CS6161: Design and Analysis of Algorithms (Fall 2020)

Approximation Algorithms

Instructor: Haifeng Xu

Outline

- Introduction and a Simple Example
- Greedy Algorithms
- Randomized Algorithms, and De-randomization
- Fully Polynomial-Time Approximation Schemes (FPTAS)

Approximation Algorithm for Optimization Problems

Analyzing algorithms/heuristics by proving *approximate* guarantee of optimality

- Typically for NP-hard problems
- But can also design simpler/faster approximate Algo for poly-time solvable problems

Terminologies

➢ How to evaluate approximate guarantee?

Minimization problems

Algorithm Algo is a (multiplicative) α-approximation to a minimization problem if for any instance I of the problem, we have

$$\text{Algo}(I) \leq \alpha \cdot OPT(I)$$

Algo is additive α-approximation if $\text{Algo}(I) \leq OPT(I) + \alpha$

Remarks:

- Typically, multiplicative approx satisfies $OPT(I) > 0$ and $\alpha \geq 1$
- Typically, additive approx makes sense only when $OPT(I)$ is bounded within interval $[-c, c]$
  - Sometimes, additive α-approximation is called α-optimal
- Both are common, and they can be very different
  - In concrete problems, choice depends on what you can show

Terminologies

- Maximization problem is similar

Maximization problems

Algorithm Algo is a (multiplicative) α-approximation to a minimization problem if for any instance I of the problem, we have

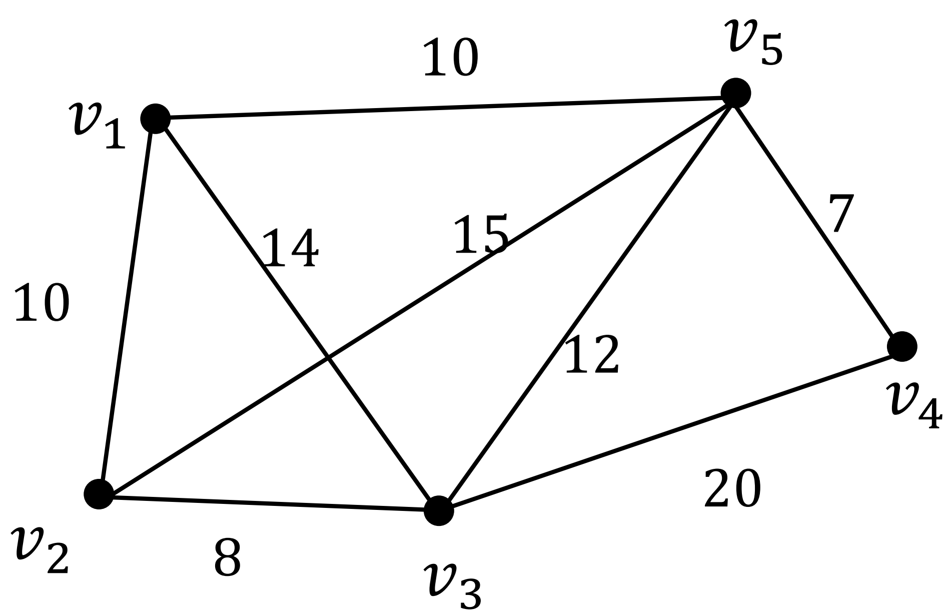
$$\mathsf{Algo}(I) \geq \alpha \cdot OPT(I)$$

Algo is additive α-approximation if $\mathsf{Algo}(I) \geq OPT(I) - \alpha$

- For multiplicative approx.: requires $OPT(I) > 0$ and $\alpha \leq 1$
- For additive approx.: still requires $OPT(I)$ is bounded within interval $[-c, c]$

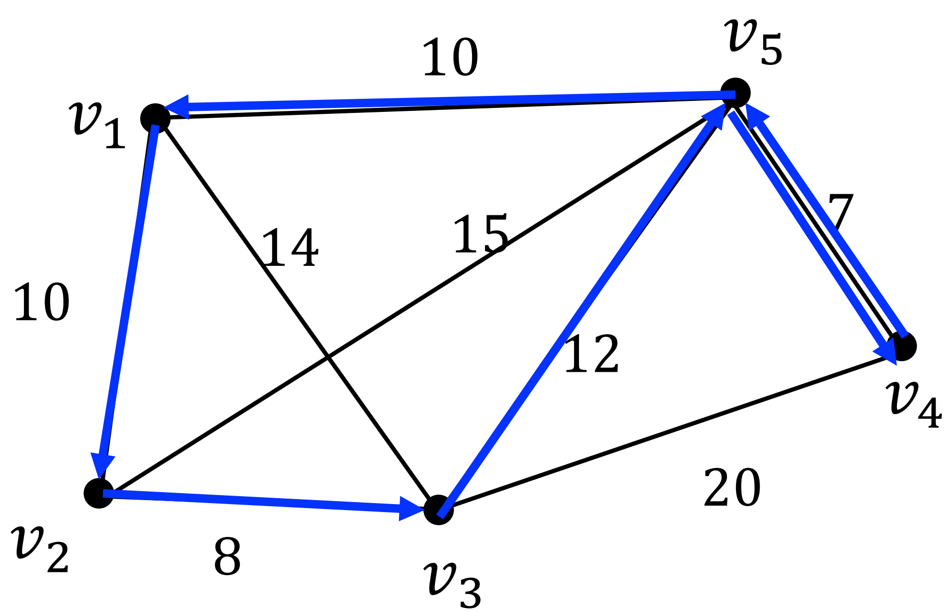
The Travelling Salesman Problem (TSP)

- Finding a shortest route that traverse through each node exactly once and return to the origin



