Approximation Algorithm

Example. A polynomial time approximation algorithm for vertex cover with an approximation ratio of 2, i.e.

$$|\text{vertex cover returned by algo}| \leq 2 \times |\text{minimum vertex cover}|$$

Solution. □

Steps

1. Find a solution that is less than or equal to $k \times OPT$.
2. If we dont know OPT, we find a lower bound for OPT, i.e. $LB \leq OPT$. And then show that solution is less than or equal to $k \times LB \leq k \times OPT$

Example. Vertex cover given a graph G, find a vertex cover of minimum size

Solution. □

LPP Define a variable x_i corresponding to each node $v_i \in V$. The LPP as follows

$$\begin{aligned} \textbf{Minimize:} \quad & x_1 + x_2 + \cdots + x_n \\ \textbf{Subject to:} \quad & x_i + x_j \geq 1 \quad \text{if } (v_i, v_j) \in E \\ & x_i \in \{0, 1\} \end{aligned}$$

Integer linear programming problem (ILP) is NP-hard. To find a solution we write the last constraint in an equivalent form

$$\begin{aligned} \textbf{Minimize:} \quad & \sum_{i=1}^{n} x_i \\ \textbf{Subject to:} \quad & x_i + x_j \geq 1 \quad \text{if } (v_i, v_j) \in E \\ & 0 \leq x_i \leq 1 \\ & x_i \in \mathbb{Z} \end{aligned}$$

We can relax ILP to LPP by dropping the $x_i \in \mathbb{Z}$ constraint

$$\begin{aligned} \textbf{Minimize:} \quad & \sum_{i=1}^{n} x_i \\ \textbf{Subject to:} \quad & x_i + x_j \geq 1 \quad \text{if } (v_i, v_j) \in E \\ & 0 \leq x_i \leq 1 \end{aligned}$$

Now assume $(x_1^*, \cdots, x_n^*)$ is the optimal solution to LPP, how do we get the final solution? We can define the cover C as follows

1. if $x_i^* \geq \frac{1}{2}$ put v_i to the cover C

2. if $x_i^* < \frac{1}{2}$ dont put v_i in cover C

In other words,

$$x_i' = \begin{cases} 1 & \text{if } x_i^* \geq \frac{1}{2} \\ 0 & \text{if } x_i^* < \frac{1}{2} \end{cases}$$

therefore

$$|C| = \sum_{i=1}^{n} x_i' \geq \sum_{i=1}^{n} x_i^*$$

Consider any edge $(v_i, v_j) \in E$ we have $x_i^* + x_j^* \geq 1$ implies that $x_i^* \geq \frac{1}{2}$ or $x_j^* \geq \frac{1}{2}$ implies that either $v_i \in C$ or $v_j \in C$. To find the **approximation ratio** we take some minimum vertex cover C' and find a solution between $|C| \leq k|C'|$. C' satisifies the ILP. Define

$$\hat{x}_i = \begin{cases} 0 & \text{if } x \notin C' \\ 1 & \text{if } x \in C' \end{cases}$$

Then

$$|C'| = \sum_{i=1}^{n} \hat{x}_i \geq \sum_{i} x_i^*$$

Observe that $x_i' \leq 2 \times x_i^*$, hence

$$\sum_{i=1}^{n} x_i' \leq 2 \sum_{i=1}^{n} x_i^*$$
$$|C'| \geq 2 \times |C|$$

Definition. ***Approximation ratio***

1. ***vertex cover*** *2*

2. ***Knapsack*** *make it as close to 1 as possible (but complexity increases accordingly)*

3. ***traveling salesman*** *There is no constant ratio unless $P = NP$*

Motivation: intractable probems are NP-hard/NPC ($P \neq NP$)

Theorem. ***Traveling salesman*** *(TS) cannot have a constant approximation ratio.*

Proof. Assume there is an approximation algorithm with a constant approximation ratio of c,i.e.

$$|\text{tour returned by algo}| \leq c \times OPT$$

The we can show that this algo can be used to solve an NP hard problem, namely the **hamiltonian-cycle** problem. Reduce $HamCycle \leq_p TS$. Let $G = (V, E)$ be a graph, input to TS. Let $G' = (V, E')$ with weights w' where

$$E' = \{(u, v) | u, v \in V\}$$

$$w'(e) = \begin{cases} 1 & \text{if } e \in E \\ cn + 1 & \text{if } e \notin E \end{cases}$$

i.e. G' is complete. Note this construction can be done in polynomial time. Assume, if G has a hamiltonian cycle, then G' has a tour of length n. Conversely, if G does not have a hamiltonian cycle, then every tour (there will always be a tour in complete graph) in G' must pass through at least one edge of length $cn + 1$, and hence have a total weight greater than cn. Now run approximation algorithm on G'. If this algo returns a tour of weights of $\leq cn$ (i.e. all edges in the tour is in G) then there is a hamiltonian cycle in G. If the algo returns a tour of total weights of more than cn, then G does not have a hamiltonian cycle □

Definition. *For Traveling salesman problem with triangle inequality, then there is a 2-approximation algorithm*

