CS880: Approximation and Online Algorithms	**Scribe:** Jialu Bao
Lecture 6: LP relaxation and rounding	**Date:** Sept.27, 2019

6.1 Finishing the proof of Rent–or–Buy Problem

6.1.1 Multi-commodity Rent-or-Buy problem

Last time we talked the Rent-or-Buy problem, where we are given a graph and a set of demands from the source to terminal nodes. We want to satisfy the demands by connecting terminals to the source (think about sending flows), and we need to either "buy" or "rent" edges used in these paths. Every edge has an associated cost c_e: when renting the edge, we need to pay c_e more for using the edge on paths connecting each new source–terminal pair; when buying the edge, we pay $M \cdot c_e$ for using the edge on whichever number of source–terminal pairs. The constant M is the same for all edges. If we have decided on paths for connecting all source–terminal pairs, then the number of paths using each edge is fixed, and we would rent or buy by choosing the lower price. However, there could be many ways to wire the paths between source-terminal pairs, so we want to find out the cheapest wiring and buy or rent accordingly.

Formally, given graph $G = (V, E)$, cost c_e on each edge e, source s, and a set of terminals T, we want to find paths P_t from s to $t \in T$ that minimize

$$\sum_{e \in \cup_t P_t} \min(M \cdot c_e, |\{t : e \in P_t\}| \cdot c_e)$$

Rent–or–Buy problem is a special case of Buy–at–Bulk network design, where the cost of using a edge could be an arbitrary concave function with respect to its load. In Rent–or–Buy, the concave function is linear in at first and becomes constant after one point. Figure 6.1.1. shows an example cost function.

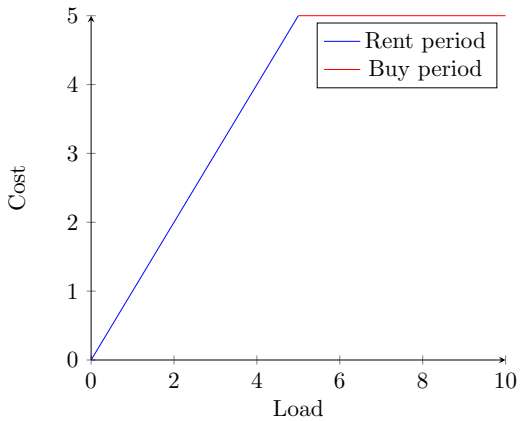

Figure 6.1.1: Ex. Cost function when $c_e = 1, M = 5$

6.1.2 Algorithm Recap

Algorithm 1 Single-source rent-or-buy

Given: Graph $G = (V, E)$, source s, terminal T, costs c_e for each edge $e \in E$

$B \leftarrow \{\}$
for $t \in T$ **do**
    Select t with probability $\frac{1}{M}$ and place it in the buy set B.
Construct a 2–approximate Steiner Tree F over $\{S\} \cup B$. Buy edges in F.
Connect every $t \in T \setminus B$ to F via shortest paths. Rent these edges.

6.1.3 Analysis

Let OPT denote the optimal solution's cost, and $c(F) := \sum_{e \in F} c_e$. To show that the algorithm gives us a constant factor approximation, it is sufficient to prove the following lemmas. See lecture 5 for the proof of **Lemma 6.1.1** and how these two lemmas lead to a 4-approximation.

Lemma 6.1.1

$$\mathbf{E}[M \cdot c(F)] \leq 2 \cdot OPT$$

Lemma 6.1.2

$$\mathbf{E}[\textbf{\textit{Rent cost of}}\ T \setminus B] \leq \mathbf{E}[\textbf{\textit{Buy cost of}}\ F] = \mathbf{E}[M \cdot c(F)]$$

Proof: Let's delay the randomness and not select B until constructing the 2-approximate Steiner Tree. We discussed earlier that Minimum Spanning Tree on terminals after metric completion is a 2-approximate Steiner Tree. So considering constructing running Prim algorithm from the source node: at each step i, Prim algorithm picks the least cost edge $e_i = (u, v)$ going out of the current tree, denoted as T_i. Assuming that $u \in T_i, v \notin T_i$ without the loss of generality, we add v into B with probability $\frac{1}{M}$. Then, let $T_{i+1} = T_i + e_i$, and iterate until the current tree spans $\{S\} \cup T$. Return that tree as $\widehat{Tr}$. Every edge in $\widehat{Tr}$ is either rented or bought.

