CMPSCI 611: Advanced Algorithms
Lecture 20: More TSP and Knapsack PTAS

Andrew McGregor
Outline

Metric TSP $3/2$ approximate
Metric Traveling Salesperson Problem

Input: Weighted complete graph $G = (V, E)$ with positive weights such that for edges $e = (u, v), e' = (v, w)$, and $e'' = (u, w)$

$$w_e + w_{e'} \geq w_{e''}$$
Metric Traveling Salesperson Problem

- **Input:** Weighted complete graph $G = (V, E)$ with positive weights such that for edges $e = (u, v)$, $e' = (v, w)$, and $e'' = (u, w)$

\[ w_e + w_{e'} \geq w_{e''} \]

- **Goal:** Find the tour (a path that visits every node exactly once and returns to starting point) of minimum total weight.
Eulerian Tours

Definition
A Eulerian tour is a path that traverses every edge of a graph exactly once and returns back to the initial vertex.
Eulerian Tours

Definition
A Eulerian tour is a path that traverses every edge of a graph exactly once and returns back to the initial vertex.

Lemma
A graph contains an Eulerian tour iff $G$ is connected and every vertex has even degree.
Metric TSP Approximation Algorithm

Algorithm

1. Compute minimum spanning tree $T_{mst}$ of $G$
Algorithm

1. Compute minimum spanning tree $T_{mst}$ of $G$
2. Let $D$ be the nodes in $T_{mst}$ that have odd degree
Metric TSP Approximation Algorithm

Algorithm

1. Compute minimum spanning tree \( T_{\text{mst}} \) of \( G \)
2. Let \( D \) be the nodes in \( T_{\text{mst}} \) that have odd degree
3. Find minimum cost perfect matching \( M \) on nodes of \( D \)

Theorem
The algorithm is a \( 3/2 \)-approximation and runs in polynomial time. The result was first proved by Christofides in 1976. In 2020, Karlin, Klein, and Gharan designed and analyzed a \( 3/2 - 10^{-36} \) approximation!
Metric TSP Approximation Algorithm

Algorithm

1. *Compute minimum spanning tree* $T_{mst}$ *of* $G$
2. *Let* $D$ *be the nodes in* $T_{mst}$ *that have odd degree*
3. *Find minimum cost perfect matching* $M$ *on nodes of* $D$
4. *Find Euler tour of* $T_{mst} + M$
Algorithm

1. Compute minimum spanning tree $T_{\text{mst}}$ of $G$
2. Let $D$ be the nodes in $T_{\text{mst}}$ that have odd degree
3. Find minimum cost perfect matching $M$ on nodes of $D$
4. Find Euler tour of $T_{\text{mst}} + M$
5. Transform into tour by short-cutting repeated vertices.

Theorem

The algorithm is a $3/2$-approximation and runs in polynomial time. The result was first proved by Christofides in 1976. In 2020, Karlin, Klein, and Gharan designed and analyzed a $3/2 - 10^{-36}$ approximation.
Metric TSP Approximation Algorithm

Algorithm
1. Compute minimum spanning tree $T_{mst}$ of $G$
2. Let $D$ be the nodes in $T_{mst}$ that have odd degree
3. Find minimum cost perfect matching $M$ on nodes of $D$
4. Find Euler tour of $T_{mst} + M$
5. Transform into tour by short-cutting repeated vertices.

Theorem
The algorithm is a $3/2$-approximation and runs in polynomial time.

The result was first proved by Christofides in 1976. In 2020, Karlin, Klein, and Gharan designed and analyzed a $3/2 - 10^{-36}$ approximation!
Analysis

Theorem

The algorithm is a $3/2$-approximation and runs in polynomial time.
Analysis

Theorem
The algorithm is a 3/2-approximation and runs in polynomial time.

Proof.
Analysis

Theorem

*The algorithm is a 3/2-approximation and runs in polynomial time.*

Proof.

- Cost of tour found is at most cost of Euler tour

\[
\text{cost(tour found) } \leq \text{cost(Euler tour) } = \text{cost}(T_{mst}) + \text{cost}(M)
\]
Analysis

Theorem

The algorithm is a 3/2-approximation and runs in polynomial time.

Proof.

- Cost of tour found is at most cost of Euler tour

\[ \text{cost(tour found)} \leq \text{cost(Euler tour)} = \text{cost}(T_{mst}) + \text{cost}(M) \]

- As before, \( \text{cost}(T_{mst}) \leq \text{cost(} \text{optimal tour}) \)
Analysis

Theorem

*The algorithm is a 3/2-approximation and runs in polynomial time.*

Proof.

