Lecture 3: Linear Programming Relaxations and Rounding

1 Approximation Algorithms and Linear Relaxations

For the time being, suppose we have a minimization problem. Many times, the problem at hand can be cast as an integer linear program: minimizing a linear function over integer points in a polyhedron. Although this by itself doesn't benefit us, one can remove the integrality constraints on variables to get a lower bound on the optimum value of the problem. Linear programs can be solved in polynomial time to give what is called a "fractional solution". This solution is then "rounded" to a integer solution, and one normally shows that the rounded solution's cost can is within a factor, say ρ, of the fractional solution's cost. It is clear then that the solution's cost is within ρ of the optimum cost.

Example. Let us take the set cover problem we did in the first class. Here is an integer program for it.

$$\min \quad \sum_{j=1}^{m} c(S_j)x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j:i\in S_j} x_j \geq 1 \qquad \forall i \in U \tag{2}$$

$$x_j \in \{0,1\} \qquad \forall j = 1...m \tag{3}$$

To get the LP relaxation, we replace the constraint (3) by $0 \leq x_j \leq 1$. Suppose x is an optimal solution to the resulting LP (to stand for linear program henceforth). By the discussion above, we get $\texttt{opt} \geq \sum_{j=1}^{m} c(S_j)x_j$.

Now suppose we are told that each element appears in at most f sets; that is, the frequency of each element is at most f. For example, in the special case of vertex cover, the frequency of each element (edge) is 2. What can we say then? Well, let's look at (2). For each element in U, there are at most f terms in the corresponding summation, and thus at least one of the x_j's must be $\geq 1/f$. In other words, the algorithm which picks all sets S_j with $x_j \geq 1/f$, that is rounds their value up to 1, is assured to be a set cover. Furthermore, the cost of this "integral" solution is at most $f \cdot \sum_{j=1}^{m} c(S_j)x_j \leq f \cdot \texttt{opt}$. Thus we get a simple f-approximation to set cover using linear programs.

Given a particular LP relaxation for an optimization problem, there is a limit on the best approximation factor obtainable by the process described above. Take an instance I of the problem (set cover, say), and let $\texttt{lp}(I)$ be the value of the LP relaxation, and $\texttt{opt}(I)$ be the value of the optimum. Since our solution can't cost less than $\texttt{opt}(I)$, the best approximation factor we can hope

for in this instance is $\frac{\texttt{opt}(I)}{\texttt{lp}(I)}$. Thus, the best approximation factor one can get for the minimization problem via this LP relaxation is at most

$$\sup_{\text{instances } I} \frac{\texttt{opt}(I)}{\texttt{lp}(I)}$$

This quantity is called the integrality gap of the LP relaxation and is very useful in judging the efficacy of a linear program. What is the integrality gap of the LP relaxation of (1) in terms of f?

1.1 Some facts regarding linear programs.

A minimization linear program, in its generality, can be denoted as

$$\begin{aligned} \min \quad & c \cdot x \\ \text{subject to} \quad & Ax \geq b \end{aligned} \tag{4}$$

The number of variables is the number of columns of A, the number of constraints is the number of rows of A. The ith column of A is denoted as A_i, the ith row as a_i. Note that constraints of A could also include constraints of the form $x_i \geq 0$. These non-negativity constraints are often considered separately (for a reason). Let's call the other constraints *non-trivial* constraints. For the discussion of this section, let n be the number of variables and let m denote the number of non-trivial constraints. Thus the number of rows in A are $(m + n)$ and $\texttt{rank}(A) = n$ (since it contains an $I_{n\times n}$ as a submatrix).

A solution x is a feasible solution if it satisfies all the constraints. Some constraints may be satisfied with equality, others as a strict inequality. A feasible solution x is called *basic* if the constraints satisfied with equality span the entire space. Let B be the equality rows of A such that $Bx = b_B$, where b_B are the entries of b corresponding to the rows of B. Note that B depends on x. x is a basic feasible solution if $\texttt{rank}(B) = n$. The following fact is a cornerstone in the theory of linear optimization.