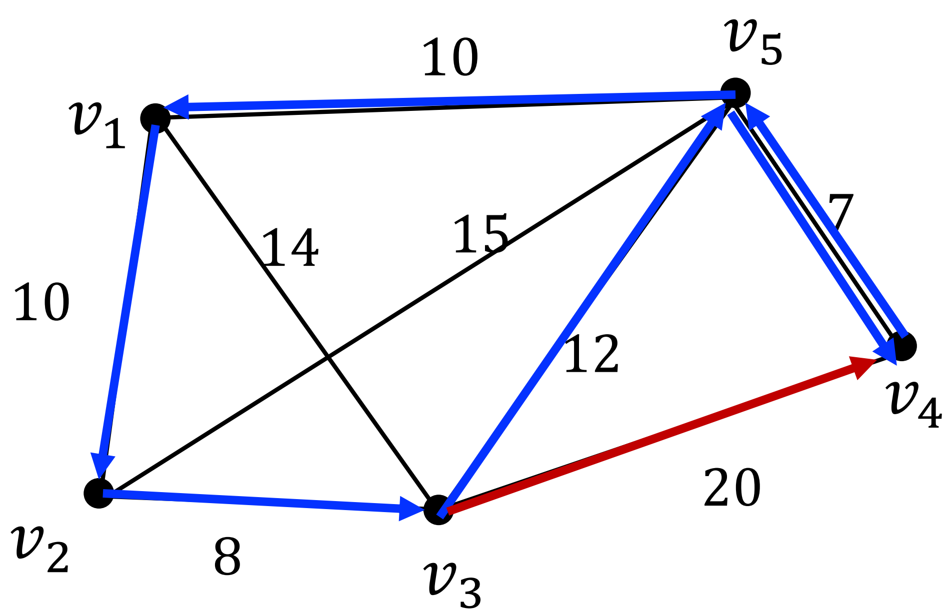
The Travelling Salesman Problem (TSP)

- Finding a shortest route that traverse through each node exactly once and return to the origin
  - Note: in Lec 7, we considered a different TSP version that allows a node to be traversed for multiple times



The Travelling Salesman Problem (TSP)

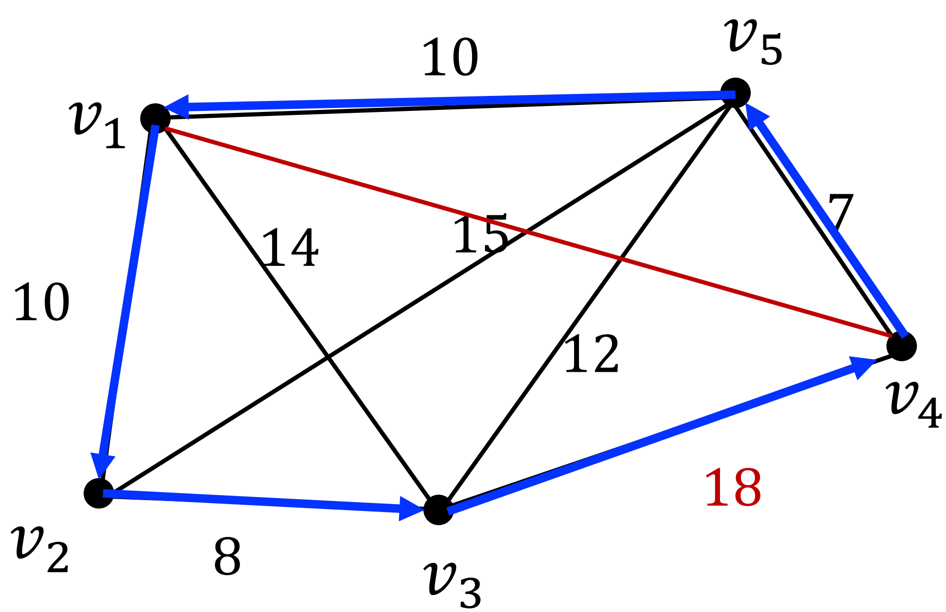
- Finding a shortest route that traverse through each node exactly once and return to the origin
  - Note: in Lec 7, we considered a different TSP version that allows a node to be traversed for multiple times
  - When would a shortest route ever want to visit the same node twice?



When triangle inequality is violated: $c_{35} + c_{54} < c_{34}$!

The Travelling Salesman Problem (TSP)

- Finding a shortest route that traverse through each node exactly once and return to the origin
  - Next: we assume triangle inequality holds (natural in reality)
  - Note: this implies there is an edge between any two nodes



The Travelling Salesman Problem (TSP)

- Finding a shortest route that traverse through each node exactly once and return to the origin
  - Next: we assume triangle inequality holds (natural in reality)
  - Note: this implies there is an edge between any two nodes

Claim: Assume triangle inequality, there always exists an optimal TSP route that visits each node exactly once (except the origin).

Proof:

- Suppose any non-origin node v_i is visited twice
- The second time v_i is visited is through … $v_j \rightarrow v_i \rightarrow v_k$ …
- The route that directly goes from … $v_j \rightarrow v_k$ … is no worse by triangle inequality

The Travelling Salesman Problem (TSP)

- Finding a shortest route that traverse through each node exactly once and return to the origin
  - Next: we assume triangle inequality holds (natural in reality)
  - Note: this implies there is an edge between any two nodes

Claim: Assume triangle inequality, there always exists an optimal TSP route that visits each node exactly once (except the origin).

