Advanced Algorithms 10/24, 2017

Lecture 6

Lecturer: Mohsen Ghaffari *Scribe: Timon Gehr*

This draft has not been checked by the lecturer yet.

We continue our discussion of approximation algorithms. So far, we have talked about greedy approximation algorithms and PTAS/FPTAS for optimization problems, as well as FPRAS for counting problems. Many of the discussed methods were rather combinatorial and quite ad-hoc. It was not necessarily clear how to invent such algorithms from scratch.

Here, we are going to discuss rounding for linear programs. This is a more systematic approach, that helps for many problems. Usually, we will not need to understand the combinatorial structure of the problem in depth to find good approximation algorithms.

The basic idea is to take a linear program[1], find an optimal (fractional) solution $\mathbf{x}^*$ to it and then round it to obtain an integer solution $\mathbf{y}$ which has a combinatorial interpretation. We then usually try to relate the optimal solution $\mathbf{x}^*$ of the linear program to the rounded solution in order to prove approximation factors.

Note that rounding of linear programs could be its own course, as it is a deep area with many people working on it. However, we will only spend one lecture on this topic, to see a few of the relevant ideas.

1 Set Cover: Deterministic and Randomized Rounding

Set cover is a very basic problem in approximation algorithms. It could be seen as the canonical problem.

A set cover instance consists of a ground set $U = \{a_1, a_2, \ldots, a_n\}$ and the sets $S_1, S_2, \ldots S_m \subseteq U$ with costs $c_1, \ldots, c_m \in 2^+$. The goal is to find a collection of sets covering all elements with minimal total cost. For simplicity, we will assume that the given instance has at least one solution, i.e. $\bigcup_i S_i = U$.

ILP for Set Cover. We can phrase set cover as an integer linear program.

Variables: $x_j \in \{0, 1\}$ for $j \in \{1, \ldots, m\}$. x_j indicates whether set S_j is taken.
Objective and Constraints:

$$\min \sum_{j=1}^{m} c_j \cdot x_j$$
$$\text{s.t. } \forall i \in \{1, \ldots, m\}, \sum_{\substack{j \in \{1,\ldots,m\} \\ i \in S_j}} x_j \geq 1.$$

The objective minimizes the total cost under the constraint that for each element, there is at least one set that covers it. From an optimal solution $\mathbf{x}^*$ to this ILP, we can recover the optimal solution to the set cover problem by forming the collection of all sets S_j whose indicator x_j^* is equal to 1.

LP Relaxation for Set Cover. We can obtain a linear program from the given ILP by requiring that $x_j \geq 0$[2] instead of $x_j \in \{0, 1\}$. This is a linear program with a polynomial number of variables and constraints, therefore we can find an optimal solution $\mathbf{x}^*$ in polynomial time.

Suppose the optimal feasible solution of the LP is $\mathbf{x}^*$, which, in general, contains fractional values. Our goal is to round the components of $\mathbf{x}^*$ to $\{0, 1\}$.

Like during our discussion of greedy algorithms for set cover, we will consider two different regimes.

[1] Often, this is just a relaxation of an integer linear program formulation of the problem.

[2] A more natural relaxation is to require $x_j \in [0, 1]$, but $x_j \leq 1$ is implicitly satisfied because we are minimizing a linear form with non-negative coefficients and setting a variable to a value larger than 1 is never necessary to satisfy a constraint.

Regime 1: Each element appears only in a small number of sets. Suppose each element appears in only f sets. To make the setting even more concrete, we will first assume that $f = 2$, which reduces the problem to the standard vertex cover problem: The sets $S_1, \ldots, S_m$ correspond to vertices of a graph whose edges correspond to elements. Each edge connects the two sets that contain it. The goal is to select a set of vertices (sets) that cover all the edges (elements).

Suppose someone gives us the optimal (fractional) solution $\mathbf{x}^*$. How can we round those values in order to recover a meaningful solution? In fact, it suffices to perform deterministic rounding of x_v^* to 1 if $x_v^* \geq 1/2$ and to 0 otherwise, as:

(1) This yields a cover: As $\mathbf{x}^*$ satisfies all LP constraints, we know for every edge $e = \{u, v\}$ that $x_u^* + x_v^* \geq 1$. This implies that at least one of x_u^* and x_v^* is at least $1/2$. Therefore, for each edge, the variable corresponding to at least one of its endpoints will be rounded up to 1, and therefore this endpoint will be in the computed collection.

