GEMS OF TCS

APPROXIMATION ALGORITHMS

Sasha Golovnev

January 28, 2020
APPROXIMATION ALGORITHMS

- Optimal exact solution \( \text{OPT} \) (ex: shortest TSP cycle)
Approximation Algorithms

- Optimal exact solution \( \text{OPT} \) (ex: shortest TSP cycle)
- \( \text{OPT} \) is too hard to find (ex: \( \text{NP} \)-hard)
APPROXIMATION ALGORITHMS

• Optimal exact solution \( \text{OPT} \) (ex: shortest TSP cycle)

• \( \text{OPT} \) is too hard to find (ex: \( \textbf{NP} \)-hard)

• A \( k \)-approximation algorithm finds a solution \( \leq k \times \text{OPT} \)
Approximation Algorithms

- Optimal exact solution $\text{OPT}$ (ex: shortest TSP cycle)

- $\text{OPT}$ is too hard to find (ex: $\text{NP}$-hard)

- A $k$-approximation algorithm finds a solution $\leq k \times \text{OPT}$

- Possibly efficiently! (ex: poly time)
Approximation Algorithms

- Optimal exact solution $\text{OPT}$ (ex: shortest TSP cycle)
- $\text{OPT}$ is too hard to find (ex: $\text{NP}$-hard)
- A $k$-approximation algorithm finds a solution $\leq k \times \text{OPT}$
- Possibly efficiently! (ex: poly time)
- When do we use approximation algorithms?
**Matchings**

- A *Matching* in a graph is a set of edges without common vertices
**Matchings**

- A *Matching* in a graph is a set of edges without common vertices.
- A *Maximal Matching* is a matching which cannot be extended to a larger matching.
**Matchings**

- A *Matching* in a graph is a set of edges without common vertices

- A *Maximal Matching* is a matching which cannot be extended to a larger matching

- A *Maximum Matching* is a matching of the largest size
Matchings. Examples

Maximal matching

not Maximum matching
Matchings. Examples

Maximum matchings
## Job Assignment

<table>
<thead>
<tr>
<th></th>
<th>Alice</th>
<th>Ben</th>
<th>Chris</th>
<th>Diana</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Programmer</td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Librarian</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professor</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>
JOB ASSIGNMENT

jobs: adm, prog, libr, prof
people: A, B, C, D

jobs → people mapping:
adm → A
prog → B
libr → C
prof → D
Perfect matching - Matching that covers all vertices
# Room Assignment

<table>
<thead>
<tr>
<th></th>
<th>R# 1</th>
<th>R# 2</th>
<th>R# 3</th>
<th>R# 4</th>
<th>R# 5</th>
<th>R# 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aaron</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bianca</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dana</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Emma</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Francis</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Algorithms**

**Maximal Matching**

Can be found in polynomial time by a greedy algorithm
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximal Matching</td>
<td>Can be found in polynomial time by a greedy algorithm</td>
</tr>
<tr>
<td>Maximum Matching</td>
<td>Can be found in polynomial time by the blossom algorithm</td>
</tr>
</tbody>
</table>
# Algorithms

<table>
<thead>
<tr>
<th><strong>Maximal Matching</strong></th>
<th>Can be found in polynomial time by a greedy algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maximum Matching</strong></td>
<td>Can be found in polynomial time by the blossom algorithm</td>
</tr>
<tr>
<td><strong>Minimum Weight Perfect Matching</strong></td>
<td>Can be found in polynomial time by Edmonds’ algorithm</td>
</tr>
</tbody>
</table>
Vertex Cover
**Vertex Covers**

- A *Vertex Cover* of a graph $G$ is a set of vertices $C$ such that every edge of $G$ is connected to some vertex in $C$. 
**Vertex Covers**

- A *Vertex Cover* of a graph $G$ is a set of vertices $C$ such that every edge of $G$ is connected to some vertex in $C$.

- A *Minimal Vertex Cover* is a vertex cover which does not contain other vertex covers.
**Vertex Covers**

- A *Vertex Cover* of a graph $G$ is a set of vertices $C$ such that every edge of $G$ is connected to some vertex in $C$.

- A *Minimal Vertex Cover* is a vertex cover which does not contain other vertex covers.