Remark

- Assume triangle inequality, it is natural to traverse each node exactly once, except the origin
- Proof also provides a simple way to convert any shortest route to a shortest route that visits each node once

The Travelling Salesman Problem (TSP)

- A multiplicative 2-approximation for TSP satisfying triangle inequality

The Travelling Salesman Problem (TSP)

- A multiplicative 2-approximation for TSP satisfying triangle inequality

APX-TSP($G = (V, E, \{c_e\}_{e \in E})$

1 Find a minimum spanning tree (MST) with root v_1 as the origin

The Travelling Salesman Problem (TSP)

- A multiplicative 2-approximation for TSP satisfying triangle inequality

APX-TSP($G = (V, E, \{c_e\}_{e \in E})$)

1 Find a minimum spanning tree (MST) with root v_1 as the origin

2 Traverse through the MST by DFS until ending at v_1

The Travelling Salesman Problem (TSP)

- A multiplicative 2-approximation for TSP satisfying triangle inequality

APX-TSP($G = (V, E, \{c_e\}_{e \in E})$)

1 Find a minimum spanning tree (MST) with root v_1 as the origin

2 Traverse through the MST by DFS until ending at v_1

3 Alone the way: skip any node that has been visited before

// If BFS visits … $v_j \rightarrow v_i \rightarrow v_k$ … where v_i visited before, the route takes … $v_j \rightarrow v_k$ … instead

The Travelling Salesman Problem (TSP)

- A multiplicative 2-approximation for TSP satisfying triangle inequality

APX-TSP($G = (V, E, \{c_e\}_{e\in E})$)

1 Find a minimum spanning tree (MST) with root v_1 as the origin

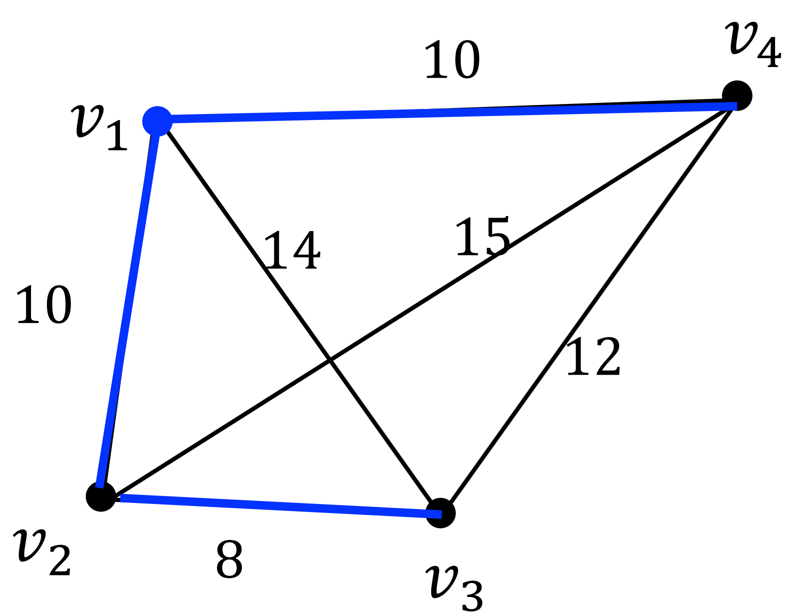
2 Traverse through the MST by DFS until ending at v_1

3 Alone the way: skip any node that has been visited before

// If BFS visits … $v_j \to v_i \to v_k$ … where v_i visited before, the route takes … $v_j \to v_k$ … instead

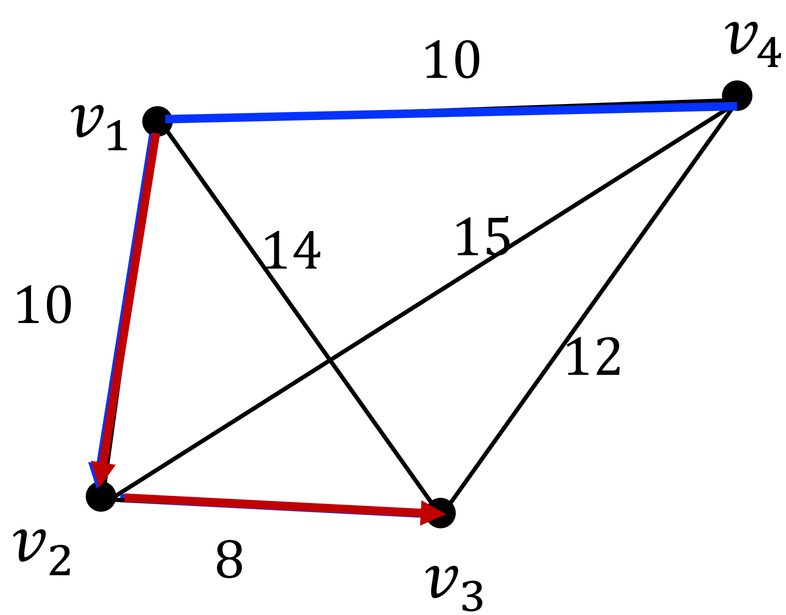
Theorem: This is a multiplicative 2-approximation for TSP that satisfies non-negative costs and triangle inequality.

