Linear Programming Relaxations and Integrality Gaps[1]

- In this lecture, we look at one of the most powerful techniques in the design of approximation algorithms : the idea of using solvable *continuous relaxations* to the problem one is studying, and then *rounding* the solution to the continuous relaxation to obtain an approximation algorithm.
- Let us illustrate this idea using the vertex cover problem. In this problem, we are given an undirected graph $G = (V, E)$. Every vertex has a non-negative cost $c(v)$. The objective is to select a subset $C \subseteq V$ such that every edge (u, v) has at least one endpoint in C, and the cost of C is as small is possible. This is a special case of the set-cover problem where the edges are elements and the edges incident to any vertex is a subset. We know that the greedy algorithm (which picks the highest degree vertex, deletes it, and repeats till all edges are covered) is a $O(\log n)$-approximation.

  Exercise: ☕ *Show that the approximation factor of the greedy algorithm on vertex cover can be as bad as $\Omega(\log n)$. More precisely, describe a family of graphs where the vertex cover returned by the greedy algorithm is at least $c \cdot \log n \cdot \texttt{opt}$ for some constant $c > 0$.*

  We now describe a 2-approximation algorithm.

- We set the vertex cover problem as a *mathematical program*. Consider a variable $x_v \in \{0, 1\}$ for every $v \in V$ which is supposed to capture the semantic of whether we pick vertex v. More precisely, if $x_v = 1$ then v is in the cover, and if $x_v = 0$ then v is not in the cover. In that case, the cost of the vertex cover is the *linear objective function* $\sum_{v \in V} c(v) x_v$. This implies the following program

$$\text{minimize} \quad \sum_{v \in V} c(v) x_v \quad : \quad x_v \in \{0, 1\}, \ \ \forall v \in V$$

  The above program, of course, is silly : the answer is 0 by setting *all* $x_v = 0$. If we apply the above semantic "backwards", we see that $x_v = 0$ for all $v \in V$ corresponds to the "cover" $C = \emptyset$. But this is not a cover at all. Indeed, we need to now assert certain *constraints* on the x_v's to encode the fact that the "$x_v = 1$ vertices must form a cover". A first cut, and this often works, is to just encode the definition : we need every edge to have an endpoint picked, and so we should have

$$\forall (u, v) \in E \quad : \quad x_u + x_v \geq 1$$

  And therefore one gets a (possibly) better program of

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} c(v) x_v & & \text{(Vertex Cover IP)} \\ & x_u + x_v \geq 1, & \forall (u, v) \in E & \\ & x_v \in \{0, 1\}, & \forall v \in V & \end{aligned}$$

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 23rd Jan, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Theorem 1. For any graph $G = (V, E)$ and any costs $c(v)$ on vertices, the value of (Vertex Cover IP) equals opt, the cost of the minimum vertex cover.

Proof. Given the optimum solution x_v^* to the above program, the set $C := \{v : x_v^* = 1\}$ is a valid cover of cost $IP := \sum_v c(v)x_v^*$ and thus $\texttt{opt} \leq IP$. In the other direction, given the optimal vertex cover C^*, taking the solution $x_v = 1$ for all $v \in C^*$ and $x_v = 0$ for $v \notin C^*$ gives a valid solution to (Vertex Cover IP) of cost opt. Therefore, $IP \geq \texttt{opt}$. □

- *Math Programs.* What we have done is we have converted the combinatorial vertex cover problem into a *mathematical program*. A math program consists of *variables* $\mathbf{x} = (x_1, \ldots, x_n)$ and an *objective function* $f(\mathbf{x})$ which is to be minimized subject to certain number *constraints* of the form $g_j(\mathbf{x}) \geq 0$. The number of variables n is called the *dimension* of the problem.

