Lecture 6: Provable Approximation via Linear Programming

Lecturer: *Matt Weinberg* Scribe: *Sanjeev Arora*

One of the running themes in this course is the notion of *approximate solutions*. Of course, this notion is tossed around a lot in applied work: whenever the exact solution seems hard to achieve, you do your best and call the resulting solution an approximation. In theoretical work, approximation has a more precise meaning whereby you *prove* that the computed solution is close to the exact or optimum solution in some precise metric. We saw some earlier examples of approximation in sampling-based algorithms; for instance our hashing-based estimator for set size. It produces an answer that is whp within $(1+\epsilon)$ of the true answer. Today we will see many other examples that rely upon linear programming (LP).

Recall that most NP-hard optimization problems involve finding 0/1 solutions. Using LP one can find *fractional* solutions, where the relevant variables are constrained to take real values in $[0, 1]$.

Recall the example of the assignment problem from last time, which is also a 0/1 problem (a job is either assigned to a particular factory or it is not) but the LP relaxation magically produces a 0/1 solution (although we didn't prove this in class). Whenever the LP produces a solution in which all variables are 0/1, then this must be the optimum 0/1 solution as well since it is the best *fractional* solution, and the class of fractional solutions contains every 0/1 solution. Thus the assignment problem is solvable in polynomial time.

Needless to say, we don't expect this magic to repeat for NP-hard problems. So the LP relaxation yields a fractional solution in general. Then we give a way to *round* the fractional solutions to 0/1 solutions. This is accompanied by a mathematical proof that the new solution is provably approximate.

1 Deterministic Rounding (Weighted Vertex Cover)

First we give an example of the most trivial rounding of fractional solutions to 0/1 solutions: round variables $< 1/2$ to 0 and $\geq 1/2$ to 1. Surprisingly, this is good enough in some settings.

In the *weighted vertex cover* problem, which is NP-hard, we are given a graph $G = (V, E)$ and a weight for each node; the nonnegative weight of node i is w_i. The goal is to find a *vertex cover*, which is a subset S of vertices such that every edge contains at least one vertex of S. Furthermore, we wish to find such a subset of minimum total weight. Let $VC_{\min}$ be this minimum weight. The following is the LP relaxation:

$$\begin{array}{lll} \min & \sum_i w_i x_i & \\ & 0 \leq x_i \leq 1 & \forall i \\ & x_i + x_j \geq 1 & \forall \{i, j\} \in E. \end{array}$$

Let OPT_f be the optimum value of this LP. It is no more than $VC_{\min}$ since every 0/1 solution (including in particular the 0/1 solution of minimum cost) is also an acceptable fractional solution.

Applying *deterministic* rounding, we can produce a new set S: every node i with $x_i \geq 1/2$ is placed in S and every other i is left out of S.

Claim 1: S is a vertex cover.

Reason: For every edge $\{i, j\}$ we know $x_i + x_j \geq 1$, and thus at least one of the x_i's is at least $1/2$. Hence at least one of i, j must be in S.

Claim 2: The weight of S is at most $2OPT_f$.

Reason: $OPT_f = \sum_i w_i x_i$, and we are only picking those i's for which $x_i \geq 1/2$. $\square$.

Thus we have constructed a vertex cover whose cost is within a factor 2 of the optimum cost *even though we don't know the optimum cost per se.*

Exercise: Show that for the complete graph the above method indeed computes a set of size no better than 2 times OPT_f.

Remark: This 2-approximation was discovered a long time ago, and despite myriad attempts we still don't know if it can be improved. Using the so-called PCP Theorems Dinur and Safra showed (improving a long line of work) that 1.36-approximation is NP-hard. Khot and Regev showed that computing a $(2-\epsilon)$-approximation is UG-hard, which is a new form of hardness popularized in recent years. The bibliography mentions a popular article on UG-hardness.

2 Simple randomized rounding: MAX-2SAT

Simple randomized rounding is as follows: if a variable x_i is a fraction then toss a coin which comes up heads with probability x_i. If the coin comes up heads, make the variable 1 and otherwise let it be 0. The expectation of this new variable is exactly x_i. Furthermore, linearity of expectations implies that if the fractional solution satisfied some linear constraint $c^T x = d$ then the new variable vector satisfies the same constraint *in the expectation*. But in the analysis that follows we will in fact do something more.

