ for all constraints. Whenever a set is overcovered, we reduce the weight of that set so we don't try as hard to cover it in the next step. Whenever a set is undercovered we increase the weight of the set so we try harder to cover it in the next step. Now, after $4L^2 \ln n/\epsilon^2$ steps we will obtain an ϵ-approximate solution $\widetilde{x}$ such that

$$\begin{aligned} \sum_S \widetilde{x}_S &= L \\ \sum_{S \ni e} \widetilde{x}_S &\geq 1 - \epsilon \\ \widetilde{x} &\geq 0 \end{aligned}$$

Note that, in this case, the constraint matrix is completely nonnegative, and we can scale up our solution to get a feasible solution $\widehat{x} = \widetilde{x}/(1-\epsilon)$ so that

$$\begin{aligned} \sum_S \widehat{x}_S &= \frac{L}{1-\epsilon} \approx L(1+\epsilon) \\ \sum_{S \ni e} \widehat{x}_S &\geq 1 \\ \widetilde{x} &\geq 0 \end{aligned}$$

17.2.5 Comments

1. The scaling we used for minimum set cover to obtain a non-optimal, feasible solution can be applied to any LP where $b > \mathbf{1}\epsilon$—indeed, we could just multiply all the x values by $\max_i 1/(b_i - \epsilon)$. This is often useful, particularly when we're going to round this LP solution and incur further losses, and hence losing this factor may be insignificant.

2. If the constraint matrix A is all positive the problem is said to be a *covering problem* (we are just interested in putting enough weight on x to cover every constraint). If the constraint matrix is all negative—or equivalently, if we have $Ax \leq b$ with an all-positive matrix A—the problem is said to be a *packing problem* (we are packing as much weight into x as possible without violating any constraint). In either case, we can use a similar scaling trick to get a non-optimal, feasible solution.

   In this case we can reduce the run-time further. Assume we have a covering problem: $\min\{c^\top x \mid Ax \geq b, x \geq 0\}$. By scaling, we can transform this into a problem of the form
   $$\min\{c^\top x \mid Ax \geq 1, x \geq 0\}$$
   The uniform values of $b_i = 1$ allows us to set the cost vectors $m_i^{(t)} = a_i^\top x^{(t)}$ instead of $m_i^{(t)} = a_i^\top x^{(t)} - 1$; this translation does not change the algorithm. But the positive cost vectors allow us to use Lemma 17.3 to reduce the runtime from $O\left(\frac{\log m}{\epsilon^2}\rho^2\right)$ to $O\left(\frac{\log m}{\epsilon^2}\rho\right)$.

3. In general, the width of our LPs may not turn out to be as nice. For example, in the *weighted* minimum set cover problem
   $$\begin{aligned} \min & \sum_S c_S x_S \\ \text{s.t.} & \sum_{S \ni e} x_S \geq 1 \quad \forall e \\ & x_S \geq 0 \end{aligned}$$
   our optimum, and hence the width, can increase to as much as $m \cdot \frac{\max_S c_S}{\min_S c_S}$. An approach developed by Garg and Könemann [GK07] can be useful to solve the problems without the width penalty.

4. The MW algorithm does not need a perfect oracle. Being able to determine given $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ if there is no $x \in K$ with $\alpha^| topx \geq \beta$, or else returning an $x \in K$ such that $\alpha^\top x \geq \beta - \epsilon'$ is sufficient for our purposes. This gives us solutions $\widetilde{x} \in K$ such that
   $$Ax \geq b - (\epsilon + \epsilon')\mathbf{1}.$$

5. There was exactly one point where we used the fact that our constraints were linear. That was concluding that
   $$\frac{1}{T}\sum_{t \leq T} a_i^\top x^{(t)} - b_i = a_i^\top \widetilde{x} - b_i$$

However, we can make a similar claim for any set of convex constraints as well: if we wanted to find $x \in K$ such that $f_i(x) \leq 0$ for $i \in [m]$, with the f_i's convex. Then as long as we could solve the oracle and find $x \in K$ with $\sum_i p_i^{(t)} f_i(x) \leq 0$ efficiently, the rest of the argument would go through. In particular, in the step where we used linearity, we could instead use

$$\frac{1}{T}\sum_{t \leq T} f_i(x^{(t)}) \leq f_i\left(\frac{1}{T}\sum_{t \leq T} x^{(t)}\right) = f_i(\widetilde{x}).$$