Example Execution



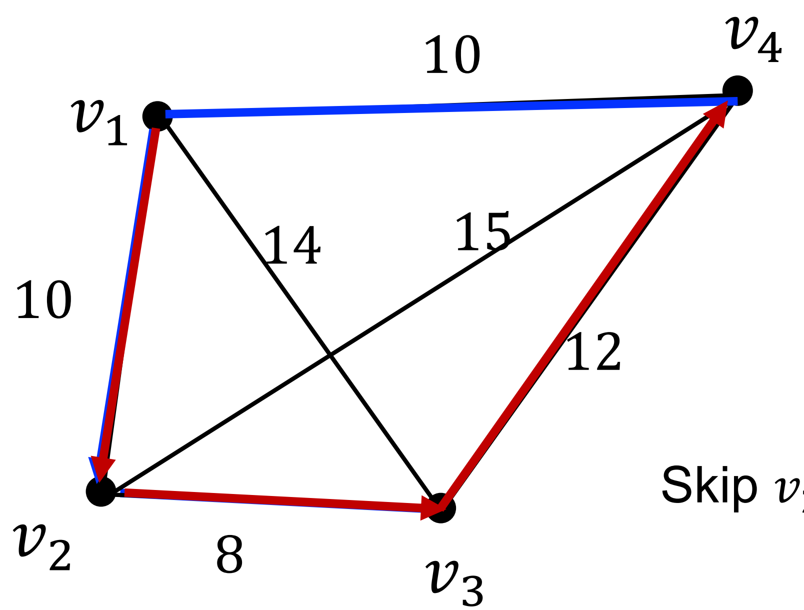
Find MST with v_1 as root

Example Execution



Traverse the MST by DFS, and skip a node if visited already

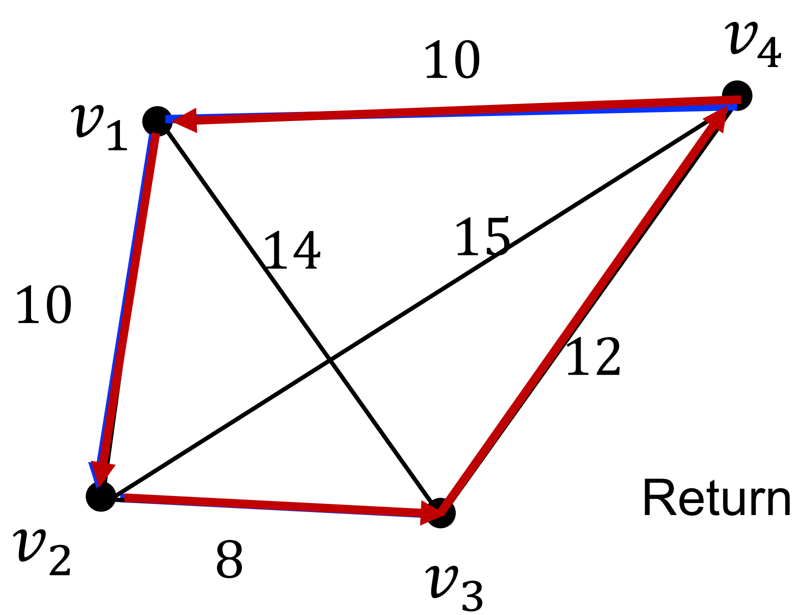
Example Execution



Skip v_2, v_1; directly go to v_4

Traverse the MST by DFS, and skip a node if visited already

Example Execution



Return to v_1 after visiting all nodes

Traverse the MST by DFS, and skip a node if visited already

The Travelling Salesman Problem (TSP)

Theorem: APX-TSP is a multiplicative 2-approximation for TSP that satisfies non-negative costs and triangle inequality.

Proof:

1. APX-TSP outputs a route that visits each node once, except v_1
   - Follows from definition
2. Total costs is at most twice the MST, because
   - (1) BFS visits each edge at most twice
   - (2) The “skipping” step will not increase cost due to triangle inequality
3. Total cost of any TSP solution is at least the MST
   - Because any TSP solution is a tour, which is a spanning tree with an additional edge
   - So its cost is at least the cost of a MST

Point 2 and 3 → cost of APX-TSP $\leq 2\ c(MST) \leq 2\ OPT(TSP)$

The Travelling Salesman Problem (TSP)

Theorem: APX-TSP is a multiplicative 2-approximation for TSP that satisfies non-negative costs and triangle inequality.

Remarks:

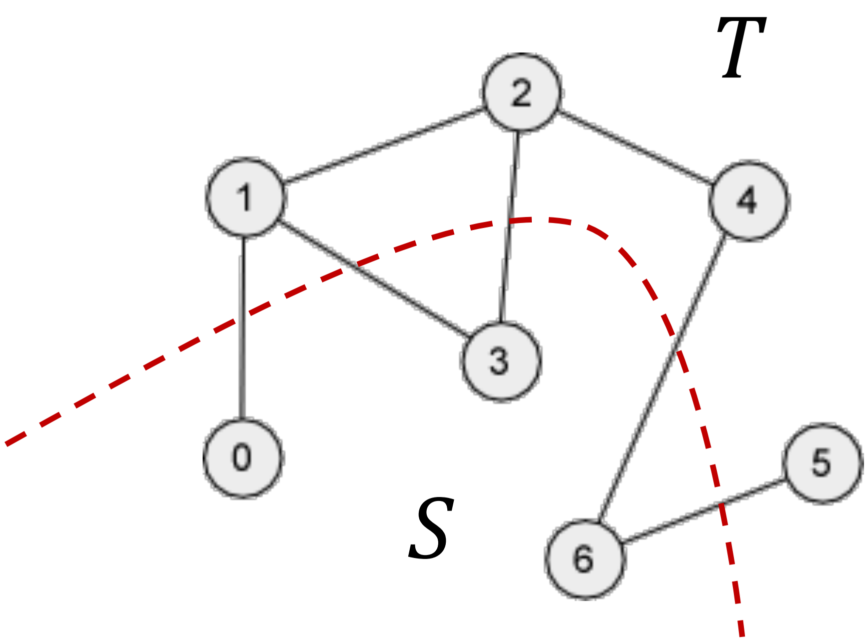
- If nodes allow multiple visits, any TSP instance can be reduced to an instance satisfying triangle inequality
  - By redefining distance between any i, j as the shortest distance in the original graph
- Best currently known algorithm is a 1.5-approximation
- If nodes allow only a single visit, impossible to obtain any guaranteed approximation unless P=NP.