When the algorithm returns, we have flipped the coin for every terminal $t \in T$ to decide whether to include it into the buy set B, with independent probability $\frac{1}{M}$, and the returned tree is a 2-approximate Steiner Tree over B. So the expected buy cost $\widehat{Tr}$ is exactly the same as the expected buy cost of F given by the original algorithm. The rent cost may be lower in the original algorithm, which find the shortest paths connecting $T \setminus B$ to S over all edges.

Consider the rent cost and buy cost in this algorithm. At every step i,

$$\mathbf{E}[\textbf{Increase in buy cost of}\ T_i] = \frac{1}{M} \cdot M \cdot c_e = c_e$$
$$\mathbf{E}[\textbf{Increase in rent cost of}\ T_i] = (1 - \frac{1}{M}) \cdot c_e \leq c_e$$

Thus, after all these steps, $\mathbf{E}[\textbf{Buy cost of}\ T_i] \leq \mathbf{E}[\textbf{Rent cost of}\ T_i]$. Thus,

$$\mathbf{E}[\textbf{Buy cost of}\ B] = \mathbf{E}[\textbf{Buy cost of}\ T_i] \leq \mathbf{E}[\textbf{Rent cost of}\ T_i] \leq \mathbf{E}[\textbf{Rent cost of}\ T \setminus B]$$

■

See [Gupta et al., 2003, Kumar et al., 2002] for more.

6.2 Linear Programming

A linear programming problem has a set of variables, a set of linear constraints, and a linear objective. For example,

$$\begin{aligned} \textbf{minimize } & x + 3y && \textit{Objective} \\ \textbf{s.t. } & y - 2x \geq 5 && \textit{Constraint} \\ & x, y \geq 0 && \textit{Constraint} \end{aligned}$$

Every linear programming problem can be expressed in the following canonical form

$$\begin{aligned} &\textbf{Maximize } \mathbf{c}^T\mathbf{x} \\ &\textbf{Subject to } A\mathbf{x} \leq \mathbf{b} \\ &\textbf{and } \mathbf{x} \geq 0 \end{aligned}$$

Feasible solutions of a linear program form a polyhedron, and the objective's solution lies on extreme points of the polyhedron. The linear programming problem can encode many other problems, and there are polynomial time algorithms finding its solutions (or determine that no $\mathbf{x}$ satisfies all linear constraints).

6.3 Integer Programming and LP relaxation

An integer (linear) programming (ILP) problem is a linear programming problem (LP) with additional constraints on some or all variables to be integers. While linear programming is solvable in polynomial time, integer programming is NP–hard. The disparity of the running time between LP and ILP enables approximate ILP's solution efficiently by LP relaxation: first ignore the integer constraint and solve the underlying LP, and then round off the LP solution to integers.

Since LP has relaxed the integrality constraints in the original problem, its feasible solutions are wider than original ILP's feasible solutions. Thus its objective function could at least attain value that is as good as ILP's optimal value (OPT). Therefore, besides providing an approximation, the optimal value of LP relaxation (LP–OPT) also provides a lower bound of OPT. We define the ratio $\frac{OPT}{LP-OPT}$ as the integrality gap. The following diagram illustrate their relationships.

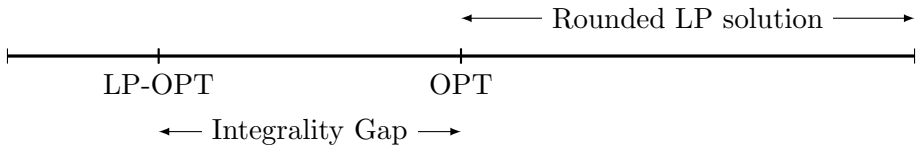

Figure 6.3.2: LP-OPT as a lower bound of OPT

6.4 Linear Relaxation for Vertex Cover

A vertex cover problem asks to find the minimal-sized set S that "cover" all edges on a graph $G = (V, E)$, i.e., for every $(u, v) \in E$, we want either $u \in S$ or $v \in S$. We can solve the vertex cover problem using the following integer programming task as a subroutine :

$$\begin{aligned} \textbf{minimize} & \sum_{v \in V} x_v \\ \textbf{s.t.} \ & x_v \in \{0, 1\} && \forall v \in V \\ & x_u + x_v \geq 1 && \forall (u, v) \in E \end{aligned}$$

Let $S = \{v \ \textbf{s.t.} \ x_v = 1\}$, then S would be a minimal vertex covering set.