(2) The cover is a 2-approximation: Let $\mathbf{y}$ and $\mathbf{y}^*$ be the rounded and optimal solutions to the ILP. We then have

$$\text{cost}_{\text{ILP}}(\mathbf{y}) = \text{cost}_{\text{LP}}(\mathbf{y}) \leq 2 \cdot \text{cost}_{\text{LP}}(\mathbf{x}^*) \leq 2 \cdot \text{cost}_{\text{ILP}}(\mathbf{y}^*).$$

The cost of y is the same in the ILP and the LP as they use the same objective function, rounding increases the total cost by at most a factor 2, and the optimal solution to the (more constrained) ILP cannot cost less than the optimal solution to the (less constrained) LP.

This can be easily generalized to give an f-approximation also for $f \neq 2$ (i.e. vertex cover in a hypergraph where each edge contains only f nodes): We round x_j^* to 1 if $x_j^* \geq f^{-1}$ and to 0 otherwise. (Note that this strategy works even if not all hyperedges have the same cardinality as long as none of them exceeds f.)

Regime 2: General regime (f is potentially high). In the general case, how can we think about fractional solutions and how to turn them into integral points that are not too far away? One useful idea is *randomized rounding*: We will interpret the fractional value as the probability of the variable being equal to 1, or more formally, we interpret x_j^* as $\Pr[y_j = 1]$ for $\mathbf{y} \in \{0, 1\}^m$ where the components of $\mathbf{y}$ are independent.

Back to set cover, we investigate the properties of this interpretation: The expected cost of $\mathbf{y}$ is

$$\mathbf{E}[\text{cost}(\mathbf{y})] = \mathbf{E}\left[\sum_{j \in \{1,\ldots,m\}} c_j \cdot y_j\right] = \sum_{j \in \{1,\ldots,m\}} c_j \cdot \mathbf{E}[y_j] = \sum_{j \in 1,\ldots,m} c_j \cdot x_j^* = \text{cost}(\mathbf{x}^*).$$

In other words, the expected cost of the solution is the cost of the optimal solution to the LP. The catch is that the vector $\mathbf{y}$ might not correspond to a cover. However, we claim that the chances that any given element is covered are good. Why? Suppose the element a_i is in some sets $S_{j_1}, \ldots, S_{j_k}$. Then, we know that $x_{j_1}^* + x_{j_2}^* + \cdots + x_{j_k}^* = \sum_{l=1}^k x_{j_l}^* \geq 1$. The probability that we will not cover the element a_i is

$$(1 - x_{j_1}^*) \cdot (1 - x_{j_2}^*) \cdots (1 - x_{j_k}^*) \leq e^{-x_{j_1}^*} \cdot e^{-x_{j_1}^*} \cdots e^{-x_{j_k}^*} = e^{-\sum_{l=1}^k x_{j_l}} \leq 1/e.$$

Now we have a solution that covers any single element with high probability and in expectation has the same cost as the optimal solution. How to turn this into a full solution? One way is to run the process multiple times and collect all the sets that were picked in one of the repetitions. For each element, the probability of not being covered decreases exponentially in the number of repetitions. If we run the process $C \cdot \log n$ times for a sufficiently large constant C, we can union bound over all elements to get a cover and use Markov to show that we do not pick too many edges. This way, we get a $(4C \cdot \log n)$-approximation with constant probability (which we can amplify by repeating).[3]

2 Minimizing Congestion in Routing

The following is a result by Raghavan and Tompson [RT87]. Raghavan was working at IBM at the time and had to solve a practical problem. The bigger-picture name of this problem is integer multi-commodity flow (by multi-commodity, we mean that instead of only one (s, t) pair, we have different (s_i, t_i) for $1 \leq i \leq k$ and the flow needs to deliver incompatible commodities from each s_i to its t_i).

[3] Depending on the range of c_j it is sometimes possible to get a better concentration.

We will first consider a slightly simpler setting: We have some directed graph $G = (V, E)$ on n nodes and m edges. Each edge $e \in E$ has some capacity $c(e)$. We also have a set of k source-destination pairs $(s_1, t_1), (s_2, t_2), \ldots (s_k, t_k)$. The pair (s_i, t_i) has an associated *demand* d_i, and we are given a collection of paths P_i connecting s_i to t_i for each $1 \leq i \leq k$.