Fact 1. *There is an optimal solution to every LP which is a basic feasible solution.*

Let x be a basic feasible solution, and let $\texttt{supp}(x) := \{i : x_i > 0\}$. Let B be the basis such that $Bx = b_B$. Note that B contains at most $n - \texttt{supp}(x)$ non-negativity constraints. Therefore, at least $\texttt{supp}(x)$ linearly independent non-trivial constraints which are satisfied with equality. This fact will be used crucially a few classes later.

Fact 2. *There is an optimal solution to every LP with at least* $\texttt{supp}(x)$ *linearly independent non-trivial constraints satisfied with equality.*

A linear program can be solved in polynomial time, polynomial in the number of variables and constraints, by using the *ellipsoid algorithm*. Actually more is true. Suppose there is a polynomial (in n) time algorithm which given a candidate solution x checks if whether $Ax \geq b$, and if not returns a violated inequality $a_i \cdot x < b_i$. Such an algorithm is called the *separation oracle*. If a separation oracle exists, then LPs can be solved in time polynomial in only the number of variables. This can possibly be (and will be) used to solve LPs with exponentially (in n) many constraints. By the way, note that if m is polynomial in n, then clearly there is a polytime separation oracle.

Let's talk a bit more about how the ellipsoid algorithm works. Actually, the ellipsoid method checks feasibility of a system of inequalities in polynomial time. This is equivalent to solving the LP: one guesses the optimum M of the LP and adds the constraint $c \cdot x \leq M$ to the system $Ax \geq b$. Suppose that the system $Ax \geq b$ is bounded; if not, it can be made bounded by intersecting with a fairly large cube. In each iteration of the ellipsoid algorithm, a candidate solution x_t is generated. (It is the centre of a carefully constructed ellipsoid containing $\{x : Ax \geq b\}$ - hence the name of the algorithm.) Now, the separation oracle either says that x_t is feasible in which case we are done, or else it returns a constraint $a_t \cdot x \geq b_t$ (or $c \cdot x \leq M$) which is violated by x_t. The nontrivial theorem regarding the ellipsoid algorithm is that in polynomial (in n) many iterations, we either get a feasible solution or a proof that the system is infeasible. If the system is infeasible, then our guess M was too low and we raise it; if the system is feasible then we lower our guess and run again. By doing a binary search, we can reach the true optimum M in polynomial time.

Consider the run of the ellipsoid algorithm when the guess was M^-, "just below" M. That is, the LP cannot take any value between M and M^-. Let $(\hat{A}, \hat{b})$ be the collection of polynomially many constraints/rhs'es returned by the separation oracle in the various iterations, after which the algorithm guarantees infeasibility. Consider the system $\hat{A}x \geq \hat{b}$; we claim that $\min\{c \cdot x : Ax \geq b\} = \min\{c \cdot x : \hat{A}x \geq \hat{b}\}$. Surely, the second is lower since we have removed constraints. However, the ellipsoid algorithm certifies infeasibility of $\{c \cdot x \leq M^-,\ \hat{A}x \geq \hat{b}\}$. So, $\min\{c \cdot x : \hat{A}x \geq \hat{b}\} = M$ as well. Thus given a cost vector c, the exponentially many constraints of the LP can be "essentially" reduced to the polynomial sized LP.