6.4.1 LP-relaxation and rounding algorithm

Vertex cover problem is known to be NP-hard, so solving that integer programming problem would be NP-hard too. Let's consider its LP–relaxation:

$$\begin{aligned} \textbf{minimize} & \sum_{v \in V} \widehat{x}_v \\ \textbf{s.t.} \ & \widehat{x}_u + \widehat{x}_v \geq 1 && \forall (u, v) \in E \end{aligned}$$

Then, let $\widehat{S} = \{v \ \textbf{s.t.} \ \widehat{x}_v \geq \frac{1}{2}\}$

Claim 6.4.1 *$\widehat{S}$ is a vertex cover, i.e., for every $(u, v) \in E$, either $u \in \widehat{S}$ or $v \in \widehat{S}$.*

Proof: For every $(u, v) \in E$, the constraints enforce $x_u + x_v \geq 1$, so either $\widehat{x}_u \geq \frac{1}{2}$, or $\widehat{x}_v \geq \frac{1}{2}$, or both at least $\frac{1}{2}$. By construction of $\widehat{S}$, either $\widehat{S} \ni u$, or $\widehat{S} \ni v$, or containing both. ■

Claim 6.4.2 $|\widehat{S}| \leq 2|S|$

Proof: Linear relaxation optimal provides a lower bound of the original OPT, so

$$|S| = \sum_{v \in V} x_v \geq \sum_{v \in V} \widehat{x}_v$$

Meanwhile, since $S \subseteq V$

$$\sum_{v \in V} \widehat{x}_v \geq \sum_{v \in S} \widehat{x}_v \geq \frac{1}{2} |\widehat{S}|$$

It directly follows that $|S| \geq \frac{1}{2}|\widehat{S}|$. ■

Thus, this LP relaxation and rounding algorithm gives a 2-approximation of the minimal vertex cover. This is the best approximation ratio we can achieve using LP-OPT as the lower bound because the integrality gap is also 2. Consider complete graph K_n, the minimal vertex cover is any subset of size $n - 1$, so OPT $= \sum_{v \in V} x_v = n - 1$. The LP–relaxation could find a solution where $x_v = \frac{1}{2}$ for every vertex $v \in V$, so LP-OPT $= \sum_{v \in V} \widehat{x}_v = \frac{n}{2}$. So,

$$\lim_{n \to \infty} \frac{\text{OPT}}{\text{LP-OPT}} = \lim_{n \to \infty} \frac{n - 1}{\frac{n}{2}} = 2$$

In this example, the rounding of the LP-relaxation would then return $\widehat{S} = V$ as the vertex cover. $\widehat{S}$ is indeed very close to the optimal $(n-1)$–sized minimal vertex cover, but using LP-OPT as the proxy lower bound of OPT, our analysis could only conclude that $\widehat{S}$ is no larger than twice size of the minimal vertex cover.

6.5 Linear Relaxation for Set Cover

Set cover is defined as follows: Given a ground set S of n elements, and a collection of subsets, $S_1, ..., S_m$, each subset S_i associated with a cost c_i, we want to find $I \subseteq [m]$ such that $\cup_{i \in I} S_i = S$ and minimize $\sum_{i \in I} c_i$.

It is a more general problem than vertex cover. We can transform a vertex cover problem on $G = (V, E)$ to a set cover problem by devising an element in S for each edge in E, and a set for each node v, and let

$$S_v = \{(u, w) \mid (u, w) \in E \textbf{ and } (u \textbf{ or } w = v)\}, c_v = 1$$

for each $v \in V$. Then the vertices associated with the sets in the minimal set cover is a minimal vertex cover. Set cover is more expressive in that it can associate different costs with different sets.