Our goal will be to obtain path-type flows: For each pair (s_i, t_i) we will pick exactly one path p from P_i and push a flow of size d_i through this path. Ideally, we would want to do this while satisfying the constraint that no edge $e \in E$ becomes overloaded: the sum of all demands associated to paths that contain e should not exceed $c(e)$. Actually, for what follows, we will allow the load of an edge to exceed its capacity by a factor λ, called *congestion*. We can then minimize λ, which gives us a well-defined way to talk about how well our computed solution approximates an optimal routing.

To clean up the discussion, we only consider the case $\lambda \geq 1$ and we assume that for each path, the demand is at most the capacity of all the edges that lie on it: $d_i \leq c(e)$ for all $e \in p$ and $p \in P_i$. This means that a single commodity will not overload an edge.

Mixed ILP for Congestion This can be written as a mixed ILP.

Variables: $\lambda \geq 1$, the congestion. $x_{ip} \in \{0, 1\}$ for $i \in \{1, \ldots, k\}, p \in P_i$. x_{ip} indicates that we picked path $p \in P_i$ to connect s_i to t_i.
Objective and Constraints:

$$\begin{aligned} &\min \lambda \\ \text{s.t.} \quad &\sum_{p \in P_i} x_{ip} = 1 \\ \text{and } \forall e \in E, &\sum_{i=1}^{k} d_i \sum_{\substack{p \in P_i \\ e \in p}} x_{ip} \leq \lambda \cdot c(e). \end{aligned}$$

The objective minimizes congestion under the constraint that we pick exactly one path from P_i for each i, and under the constraint that no edge can exceed its capacity by more than a factor λ.

The number of variables and constraints is polynomially bounded in the input size, namely by the total number of paths that we need to consider.

LP Relaxation for Congestion We will again relax the domain of x_{ip} and only require $x_{ip} \geq 0$ instead of $x_{ip} \in \{0, 1\}$. ($x_{ip} \leq 1$ is then implied by the first constraint.) We get an LP and, in polynomial time, a (fractional) optimal solution $(\lambda^*, \mathbf{x}^*)$ to this LP.

Randomized Rounding We can apply randomized rounding in the following way: for each pair (s_i, t_i), we pick a path $p \in P_i$ with probability proportional to x^*_{ip}. We pick exactly one path for each i. Let y_{ip} be indicator variables for the choice of the path. The choices of paths for different i are independent. Let κ denote the congestion in the resulting routing (i.e., the maximal factor by which some edge exceeds its capacity).

What approximation factor do we get for congestion when we use this randomized rounding scheme? Note that we have assumed that the d_i are not too large, i.e. $d_i/c(e) \leq 1$, and that κ is the minimal value that satisfies

$$\sum_i d_i/c(e) \sum_{\substack{p \in P_i \\ e \in P}} y_{i,p} \leq \kappa,$$

for all edges. This is a summation of independent random variables, where each contributes at most 1 (as $d_i/c(e) \leq 1$ and for each i, at most one of the indicators $y_{i,p}$ is 1). As $(\lambda^*, \mathbf{x}^*)$ is a solution to the LP, we can conclude using linearity of expectation that for each edge $e \in E$, the expected congestion is at most λ^*. If we union bound over all edges to conclude that none of them is loaded and then apply the most basic Chernoff bound, we obtain that $\kappa \leq O(\log n) \cdot \lambda^*$ with probability at least $1 - 1/\text{poly}(n)$. However, we can do a bit better if we use a more careful Chernoff bound, namely

$$\Pr[Y \geq (1 + \delta)\mathbf{E}[Y]] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu,$$

for $\mu = E[Y]$ and $\delta \geq 0$. (This is another form of Chernoff, valid for the upper tail.)

It suffices to set $\delta = \Theta(\log n / \log \log n)$. Then, the probability of deviation becomes smaller than $1/\text{poly}(n)$, and we get a $O(\log n / \log \log n)$-approximation for minimizing congestion.

Generalization: Paths not given nor bounded. We were only able to efficiently find a solution because the number of paths, and hence the number of variables, was polynomially bounded. We now relax this assumption. Namely, we want to solve the following problem:

Given k pairs of source and destination, pick one path in G between each (s_i, t_i) to minimize overall congestion.