A 2CNF formula consists of n boolean variables $x_1, x_2, \ldots, x_n$ and *clauses* of the type $y \vee z$ where each of y, z is a *literal*, i.e., either a variable or its negation. The goal in MAX2SAT is to find an assignment that *maximises* the number of satisfied clauses. (Aside: If we wish to satisfy all the clauses, then in polynomial time we can check if such an assignment exists. Surprisingly, the maximization version is NP-hard.) The following is the LP relaxation where J is the set of clauses and y_{j1}, y_{j2} are the two literals in clause j. We have a variable z_j for each clause j, where the intended meaning is that it is 1 if the assignment decides to satisfy that clause and 0 otherwise. (Of course the LP can choose to give z_j a fractional value.)

$$
\begin{aligned}
\max \quad & \textstyle\sum_{j \in J} z_j \\
& 1 \geq x_i \geq 0 \quad \forall i \\
& z_j \leq 1 \quad \forall j \\
& y_{j1} + y_{j2} \geq z_j \quad \forall j
\end{aligned}
$$

Where y_{j1} is shorthand for x_i if the first literal in the jth clause is the ith variable, and shorthand for $1 - x_i$ if the literal is the negation of the i variable. (Similarly for y_{j2}.)

If MAX-2SAT denotes the number of clauses satisfied by the best assignment, then it is no more than OPT_f, the value of the above LP. Let us apply randomized rounding to the fractional solution to get a 0/1 assignment. How good is it?

Claim: $\mathbf{E}$[number of clauses satisfied] $\geq \frac{3}{4} \times OPT_f$.

We show that the probability that the jth clause is satisfied is at least $3z_j/4$ and then the claim follows by linear of expectation.

If the clause is of size 1, say x_r, then the probability it gets satisfied is x_r, which is at least z_j. Since the LP contains the constraint $x_r \geq z_j$, the probability is certainly at least $3z_j/4$.

Suppose the clauses is $x_r \vee x_s$. Then $z_j \leq x_r + x_s$ and in fact it is easy to see that $z_j = \min\{1, x_r + x_s\}$ at the optimum solution: after all, why would the LP not make z_j as large as allowed; its goal is to maximize $\sum_j z_j$. The probability that randomized rounding satisfies this clause is exactly $1 - (1 - x_r)(1 - x_s) = x_r + x_s - x_r x_s$. Moreover, $(x_r + x_s)^2 - (x_r - x_s)^2 = 4x_s x_r$, so $x_s x_r \leq (x_r + x_s)^2/4$, and the probability that the clause is satisfied is at least $x_r + x_s - (x_r + x_s)^2/4$.

If $x_r + x_s \leq 1$, then this is clearly at least $3(x_r + x_s)/4$. If $x_r + x_s \geq 1$, then this is at least 3/4 (the partial derivative wrt x_s and x_r are both non-negative while $(x_r + x_s) \leq 2$). In either case it's at least $3z_j/4$. $\square$.

Remark: This algorithm is due to Goemans-Williamson, but the original 3/4-approximation is due to Yannakakis. The 3/4 factor has been improved by other methods to 0.94.

3 More Clever Rounding: Job Scheduling

Here, we'll consider a more clever rounding scheme that also starts from an LP relaxation due to Shmoys and Tardos. Consider the problem of scheduling *jobs* on *machines*. That is, there are n jobs and m machines. Processing job i on machine j takes time p_{ij}. Your goal is to finish *all* jobs as quickly as possible: that is, if $x_{ij} = 1$ whenever job i is assigned to machine j (and 0 otherwise), minimize $\max_j\{\sum_i x_{ij} p_{ij}\}$. This lends itself to a natural LP relaxation:

$$\begin{aligned} \min T& \\ x_{ij} \in [0,1] \quad & \forall i, j \\ \sum_j x_{ij} \geq 1 \quad & \forall i \\ T \geq \sum_i p_{ij} x_{ij} \quad & \forall j \end{aligned}$$

That is, we want to minimize the maximum load on any machine, subject to every job being assigned (at least) once. Unfortunately, this LP has a huge *integrality gap*. That is, the best fractional solution might be significantly better than the best integral solution. Why? Maybe there's only one job with $p_{1j} = 1$ for all machines j. Then the best fractional solution will set $x_{1j} = 1/m$ for all machines and get $T = 1/m$. But clearly the best integral schedule takes time 1. The problem is that we're asking for too much: if there's a single job that itself takes time $t >> T$ to process on every machine, we can't possibly hope to get a good approximation to T with an integral schedule. Instead, we'll consider the following modified relaxation:

$$\begin{aligned}
\min T \\
x_{ij} \in [0,1] \quad \forall i,j \\
\sum_j x_{ij} \geq 1 \quad \forall i \\
T \geq \sum_i p_{ij} x_{ij} \quad \forall j \\
x_{ij} = 0 \quad \forall i,j \text{ such that } p_{ij} > t
\end{aligned}$$

The problem with the previous example was that a single job had processing time 1, but $T = 1/m$ and we asked for a new schedule with processing time $O(1/m)$. Instead, we'll ask for one of time $T + t$. Note that if the optimal schedule has total processing time P, then the maximum time it takes to process any job is some $t \leq P$. So if we solve the above LP with this given t, the optimal schedule will be considered, and we'll have $T \leq P$ and $t \leq P$ for a 2-approximation. Note also that there are only nm different processing times in the input, so we can just try all of them and guarantee that one of them will give the correct guess of t.