- Cost of tour found is at most cost of Euler tour
  
  \[ \text{cost(tour found)} \leq \text{cost(Euler tour)} = \text{cost}(T_{\text{mst}}) + \text{cost}(M) \]

- As before, \( \text{cost}(T_{\text{mst}}) \leq \text{cost(}\text{optimal tour}) \)

- Cost of \( M \) is at most half cost of optimal tour
  
  \[ \text{cost}(M) \leq \frac{\text{cost(}\text{optimal tour})}{2} \]
Analysis

Theorem
The algorithm is a $3/2$-approximation and runs in polynomial time.

Proof.

- Cost of tour found is at most cost of Euler tour
  \[
  \text{cost(tour found)} \leq \text{cost(Euler tour)} = \text{cost}(T_{\text{mst}}) + \text{cost}(M)
  \]

- As before, \( \text{cost}(T_{\text{mst}}) \leq \text{cost(optimal tour)} \)

- Cost of \( M \) is at most half cost of optimal tour
  \[
  \text{cost}(M) \leq \frac{1}{2} \text{cost(optimal tour)}
  \]

Let \( D = \{d_1, \ldots, d_k\} \) be ordered according to optimal tour.

\[
\text{cost(optimal tour)} \geq w_{d_1, d_2} + w_{d_2, d_3} + \ldots + w_{d_k, d_1}
= (w_{d_1, d_2} + w_{d_3, d_4} + \ldots w_{d_{k-1}, d_k}) +
  (w_{d_2, d_3} + w_{d_4, d_5} + \ldots w_{d_k, d_1})
\]
PTAS for Knapsack Problem

General Knapsack Problem:

1. Input: A set of items numbered 1, 2, . . . , n, where each the i-th item has weight \( w_i \) and value \( v_i \). \( C \) is the capacity of your knapsack. (Assume each \( w_i \leq C \).)

2. Goal: Find a subset \( B \) of the items with maximum total value subject to \( \sum_{i \in B} w_i \leq C \).
Dynamic Programming Approach

- Let $v_{knap}(i, v)$ be the minimum weight required to achieve a value of at least $v$ using items 1, ..., $i$. 

  \[
  \begin{align*}
  v_{knap}(1, v) &= \begin{cases} 
  w_1 & \text{for } v \leq v_1 \\
  \infty & \text{for } v > v_1
  \end{cases} \\
  v_{knap}(i+1, v) &= \min\{v_{knap}(i, v), v_{knap}(i, v-v_i+1) + w_{i+1}\}
  \end{align*}
  \]

  where $v_{knap}(i, u) = 0$ if $u < 0$. 

  Let $V = \max_i(v_i)$ and note that the maximum value obtainable is $\leq V \cdot n$. 

  Dynamic programming solution has $O(n^2 V)$ complexity.
Dynamic Programming Approach

- Let $vknap(i, v)$ be the minimum weight required to achieve a value of at least $v$ using items 1, ..., $i$.

- Then

\[
vknap(1, v) = \begin{cases} 
  w_1 & \text{for } v \leq v_1 \\
  \infty & \text{for } v > v_1 
\end{cases}
\]

and

\[
vknap(i + 1, v) = \min\{vknap(i, v), vknap(i, v - v_{i+1}) + w_{i+1}\}
\]

where $vknap(i, u) = 0$ if $u < 0$. 
Dynamic Programming Approach

Let $v_{knap}(i, v)$ be the minimum weight required to achieve a value of at least $v$ using items $1, \ldots, i$.

Then

$$v_{knap}(1, v) = \begin{cases} w_1 & \text{for } v \leq v_1 \\ \infty & \text{for } v > v_1 \end{cases}$$

and

$$v_{knap}(i + 1, v) = \min\{v_{knap}(i, v), v_{knap}(i, v - v_{i+1}) + w_{i+1}\}$$

where $v_{knap}(i, u) = 0$ if $u < 0$.

Let $V = \max_i(v_i)$ and note that max value obtainable is $\leq Vn$.
Dynamic Programming Approach

Let \( vknap(i, v) \) be the minimum weight required to achieve a value of at least \( v \) using items \( 1, \ldots, i \).

Then

\[
vknap(1, v) = \begin{cases} 
w_1 & \text{for } v \leq v_1 \\
\infty & \text{for } v > v_1 
\end{cases}
\]

and

\[
vknap(i + 1, v) = \min\{vknap(i, v), vknap(i, v - v_{i+1}) + w_{i+1}\}
\]

where \( vknap(i, u) = 0 \) if \( u < 0 \).

Let \( V = \max_i(v_i) \) and note that max value obtainable is \( \leq Vn \).