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) && \text{(Very General Math Program)} \\ & g_j(\mathbf{x}) \geq 0, \qquad \forall j = 1, \ldots, m \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

  To be a bit more clear at the risk of being pedantic, let us go over the vertex cover problem again. Here, the dimension is $|V|$, and $\mathbf{x} = (x_v \ : \ v \in V)$. The objective function is $f(\mathbf{x}) := \sum_{v \in V} c(v)x_v$. There are *two* kinds of constraints. The first kind has $|E|$ constraints, one for every edge. And they look like $g_{(u,v)}(\mathbf{x}) := x_u + x_v - 1$. The second kind of constraints is more interesting. These have to capture the fact that $x_v \in \{0, 1\}$. One way to capture this is to include $x_v^2 - x_v = 0$, which itself is two "inequality" constraints. So, for every vertex v we have two constraints with the functions $g_v^+(\mathbf{x}) := x_v^2 - x_v$, and the other $g_v^-(\mathbf{x}) := x_v - x_v^2$.

- *Linear Programs (LP).*It should be clear to the reader by now that solving the general math program is a very hard problem. Indeed, it generalizes vertex cover, and is thus NP-hard. However, one of the most powerful algorithmic tools we have is that a large class of math programs are solvable efficiently. In particular, when the functions f and the g_j's are *linear* functions of $\mathbf{x}$, the math program is polynomial time solvable. Such programs are called linear programs. Recall that since a linear function $f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x}$ for some n-dimensional vector $\mathbf{c}$, a linear program is often written as follows.

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\top \mathbf{x} = \sum_{i=1}^{n} c_i x_i && \text{(Linear Program)} \\ & A\mathbf{x} \geq \mathbf{b}, \qquad A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m \\ & \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

- ***Linear Programming (LP) Relaxations.*** Going back to the vertex cover problem, we see that the math program's objective was linear and one kind of constraints was linear. The problematic constraints were the "quadratic" constraints which encoded the $x_v \in \{0, 1\}$ constraints. The idea of *relaxations* is to replace these "hard" constraints by linear ones which capture some essence of this constraint. This step is often called "linearizing" the hard constraint. In this case, one just replaces these by

adding $0 \le x_v \le 1$. This gives us the following linear program. Since we have *relaxed* the constraint $x_v \in \{0, 1\}$, the value of the linear program is ***at most*** the value of the integer program which is $\mathtt{opt}$ and thus provides a *lower bound* to the optimum solution.

$$\begin{aligned} \mathsf{lp} := \text{minimize} \quad & \sum_{v \in V} c(v) x_v && \text{(Vertex Cover LP)} \\ & x_u + x_v \ge 1, \qquad \forall (u, v) \in E && (1) \\ & 0 \le x_v \le 1, \qquad \forall v \in V \end{aligned}$$

Theorem 2. The value of (Vertex Cover LP) is a polynomial time computable ***lower bound*** on $\mathtt{opt}$.

Once again, the constraint matrix A has $|E| + 2|V|$ rows. The first $|E|$ rows correspond to edges, where the (u, v)th row has a 1 in the u and v entries and 0 everywhere else, the second $|V|$ rows are an identity matrix, and the next $|V|$ rows are the negative identity matrix. The $\mathbf{b}$ vector has 1's in the first $|E|$ entries, 0 in the next $|V|$ and -1 in the last $|V|$ entries. For the first time reader, it is instructive to write down these matrices for small graphs just to get used to them.

- ***Rounding.*** We have now seen how to get a lower bound on $\mathtt{opt}$ for the vertex cover problem via an LP relaxation. The next step is to *use* the solution to obtain an actual vertex cover, and *analyze* it by comparing the value of the algorithm's cover to the value of the LP relaxation. This process of moving from the "fractions on every vertex" to a true vertex cover is called rounding. For vertex cover, the rounding is really easy.

```
1: procedure VERTEX COVER ROUNDING(G = (V, E); costs c(v) on vertices):
2:     Solve (Vertex Cover LP) to obtain x_v ∈ [0, 1] for all v ∈ V.
3:     C ← {v ∈ V : x_v ≥ 0.5}.
4:     return C.
```

Theorem 3. VERTEX COVER ROUNDING is a 2-approximation algorithm.