It is possible to naively apply the insights we have collected so far by using the set of all paths from s_i to t_i as the set P_i. However, the number of paths in P_i can in general be exponential in n, and therefore this solution will not be efficient. There are at least two ways around the problem:

- Use the ellipsoid method on the dual problem. (We will not discuss this here.)
- Write the problem as a flow LP. This will still give a $O(\log n / \log\log n)$-approximation in the same way, but in polynomial time.

Flow LP for Multi-Commodity Flow (Version with Congestion). As indicated, there is a more efficient LP that will be sufficient for our purposes.

Variables: $\lambda \geq 0$, the congestion. $f(e, i) \in [0, 1]$, denoting the fraction of the total flow of commodity i between s_i and t_i that passes over edge e.
Objective and Constraints:

$$\begin{aligned} &\min \lambda \\ \text{s.t. } &\forall e \in E, \sum_{i=1}^{k} d_i \cdot f(e,i) \leq \lambda \cdot c(e) \\ &\text{and } \forall i \in \{1,\dots,k\}, \sum_{e \in \text{out}(s_i)} f(e,i) - \sum_{e \in \text{in}(s_i)} f(e,i) = 1 \\ &\qquad \text{and } \forall v \in V \setminus \{s_i, t_i\}, \sum_{e \in \text{out}(v)} f(e,i) - \sum_{e \in \text{in}(v)} f(e,i) = 0 \end{aligned}$$

The LP defines the problem in terms of a polynomial number of local constraints: The total flow over an edge e (which is a sum over all different commodities) must be below $\lambda \cdot c(e)$, the entire flow of commodity i between s_i and t_i must leave the source node s_i, and all nodes except sources and destinations need to preserve the flow of each commodity. (Note that the constraint that the entire flow of commodity i between s_i and t_i needs to enter t_i is implicitly satisfied.)

We can solve the LP in polynomial time to get an optimal solution $(\lambda^*, f^*(e, i))$.[4] Note that this λ^* is also the optimal λ in the naive solution that enumerates all paths; the new LP encodes the same basic setting, but more efficiently. However, the solution no longer contains explicit fractional indicators for paths. How can we apply the ideas we derived in the setting where the paths were given explicitly?

Flow Decomposition There is an easy way to decompose the flow f^* into a polynomial number of path-type flows: We consider an arbitrary path from s_i to t_i using only edges e for which $f^*(e, i) > 0$. We take the edge with minimum flow of commodity i, and use that flow value as the weight of the path. We can then subtract the path from the flow of commodity i. We repeat those steps until the flow of commodity i is zero. For each fixed i, this procedure terminates after at most m iterations, because in each iteration, at least one edge disappears from consideration. We can use the obtained paths as the set P_i and the weight of a path $p \in P_i$ as the fractional indicator x^*_{ip}. The size of all P_i is polynomially bounded. This shows that there are optimal solutions to the exponentially large LP from before where only polynomially many paths have support, and it gives an easy way to construct such solutions. Randomized rounding can then be applied just as before, and it will be efficient.

3 Scheduling on Unrelated Parallel Machines

We will now discuss a generalization of the makespan problem. In order to get a good approximation, we will again use an LP formulation. However, this time, randomized rounding will not be sufficient, and

[4] We assume that our LP solver produces a basic feasible solution, to avoid dealing with the case where the solution contains cyclic flows.

we will instead use a more elaborate form of rounding. Randomized rounding does not always work, and this time, we need to understand some of the combinatorial structure of the problem in order to round in a good way. Note that even for randomized rounding, there are many ways to do it: For example, we could also define a function $f : [0,1] \to [0,1]$ other than the identity that maps the components of the optimal solution x^* to probabilities. One instance where this "non-linear" randomized rounding gives a better solution is given as an exercise.

Setting. The setting is close to what we discussed in the past: We have a collection J of n jobs/tasks, and we have a set M of m machines to process them. Previously, each job had a fixed size regardless of which machine processes it. Now we think about a more general case: $t_{ij} \geq 0$ is the time for machine j to process job i. The t_{ij} can be arbitrary. This means that some machines can be a lot better than others on given jobs, and a lot worse on others. This is what we mean when we speak of "unrelated" machines.

How can we formulate this problem as an LP?

