BBM402-Lecture 11: More Approximation Algorithms

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs473/fa2016/lectures.html

Formal definition of approximation algorithm

An algorithm $\mathcal{A}$ for an optimization problem X is an α-approximation algorithm if the following conditions hold:

- for each instance I of X the algorithm $\mathcal{A}$ correctly outputs a valid solution to I
- $\mathcal{A}$ is a polynomial-time algorithm
- Letting $OPT(I)$ and $\mathcal{A}(I)$ denote the values of an optimum solution and the solution output by $\mathcal{A}$ on instances I, $OPT(I)/\mathcal{A}(I) \leq \alpha$ *and* $\mathcal{A}(I)/OPT(I) \leq \alpha$. Alternatively:
  - If X is a minimization problem: $\mathcal{A}(I)/OPT(I) \leq \alpha$
  - If X is a maximization problem: $OPT(I)/\mathcal{A}(I) \leq \alpha$

Definition ensures that $\alpha \geq 1$

To be formal we need to say $\alpha(n)$ where $n = |I|$ since in some cases the *approximation ratio* depends on the size of the instance.

Formal definition of approximation algorithm

Unfortunately notation is not consistently used. Some times people use the following convention:

- If X is a minimization problem then $\mathcal{A}(I)/OPT(I) \leq \alpha$ and here $\alpha \geq 1$.
- If X is a maximization problem then $\mathcal{A}(I)/OPT(I) \geq \alpha$ and here $\alpha \leq 1$.

Usually clear from the context.

Part I

Approximation for Load Balancing

Load Balancing

Given n jobs $J_1, J_2, \ldots, J_n$ with sizes $s_1, s_2, \ldots, s_n$ and m identical machines $M_1, \ldots, M_m$ assign jobs to machines to minimize maximum load (also called makespan).

Problem sometimes referred to as multiprocessor scheduling.
Example: $\mathbf{3}$ machines and $\mathbf{8}$ jobs with sizes $\mathbf{4, 3, 1, 2, 5, 6, 9, 7}$.

Load Balancing

Given n jobs $J_1, J_2, \ldots, J_n$ with sizes $s_1, s_2, \ldots, s_n$ and m identical machines $M_1, \ldots, M_m$ assign jobs to machines to minimize maximum load (also called makespan).

Formally, an assignment is a mapping $f : \{1, 2, \ldots, n\} \to \{1, \ldots, m\}$.

- The load $\ell_f(j)$ of machine M_j under f is $\sum_{i:f(i)=j} s_i$
- Goal is to find f to minimize $\max_j \ell_f(j)$.

Greedy List Scheduling

List-Scheduling

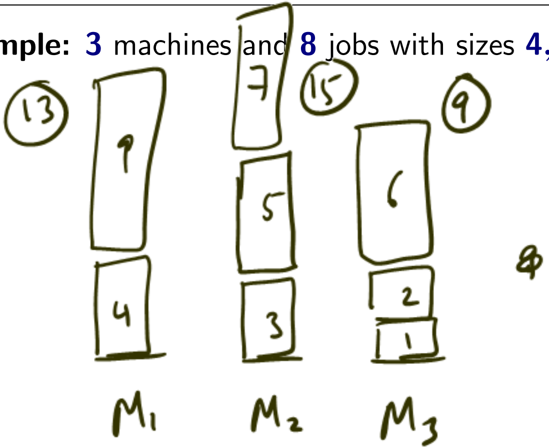
```
Let J_1, J_2, ..., J_n be an ordering of jobs
for i = 1 to n do
    Schedule job J_i on the currently least loaded machine
```

Greedy List Scheduling

List-Scheduling

```
Let J_1, J_2, ..., J_n be an ordering of jobs
for i = 1 to n do
    Schedule job J_i on the currently least loaded machine
```

Example: 3 machines and 8 jobs with sizes $4, 3, 1, 2, 5, 6, 9, 7$.



Example

Example: $\mathbf{3}$ machines and $\mathbf{8}$ jobs with sizes $\mathbf{4, 3, 1, 2, 5, 6, 9, 7}$.
Different list: $\mathbf{9, 7, 6, 5, 4, 3, 2, 1}$

Two lower bounds on OPT

OPT is the optimum load

- average load: $OPT \geq \sum_{i=1}^{n} s_i / m$. Why?
- maximum job size: $OPT \geq \max_{i=1}^{n} s_i$. Why?

Analysis of Greedy List Scheduling

Theorem

Let L be makespan of Greedy List Scheduling on a given instance. Then $L \leq 2(1 - 1/m)OPT$ where OPT is the optimum makespan for that instance.