17.3 Solving SDPs with Multiplicative Weights

Suppose we now move to solving SDPs of the form

$$\begin{aligned} \min\, & C \bullet X \\ \text{s.t. } & A_i \bullet X \geq b_i \\ & X \succeq 0 \end{aligned}$$

note that the first few constraints are linear constraints. It is only the psd-ness constraint that is non-linear—so we only need to modify our MW algorithm by absorbing the $X \succeq 0$ constraint into the oracle. It will be also convenient to require the constraint $\mathrm{tr}(X) = 1$ as well: usually we can guess the trace of the solution X. (If the trace of the solution we seek is not 1 but R, we can scale the problem by R to get unit trace.) Then the oracle we must implement is this:

> Let $K := \{X \mid X \succeq 0, \mathrm{tr}(X) = 1\}$. Given a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\beta \in \mathbb{R}$, does there exist $X \in K$ such that $\mathbf{A} \bullet X \geq \beta$?

(Again, $\mathbf{A}, \beta$ will be obtained in the algorithm by setting $\mathbf{A}^{(i)} := p_i^{(t)} A_i$, and $\beta^{(i)} := p_i^{(t)} b_i$.) But we know from Lecture 12 that this is equivalent to asking whether the maximum eigenvalue of the symmetric matrix $\mathbf{A}$ is at least β. Indeed, if this is so, and if $\lambda_{\max}$ is the maximum eigenvalue of $\mathbf{A}$ with unit eigenvector x, then

$$\begin{aligned} \mathbf{A} \bullet (xx^\top) &= \mathrm{tr}(\mathbf{A}^\top xx^\top) \\ &= \mathrm{tr}(\mathbf{A}xx^\top) \\ &= \mathrm{tr}(\lambda_{\max} xx^\top) \\ &= \lambda_{\max} \end{aligned}$$

so our oracle should return $X = xx^\top$, else it should return `Infeasible`. Moreover, using the Observation #4 on the previous page, it suffices to return x such that $x^\top \mathbf{A} x \geq \lambda_{\max} - \epsilon$. How fast this can be done depends on the particular structure of the matrix $\mathbf{A}$; in the next section we see that for the max-cut problem, the matrix $\mathbf{A}$ itself is psd, and hence we can find such an x relatively quickly.

17.3.1 Example: Max Cut

This part is loosely based on the paper of Klein and Lu [KL96]. Recall the Max Cut SDP we derived in Lecture 12:

$$\begin{aligned} \max \frac{1}{4} L \bullet X \\ \text{s.t. } (e_i e_i^\top) \bullet X = 1 \quad \forall i \\ X \succeq 0 \end{aligned}$$

As usual, we will think of the edge weights as summing to 1: this means that $\text{tr}(L) = \sum_i L_{ii} = -\sum_{i \neq j} L_{ij} = 1$. If we let $b = \text{OPT}$ and scale X by $1/n$, we are looking for feasibility of the constraints:

$$\begin{aligned} \frac{n}{4b} L \bullet X \geq 1 \\ n(e_i e_i^\top) \bullet X = 1 \quad \forall i \\ X \succeq 0 \end{aligned}$$

Finally, if we take $K = \{X \mid X \succeq 0, \text{tr}(X) = 1\}$, the above SDP is equivalent to finding $X \in K$ such that

$$\begin{aligned} \frac{n}{4b} L \bullet X \geq 1 \\ n(e_i e_i^\top) \bullet X \geq 1 \quad \forall i \end{aligned}$$

(This is because $\text{tr}(X) = 1$ means $\sum_i X_{ii} = 1$. Since we have the constraints $n(e_i e_i^\top) \bullet X = nX_{ii} \geq 1$, this means $X_{ii} = 1/n$ for all i.) By the discussions of the previous section, our oracle will need to check whether there exists $X \in K$ such that $D^{(t)} \bullet X \geq 1$, where

$$D^{(t)} = p_0^{(t)} \frac{n}{4b} L + \sum_{i=1}^{n} p_i^{(t)} n(e_i e_i^\top).$$

And again, is is equivalent to checking whether $\lambda_{\max}(D^{(t)}) \geq 1$.