Naive LP. The most obvious way is to take x_{ij} to be an indicator for assigning job i to machine j and then optimizing the following objective:

$$\begin{aligned} &\min t \\ \text{s.t. } &\forall \text{machine } j, \sum_{i=1}^{n} t_{ij} \cdot x_{ij} \leq t \\ \text{and } &\forall \text{job } i, \sum_{j=1}^{m} x_{ij} \geq 1 \\ \text{and } &\forall \text{job i, machine } j,\ x_{ij} \geq 0 \end{aligned}$$

(Here, t is an additional variable giving an upper bound for the finishing time of the last job.)

The problem with this LP is that the best fractional solution and the best integral solution can be far apart (the LP has a large "integrality gap"). Namely, the fractional solution to the LP is allowed to distribute a big job among different machines. In particular, consider an instance with a single large job with the same processing time on all machines. The integral solution needs to assign this job to one of the machines, while the fractional solution can evenly split it among all m machines. Therefore, we can lose a factor as big as m.

Note that we get a correct solution for both the fractional and the integral case. The problem is that we want to relate the two solutions in order to prove a small approximation factor, and this is not possible for the given LP.

Improved LPs. We will now change the LP a bit. Suppose somebody tells us that the processing time t is at most a certain λ. We will find a way to check this claim up to an approximation factor. Once we have the upper bound λ, only some of the assignments make sense, namely

$$S_\lambda = \{(i,j) \mid t_{ij} \leq \lambda\}.$$

If a single job i takes more than λ time on a given machine j, we cannot schedule it on this machine at all. The set S_λ contains all assignments of single jobs to single machines that are not ruled out in this way. We can now write an LP on $|S_\lambda|$ variables that is specific to a given value of λ.

$$\begin{aligned} &\text{LP}(\lambda): \\ &\forall \text{machine } j, \sum_{(i,j) \in S_\lambda} t_{ij} \cdot x_{ij} \leq \lambda \\ &\forall \text{job } i, \sum_{(i,j) \in S_\lambda} x_{ij} \geq 1 \\ &\forall (i,j) \in S_\lambda,\ x_{ij} \geq 0 \end{aligned}$$

This time, we just want to check for feasibility. We have constraints (defining a polytope), but there is no objective function. Using binary search[5], we can find the smallest λ^* for which we can find a fractional solution of LP(λ^*). Note that it is easy to initialize the binary search, as there are trivial lower and upper bounds on λ^*, for example 0 and the sum of all processing times.

Now, somebody gives us a fractional solution of LP(λ^*): Suppose this solution is $\mathbf{x}^* \in [0,1]^{|S_\lambda|}$. Instead of just assuming that $\mathbf{x}^*$ is an arbitrary solution, we will also assume that $\mathbf{x}^*$ is a vertex of the polytope.[6]

Rounding Algorithm. We want to round $\mathbf{x}^*$ to an integral assignment of jobs to machines. This is a place where the rounding will be not very direct. However, there are some assignments for which rounding is obvious, in the sense that $x_{ij} = 1$. For those cases, we can just assign the job i to machine j. So in the following we can assume that all variables have fractional values. The support of $\mathbf{x}^*$ forms a graph H on jobs and machines. (The edge (i,j) is present iff $x^*_{ij} \in (0,1)$.) This is a bipartite graph with jobs on one side and machines on the other side.

We will prove that there is a perfect matching in H, and that we obtain a 2-approximation by combining the obvious assignments with this perfect matching. More explicitly, the algorithm is this:

1. For edges (i,j) such that $x_{ij} = 1$, assign job i to machine j. Let I be the set of jobs assigned in this step. ($I \subseteq J$.)
2. Let H be the bipartite graph on jobs and machines where job i and machine j are connected iff $x_{i,j} \in (0,1)$.
3. Find a matching in H that is perfect for the remaining jobs F. ($F = J \setminus I$.)
4. For each matching edge (i,j), assign job i to machine j.

Jobs assigned in step 1 take at most time λ^* to complete. With the matching, each machine gets at most one more job, whose cost is also at most λ^* (by definition of the set S^*_λ). Therefore, we can construct a solution with cost at most $2 \cdot \lambda^*$. As λ^* is a lower bound on the optimal cost, this proves that the algorithm is a 2-approximation. This is a much more careful rounding method than randomized rounding.

Correctness of Rounding. It remains to be argued that the perfect matching exists.

Definition 1. *A* pseudo-forest *is a graph for which each connected component is a pseudo-tree, where a pseudo-tree is either a tree or a tree with an additional edge*

Claim: H is a bipartite pseudo-forest, where each leaf is a machine. It is easy to prove that such bipartite pseudo-forests admit a matching that is perfect for the jobs.