Analysis of Greedy List Scheduling

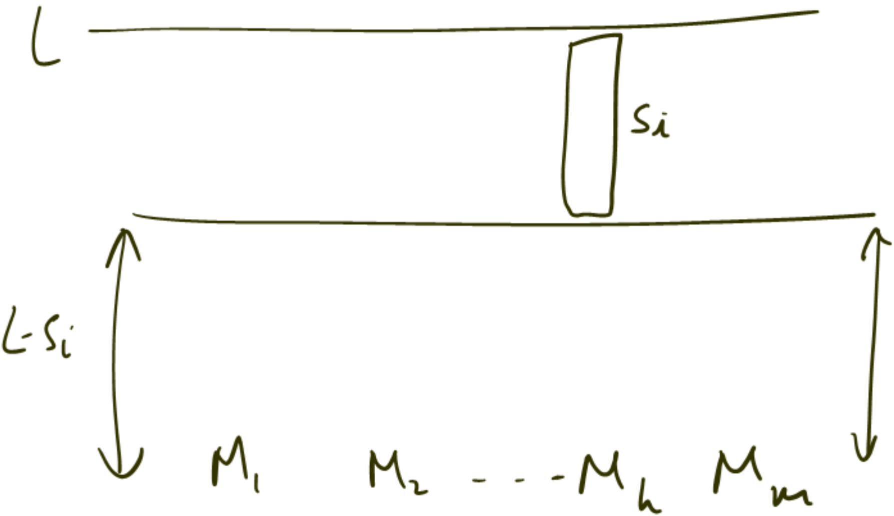
Theorem

Let L be makespan of Greedy List Scheduling on a given instance. Then $L \leq \cancel{2(1 - 1/m)} OPT$ where OPT is the optimum makespan for that instance. $\left(2 - \frac{1}{m}\right)$

- Let M_h be the machine which achieves the load L for Greedy List Scheduling.
- Let J_i be the job that was last scheduled on M_h.
- Why was J_i scheduled on M_h? It means that M_h was the least loaded machine when J_i was considered. Implies all machines had load at least $L - s_i$ at that time.

$$L - s_i \le \sum_{l=1}^{n} s_l \,/\, m \le OPT$$

$$s_i \le OPT$$



Analysis continued

Lemma

$L - s_i \leq (\sum_{\ell=1}^{i-1} s_\ell)/m.$

Proof.

Since all machines had load at least $L - s_i$ it means that $m(L - s_i) \leq \sum_{\ell=1}^{i-1} s_\ell$ and hence

$$L - s_i \leq (\sum_{\ell=1}^{i-1} s_\ell)/m.$$

□

Analysis continued

But then

$$
\begin{aligned}
L &\leq (\sum_{\ell=1}^{i-1} s_\ell)/m + s_i \\
&\leq (\sum_{\ell=1}^{n} s_\ell)/m + (1 - \frac{1}{m})s_i \\
&\leq OPT + (1 - \frac{1}{m})OPT \\
&\leq \cancel{2(1 - \frac{1}{m})}OPT.
\end{aligned}
$$

$(2 - \frac{1}{m})$ OPT

A Tight Example

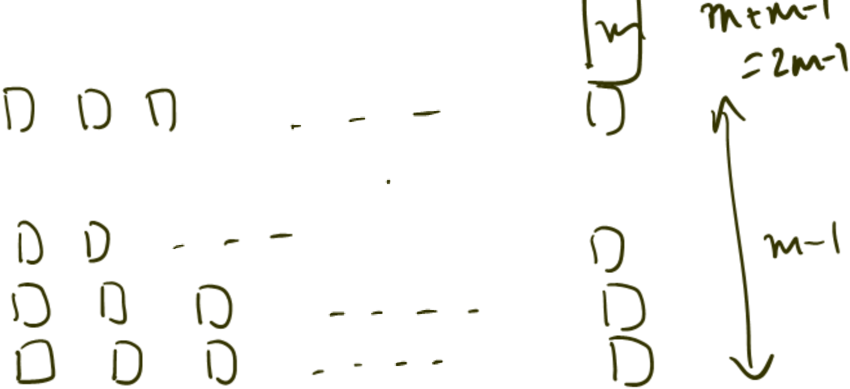
Question: Is the analysis of the algorithm tight? That is, are there instances where L is $\cancel{2(1-1/m)}OPT$?

$\left(2 - \frac{1}{m}\right) OPT$

A Tight Example

Question: Is the analysis of the algorithm tight? That is, are there instances where L is $2(1 - 1/m)OPT$?

Example: $m(m-1)$ jobs of size 1 and one big job of size m where m is number of machines.



A Tight Example

Question: Is the analysis of the algorithm tight? That is, are there instances where L is $2(1 - 1/m)OPT$?

Example: $m(m-1)$ jobs of size 1 and one big job of size m where m is number of machines.

- $OPT = m$. Why?
- If the list has large job at end the schedule created by Greedy is $m + m - 1 = 2m - 1$.

Ordering jobs from largest to smallest

Obvious heuristic: Order jobs in decreasing size order and then use Greedy.

Does it lead to an improved performance in the worst case? How much?

Ordering jobs from largest to smallest

Obvious heuristic: Order jobs in decreasing size order and then use Greedy.

Does it lead to an improved performance in the worst case? How much?

Theorem

Greedy List Scheduling with jobs sorted from largest to smallest gives a $\mathbf{4/3}$*-approximation and this is essentially tight.*

Analysis

Not so obvious.

If we only use average load and maximum job size as lower bounds on OPT then we cannot improve the bound of 2

Example: $m + 1$ jobs of size 1

- $OPT = 2$
- average load is $1 + 1/m$ and max job size is 1

Analysis

Not so obvious.