Outline

- Introduction and a Simple Example
- Greedy Algorithms
- Randomized Algorithms, and De-randomization
- Fully Polynomial-Time Approximation Schemes (FPTAS)

The Max-Cut Problem

- We learned the Min-Cut problem
- Max-Cut is slightly different (though relevant, obviously)
  - An undirected graph
  - A cut is still a partition of nodes into two disjoint sets S, T
  - Goal: maximize #edges crossing the cut



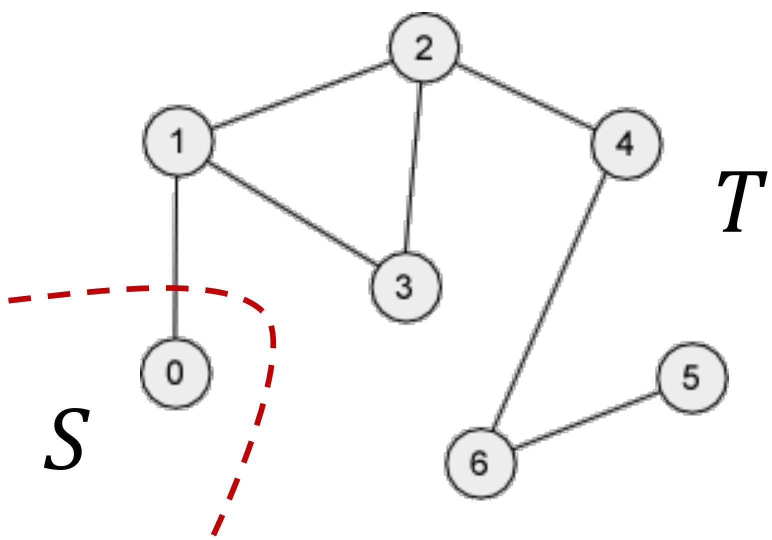
Note:

- Different from Min-Cut, Max-Cut is NP-hard and thus does not have a poly time algorithm

A Simple Greedy Algorithm

MaxCut-Greedy$(G = (V, E))$

1 Start with an arbitrary cut S, T

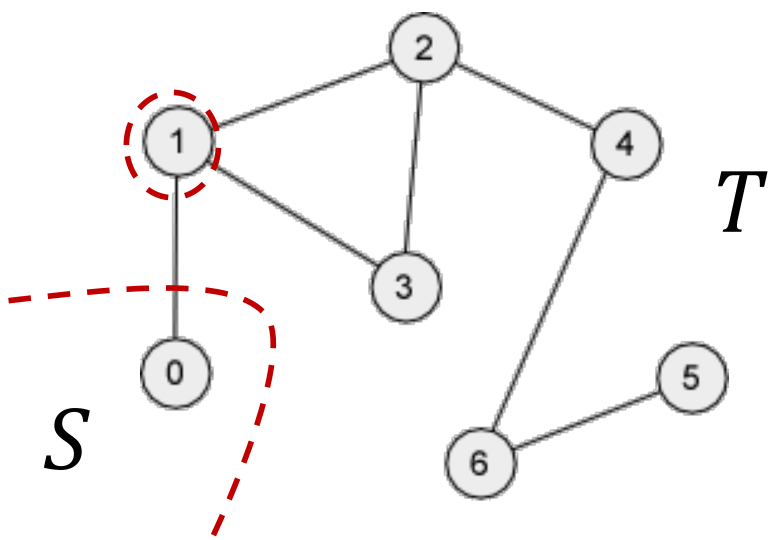


A Simple Greedy Algorithm

MaxCut-Greedy$(G = (V, E))$

1 Start with an arbitrary cut S, T

2 While $\exists u \in S$ such that moving u from S to V, or moving some $u \in V$ to S, increases the size of the cut, do it



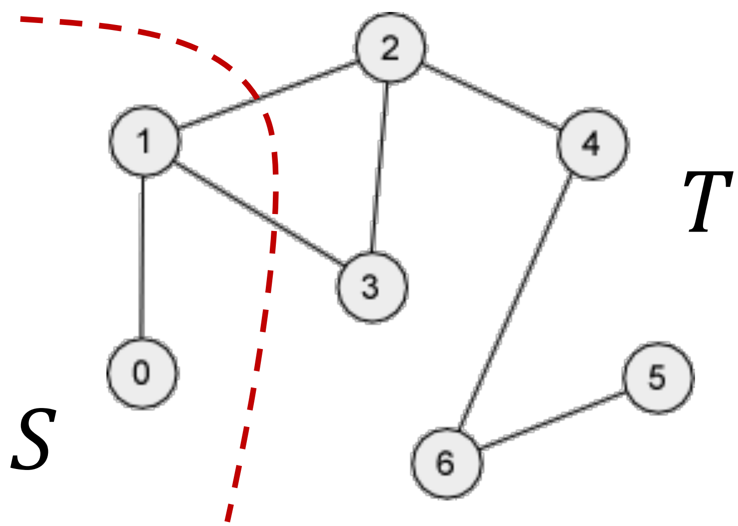
A Simple Greedy Algorithm

MaxCut-Greedy($G = (V, E)$)

1 Start with an arbitrary cut S, T

2 While $\exists u \in S$ such that moving u from S to V, or moving some $u \in V$ to S, increases the size of the cut, do it

3 Output the cut S, T when no improvement is possible



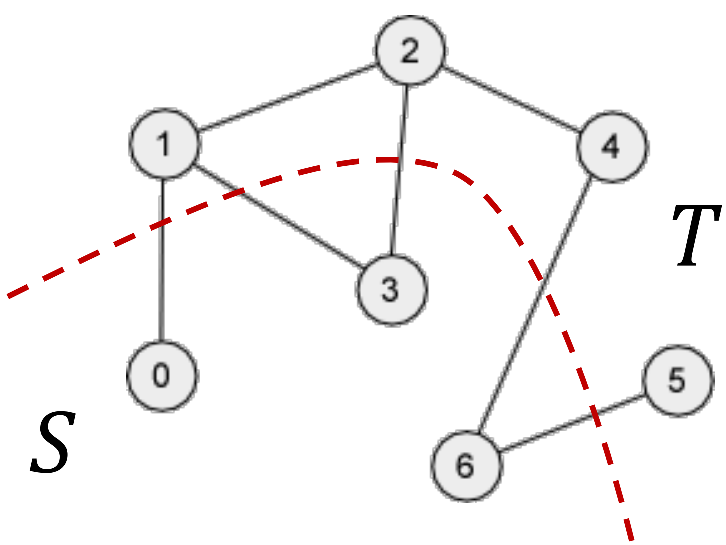
Step 2 terminates in at most $|E|$ steps

Greedy is a ½ -Approximation

Theorem: *MaxCut-Greedy* is a multiplicative ½ -approximation for *MaxCut* Problem.