- A *Minimum Vertex Cover* is a vertex cover of the smallest size.
VERTEX COVERS: EXAMPLES
VERTEX COVERS: EXAMPLES

VC
not minimal VC

Diagram showing a graph with vertex covers marked and a note indicating a non-minimal vertex cover.
VERTEX COVERS: EXAMPLES
VENUE COVERS: EXAMPLES

minimal VC
not minimum VC
VERTEX COVERS: EXAMPLES

minimum VC
ANTIVIRUS SYSTEM
### Minimal Vertex Cover

Can be found in polynomial time by a greedy algorithm.
# Algorithms

<table>
<thead>
<tr>
<th>Minimal Vertex Cover</th>
<th>Can be found in polynomial time by a greedy algorithm</th>
</tr>
</thead>
</table>

| Minimum Vertex Cover | Is **NP**-hard. We only know exponential-time algorithms |
APPROXIMATION ALGORITHM

• $M \leftarrow$ maximal matching in $G$
Approximation Algorithm

- $M \leftarrow$ maximal matching in $G$

- return all vertices in $M$

1. It runs in poly time
2. It's 2-approximate
EQUIVALENT ALGORITHM

• $C \leftarrow \emptyset$
**Equivalent Algorithm**

- $C \leftarrow \emptyset$

- while $E \neq \emptyset$

\[ G = (V, E) \]
**Equivalent Algorithm**

- $C \leftarrow \emptyset$

- while $E \neq \emptyset$
  - $\{u, v\} \leftarrow$ any edge from $E$
 Equivalent Algorithm

- $C \leftarrow \emptyset$

- while $E \neq \emptyset$
  
  - $\{u, v\} \leftarrow$ any edge from $E$
  
  - add $u, v$ to $C$
**Equivalent Algorithm**

- $C \leftarrow \emptyset$

- while $E \neq \emptyset$
  - $\{u, v\} \leftarrow$ any edge from $E$
  - add $u, v$ to $C$
  - delete from $E$ all edges incident to $u$ or $v$
- return $C$

*Runs in poly time*
Lemma

This algorithm runs in polynomial time and is 2-approximate: it returns a vertex cover that is at most twice larger then a minimum vertex cover.

OPT is the size of minimum vertex
we select ≤ 2OPT vertices
Matching C

Minimum VC has size OPT

\[ \text{OPT} \geq \frac{1}{2} \text{# vertices in matching} \]

Our output = \# vertices matching \leq 2 \text{OPT}
1. Runs in poly-time
2. Outputs some vertex cover
3. Its VC ≤ 2. Minimum VC

This edge could be added to matching
Final Remarks

• The analysis is tight: there are graphs with matchings twice larger than vertex covers
Final Remarks

• The analysis is tight: there are graphs with matchings twice larger than vertex covers.

• No 1.99-approximation algorithm is known.

Under reasonable conj, there is no better approx alg in poly time.
Break

Matchings:

Vertex covers:
Traveling Salesman
If $P \neq NP$, then there is no $k$-approximation algorithm for the general version of TSP for any constant $k$. 
If $P \neq NP$, then there is no $k$-approximation algorithm for the general version of TSP for any constant $k$

- **Euclidean TSP**: $w(u, v) = w(v, u)$ and $w(u, v) \leq w(u, z) + w(z, v)$
**APPROXIMATION**

- If $P \neq NP$, then there is no $k$-approximation algorithm for the general version of TSP for any constant $k$

- **Euclidean TSP:** $w(u, v) = w(v, u)$ and $w(u, v) \leq w(u, z) + w(z, v)$

- We will design a 2-approximation algorithm: it quickly finds a cycle that is at most twice longer than an optimal one
A **tree** is a connected graph without cycles
**Definition**

- A **tree** is a connected graph without cycles.
- A **tree** is a connected graph on \( n \) vertices with \( n - 1 \) edges.
**Definition**

- A **tree** is a connected graph without cycles.
- A **tree** is a connected graph on $n$ vertices with $n-1$ edges.
- A **Spanning Tree** of a graph $G$ is a subgraph of $G$ that (i) is a tree and (ii) contains all vertices of $G$. 
DEFINITION

- A **tree** is a connected graph without cycles
- A **tree** is a connected graph on $n$ vertices with $n - 1$ edges
- A **Spanning Tree** of a graph $G$ is a subgraph of $G$ that (i) is a tree and (ii) contains all vertices of $G$
- A **Minimum Spanning Tree** (MST) of a weighted graph $G$ is a spanning tree of the smallest weight

Kruskal’s, Prim’s
MINIMUM SPANNING TREE: EXAMPLES
Minimum Spanning Tree: Examples

MST
**Minimum Spanning Trees**

**Lemma**

Let $G$ be an undirected graph with non-negative edge weights. Then

$$\text{MST}(G) \leq \text{TSP}(G).$$

---

TSP

- weight of this path
- min weight of a path
- min weight of a tree

= MST$(G)$

If I can find some cycle of length

$\leq 2\text{MST}$

$\leq 2\text{TSP}$

(path is a tree)
**Lemma**

Let $G$ be an undirected graph with non-negative edge weights. Then $\text{MST}(G) \leq \text{TSP}(G)$.  