We end our discussion with one final point. We claimed that there exists an optimum solution which is basic feasible. However, the ellipsoid algorithm is not guaranteed to return a basic feasible solution. Can we get a optimal basic feasible solution in polynomial time? More generally, given a point $x \in \mathcal{P} := \{x : Ax \geq b\}$ which is guaranteed to minimize $c \cdot x$ over all points of $\mathcal{P}$, can we move to a basic feasible solution of $\mathcal{P}$ of the same value. (We have reverted to the old notation where A includes the non-negativity constraints as well). Consider the equality submatrix $Bx = b_B$. If B is full rank, we are at a basic feasible solution. If not, let v be a vector in the null space of B; that is $a_i \cdot v = 0$ for all $a_i \in B$. Such a vector can be found by solving a system of linear equations, for instance. Suppose $c \cdot v \geq 0$. Consider $x' = x - \alpha v$ where α is a positive scalar such that for all $i \notin B$, $a_i \cdot (x - \alpha v) \geq b_i$. Since $\mathcal{P}$ is bounded, we have that $a_i \cdot v > 0$ for some i. Choose $\alpha := \min \frac{a_i \cdot x - b}{a_i \cdot v}$ over the i's $\notin B$ with $a_i \cdot v > 0$. What can we say about x'? It is feasible. $c \cdot x' \leq c \cdot x$ (implying $c \cdot v = 0$). Since $a_i \cdot v > 0$ and v is in the null space of B, a_i and B form a linearly independent system. So, we move to a solution with an extra linearly independent constraint in the equality matrix. Repeat this till one gets a basic feasible solution.

Fact 3. *An optimum basic feasible solution to a LP can be found in polynomial time.*

Remark 1. *A word of caution. Whatever written above about the ellipsoid method should be taken with a pinch of salt (although the above fact is true). After all, I have not described how the "carefully constructed" ellipsoids are generated. To make all this work in polynomial time is more non-trivial and we refer the reader to the book "Geometric Algorithms and Combinatorial Optimization" by Grötschel, Lovász, and Schrijver.*

Let's get back to approximation algorithms.

2 Facility Location

LP Relaxation

$$\begin{aligned} \min \quad & \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in C} c(i,j) x_{ij} && (x_{ij}, y_i \geq 0) && (5) \\ \text{subject to} \quad & \sum_{i \in F} x_{ij} \geq 1 && \forall j \in C && (6) \\ & y_i \geq x_{ij} && \forall i \in F,\ \forall j \in C && (7) \end{aligned}$$

Let (x, y) be a fractional solution to the above LP. We define some notation. Let $F_{LP} := \sum_{i \in F} f_i y_i$. Let $C_j := \sum_{i \in F} c(i,j) x_{ij}$. Let $C_{LP} = \sum_{j \in C} C_j$. We now show an algorithm which returns a solution of cost at most $4(F_{LP} + C_{LP}) \leq 4\texttt{opt}$. The algorithm proceeds in two stages. The first stage is called *filtering* which will "take care" of the x_{ij}'s, the second stage is called *clustering* which will "take care" of the y_i's.

Filtering. Given a client j, order the facilities in increasing order of $c(i,j)$. That is, $c(1,j) \leq c(2,j) \leq \cdots \leq c(n,j)$. The fractional cost of connecting j to the facilities is C_j; our goal is to pay not much more than this cost (say within a factor $\rho > 1$ which we will figure out what it should be later). So, we look at the set of facilities $N_j(\rho) := \{i : c(i,j) \leq \rho \cdot C_j\}$. Now it is possible that $x_{ij} > 0$ for some $i \notin N_j(\rho)$. However, we can change the solution so that j is fractionally connected only to facilities in $N_j(\rho)$. This is called filtering the fractional solution.

Define $\overline{x}_{ij}$ as follows. $\overline{x}_{ij} = \frac{\rho}{\rho - 1} \cdot x_{ij}$ for $i \in N_j(\rho)$, $\overline{x}_{ij} = 0$ for $i \notin N_j(\rho)$.

Claim 1. *$\overline{x}$ satisfies* (6)*.*

Proof. Let us consider the sum $C_j = \sum_i c(i,j) x_{ij} = \sum_{i \in N_j(\rho)} c(i,j) x_{ij} + \sum_{i \notin N_j(\rho)} c(i,j) x_{ij}$. Since $c(i,j) > \rho \cdot C_j$ for all $i \notin N_j(\rho)$, we get $\sum_{i \notin N_j(\rho)} x_{ij} < \frac{1}{\rho}$, for otherwise the second term in the RHS above exceeds C_j. This implies $\sum_{i \in N_j(\rho)} x_{ij} \geq (1 - \frac{1}{\rho})$ implying $\sum_{i \in F} \overline{x}_{ij} \geq 1$. □

Suppose we could guarantee that a client j is always connected to a facility i with $\overline{x}_{ij} > 0$, then we could argue that the cost paid to connect this pair is at most $\rho \cdot C_j$. However, we have to argue about the facility opening costs as well. This is done using a clustering idea.