Implementing the oracle. It is useful to note that $D^{(t)}$ is positive semidefinite: indeed, it is the sum of the Laplacian (which is psd), and a bunch of matrices $e_i e_i^\top$ (which are psd).

> **Note:** In Homework #6, you will show that for any psd matrix D, the "power method" starting with a random unit vector can find $x \in K$ such that $D \bullet (xx^\top) \in [\lambda_{\max}(D)/(1+\epsilon), \lambda_{\max}(D)]$. The algorithm succeeds with high probability, and runs in time $O(\epsilon^{-1} m \log n)$ time, where m is the number of edges in G (and hence the number of non-zeroes in L).

So we can run this algorithm: if it answers with an x such that $D^{(t)} \bullet (xx^\top)$ is smaller than $1/(1+\epsilon)$, we answer saying $\lambda_{\max}(D^{(t)}) < 1$. Else we return the vector x: this has the property that $D^{(t)} \bullet (xx^\top) \geq 1/(1+\epsilon) \geq 1 - \epsilon$. Now, using the Observation #4 on the previous page, we know this will suffice to get a solution that has an $O(\epsilon)$ infeasibility.

Bounding the width. The width of our algorithm is the maximum possible magnitude of $D^{(t)} \bullet X$ for $X \in K$, i.e., the maximum possible eigenvalue of $D^{(t)}$. Since $D^{(t)}$ is positive

semidefinite all of its eigenvalues are non-negative. Moreover, $\mathrm{tr}(L) = 1$, and also $\mathrm{tr}(e_i e_i^\top) = 1$. So

$$\begin{aligned}\lambda_{\max}(D^{(t)}) &\leq \sum \lambda_i(D^{(t)}) = \mathrm{tr}(D^{(t)}) \\ &= \mathrm{tr}\left(p_0^{(t)}\frac{n}{4b}L + \sum_{i=1}^{n} p_i^{(t)} n(e_i e_i^\top)\right) \\ &= p_0^{(t)}\frac{n}{4b}\mathrm{tr}(L) + \sum_{i=1}^{n} p_i^{(t)}\, n\,\mathrm{tr}(e_i e_i^\top) \\ &= n(1 + 1/4b).\end{aligned}$$

Finally, the max-cut values we are interested in lie between $1/2$ (since the max-cut is at least half the edge-weight) and 1. So $b \in [1/2, 1]$, and the width is $O(n)$.

Running Time. Setting the width $\rho = O(n)$ gives us a runtime of

$$O\left(\frac{n^2 \log n}{\epsilon^2} T_{oracle}\right)$$

which we can reduce to

$$O\left(\frac{n \log n}{\epsilon^2} T_{oracle}\right)$$

using Lemma 17.3, since our cost vectors can be made all nonnegative. Finally, plugging in our oracle gives a final runtime of

$$O\left(\frac{mn \log^2 n}{\epsilon^3}\right),$$

where m is the number of edges in our graph.

Note: We can now scale the "average" matrix $\widetilde{X}$ by n to get a matrix $\widehat{X}$ satisfying:

$$\begin{aligned}\tfrac{1}{4}L \bullet \widehat{X} &\geq b(1-\epsilon) \\ \widehat{X}_{ii} &\geq 1-\epsilon \qquad \forall i \\ \mathrm{tr}(\widehat{X}) &= n \\ \widehat{X} &\succeq 0\end{aligned}$$

The attentive reader will observe that this is not as nice as we'd like. We'd really want each $\widehat{X}_{ii} \in [1-\epsilon, 1+\epsilon]$—then we could transform this solution into one where $X_{ii} = 1$ and $\frac{1}{4}L \bullet \widehat{X} \geq b(1 - \epsilon^{O(1)})$.

What we have only guarantees that $X_{ii} \in [1-\epsilon, 1+n\epsilon]$, and so we'd need to set $\epsilon \leq 1/n$ for any non-trivial guarantees. This would still give us a run-time of $O(\epsilon^{-3} mn^4 \mathrm{poly}\log n)$—still polynomial (and useful to examplify the technique), but it could be better. One can avoid this loss by defining K differently—in fact, in a way that is similar to Section 17.2.1—the details can be found in [KL96]. One can do even better using the *matrix* multiplicative weights algorithms: see, e.g., [AK07, Ste10].

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 17.1

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1