**Proof**

By removing any edge from an optimum TSP cycle one gets a spanning tree of $G$.  

**Minimum Spanning Trees**
Eulerian Cycle

An Eulerian cycle (or path) visits every edge exactly once
**Eulerian Cycle**

<table>
<thead>
<tr>
<th><strong>Criteria</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A connected undirected graph contains an Eulerian cycle, if and only if the degree of every node is even</td>
<td></td>
</tr>
</tbody>
</table>
Example

Non-Eulerian graph
ALGORITHM

・ $T \leftarrow$ minimum spanning tree of $G$
ALGORITHM

- $T \leftarrow$ minimum spanning tree of $G$
- $D \leftarrow T$ with each edge doubled
Algorithm

- $T \leftarrow$ minimum spanning tree of $G$
- $D \leftarrow T$ with each edge doubled
- find an Eulerian cycle $C$ in $D$
Algorithm

- $T \leftarrow$ minimum spanning tree of $G$
- $D \leftarrow T$ with each edge doubled
- find an Eulerian cycle $C$ in $D$
- return a cycle that visits the nodes in the order of their first appearance in $C$
Euclidean TSP
Our solution

\[ \leq \text{MST} \cdot 2 \leq \text{TSP} \cdot 2 \]

Example

Recall Lemma

\[ \text{MST} \leq \text{TSP} \]

Metric TSP: all weights in the graph satisfy the triangle inequality
Our solution $\leq \text{MST} + \text{bunch edges} \leq 2 \cdot \text{TSP}$
EXAMPLE
Approximation Guarantee

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>The algorithm is 2-approximate.</td>
</tr>
</tbody>
</table>
Approximation Guarantee

Lemma
The algorithm is 2-approximate.

Proof
- The total length of the MST $T \leq \text{OPT} = \text{TSP}$
## Approximation Guarantee

**Lemma**
The algorithm is 2-approximate.

**Proof**
- The total length of the MST $T \leq \text{OPT}$
- We start with Eulerian cycle of length $2|T|$
**Approximation Guarantee**

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>The algorithm is 2-approximate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The total length of the MST $T \leq \text{OPT}$</td>
</tr>
<tr>
<td>• We start with Eulerian cycle of length $2</td>
</tr>
<tr>
<td>• Shortcuts can only decrease the total length</td>
</tr>
</tbody>
</table>
IMPROVEMENT
IMPROVEMENT
IMPROVEMENT
Instead of doubling all edges, increase by 1 the degrees of odd-degree vertices only.

Perfect matching odd-degree vertices of minimum weight.
IMPROVEMENT
IMPROVEMENT
Algorithm

• $T \leftarrow$ minimum spanning tree of $G$
Algorithm

- $T \leftarrow$ minimum spanning tree of $G$
- $M \leftarrow$ minimum weight perfect matching on odd-degree vertices of $T$
ALGORITHM

- $T \leftarrow$ minimum spanning tree of $G$
- $M \leftarrow$ minimum weight perfect matching on odd-degree vertices of $T$
- find an Eulerian cycle $C$ in $T \cup M$
**Algorithm**

- $T \leftarrow$ minimum spanning tree of $G$
- $M \leftarrow$ minimum weight perfect matching on odd-degree vertices of $T$
- find an Eulerian cycle $C$ in $T \cup M$
- return a cycle that visits the nodes in the order of their first appearance in $C$
Lemma

The algorithm is $3/2$-approximate.
**Approximation Guarantee**

**Lemma**
The algorithm is $3/2$-approximate.

**Proof**
- The total length of the MST $T \leq \text{OPT}$
Approximation Guarantee

Lemma

The algorithm is 3/2-approximate.

Proof

- The total length of the MST $T \leq \text{OPT}$
- The weight of the matching $M \leq \text{OPT} / 2$
**Approximation Guarantee**

| Lemma |
The algorithm is 3/2-approximate. |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof</td>
</tr>
<tr>
<td>• The total length of the MST ( T \leq OPT )</td>
</tr>
<tr>
<td>• The weight of the matching ( M \leq OPT /2 )</td>
</tr>
<tr>
<td>• Shortcuts can only decrease the total length</td>
</tr>
</tbody>
</table>
Final Remarks

• Euclidean TSP can be approximated to within any factor \((1 + \varepsilon)\) \(\frac{1}{1.00001}\)
Final Remarks

• Euclidean TSP can be approximated to within any factor \((1 + \varepsilon)\)

• The currently best known approximation algorithm for TSP with triangle inequality has approximation factor of \(3/2 - 10^{-36}\) (July 2020)