Clustering. Suppose we could find sets of disjoint facilities $F_1, F_2, \ldots$ such that in each F_k, we have $\sum_{i \in F_k} y_i \geq 1$, and we only open the minimum cost facility i_k^* in each F_k. Then, our total facility opening cost would be $\sum_k f_{i_k^*} \leq \sum_k \sum_{i \in F_k} f_i y_i \leq F_{LP}$. In fact, if the 1 is replaced by α, then our facility opening cost is at most F_{LP}/α. We now show how to obtain such a clustering.

The clustering algorithm proceeds in rounds. In round ℓ, the client with minimum C_j in C is picked. The facility F_ℓ is defined as the set of facilities $\{i : \overline{x}_{ij_\ell} > 0\}$. All clients j such that there exists an $i \in F_\ell$ with $\overline{x}_{ij} > 0$ are deleted from C. Let all these clients be denoted by the set $N^2(j_\ell)$. The procedure continues till C becomes empty. Note that F_k's are disjoint.

The algorithm opens the cheapest facility $i_\ell \in F_\ell$ and connects all clients $j \in N^2(j_\ell)$ to i_ℓ. Let's argue about the facility opening cost. Note that for each F_ℓ, we have $\sum_{i \in F_\ell} y_i \geq \sum_{i \in F_\ell} x(i, j_\ell) \geq \frac{\rho - 1}{\rho} \sum_{i \in F_\ell} \overline{x}_{ij_\ell} \geq \frac{\rho - 1}{\rho}$ by Claim 1. Thus, the total facility opening cost is at most $\frac{\rho}{\rho - 1} \cdot F_{LP}$.

Consider a client j. Either it is a j_ℓ, in which case the assignment cost is at most $\rho \cdot C_j$. Otherwise, it is connected to $i_\ell \in F_\ell$ for some ℓ. However, there is some i (not necessarily i_ℓ) in F_ℓ such that $x_{ij} > 0$. Now, $c(i_\ell, j) \leq c(i,j) + c(i, j_\ell) + c(i_\ell, j_\ell) \leq \rho \cdot C_j + 2\rho \cdot C_{j_\ell} \leq 3\rho \cdot C_j$ since $C_j \geq C_{j_\ell}$. The latter's because the j's are considered in increasing order of C_j.

So, $\texttt{alg} \leq \frac{\rho}{\rho - 1} F_{LP} + 3\rho C_{LP}$. If $\rho = 4/3$, then the above gives $\texttt{alg} \leq 4(F_{LP} + C_{LP})$.

3 Generalized Assignment Problem

In GAP, we are given m items J, and n bins I. Each bin i can take a maximum load of B_i. Each item j weighs w_{ij} in bin i, and gives profit p_{ij} when put in it. The goal is to find a max-profit feasible allocation.

LP Relaxation

$$\begin{aligned}
\max \quad & \sum_{i \in I, j \in J} p_{ij} x_{ij} && (x_{ij} \geq 0) && (8)\\
\text{subject to} \quad & \sum_{j \in J} w_{ij} x_{ij} \ \leq \ B_i && \forall i \in I && (9)\\
& \sum_{i \in I} x_{ij} \ \leq \ 1 && \forall j \in J && (10)
\end{aligned}$$

Solve the LP to get a fractional solution x. Suppose all the weights w_{ij} were B_i, say. Then the first constraint is equivalent to $\sum_{j \in J} x_{ij} \leq 1$. Consider the bipartite graph (I, J, E) where we have an edge (i,j) if $x_{ij} > 0$. Then, the solution x above is a maximum weight *fractional matching* in this bipartite graph. This implies that there is an *integral matching* of the same weight, and thus there is an allocation equalling the LP value (and hence the optimum value). This is a non-trivial combinatorial optimization fact and is key to the approximation algorithm below. Let's come to this fact later.

