1 Overview

Approximation algorithms can be designed using two fundamental techniques that are based on linear programming: a) Linear Programming (LP) relaxation and rounding and b) the primal-dual method. Here we will discuss the former, while the latter will be covered in succeeding lectures. The basic idea of LP relaxation and rounding is simple. We first formulate an optimization problem as an Integer Program (IP). Then we relax the integral constraints to transform the IP to an LP. Next, we solve the LP and obtain an optimal solution $x^*_{LP}$. From this solution we will construct a feasible solution $x_{IP}$ to the IP. This construction step is often done by rounding the entries of $x^*_{LP}$ to integral values so that $x_{IP}$ is feasible for the IP. Rounding can be done deterministically or probabilistically. When the latter approach is taken, we have the so-called randomized rounding method. The quality of approximation algorithms based on LP relaxation and rounding is upper bounded by the maximum ratio between the cost of the IP optimal solution $x^*_{IP}$ and the cost of the LP optimal solution $x^*_{LP}$. This quantity $\frac{\text{cost}(x^*_{IP})}{\text{cost}(x^*_{LP})}$ is known as the integrality gap. In this lecture we will see how approximation algorithms based on LP relaxation and rounding are constructed for three problems: Vertex Cover, Set Cover and Load Balancing on Unrelated Machines.

2 Approximation algorithms based on LP Relaxation and Rounding

2.1 Vertex Cover

Given a graph $G = (V,E)$, $|V| = n$, $|E| = m$, we want to find a minimum vertex cover $C \subseteq V$ such that every edge of $G$ has at least one end in $C$.

This problem can be naturally formulated as the following Integer Program:

$$\begin{align*}
\min & \quad \sum_{v \in V} x(v) \\
\text{subject to} & \quad x(v) + x(u) \geq 1 \quad \forall (v,u) \in E \quad . \\
& \quad x(v) \in \{0,1\} \quad \forall v \in V
\end{align*}$$

(1)

(2)

(3)

We can transform the above IP to an LP by relaxing the last constraint (Equation 3), which forces variables to be binary:

$$\begin{align*}
\min & \quad \sum_{v \in V} x(v) \\
\text{subject to} & \quad x(v) + x(u) \geq 1 \quad \forall (v,u) \in E \quad . \\
& \quad x(v) \geq 0 \quad \forall v \in V
\end{align*}$$

(4)

(5)

(6)

Note that we don’t need to upper bound variables $x(v)$ in Equation 6 since this is a minimization LP.

We now discuss a rounding scheme to transform the optimal LP solution $x^*_{LP}$ into an approximate integral
solution $x_{IP}$. We have to keep in mind that any rounding scheme must preserve feasibility, that is, the constructed IP solution must be feasible.

Consider the following thresholding rounding scheme for the vertex cover problem:

$$x_{IP}(v) = \begin{cases} 
1 & \text{if } x^*_{LP}(v) \geq \frac{1}{2} \\
0 & \text{otherwise}
\end{cases}. \quad (7)$$

This rounding scheme produces a feasible solution for the IP because it always satisfies the constraint in Equation (2). The rounding results in a 2-approximation algorithm. You can intuitively think of the worst case scenario when all entries $x^*_{LP}(v) = 0.5$ and the IP optimal solution $x_{IP}^*$ will have the same cost as the LP optimal solution $x^*_{LP}$. In these settings the objective cost$(x_{IP}) = 2\text{cost}(x_{IP}^*)$.

### 2.2 Set Cover

Given a collection of sets $S = \{S_1, \ldots, S_n\}$ each containing elements from a universe $U = \{e_1, \ldots, e_m\}$ and each set having cost $c(s)$, find a minimum cost cover $C = \{S_i \mid i \in S\}$ such that $C$ covers all elements in $U$. We formulate this problem as the following Integer Program:

$$\min \ \sum_{s \in S} c(s)x(s) \quad (8)$$

subject to

$$\sum_{s \in S} x(s) \geq 1 \quad \forall e \in U. \quad (9)$$

$$x(s) \in \{0, 1\} \quad \forall s \in S. \quad (10)$$

Relaxing the last constraint, we get the following LP:

$$\min \ \sum_{s \in S} c(s)x(s) \quad (11)$$

subject to

$$\sum_{s \in S} x(s) \geq 1 \quad \forall e \in U. \quad (12)$$

$$x(s) \geq 1 \quad \forall s \in S. \quad (13)$$

Consider this randomized rounding scheme for the LP:

$$x_{IP}(s) = \begin{cases} 
1 & \text{w.p. } x^*_{LP}(s) \\
0 & \text{otherwise}
\end{cases}. \quad (14)$$

