Approximation Algorithm

NP-complete! What’s next?

- Lots of practical applications generate problems that are NP-complete
- We can’t afford to give up
- Three workarounds
  1. For small inputs, write an exponential algorithm
  2. Isolate important special cases that are solvable in polynomial time
  3. Find near optimal solutions in polynomial time, if you can guarantee the bound of near optimality, it’s an Approximation Algorithm, without provable bound – we will call it heuristic

Approximation Ratio

An algo for a problem has an approximation ratio of *p(n)*, if for any input size *n*, the cost *C* of the solution produced by the algo is within a factor of *p(n)* of the cost C^* of an optimal solution, i.e.,

$$max\left(\underset{\underset{\text{Minimization problems}}{\nearrow}}{\frac{C}{C^*}}, \underset{\underset{\text{Maximization problems}}{\nwarrow}}{\frac{C^*}{C}}\right) \leq p(n)$$

Contd.

- We call this a *p(n)-approximation* problem
- So, *p(n)* > 1 always
- A *1-approximation* algo is the *optimal* solution
- For some problems, *constant-approximation* exists, for some others *p(n)* is a function of *n*
  - Eg: For set cover problem, *p(n)* is $\log_2 n$

Approximation Algorithms

- Some NP-complete problems allow poly-time approximation algorithms to achieve increasingly smaller approximation ratios using more and more time
- An Approximation Scheme for an opt. problem is an approximation algorithm that takes as input, an instance of the problem as well as a value ε > 0, such that for any fixed ε, the scheme is a (1 + ε)-approximation algorithm
- It is called a poly-time approx. scheme if for any fixed ε > 0, the scheme runs in time polynomial in *n*
  - Eg: Sometimes the running time can rise very fast with ε like $O\left(n^{\frac{2}{\varepsilon}}\right)$

Contd.

- We would like it to be polynomial in $\frac{1}{\varepsilon}$ as well as in *n*
- We call the approx. scheme fully polynomial time if it's running time is polynomial in both $\frac{1}{\varepsilon}$ and *n*
  - Eg: $O\left(\left(\frac{1}{\varepsilon}\right)^2 n^3\right)$ - a constant factor decrease is possible by a constant factor increase in running time

Algorithm 1: Approx-Vertex-Cover(G)

```
begin
    $C \leftarrow \phi$
    $E' \leftarrow E[G]$
    while $E' \neq \phi$ do
        let $(u, v)$ be an arbitrary edge of $E'$
        $C \leftarrow C \cup \{(u, v)\}$
        remove from $E'$ every edge incident on either $u$ or $v$
    end
    return $C$
end
```

Running time $O(V + E)$

Theorem: The algo is 2-approximation algo

- Set *C* is a Vertex-Cover as it is verified till E’ is null.

Let *A* denote the set of edges picked in line 5.

No two edges in *A* share an endpoint.

So, lower bound $|C^*| \geq |A|$

$$|C| = 2\,|A|$$

$$\therefore \quad |C| \leq 2\,|C^*|$$

Hence, $\frac{|C\quad|}{|C*|} \leq 2$

Approximation Algorithm for 0/1 Knapsack Problem

- Consider the following heuristic

Assume that the objects are in non-increasing order of $\frac{p_i}{w_i}$. *If object i fits, then set $x_i = 1$, else set $x_i = 0$.*

- Suppose $(p1, p2) = (100, 20)$, $(w1, w2) = (4, 1)$ and $M = 4$.
- Since ${}^{100}/_{4} > {}^{20}/_{1}$, consider objects in the order of 1, 2
- The result is $(x1, x2) = (1, 0)$, which is also *optimal*.

However, consider another instance.

- $n = 2$, $(p1, p2) = (2, r)$, $(w1, w2) = (1, r)$ and $M = r$.
- When $r > 2$, the optimal solution is $(x1, x2) = (0, 1)$.
- Its value $F^*(I) = r$, and it should be actually $(x1, x2) = (0, 1)$, but our algorithm generates $(1, 0)$, so $F(I) = 2$

Contd.

- Recall, $F^*(I) = r$ and $F(I) = 2$
- This is a maximization problem, so

$$p(n) = \frac{F^*(I)}{F(I)} = \frac{r}{2}$$

Since *r* can be anything, this doesn't guarantee any approximation bound. Hence the earlier algorithm for selecting items in *non-increasing order* of $\frac{p_i}{w_i}$, was only a heuristic, not an approximation algorithm

Approx. Scheme for 0/1 Knapsack

Order is k – For higher the value of k, it runs for more time but can give better profit. Note that Algo 2 tries all combinations and hence slow, but Algo 3 is is single-pass and hence is fast but cannot guarantee quality.