Of course w_{ij} is not equal to B_i, although we can assume $w_{ij} \leq B_i$ since if not we know x_{ij} has to be 0 for such a pair. (Note this must be put into the above LP explicitly, that is, we must set $x_{ij} = 0$ for such pairs as constraints.) Thus, in general, $\sum_{j \in J} x_{ij} \geq 1$. Let $n_i := \lceil \sum_{j \in J} x_{ij} \rceil$. The algorithm proceeds as follows: it uses x to define a fractional matching on a different bipartite graph, use it to get an integral matching of equal weight in it, and then do a final step of pruning.

New Bipartite Graph. One side of the bipartition is J. The other side is I' which consists of n_i copies of each bin i in I. For each bin $i \in I$, consider the items in J in *increasing* weight order. That is, suppose, $w_{i1} \leq w_{i2} \leq \cdots \leq w_{im}$. Consider the fractions $x_{i1}, x_{i2}, \ldots, x_{im}$ in this order. Let $j_1, j_2, \ldots, j_{n_i - 1}$ be the "boundary" items defined as follows. $x_{i1} + \cdots + x_{j_1} \geq 1$, and $x_{i1} + \cdots + x_{j_1 - 1} < 1$; $x_{i1} + \cdots + x_{j_2} \geq 2$, and $x_{i1} + \cdots + x_{j_1 - 2} < 2$; and so on. Formally, for each $1 \leq \ell \leq n_i - 1$,

$$\sum_{j=1}^{j_\ell} x_{ij} \geq \ell; \quad \sum_{j=1}^{j_\ell - 1} x_{ij} < \ell$$

Recall there are n_i copies of the bin i in I'. The ℓth copy, call it i_ℓ, has an edge to items $j_{\ell-1}$ to j_ℓ (j_0 is the item 1). The n_ith copy has an edge to items $j_{n_i - 1 - 1}$ to m.

New Fractional Matching. x' is so defined such that for every copy i_ℓ, the total fractional weight incident on it is at most 1 (in fact, it'll be exactly 1 for all copies but the n_ith copy). The total fractional weight incident on item j is the same as that induced by x. It's clear how to do it given the way edges are defined above; formally it's the mess given below.

$$\begin{aligned} x'_{i_\ell,j} &= x_{ij}, && \text{for } j_{\ell-1} < j < j_\ell \\ x'_{i_\ell,j_{\ell-1}} &= x_{i,j_{\ell-1}} - x'_{i_{\ell-1},j_{\ell-1}} && \text{(if } \ell = 1\text{, then the second term is 0).} \\ x'_{i_\ell,j_\ell} &= 1 - \sum_{j=j_{\ell-1}}^{j_\ell-1} x'_{i_\ell,j} \end{aligned}$$

Integral Matching and Pruning. Note that x' defines a fractional matching in (I', J, E'). Thus, there is an allocation of items to I' whose profit is at least the LP profit (and thus at least $\texttt{opt}$). The assignment to I' implies an assignment to I in the natural way: bin i gets all items allocated to the n_i copies in I'. Is this feasible? Not necessarily. But, the total weight allocated to bin i is not too much larger than B_i.

Claim 2. *The total weight of items allocated by the integral matching above is at most $B_i + \Delta_i$, where $\Delta_i := \max_{j \in J} w_{ij}$.*

Proof. We use the fact that the items (for bin i) were ordered in increasing order of weights. Let J_ℓ be the set of items from $j_{\ell-1}$ to j_ℓ, and let J_{n_i} be the items from j_ℓ to m. Since x is a feasible solution to the LP, we get

$$B_i \geq \sum_{j \in J} w_{ij} x_{ij} = \sum_{\ell=1}^{n_i} \sum_{j \in J_\ell} w_{ij} x'_{i_\ell,j} \geq \sum_{\ell=1}^{n_i} \left(w_{i,j_{\ell-1}} \cdot \sum_{j \in J_\ell} x'_{i_\ell,j} \right) \geq w_{i,j_1} + \cdots + w_{i,j_{n_i-1}}$$

The second inequality uses the increasing order of weights, the last uses the fact that x' forms a fractional matching.