Dynamic programming solution has \( O(n^2V) \) complexity
Approximation Algorithm

1. **New values:** Define $v_i'$ by setting $k$ lowest order bits of $v_i$ to zero.

Lemma

If $B'$ be set returned and let $B$ be the optimal set:

$$\sum_{i \in B} v_i \sum_{i \in B'} v_i' \leq 1 + \frac{n^2}{2^k V} - \frac{n^2}{2^k}$$

Proof.

1. Since $B'$ is optimal for new values:

$$\sum_{i \in B'} v_i \geq \sum_{i \in B'} v_i' \geq \sum_{i \in B} (v_i - 2^k) \geq \left(\sum_{i \in B} v_i\right) - 2^k n$$

2. Therefore

$$\sum_{i \in B} v_i \sum_{i \in B'} v_i' \leq \sum_{i \in B} v_i \left(\sum_{i \in B} v_i\right) - 2^k n = 1 + \frac{n^2}{2^k V} - \frac{n^2}{2^k}$$
Approximation Algorithm

1. **New values:** Define $v'_i$ by setting $k$ lowest order bits of $v_i$ to zero.
2. Run dynamic programming solution with the new values

**Lemma**

*If $B'$ be set returned and let $B$ be the optimal set:* 

$$\sum_{i \in B} \frac{v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V-n2^k}$$
Approximation Algorithm

1. New values: Define $v'_i$ by setting $k$ lowest order bits of $v_i$ to zero.
2. Run dynamic programming solution with the new values

Lemma

If $B'$ be set returned and let $B$ be the optimal set: 

$$\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V-n2^k}$$

Proof.

1. Since $B'$ is optimal for new values:

$$\sum_{i \in B'} v_i \leq \sum_{i \in B'} v'_i$$
Approximation Algorithm

1. New values: Define $v'_i$ by setting $k$ lowest order bits of $v_i$ to zero.
2. Run dynamic programming solution with the new values

Lemma

If $B'$ be set returned and let $B$ be the optimal set: \[ \frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V - n2^k} \]

Proof.

1. Since $B'$ is optimal for new values:

\[ \sum_{i \in B'} v_i \geq \sum_{i \in B'} v'_i \geq \sum_{i \in B} v'_i \]
Approximation Algorithm

1. New values: Define $v'_i$ by setting $k$ lowest order bits of $v_i$ to zero.
2. Run dynamic programming solution with the new values

Lemma

If $B'$ be set returned and let $B$ be the optimal set: $\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V-n2^k}$

Proof.

1. Since $B'$ is optimal for new values:

$$\sum_{i \in B'} v_i \geq \sum_{i \in B'} v'_i \geq \sum_{i \in B} v'_i \geq \sum_{i \in B} (v_i - 2^k) \geq \left(\sum_{i \in B} v_i\right) - 2^k n$$
**Approximation Algorithm**

1. **New values:** Define $v'_i$ by setting $k$ lowest order bits of $v_i$ to zero.
2. Run dynamic programming solution with the new values

**Lemma**

If $B'$ be set returned and let $B$ be the optimal set: \[rac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V - n2^k}\]

**Proof.**

1. Since $B'$ is optimal for new values:

\[
\sum_{i \in B'} v_i \geq \sum_{i \in B'} v'_i \geq \sum_{i \in B} v'_i \geq \sum_{i \in B} (v_i - 2^k) \geq \left(\sum_{i \in B} v_i\right) - 2^k n
\]

2. Therefore

\[
\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq \frac{\sum_{i \in B} v_i}{\left(\sum_{i \in B} v_i\right) - 2^k n}
\]
Approximation Algorithm

1. New values: Define \( v_i' \) by setting \( k \) lowest order bits of \( v_i \) to zero.
2. Run dynamic programming solution with the new values

Lemma

If \( B' \) be set returned and let \( B \) be the optimal set: \[
\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V - n2^k}
\]

Proof.

1. Since \( B' \) is optimal for new values:

\[
\sum_{i \in B'} v_i \geq \sum_{i \in B'} v_i' \geq \sum_{i \in B} v_i' \geq \sum_{i \in B} (v_i - 2^k) \geq \left( \sum_{i \in B} v_i \right) - 2^k n
\]

2. Therefore

\[
\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq \frac{\sum_{i \in B} v_i}{(\sum_{i \in B} v_i) - 2^k n} = 1 + \frac{2^k n}{(\sum_{i \in B} v_i) - 2^k n}
\]

10/13
Approximation Algorithm

1. **New values**: Define $v'_i$ by setting $k$ lowest order bits of $v_i$ to zero.
2. Run dynamic programming solution with the new values