Algorithm 2: Epsilon-Approx(p, W, M, n, k)

```
// $k$ defines the order of algorithm, $k > 0$
// $M$ is the size of knapsack
begin
    $p_{max} \leftarrow 0$
    forall combinations $I$ of size $\leq k$ and weight $\leq M$ do
        $p_I \leftarrow \sum_{i \in I} p_i$
        $p_{max} \leftarrow max(p_{max}, p_I + LBound(I, p, W, M, n))$
    end
    return $p_{max}$
end
```

Algorithm 3: $LBound(I, p, W, M, n)$

```
1  begin
2      $s \leftarrow 0$
3      $t \leftarrow m - \sum_{i \in I} W_i$
4      for $i \leftarrow 1$ to $n$ do
5          if $i \notin I$ AND $W[i] \leq t$ then
6              $s \leftarrow s + p[i]$
7              $t \leftarrow t - W[i]$
8          end
9      end
10     return $s$
11 end
```

Traveling Salesman Problem

- Given a complete undirected graph *G* = (*V*, *E*) that has a cost *c(u, v)*, *c(u, v) ε* Z^+ for each edge *(u, v) ε E,* we have to find a *Hamiltonian* cycle of *G* with min. cost.
- Let *C(A)* denote the total cost of the edges in the subset A ⊆E.

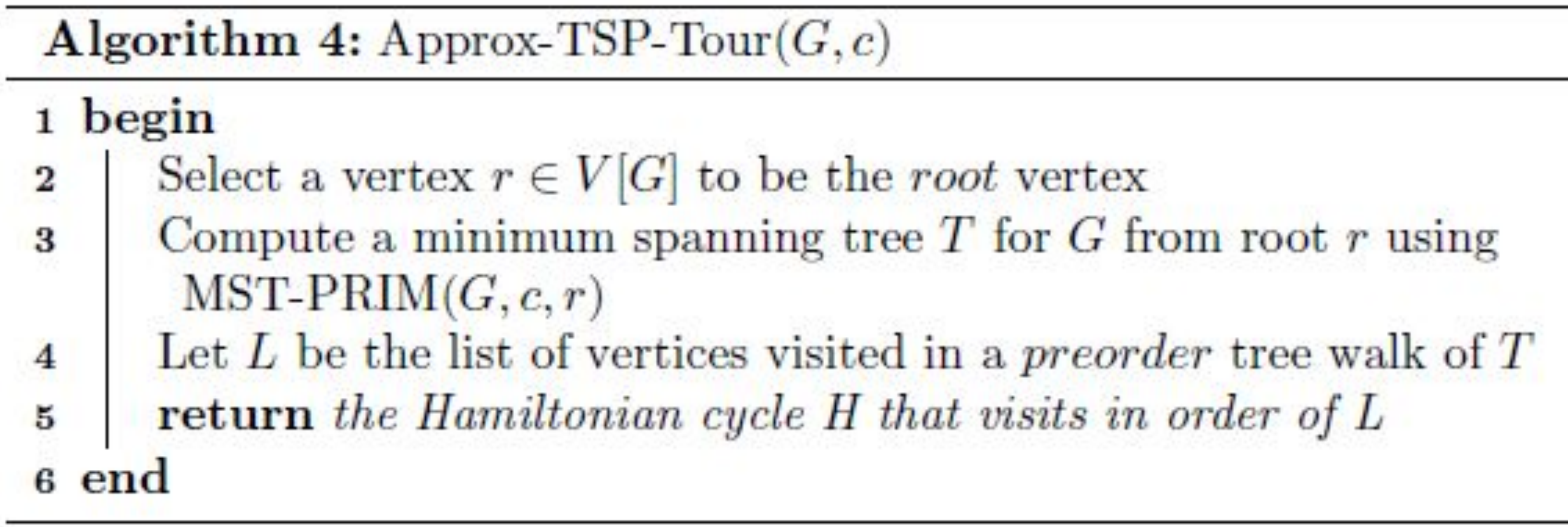
$$C(A) = \sum_{(u,v)\in A} c(u,v)$$

- If *c(u, w)* ≤ c(u, v) *+ c(v, w)* ∀ *u, v, w ε V,* we say the TSP satisfies *Triangle Inequality.*