Now, bin i gets at most one item from each J_ℓ (since each copy i_ℓ gets at most item from it's neighbors in J_ℓ.) Note that the heaviest item in J_ℓ weighs w_{i,j_ℓ}, and $\Delta_i = w_{i,n_i}$. Thus, the load on bin i is at most $w_{i,j_1} + \cdots + w_{i,n_i-1} + w_{i,n_i}$ which is at most $B_i + \Delta_i$. □

To summarize, we have found an allocation whose profit is at least $\texttt{opt}$ and each bin i has load at most $B_i + \Delta_i$. For each bin i now consider the heaviest item j allocated to it. We know that $w_{ij} \leq B_i$. We also know weight of all the *other* items allocated to it is $\leq B_i$. To make the allocation feasible, keep either j or the rest, whichever gives more profit. In this pruned allocation, each bin thus gets profit at least half of what it got in the earlier infeasible allocation. Thus, the profit of this new feasible allocation is at least $\texttt{opt}/2$. Thus, the above algorithm is a 1/2-approximation.

3.1 Fractional Matching to Integral Matching

Given a fractional matching x, construct the fractional bipartite graph $G = (I, J, E_f)$ where there is an edge between $i \in I$ and $j \in J$ with $0 < x_{ij} < 1$. We now describe a procedure which takes x and converts it to x' such that two things occur: a) the number of edges in the corresponding E_f

is strictly less, and b) $\sum_{i,j} w_{ij}x_{ij} \leq \sum_{i,j} w_{ij}x'_{ij}$. The fact follows.

Procedure ROTATE.

1. Pick a path or cycle in $G = (I, J, E_f)$. Call the edges picked F. Decompose F into two matchings M_1 and M_2.
2. Let
$$\varepsilon_1 := \min\left(\min_{(i,j)\in M_1} x_{ij}, \min_{(i,j)\in M_2}(1 - x_{ij})\right)$$
$$\varepsilon_2 := \min\left(\min_{(i,j)\in M_2} x_{ij}, \min_{(i,j)\in M_1}(1 - x_{ij})\right)$$
3. Define $x^{(1)}$ as follows. For each $(i,j) \in M_1$, $x^{(1)}_{ij} = x_{ij} - \varepsilon_1$, for each $(i,j) \in M_2$, $x^{(1)}_{ij} = x_{ij} + \varepsilon_1$, for all other edges $x^{(1)}_{ij} = x_{ij}$. Define $x^{(2)}$ similarly.
4. Let x' be the $x^{(1)}$ or $x^{(2)}$ with larger $\sum_{i,j} w_{ij}x'_{ij}$.

It is clear that the number of edges in E_f decreases in this procedure. Also note that the sum $\sum_{i,j} w_{ij}x'_{ij}$ is at least $\sum_{i,j} w_{ij}x_{ij}$. This is because the "increase" in weight in $x^{(1)}$ over x is precisely $\varepsilon_1\left(\sum_{(i,j)\in M_2} w_{ij} - \sum_{(i,j)\in M_1} w_{ij}\right)$, and that in $x^{(2)}$ is $\varepsilon_2\left(\sum_{(i,j)\in M_1} w_{ij} - \sum_{(i,j)\in M_2} w_{ij}\right)$. One of them is at least 0. The following claims ends the analysis of the procedure.

Claim 3. *x' is a feasible fractional matching.*

Proof. If F forms a cycle, then it's clear that the fractional "load" on any vertex remains unchanged. If F forms a path, then we need to only concern about end vertices. Let i be such a vertex. Note that there are no edges (i, j') with $x_{ij'} = 1$ incident on it, and exactly one edge $(i,j) \in E_f$ incident on it. In the end, $x'_{ij} \leq 1$. □