Definition. ***Knapsack problem*** *Given a set of objects $\{1, \cdots, n\}$ with weight w_i and value v_i and a weight limit W. We want to find a subset of objects in $S \in \{1, \cdots, n\}$ such that (optimization problem)*

$$\sum_{i \in S} w_i \leq W \quad \textit{and} \quad \sum_{i \in S} v_i \textit{ is maximized}$$

Another variation (decision problem) introduces a value limit V, and we want to find S such that

$$\sum_{i \in S} w_i \leq W \quad \textit{and} \quad \sum_{i \in S} v_i \geq V$$

Theorem. *0-1 Knapsack is NP-hard*

Proof. Reduce subset-sum (SS) to Knapsack (decision variation). Given a set $S = \{s_1, \cdots, s_n\}$ and a number t. Define

1. $w_i = s_i$
2. $v_i = s_i$
3. $W = t$

4. $V = t$

and let S' be a solution to SS

$$\sum_{i \in S'} w_i \leq W \iff \sum_{i \in S'} s_i \leq t$$

$$\sum_{i \in S'} v_i \geq V \iff \sum_{i \in S'} s_i \geq t$$

so $SS \leq_p Knapsack$ and SS is NP-hard implies *knapsack* is NP-hard □

Definition. *Note Knapsack problem is NP-hard, we had found a Dynamic programming solution (pseudo-polynomial).*

Solution. □

Define $P = max\{v_1, \cdots, v_n\}$ for each $i \in \{1, \cdots, n\}$ and each $v \in \{1, \cdots, nP\}$. Let $s_{i,v}$ be a subset of $\{1, \cdots, i\}$ that has a total value of exactly v and has the minimum weight (keep value same and minimize the weight). Let $A[i, v]$ be total weight of $S_{i,v}$. We have base case,

$$A[1, v_1] = w_1 \qquad A[i, v] = \infty$$

and recurrence

$$A[i+1, v] = \begin{cases} w_{i+1} + A[i, V - v_{i+1}] & \textit{if } i+1 \in S_{i+1,v} \wedge v + i + 1 < v \\ A[i, v] & \textit{otherwise} \end{cases}$$

Hence

$$A[i+1, v] = Min\{A[i, v], w_{i+1} + A[i, v - v_{i+1}]\}$$

Complexity is $O(n \times nP) = O(n^2 P)$. , where P depends on $v_1, \cdots, v_n$. So a pseudo-polynomial algo.

Function `Knapsack-Approximation` $(\{v_i\}, \{w_i\}, W, \epsilon)$
    $P \leftarrow Max\{v_1, \cdots, v_n\}$
    $k \leftarrow \dfrac{\epsilon P}{n}$
    for $i = i$ **to** n **do**
        $v'_i \leftarrow \lfloor \dfrac{v_i}{k} \rfloor$
    With the new v'_i as the new values, use DP to find the most valuable set S'
    return S'

Definition. ***Complexity*** *the critical DP part is $O(n^2 P')$ where*

$$P' = Max\{v'_1, \cdots, v'_n\} = Max\{\lfloor \frac{v_i}{k} \rfloor, \cdots, \} = \frac{P}{k} = \frac{n}{\epsilon}$$

so $O(nP') = O(n^2 \dfrac{n}{\epsilon}) = O(\dfrac{n^3}{\epsilon})$. the closer ϵ goes to zero, complexity go to ∞

Lemma. *The set S' satisifies $OPT \geq v(S') \geq (1-\epsilon) \times OPT$*

Proof. Let O be an optimal solution returning the maximum value possible. Note $v_i = kv_i' + \delta$, where $0 \leq \delta < k$, then

$$kv_i' \leq v_i$$

and the difference is at most k. on the new weights v_i', DP returns the best solution, i.e. $v'(O) \leq v'(S')$. For each object in O, the difference in value is at most k, so

$$v(O) - kv'(O) \leq nk$$

Finally,

$$v(S') \geq kv'(O) \geq v(O) - nk = OPT - \epsilon P \geq OPT - \epsilon OPT = (1-\epsilon) \times OPT$$

□

Random algorithm

Definition. ***Monte Carlo simulation***
π is ratio of circumference to diameter. We can find the an approximation to π by running lots of random simulation. Generate random coordinate in unit square containing a unit circle. Let N be number of coordinates inside the unit circle then

$$\frac{N}{totalnumberofsimulation}$$

is close to π

Example. Find a maximum independent set in a tree.

Lemma. *If a node v is a leaf, then there exists a maximum cardinality independent set containing v, i.e. every leaf is in the set*

Proof. Exchange argument. Consider a max cardinality independent set S.

1. If $v \in S$, then done
2. If $v \notin S$, consider edge $(u,v) \in E$ in tree
   (a) if $u \in S$, $S' = S \setminus \{u\} \cup \{v\}$, still independent. $|S'| = |S|$
   (b) if $u \notin S$, then $S' = S \cup \{v\}$ is independent $|S'| > |S|$.

□

```
Function Independent-Set-In-A-Forest(F)
    S ← ∅
    while F has at least one edge do
        e ← (u, v) where v is a leaf
        S ← S ∪ {v}
        F = F \ {u, v}
    return S ∪ {nodes remaining in F}
```