- If the vertices are geometrical points on a plane, then *Eucledian* distance as cost obeys the triangle inequality

Approx Algo for TSP

- The general TSP doesn’t have a constant approx algo
- But the TSP with triangle inequality has a 2-approx algo



Algorithm 4: Approx-TSP-Tour(G, c)

```
1 begin
2     Select a vertex $r \in V[G]$ to be the *root* vertex
3     Compute a minimum spanning tree $T$ for $G$ from root $r$ using
        MST-PRIM($G, c, r$)
4     Let $L$ be the list of vertices visited in a *preorder* tree walk of $T$
5     return the Hamiltonian cycle $H$ that visits in order of $L$
6 end
```

Example of the relation between C(H) and C(preorder Walk)

Considering the graph obeys triangle inequality,

ra+re>ae
eb+ec>bc
ec+ed>cd

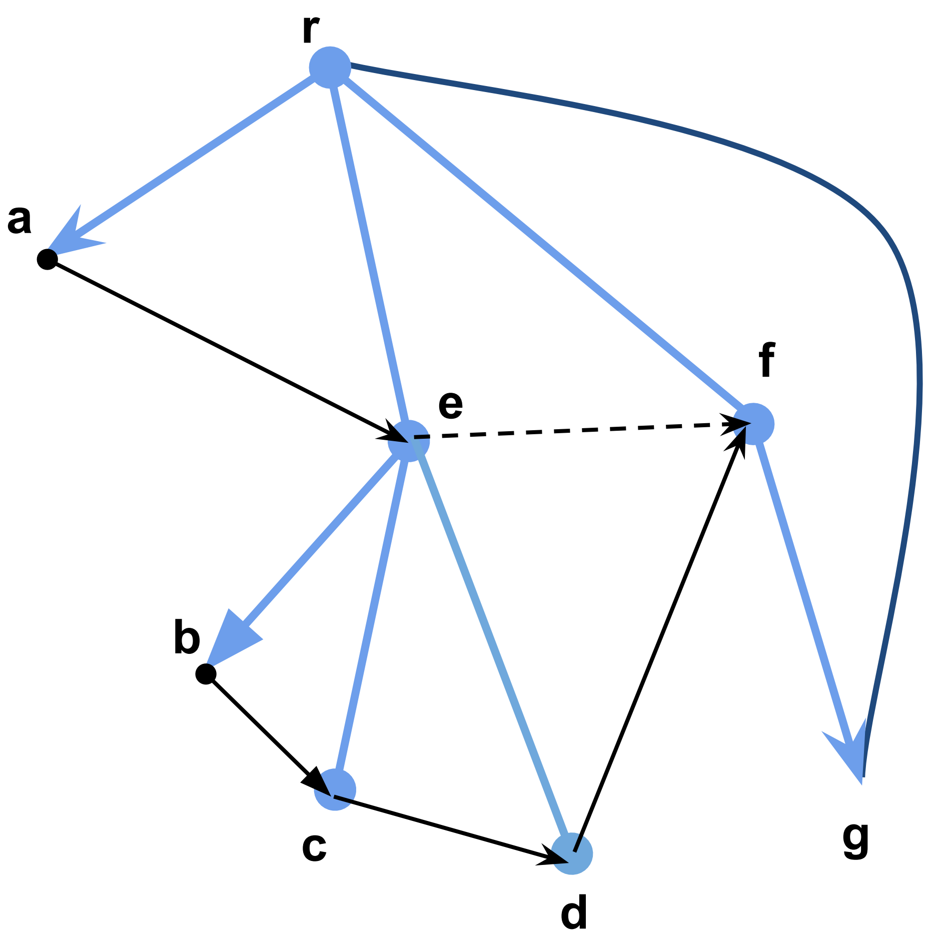
ed+ef>df
re+rf>ef

implies, ed+re+rf>df

fg+rf>rg

adding ra+eb+fg on both side,
We get the Hamiltonian cycle,

and on left we have every edge on the graph concerned traversed twiced.



Proof of 2-approx. for Δ TSP

- $C(MST) < C(H^*)$
- $C(W_{preorder}) = 2C(MST)$
- Let the walk $W_{preorder}$ be rarebecederfgr
- Delete the 2nd visit to vertices to get the Hamiltonian cycle H from the $W_{preorder}$
- So H = raebcdfgr
- By Δ inequality $C(H) \leq C(W_{preorder}) = 2C(MST) < 2C(H^*)$ (see the diagram)
- Hence $C(H)/C(H^*) \leq 2$