In expectation, this algorithm looks exact because $\mathbb{E}[c(s)x_{IP}(s)] = c(s)x^*_{LP}(s)$. However, some element might not be covered.

$$\mathbb{P}[e \text{ not covered}] = 1 - \mathbb{P}[e \text{ covered}] \quad (15)$$

$$= 1 - \prod_{s \in S} (1 - x^*_{LP}(s)) \quad (16)$$

The constraint that every element must be covered imposes $\sum_{s \in S} x(s) \geq 1$, so we can derive:

$$\prod_{s \in S} (1 - x^*_{LP}(s)) \leq \frac{1}{e}. \quad (17)$$
The derivation is made for the worst case scenario where:

$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e}.$$  \hspace{1cm} (18)

To solve the problem of a constant fraction of elements not being covered by the approximation algorithm, either the algorithm can be repeated $\log n$ times or the rounding probabilities can be boosted by $\log n$:

$$x_{IP}(s) = \begin{cases} 1 & \text{w.p. } x_{IP}^{*}(s) \log n \\ 0 & \text{otherwise} \end{cases}.$$ \hspace{1cm} (19)

The algorithm in turn becomes a Monte-Carlo algorithm with an approximation factor of $\log n$. The algorithm can be also transformed to a Las Vegas algorithm. For each uncovered element simply add to the solution the least cost set covering that element.

### 2.3 Load Balancing on Unrelated Machines

Consider a set of $n$ jobs $J_1, \ldots, J_n$, which are to be processed on $m$ unrelated parallel machines $M_1, \ldots, M_m$. Each job $J_j$ has a processing time $P_{ij} \in \mathbb{Z}^+$ on machine $M_i$. Each machine can only process one job at a time and we assume that any job processing cannot be interrupted. The objective is to devise a schedule so that the total completion time $\lambda$ for all jobs is minimized. The total completion time is often called the makespan.

This scheduling problem can be naturally expressed as an Integer Program:

$$\min \lambda$$ \hspace{1cm} (20)

subject to

$$\sum_{i \in M} x_{ij} \leq \lambda \quad \forall i \in M \hspace{1cm} (21)$$

$$\sum_{i \in M} x_{ij} \geq 1 \quad \forall j \in J \hspace{1cm} (22)$$

$$x_{ij} \in \{0, 1\} \hspace{1cm} (23)$$

We can relax the above IP to an LP as follows:

$$\min \lambda$$ \hspace{1cm} (24)

subject to

$$\sum_{i \in M} x_{ij} \leq \lambda \quad \forall i \in M \hspace{1cm} (25)$$

$$\sum_{i \in M} x_{ij} \geq 1 \quad \forall j \in J \hspace{1cm} (26)$$

$$x_{ij} \geq 0 \hspace{1cm} (27)$$

The optimal solution to this LP can have a large integrality gap. For illustration consider a scenario with ten jobs and ten machines, where one job has a processing time of 10 days while the other jobs have a processing time of 1 second independent of the machine. The fractional solution $x_{IP}^{*}$ has a makespan of about one day when breaking the 10 day job into even chunks, while the optimal integral solution $x_{IP}^{*}$ produces a schedule of 10 days. To improve this large integrality gap we now introduce a stronger feasibility LP.

$$\min \lambda^*$$ \hspace{1cm} (28)

subject to

$$\sum_{i \in M} x_{ij} \leq \lambda^* \quad \forall i \in M \hspace{1cm} (29)$$

$$\sum_{i \in M} x_{ij} \geq 1 \quad \forall j \in J \hspace{1cm} (30)$$

$$x_{ij} \geq 0 \hspace{1cm} (31)$$

#19-3
Processing times have been redefined as follows so that the makespan cannot be shorter than the time required to complete the longest job:

\[
\tilde{P}_{ij} = \begin{cases} 
  P_{ij} & \text{if } P_{ij} \leq \lambda^* \\
  \infty & \text{if } P_{ij} > \lambda^*
\end{cases}
\]  

(32)

A feasible schedule is constructed by guessing different values of the makespan \(\lambda^*\). We will show that the approximation factor is \(\alpha = 2\) in this problem with a different technique.