Proof:

- The move of any node v only affects edges connects to v



Q: When does a node want to move, and when does not? ?

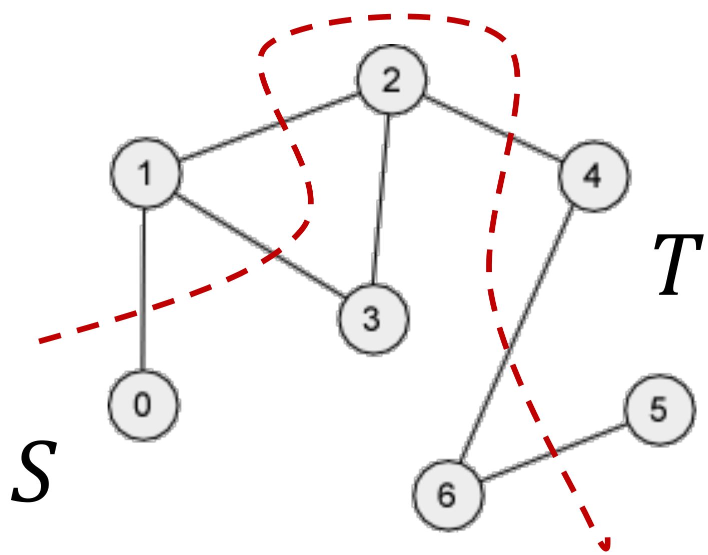
- v_1 will not move
- v_2 will move

Greedy is a ½ -Approximation

Theorem: *MaxCut-Greedy* is a multiplicative ½ -approximation for *MaxCut* Problem.

Proof:

- The move of any node v only affects edges connects to v
- When no nodes want to move, for each node v at least half of its edges are cut.



Greedy is a ½ -Approximation

Theorem: *MaxCut-Greedy* is a multiplicative ½ -approximation for *MaxCut* Problem.

Proof:

- The move of any node v only affects edges connects to v
- When no nodes want to move, for each node v at least half of its edges are cut
- Thus, in total, the number of edges cut is at least

$$\frac{\sum_v \frac{d_v}{2}}{2} = \frac{|E|}{2}$$

Each edge has 2 nodes,
thus is double counted

Greedy is a ½ -Approximation

Theorem: $MaxCut\text{-}Greedy$ is a multiplicative ½ -approximation for $MaxCut$ Problem.

Proof:

- The move of any node v only affects edges connects to v
- When no nodes want to move, for each node v at least half of its edges are cut
- Thus, in total, the number of edges cut is at least

$$\frac{\sum_v \frac{d_v}{2}}{2} = \frac{|E|}{2}$$

- This is at least half of the max cut size, which is at most $|E|$

Outline

- Introduction and a Simple Example
- Greedy Algorithms
- Randomized Algorithms, and De-randomization
- Fully Polynomial-Time Approximation Schemes (FPTAS)

A Simple Randomized Algorithm for MaxCut

$\mathsf{MaxCut\text{-}Rand}(G = (V, E))$

1 Assign each node $v \in V$ to S or T uniformly at random

2 Output the cut S, T

Theorem: expected size of the cut by $\mathsf{MaxCut\text{-}Rand}$ is $\frac{1}{2}|E|$.

Proof:

- Due to randomness, an edge may or may not be cut
- Let random var $\mathbb{I}_e \in \{0,1\}$ denote whether e was cut or not
- Expected size of cut = $\mathbb{E}(\sum_e \mathbb{I}_e)$ = $\sum_e \mathbb{E}(\mathbb{I}_e)$

By linearity of expectation

A Simple Randomized Algorithm for MaxCut

MaxCut-Rand$(G = (V, E))$

1 Assign each node $v \in V$ to S or T uniformly at random

2 Output the cut S, T

Theorem: expected size of the cut by MaxCut-Rand is $\frac{1}{2}|E|$.

Proof:

- Due to randomness, an edge may or may not be cut
- Let random var $\mathbb{I}_e \in \{0,1\}$ denote whether e was cut or not
- Expected size of cut = $\mathbb{E}(\sum_e \mathbb{I}_e)$ = $\sum_e \mathbb{E}(\mathbb{I}_e)$

$$\mathbb{E}(\mathbb{I}_e) = \Pr(\mathbb{I}_e = 1)$$

Since $\mathbb{I}_e \in \{0,1\}$

A Simple Randomized Algorithm for MaxCut

MaxCut-Rand$(G = (V, E))$

1 Assign each node $v \in V$ to S or T uniformly at random

2 Output the cut S, T

Theorem: expected size of the cut by MaxCut-Rand is $\frac{1}{2}|E|$.

Proof:

- Due to randomness, an edge may or may not be cut
- Let random var $\mathbb{I}_e \in \{0,1\}$ denote whether e was cut or not
- Expected size of cut = $\mathbb{E}(\sum_e \mathbb{I}_e)$ = $\sum_e \mathbb{E}(\mathbb{I}_e)$

$$\mathbb{E}(\mathbb{I}_e) = \Pr(\mathbb{I}_e = 1) = \Pr(u, v \text{ in different set}; e = (u, v))$$

By definition

A Simple Randomized Algorithm for MaxCut

MaxCut-Rand$(G = (V, E))$

1 Assign each node $v \in V$ to S or T uniformly at random

2 Output the cut S, T

Theorem: expected size of the cut by MaxCut-Rand is $\frac{1}{2}|E|$.