The integer programming for the set cover is

$$\begin{aligned} \textbf{minimize } & \sum_i x_i c_i & \\ \textbf{s.t. } & \sum_{i:S_i \ni j} x_i \geq 1 & \textbf{for every } j \in S \\ & x_i \in \{0, 1\} & \textbf{for every } i \in [m] \end{aligned}$$

We want to design an approximation algorithm using similar procedure: find a solution of the LP relaxation, round it to integer solutions, and bound the loss in rounding. Its LP relaxation is

$$\begin{aligned} \textbf{minimize } & \sum_i x_i c_i & \\ \textbf{s.t. } & \sum_{i:S_i \ni j} x_i \geq 1 & \textbf{for every } j \in S \\ & x_i \geq 0 & \textbf{for every } i \in [m] \end{aligned}$$

Suppose we find a set of x_i's, it's rather tricky to find a good rounding threshold t that we can round x_i to 1 if $x_i \geq t$ and round x_i to 0 otherwise. Any threshold greater than $\frac{1}{m}$ would potentially allow some set of S_i that are not set cover, but $\frac{1}{m}$ seems to be too small that the analysis could not bound tighter than m-approximation. To address that, we use a new technique, randomized rounding.

LP-OPT is the minimal value $\sum_i x_i c_i$ achieves in the linear programming problem.

Claim 6.5.1

$$\mathbf{E}\left[\sum_{i \in I} c_i\right] = \textit{LP-OPT}$$

Algorithm 2 Randomized Rounding of Set Cover's LP-relaxation

Input: The optimal solution of x_i for every $i \in [m]$ in the LP-relaxation

1: **while** I is not a Set Cover **do**
2: Initialize I as empty
3: **for** $i \in [m]$ **do**
4: Add i to I independently with probability X_i
return I

Proof: Let D_i be a indicator variable that is 1 if $i \in I$ and is 0 otherwise. Then

$$\begin{aligned}\mathbf{E}\left[\sum_{i\in I} c_i\right] &= \mathbf{E}\left[\sum_{i\in[m]} D_i \cdot c_i\right] \\ &= \sum_{i\in[m]} \mathbf{E}[D_i] \cdot c_i \\ &= \sum_{i\in[m]} \mathbf{Pr}[i \in I] \cdot c_i\end{aligned}$$

$\mathbf{Pr}[i \in I] = x_i$ by construction, so

$$\begin{aligned}\mathbf{E}\left[\sum_{i\in I} c_i\right] &= \sum_{i\in[m]} \mathbf{Pr}[i \in I] \cdot c_i \\ &= \sum_{i\in[m]} x_i \cdot c_i = \text{LP-OPT}\end{aligned}$$

■

Claim 6.5.2 *With high probability, the number of iterations needed until I is a set cover is $O(\log n)$*

Proof: First, consider the probability that some element j is left uncovered,

$$\mathbf{Pr}[j \notin \cup_{i\in I} S_i] = \mathbf{Pr}[\forall i \textbf{ s.t. } S_i \ni j, i \notin I] = \prod_{i:S_i\ni j} (1 - x_i)$$

By constraints in the linear programming, $x_i \geq 0$ for any i, and $\sum_{i:S_i\ni j} x_i \geq 1$. Thus, $\prod_{i:S_i\ni j}(1-x_i)$ is attained when all x_i are equal, i.e., $x_i = \frac{1}{k}$, where k is the number of sets S_i containing j. Then,

$$\mathbf{Pr}[j \notin \cup_{i\in I} S_i] = \prod_{i:S_i\ni j} (1 - x_i) \leq (1 - \frac{1}{k})^k \leq \frac{1}{e}$$

Since each iteration is independent, the probability that j is uncovered after t trials is at most $\frac{1}{e^t}$. By union bound, the probability that exists a j, such that j is uncovered after t trials is no more than $n \cdot \frac{1}{e^t}$.

$$\begin{aligned}&\mathbf{Pr}[\exists j \textbf{ that is uncovered after } t \textbf{ trails}] = \mathbf{Pr}[\textbf{number of steps} \geq t] \\ \Longrightarrow\ &\mathbf{Pr}[\textbf{number of steps} \geq 2\log_e n] \leq n \cdot \frac{1}{e^{2\log_e n}} = \frac{1}{n}\end{aligned}$$

Thus, with probability $(1 - \frac{1}{n})$, the algorithm finds a set cover, and the set cover has expected cost same as LP-OPT in $O(\log n)$ time. ■

See [Raghavan and Tompson, 1987, Feige, 1998] for more.

References

[Feige, 1998] Feige, U. (1998). A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652.

[Gupta et al., 2003] Gupta, A., Kumar, A., Pál, M., and Roughgarden, T. (2003). Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 606–615. IEEE.

[Kumar et al., 2002] Kumar, A., Gupta, A., and Roughgarden, T. (2002). A constant-factor approximation algorithm for the multicommodity rent-or-buy problem. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 333–342. IEEE.

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