Proof. Let lp be the solution to (Vertex Cover LP). Let C be the solution returned by the algorithm. We show that (a) C is a vertex cover, and (b) $\sum_{v \in C} c(v) \le 2\mathsf{lp}$, which would prove the theorem since $\mathsf{lp} \le \mathtt{opt}$. (b) follows since $v \in C$ iff $x_v \ge 0.5$, and thus $\sum_{v \in C} c(v) \le \sum_{v \in C} c(v) \cdot (2x_v) \le 2\mathsf{lp}$. (a) follows since for any edge (u, v), $x_u + x_v \ge 1$, and thus at least one of x_u or x_v must be ≥ 0.5. That is, at least one of them is in C implying C is a vertex cover. □

- ***Integrality Gaps.*** We have thus shown that for the vertex cover problem $\mathsf{lp} \le \mathtt{opt} \le 2\mathsf{lp}$. The LP-relaxation gives a lower bound which is not too bad, it is at most 2 times the optimum. Can this be improved? In a sense, this question is asking what the quality of the LP-relaxation is, or how "tight" the relaxation is. This concept is called the *integrality gap* of the LP-relaxation.

$$\mathsf{IG} := \sup_{G:\ \text{graph}} \frac{\mathtt{opt}(G)}{\mathsf{lp}(G)} \tag{2}$$

Theorem 3 shows that IG of (Vertex Cover LP) is at most 2. The following example shows that $\mathsf{IG} \to 2$. More precisely, if we define IG_n to be the above ratio when the supremum is taken over graphs with n vertices, the example below shows $\mathsf{IG}_n \geq 2 - \frac{2}{n}$.

Example 1. *Consider the clique K_n on n vertices. Recall, a clique is a graph with all edges. Note that $\mathtt{opt}(K_n) = n - 1$; if we miss two vertices then we miss covering the edge joining them. On the other hand $\mathsf{lp}(K_n) \leq \frac{n}{2}$; to see this note that $x_v = \frac{1}{2}$ for all $v \in V$ is a feasible solution to (Vertex Cover LP). Thus, $\mathsf{IG}_n \geq \frac{n-1}{n/2} = 2 - \frac{2}{n}$.*

- ***Strengthening the Relaxation.*** For the LP relaxation (Vertex Cover LP), the above discussion shows that the integrality gap $\to 2$ as $n \to \infty$. Therefore, there cannot exist an algorithm which returns a solution S with $\mathtt{cost}(S) \leq 1.99 \cdot \mathsf{lp}$. In particular, (Vertex Cover LP) "cheats" up to factor 2. At this point, one should ask : can we stop this cheating?

  What does this question mean? Well, let's go back to Example 1. We could argue that $\mathtt{opt} = n - 1$, and yet (Vertex Cover LP) gets a much smaller value by putting $\frac{1}{2}$ at all points. We know that no $\{0, 1\}$-solution could get such a low value. This means that the constraints (1) are too "slack" which allow this all $1/2$-point as a solution. To prevent this, one can *strengthen* the LP-relaxation by adding what is called a ***valid inequality***.

  Definition 1 (Valid Inequality)**.** *For an LP relaxation of an integer program, a valid inequality is one of the form $\mathbf{a}^\top \mathbf{x} \geq b$ such that all* integer valued $\mathbf{x}$*'s satisfy this inequality.*

  So what we want is to find a valid inequality which *every* $\{0, 1\}$-solution satisfies, but the "cheating solution", in this case the all $1/2$ solution, doesn't. We can then add this valid inequality to our LP-relaxation and re-solve. We would then be guaranteed that *this* cheating solution won't arise again. However, a new one may arise, in which case we rinse and repeat. Till, hopefully, we get something better. Note that this idea can also be applied instance-by-instance; for the particular problem instance you are solving it hand, you can look at the LP solution, assert "ah, this is a valid inequality that I can add to prevent this", add it and repeat. This method, also known as the method of "cutting planes" is a (very high level) description of a generic technique used in solving integer programs in practice.