Proof:

- Due to randomness, an edge may or may not be cut
- Let random var $\mathbb{I}_e \in \{0,1\}$ denote whether e was cut or not
- Expected size of cut = $\mathbb{E}(\sum_e \mathbb{I}_e)$ = $\sum_e \mathbb{E}(\mathbb{I}_e)$

$$\mathbb{E}(\mathbb{I}_e) = \Pr(\mathbb{I}_e = 1) = \Pr(u, v \text{ in different set}; e = (u, v)) = 1/2$$

A Simple Randomized Algorithm for MaxCut

MaxCut-Rand$(G = (V, E))$

Assign each node $v \in V$ to S or T uniformly at random

Output the cut S, T

Theorem: expected size of the cut by MaxCut-Rand is $\frac{1}{2}|E|$.

Proof:

- Due to randomness, an edge may or may not be cut
- Let random var $\mathbb{I}_e \in \{0,1\}$ denote whether e was cut or not
- Expected size of cut = $\mathbb{E}(\sum_e \mathbb{I}_e)$ = $\sum_e \mathbb{E}(\mathbb{I}_e)$ = $\frac{1}{2}|E|$

$$\mathbb{E}(\mathbb{I}_e) = \Pr(\mathbb{I}_e = 1) = \Pr(u, v \text{ in different set}; e = (u, v)) = 1/2$$

De-Randomization

- Randomization is not essential in previous algorithm
- Can produce a deterministic algorithm by de-randomization
- Suppose we assign node in order v_1, v_2 ..., we have

$$\begin{aligned}\frac{1}{2}|E| &= \mathbb{E}(\text{cut size}) \\ &= \mathbb{E}(\text{cut size}|v_1 \to S)\Pr(v_1 \to S) + \mathbb{E}(\text{cut size}|v_1 \to T)\Pr(v_1 \to T)\end{aligned}$$

Basic properties of conditional probabilities

De-Randomization

- Randomization is not essential in previous algorithm
- Can produce a deterministic algorithm by de-randomization
- Suppose we assign node in order v_1, v_2 ..., we have

$$\frac{1}{2}|E| = \mathbb{E}(\text{cut size})$$
$$= \mathbb{E}(\text{cut size}|v_1 \to S) \Pr(v_1 \to S) + \mathbb{E}(\text{cut size}|v_1 \to T) \Pr(v_1 \to T)$$
$$= \mathbb{E}(\text{cut size}|v_1 \to S) \cdot 1/2 + \mathbb{E}(\text{cut size}|v_1 \to T) \cdot 1/2$$

Since we assigned nodes uniformly at random

- One of $\mathbb{E}(\text{cut size}|v_1 \to S)$ and $\mathbb{E}(\text{cut size}|v_1 \to T)$ must be at least $\frac{1}{2}|E|$
- We assign v_1 deterministically according to the term with $\geq \frac{1}{2}|E|$ value
  - How to compute $\mathbb{E}(\text{cut size}|v_1 \to S)$? → Monte Carlo simulation

De-Randomization

- Randomization is not essential in previous algorithm
- Can produce a deterministic algorithm by de-randomization
- Suppose we assign node in order v_1, v_2 ..., we have

More generally, at iteration i for node v_i, we have

$$\begin{aligned}\frac{1}{2}|E| &\le \mathbb{E}(\text{cut size}|S_{i-1}\ ) \\ &= \mathbb{E}(\text{cut size}|S_{i-1}, v_i \to S_i) \cdot 1/2 + \mathbb{E}(\text{cut size}|S_{i-1}, v_i \to T_i) \cdot 1/2\end{aligned}$$

Assign v_i deterministically according to the term with $\ge \frac{1}{2}|E|$ value

Remarks

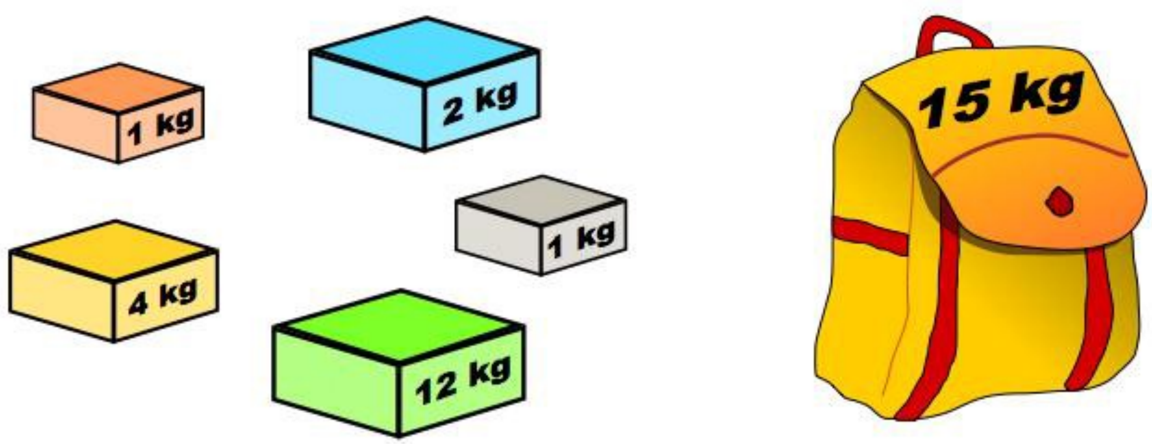
- Randomization was not essential in previous algorithm and thus did not bring additional advantages
  - Very fundamental research question: Can randomization bring strictly more power in algorithm design?
- Best approximation for max cut is 0.878 currently
  - Based on very sophisticated techniques: semidefinite programming and randomized routing
  - Believed to be the best possible