Now for the rounding. For each machine j, let $w_j = \lceil \sum_i x_{ij} \rceil$. Make a bipartite graph with jobs on the left and machines on the right. Make $\lceil w_j \rceil$ copies of the machine j node, call them $j_1, \ldots, j_{w_j}$. Make a single node on the right for each job.

For each machine j, sort the jobs in decreasing order of p_{ij}, so that $p_{(1)j} \geq p_{(2)j} \cdots \geq p_{(n)j}$. Place edges from jobs to machine j in the following manner:

1. Initialize current-node $c := 1$. Initialize current-job $i := 1$. Initialize job-weight $w := x_{(1)j}$. Initialize node-weight-remaining $r := 1$.
2. While $(i \leq n)$:
   (a) If $w \leq r$, add an edge from job (i) to j_c of weight w. Update $r := r - w$, update $i := i + 1$, $w := x_{(i)j}$ (the newly updated i). Keep $c := c$.
   (b) Else, add an edge from job (i) to c of weight r. Update $w := w - r$, update $r := 1$, update $c := c + 1$. Keep $i := i$.

In other words, starting from the slowest jobs, we put edges totalling weight x_{ij} from job i to (possibly multiple) nodes for machine j. We do so in a way such that the slowest jobs are on the earliest-indexed copies, and that each copy has total incoming weight at most 1 (actually all but the last copy have incoming weight exactly one, and the last copy has weight at most one). Now our rounding algorithm simply takes any matching with n edges, ignoring the weights (i.e. matches every job somewhere) in this graph. We first need to claim that such a matching exists, then claim that the total processing time is not too large.

Proposition 1
In the bipartite graph defined above, there exists a matching of size n.

Proof: Because the total edge weight coming out of job i into a copy of machine j is x_{ij} for all i, j, the total edge weight coming out of job i in total is 1. Moreover, the total edge weight coming into each copy of machine j is at most 1. Therefore, we have constructed a fractional matching of size n, and there is also an integral matching of size n (this is the same fact we didn't prove in Lecture 5).

To see this a little more concretely, recall that one way to find a matching in a bipartite graph is via max-flow. Our "fractional" matching has explicitly defined a flow in the corresponding max-flow graph that has size n, so there must also exist a matching of size n. $\square$

The above argues that the algorithm is well-defined (note that the proof is not "complete" in the sense that we didn't prove that fractional matchings imply integral matchings, and not everyone already saw how to find bipartite matchings via max-flow. But it's "formal" in the sense that the proof is complete with either of these outside theorems). Now we need to argue that the total processing time is good.

Proposition 2
The total processing time until all jobs are completed in any schedule output by the algorithm is at most $T + t$.

Proof: We'll show that for all machines j, the total processing time of jobs assigned to j is at most $T + t$ (which is equivalent to the proposition statement). Note first that *every* job with an edge to node j_c has a lower processing time than *any* job with an edge to node j_{c-1}. So let T_c denote the processing time of the slowest job with an edge to j_c. Then we have $T \geq \sum_i x_{ij} p_{ij} \geq \sum_{c=2}^{w_j} T_c$. This is because the jobs assigned to node j_c account for $\sum_i x_{ij} = 1$, and each have $p_{ij} \geq T_{c+1}$. Finally, observe that $T_1 \leq t$, as by definition we didn't allow any jobs to be placed on machines where their processing time exceeded t. So $T + t \geq \sum_c T_c$. Finally, observe that the maximum possible processing time of the unique job assigned to node j_c is T_c, so the total processing time of machine j is $\sum_c T_c \leq T + t$. $\square$

This is a really influential rounding scheme that accomplishes much more than just what is proved here - see the original paper and follow-ups for details.

Bibliography

1. *New 3/4-approximation to MAX-SAT* by M. X. Goemans and D. P. Williamson, SIAM J. Discrete Math 656-666, 1994.

2. *Randomized rounding: A technique for provably good algorithms and algorithmic proofs* by P. Raghavan and C. T. Tompson, Combinatorica pp 365-374 1987.

3. *On the hardness of approximating minimum vertex cover* by I. Dinur and S. Safra, Annals of Math, pp 439485, 2005.

4. *Approximately hard: the Unique Games Conjecture.* by E. Klarreich.
Popular article on `https://www.simonsfoundation.org/`

5. *An approximation algorithm for the generalized assignment problem.* by D.B. Shmoys and E. Tardos, Mathematical Programming, pp 461-474, 1993.