  Theoretically speaking, we would like to add a valid inequality, or perhaps a collection of valid inequalities, such that upon doing so, we can assert the integrality gap goes down. That is, for *every instance*, we get a better rounding algorithm. What could such a collection be for vertex cover? How would we find one? One way is to look at the integrality gap example at hand, in this case Example 1, and ask oneself which valid inequality can prevent this cheating? Turns out, we can add the following:

$$\forall \{u, v, w\} \text{ such that } (u, v), (v, w), (w, u) \in E, \quad x_u + x_v + x_w \geq 2 \tag{3}$$

  In plain English, (3) states that in any *triangle* in the graph, every valid vertex cover, or put differently and $\{0, 1\}$-solution to (Vertex Cover LP), must pick at least 2 vertices. So, we can add all these potentially $\binom{n}{3}$ constraints to (Vertex Cover LP). What is the integrality gap of this new LP relaxation? Well, it is still ≤ 2 since upon adding the constraints the new LP value, call it lp', only becomes bigger. And, in Example 1, the all $1/2$-solution is no longer feasible. Indeed, a little thought would show that on the clique K_n, the all $2/3$-solution is the best, and so $\mathsf{lp}' = \frac{2n}{3}$. Thus, the integrality gap of this new LP is ≥ 1.5. Can it be that it is $= 1.5$? Can one find a better rounding algorithm? Or even analyze this existing rounding algorithm itself?

  Turns out, the answer is no. But the example is not that trivial to find.

Exercise: ☕☕☕ *Prove that the integrality gap of* (Vertex Cover LP) *with* (3) *applied, is also* ≥ 1.99 *(indeed any* $2 - \varepsilon$*). In particular, find a graph with* **no triangles** *with "large" vertex cover. If there are no triangles, then* (3) *is useless.*

Notes

The idea of strengthening linear programming relaxations have been studied extensively in the past two decades. Indeed, there are *systematic* ways of taking a linear programming relaxation of any problem and add valid inequalities in rounds in such a way that in n rounds, where n is the dimension of the problem, the LP relaxation obtained has no integrality gap! The flip side is that in the tth round one may add $n^{O(t)}$ inequalities. So unless t is a constant, this would not immediately lead to polynomial time algorithms. Such systematic ways are often called *lift-and-project hierarchies*; the two most well studied linear programming techniques are the Sherali-Adams [6] and the Lovasz-Schrijver [5] hierarchies.

The study of proving integrality gaps for these hierarchies was initiated in the seminal paper [1] of Arora, Bollobas, and Lovasz (with a later journal version [2] including Tourlakis). This paper proved that the Lovasz-Schrijver hierarchy wouldn't bring down the integrality gap of vertex cover to 1.99 even after $O(\log n)$ rounds. A similar result for the Sherali-Adams hierarchy was proved in the paper [4] by Charikar, Makarychev, and Makarychev. More recently, the paper [3] by Bazzi, Fiorini, Pokutta, and Svensson proves that **no** polynomial sized LP relaxation for the vertex cover problem can have integrality gap 1.99. It is a remarkable result, and builds on lots of exciting and deep work on understanding the powers of linear programs, and [3] is as good a place to start ones journey into this wonderful area.

References

[1] S. Arora, B. Bollobás, and L. Lovász. Proving integrality gaps without knowing the linear program. In *Proc., IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 313–322, 2002.

[2] S. Arora, B. Bollobás, L. Lovász, and I. Tourlakis. Proving integrality gaps without knowing the linear program. *Theory of Computing*, 2(2):19–51, 2006.

[3] A. Bazzi, S. Fiorini, S. Pokutta, and O. Svensson. No small linear program approximates vertex cover within a factor 2- . *Math. Oper. Res.*, 44(1):147–172, 2019.

[4] M. Charikar, K. Makarychev, and Y. Makarychev. Integrality gaps for sherali-adams relaxations. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 283–292, 2009.

[5] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization (SIOPT)*, 1(2):166–190, 1991.

[6] H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics (SIDMA)*, 3(3):411–430, 1990.