Outline

- Introduction and a Simple Example
- Greedy Algorithms
- Randomized Algorithms, and De-randomization
- Fully Polynomial-Time Approximation Schemes (FPTAS)

Recall the 0-1 Knapsack Problem

- You have a knapsack with weight capacity W
- There are n items – item i has value p_i and weight w_i
- Question: find a subset of items with maximum total value, subject to total capacity at most W
  - Will assume all numbers are integers



Recall: A DP for 0-1 Knapsack Problems

$W[i,p]$ = min weight of a subset of $\{1,2,\cdots,i\}$ with total value at least p

DP-01Knapsack$(W,\{\mathrm{p_i},w_i\}_{i=1,\cdots,n})$

1 Initialization: $W[0,0]=0, W[0,p]=\infty, \forall p>0$

2 DP update:

$$W[i,v]=\min\begin{cases} W[i-1,v] \\ w_i+W[i-1,v-p_i] \end{cases}$$

- Solution is the maximum v with $W[i,v]\le W$
- Pseudo-polynomial running time: $O(n^2P)$ where $P=\max_i\ p_i$

Recall: A DP for 0-1 Knapsack Problems

$W[i,p]$ = min weight of a subset of $\{1,2,\cdots,i\}$ with total value at least p

DP-01Knapsack($W, \{\mathrm{p_i}, w_i\}_{i=1,\cdots,n}$)

1 Initialization: $W[0,0] = 0, W[0,p] = \infty, \forall p > 0$

2 DP update:

$$W[i,v] = \min \begin{cases} W[i-1,v] \\ w_i + W[i-1, v-p_i] \end{cases}$$

Running time issue: too many possibilities of item values to consider in DP table

Idea: "round" the item values to a close neighbor value, to reduce possibilities of total item values

Approximate DP for 0-1 Knapsack Problems

$W[i,p]$ = min weight of a subset of $\{1,2,\cdots,i\}$ with total value at least p

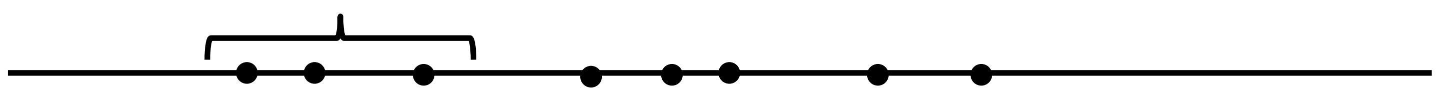
DP-01Knapsack$(W,\{\mathrm{p_i},w_i\}_{i=1,\cdots,n})$

1 Initialization: $W[0,0]=0, W[0,p]=\infty, \forall p>0$

2 DP update:

$$W[i,v]=\min\begin{cases} W[i-1,v] \\ w_i+W[i-1,v-p_i]\end{cases}$$

All values here rounded to $k\frac{\epsilon P}{n}$



Interval length $\frac{\epsilon P}{n}$

Approximate DP for 0-1 Knapsack Problems

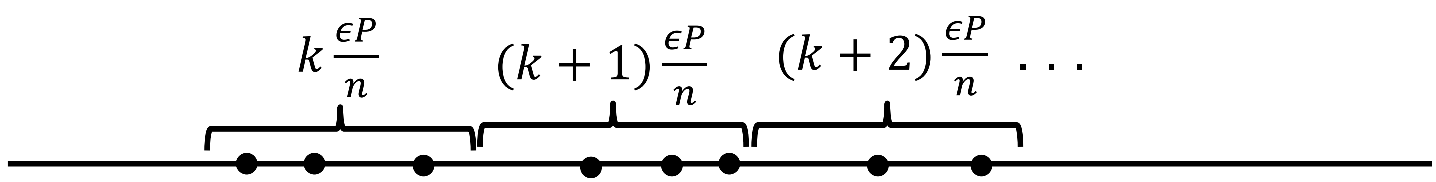
$W[i,p]$ = min weight of a subset of $\{1,2,\cdots,i\}$ with total value at least p

DP-01Knapsack$(W, \{\mathrm{p_i}, w_i\}_{i=1,\cdots,n})$

1 Initialization: $W[0,0] = 0, W[0,p] = \infty, \forall p > 0$

2 DP update:

$$W[i,v] = \min \begin{cases} W[i-1,v] \\ w_i + W[i-1, v-p_i] \end{cases}$$

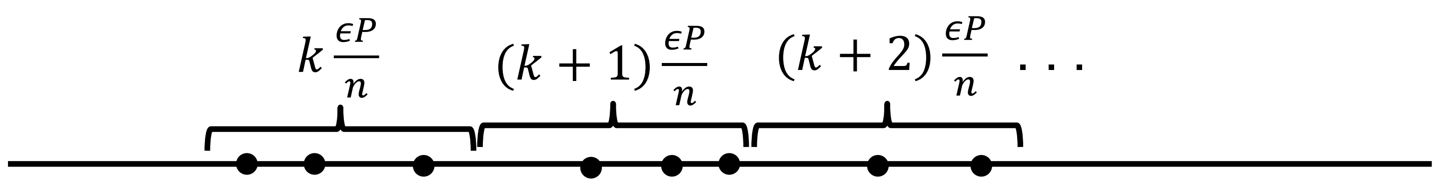


Interval length $\frac{\epsilon P}{n}$

Approximate DP for 0-1 Knapsack Problems

Thm: DP with these rounded item values runs in $O(n^3/\epsilon)$ time and is a multiplicative $(1-\epsilon)$ $-$approximation

- Tradeoff between approximation quality and running time
- This is called a Fully polynomial time approximation scheme (FPTAS)



Thank You

Haifeng Xu
University of Virginia
hx4ad@virginia.edu