Lemma 2. *A bipartite pseudo-forest whose leaves are machines contains a matching that covers all the jobs.*

Proof. We argue separately for each connected component. If a connected component has a leaf that is not adjacent to a cycle, it is always possible to pick a leaf not adjacent to a cycle such that if we remove it and its (unique) neighbour, the produced components consist of at most one pseudo-tree with machines as leaves and possibly a few isolated machines.

We can repeatedly pick a machine that is such a leaf and match it to its neighbour. We then delete both the matched job and its machine, and further, we delete all machines that have become isolated. The resulting component is again a bipartite pseudo-tree whose leaves are machines. If we repeat this process until no more steps can be taken, we are left with a graph that is an even cycle, possibly with a few leaves attached. As those leaves are machines, we can ignore them and use one of the two perfect matchings of the cycle to assign the remaining jobs. By doing this for all components, we can assign all jobs to machines. □

We still need to prove that H is in fact a pseudo-forest.

[5] Feasibility of LP(λ) is a monotone property in λ, as for $\lambda \leq \lambda'$, a solution of LP(λ) can be extended to a solution of LP(λ').

[6] Also known as: a basic feasible solution, an extreme point, a generator of the polytope, a solution that cannot be written as a convex combination of other solutions, etc.

Lemma 3. *The graph H is a pseudo-forest.*

Proof. We first prove that H is a pseudo-tree if it is connected, and then we show how to extend the argument to the case where H has multiple connected components.

H connected. If H is connected, it suffices to show that $|E(H)| \leq |V(H)|$. We will use the fact that $\mathbf{x}^*$ is a vertex of the polytope. Let $r = |S_{\lambda^*}|$ be the number of variables. As $\mathbf{x}^*$ is a vertex, there must exist r linearly independent tight constraints. Among those r constraints, there can be at most m constraints on machines and n constraints on job assignments (c.f. the LP). Therefore, at least $r - (n+m)$ constraints of the form $x_{ij} \geq 0$ must be tight. Hence the number of variables that are non-zero in $\mathbf{x}^*$ is at most $r - (r - (n+m)) = n + m$. In particular, the number of fractional variables, corresponding to edges in $E(H)$, is at most $n + m = |V(H)|$.

(Aside: Recall that we defined I as the set of integrally-assigned jobs, while F is the set of fractionally-assigned jobs. As all jobs fall into exactly one of those categories, we have $|I| + |F| = n$. Each integral job i is associated to at least one non-zero variable $x^*_{ij_0}$, while a fractional job is associated to at least two non-zero variables $x^*_{ij_1}$ and $x^*_{ij_2}$ (because of the constraint that $\sum_j x_{ij} \geq 1$). We derive the inequality $|I| + 2 \cdot |F| \leq n + m$ and can conclude that $|I| \geq n - m$. This means that if the number of machines is small, many jobs will be assigned non-fractionally.)

H disconnected. Now we extend the argument to cover the case where H has multiple connected components. Given some such component H', we can restrict the solution $\mathbf{x}^*$ to this component, by ignoring all variables corresponding to assignments of single jobs to single machines that are not both in $V(H')$. We call this restricted vector $\mathbf{x'}^*$. Claim: $\mathbf{x'}^*$ is a vertex of the polytope obtained by only considering variables associated to edges connecting vertices in $V(H')$ and writing the analogue of the LP(λ^*) constraints for them. Proof: Otherwise, we can pick two feasible solutions $\mathbf{x'^*_1}$ and $\mathbf{x'^*_2}$ of the restricted LP such that $\mathbf{x'}^* = \frac{1}{2}(\mathbf{x'^*_1} + \mathbf{x'^*_2})$. We can then extend $\mathbf{x'^*_1}$ and $\mathbf{x'^*_2}$ to solutions for the unrestricted LP by filling in the missing components from $\mathbf{x}^*$. The resulting solutions have $\mathbf{x}^*$ as their middle point, which contradicts the assumption that $\mathbf{x}^*$ is a vertex of the polytope associated to the unrestricted LP. Therefore, the reasoning from above applies independently to all connected components of H and H is a pseudo-forest. □

All leaves of H are machines, because fractionally-assigned jobs have at least two neighbours. Using the two lemmas, we conclude that the rounding algorithm is correct.

References

[RT87] Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.