**Property 1.** After rounding, the load \(l_i\) on machine \(M_i\) is at most

\[
l_i \leq \sum y^*_i \tilde{P}_{ij} + \max_{\tilde{P}_{ij} \neq \infty} \tilde{P}_{ij}
\]

(33)

\[
\leq 2\lambda^*.
\]

(34)

The first term of Equation 33, which is the fractional LP solution, is exactly \(\lambda^*\). The second term is a combinatorial term and does not depend on the LP. By the definition of \(\tilde{P}_{ij}\), this term is at most \(\lambda^*\). To argue about the consistency of this property further, we will make a slightly longer and more detailed argument that will complete the proof.

Consider a basic feasible solution (BFS) \(y^*\) to the LP. Observe that if \(\tilde{P}_{ij} = \infty\) then \(y^*_i = 0\). All such edges \(\tilde{P}_{ij}\) are not in the solution. We now ask: what is the number of edges \((i, j)\) such that \(y^*_i > 0\)? There are a total of \(mn\) variables in the LP. We know by the feasibility of the LP that in turn \(mn\) constraints must be tight. In total there are \(m + n + mn\) constraints in the LP, so we can conclude that:

\[
|\{(i, j) \in E | y^*_i > 0\}| \leq m + n.
\]

Define the bipartite graph \(H = (M \cup J, E)\) where \(E = \{(i, j) : y^*_i > 0\}\). We know \(|E| < m + n\). Suppose \(k\) jobs have degree at least 2 in \(H\). \(|E| \geq 2k + (m - k)\). This is because every job has degree at least 1 and \(k\) jobs have degree at least 2. It follows from \(|E| \leq m + n\) that \(k \leq m\).

If the degree is 1 for some job, than the \(y_{ij} = 1\) as there is only a single choice to make. For any job \(j\) with degree 1 in \(H\), we assign it to its unique neighbor and remove it from \(H\). Such assignments are already integral.

The reduced graph has \(n'\) jobs and \(m\) machines. \(n' \leq m\). For this graph we know that \(0 < y_{ij} < 1\) and \(\tilde{P}_{ij} < \infty\). Our goal is to show that this graph contains a perfect (integral) matching on the jobs. We will show that every machine is mapped to all jobs for which it is fully responsible and to one job that is fractionally assigned to it.

**Definition 1.** A connected graph is called a pseudo-tree if \(|E| = |V|\).

A pseudo-tree can be thought of as a tree with an extra edge.

**Lemma 1.** Any connected component of \(H\) is a pseudo-tree, i.e. each connected component on \(k\) vertices has at most \(k\) edges.

**Proof.**

\[
\text{#edges} \leq (m + n) - (n - n')
\]

(35)

\[
= m + n'
\]

(36)

\(\square\)
Looking at a connected component $C$, remove everything else. On that component’s LP, the solution $y^*_C$ is BFS. If the solution $y^*_C$ is not a BFS on the component, it won’t be a solution on the entire graph. The reason is that we could write the solution as a convex combination of feasible points that can be extended to the full LP.

We now shot that every pseudo-tree has a perfect matching and conclude our proof of the 2-factor approximation.

**Lemma 2.** Every bipartite pseudo-tree has a perfect matching.

*Proof.* Leaves are machines since each job has degree at least 2. Match such degree 1 machines with their unique neighboring job. Remove two vertices. What remains is still a pseudo-tree. We can recurse and complete the argument using induction. When there are no degree 1 vertices left, the graph is cyclic. Since the graph is bipartite, then it must be composed of even length cycles. For this case, simply take alternating edges in each cycle and use them in the final matching. The final matching is still a perfect matching. Thus, each machine gets at most one more job than what it got integrally in the fractional solution. By parametric pruning we complete the proof that the algorithm is a 2-approximation.

3 **Summary**

In this lecture we demonstrated how approximation algorithms can be designed using the technique of LP relaxation and rounding. We initially introduced basic terminology and gave a high level overview of the technique. We then covered in more detail how deterministic and randomized rounding could be used to design approximation algorithms for the problem of Vertex Cover, Set Cover and Load Balancing on Unrelated Machines.

**References**