**Lemma**

If $B'$ be set returned and let $B$ be the optimal set:

$$\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V - n2^k}$$

**Proof.**

1. Since $B'$ is optimal for new values:

$$\sum_{i \in B'} v_i \geq \sum_{i \in B'} v'_i \geq \sum_{i \in B} v'_i \geq \sum_{i \in B} (v_i - 2^k) \geq \left(\sum_{i \in B} v_i\right) - 2^k n$$

2. Therefore

$$\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq \frac{\sum_{i \in B} v_i}{(\sum_{i \in B} v_i) - 2^k n} = 1 + \frac{2^k n}{(\sum_{i \in B} v_i) - 2^k n} \leq 1 + \frac{2^k n}{V - 2^k n}$$
Finishing off Analysis

Claim

If \( k \leq \log(\epsilon V/(2n)) \) and \( \epsilon \leq 1 \) then \( 1 + \frac{2^k n}{V - 2^k n} \leq 1 + \epsilon \).
Finishing off Analysis

Claim

If $k \leq \log(\epsilon V/(2n))$ and $\epsilon \leq 1$ then $1 + \frac{2^kn}{V-2^kn} \leq 1 + \epsilon$.

1. Let $k = \lfloor \log(\epsilon V/(2n)) \rfloor$
Finishing off Analysis

Claim
If \( k \leq \log(\epsilon V/(2n)) \) and \( \epsilon \leq 1 \) then
\[
1 + \frac{2^k n}{\sqrt{V-2^k n}} \leq 1 + \epsilon.
\]

1. Let \( k = \lfloor \log(\epsilon V/(2n)) \rfloor \)
2. Solve for \( v' \) by solving for another set of values \( v'' \) where

\[
v''_i = v'_i / 2^k
\]
Finishing off Analysis

Claim
If $k \leq \log(\epsilon V/(2n))$ and $\epsilon \leq 1$ then $1 + \frac{2^k n}{V - 2^k n} \leq 1 + \epsilon$.

1. Let $k = \lfloor \log(\epsilon V/(2n)) \rfloor$
2. Solve for $v'$ by solving for another set of values $v''$ where
   
   $v''_i = v'_i / 2^k$

3. The maximum value for $v''$ satisfies:
   
   $\max v''_i \leq V / 2^k \leq 2V / (\epsilon V/(2n)) = 4n/\epsilon$

   so the run time is $O(n^3/\epsilon)$
Summary of Approximation Algorithms

- **Algorithms:**
  - 2-approximation for vertex cover
  - 2-approximation for max-cut
  - $3/2$-approximation for metric traveling salesman
  - $O(\log n)$-approximation for weighted set-cover
  - FPTAS for knapsack
Summary of Approximation Algorithms

- Algorithms:
  - 2-approximation for vertex cover
  - 2-approximation for max-cut
  - $3/2$-approximation for metric traveling salesman
  - $O(\log n)$-approximation for weighted set-cover
  - FPTAS for knapsack

- A poly-time reduction may not be “approximation preserving”
Summary of Approximation Algorithms

- **Algorithms:**
  - 2-approximation for vertex cover
  - 2-approximation for max-cut
  - 3/2-approximation for metric traveling salesman
  - \( O(\log n) \)-approximation for weighted set-cover
  - FPTAS for knapsack

- A poly-time reduction may not be “approximation preserving”
- For a reference of what approximation factors are known check out:
  - [http://www.csc.kth.se/~viggo/wwwcompendium/](http://www.csc.kth.se/~viggo/wwwcompendium/)
Alternative Approaches to NP-hard problems

- Restrict the input:
  - Assuming input graph is acyclic, of bounded degree, or planar
  - Solving metric TSP where the points are in Euclidean space
- Assume a probability distribution over input: *Average case analysis*
- Assume all integers in the input are polynomial in the input size...
Alternative Approaches to NP-hard problems

- Restrict the input:
  - Assuming input graph is acyclic, of bounded degree, or planar
  - Solving metric TSP where the points are in Euclidean space
- Assume a probability distribution over input: *Average case analysis*
- Assume all integers in the input are polynomial in the input size...

**Definition**
An algorithm runs in *pseudo-polynomial time* if the running time is polynomial in the input size and any integer in the input.
Alternative Approaches to NP-hard problems

- Restrict the input:
  - Assuming input graph is acyclic, of bounded degree, or planar
  - Solving metric TSP where the points are in Euclidean space
- Assume a probability distribution over input: *Average case analysis*
- Assume all integers in the input are polynomial in the input size...

**Definition**
An algorithm runs in *pseudo-polynomial time* if the running time is polynomial in the input size and any integer in the input.

**Definition**
A problem is *strongly NP-complete* if it remains NP-complete even when all integers in an input of length $n$ are polynomial in $n$