If we only use average load and maximum job size as lower bounds on OPT then we cannot improve the bound of 2

Example: $m + 1$ jobs of size 1

- $OPT = 2$
- average load is $1 + 1/m$ and max job size is 1

Need another lower bound

Another useful lower bound

Lemma

Suppose jobs are sorted, that is $s_1 \geq s_2 \geq \ldots \geq s_n$ *and* $n > m$ *then* $OPT \geq s_m + s_{m+1} \geq 2s_{m+1}$.

Another useful lower bound

Lemma

Suppose jobs are sorted, that is $s_1 \geq s_2 \geq \ldots \geq s_n$ *and* $n > m$ *then* $OPT \geq s_m + s_{m+1} \geq 2s_{m+1}$.

Proof.

Consider the first $m+1$ jobs $J_1, \ldots, J_{m+1}$. By pigeon hole principle two of these jobs on same machine. Load on that machine is at least the sum of the smallest two job sizes in the first $m+1$ jobs. $\square$

Proving a $3/2$ bound

Using the new lower bound we will prove a weaker upper bound of $3/2$ rather than the right bound of $4/3$.

As before let M_j be the machine achieving the makespan L and let J_i be the last job assigned to M_j. we have $L - s_i \leq \frac{1}{m}\sum_{\ell=1}^{i-1} s_\ell$. Now a more careful analysis.

- Case 1: If s_i is only job on M_j then $L \leq s_i \leq OPT$.
- Case 2: At least one more job on M_j before s_i.
  - We have seen that $L - s_i \leq OPT$.
  - **Claim:** $s_i \leq OPT/2$
  - Together, we have $L \leq OPT + s_i \leq 3OPT/2$.

Proof of Claim

Since M_j had a job before s_i we have $i > m$.

Hence $s_i \leq s_{m+1}$ becase jobs were sorted. Since $OPT \geq 2s_{m+1}$, we have $s_i \leq s_{m+1} \leq OPT/2$.

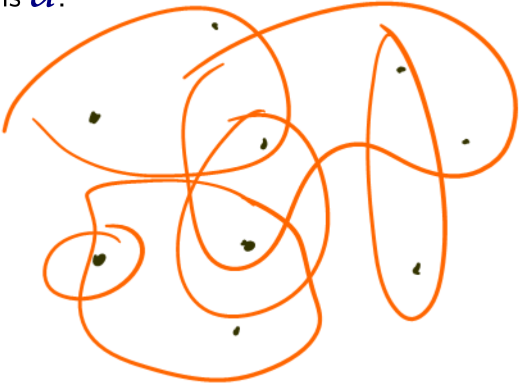
Part II

Approximation for Set Cover

Set Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ such that $\cup_i S_i = \mathcal{U}$.

Goal: Pick fewest number of subsets to cover all of $\mathcal{U}$ (equivalently, whose union is $\mathcal{U}$.



Set Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ such that $\cup_i S_i = \mathcal{U}$.

Goal: Pick fewest number of subsets to cover all of $\mathcal{U}$ (equivalently, whose union is $\mathcal{U}$.

Greedy$(\mathcal{U}, S_1, S_2, \ldots, S_m)$

```
    Uncovered = U
    While Uncovered ≠ ∅ do
        Pick set S_j that covers max number of uncovered elements
        Add S_j to solution
        Uncovered = Uncovered − S_j
    endWhile
    Output chosen sets
```

Analysis of Greedy

- Let k^* be minimum number of sets to cover $\mathcal{U}$. Let k be number of sets chosen by Greedy.
- Let α_i be number of new elements covered in iteration i.
- Let β_i be number of elements uncovered at end of iteration i. $\beta_0 = n$.

β_0 α_1 α_2

$\underline{n}$ $\beta_1 = n - \alpha_1$ $\beta_2 = \beta_1 - \alpha_2$

Analysis of Greedy

- Let k^* be minimum number of sets to cover $\mathcal{U}$. Let k be number of sets chosen by Greedy.
- Let α_i be number of new elements covered in iteration i.
- Let β_i be number of elements uncovered at end of iteration i. $\beta_0 = n$.

Lemma

$\alpha_i \geq \beta_{i-1}/k^*$.

Proof.

Let $\mathcal{U}_i$ be uncovered elements at start of iteration i. All these elements can be covered by some k^* sets since all of $\mathcal{U}$ can be covered by k^* sets. There exists one of those sets that covers at least $\mathcal{U}_i/k^*$ elements. Greedy picks the best set and hence covers at least that many elements. Note $\mathcal{U}_i = \beta_{i-1}$. □

Analysis of Greedy contd

Lemma

$\alpha_i \geq \beta_{i-1}/k^*$.

$$\beta_i = \beta_{i-1} - \alpha_i \leq \beta_{i-1} - \beta_{i-1}/k^* = (1 - 1/k^*)\beta_{i-1}.$$

Hence by induction,

$$\beta_i \leq \beta_0(1 - 1/k^*)^i = n(1 - 1/k^*)^i.$$

Thus, after $k = k^* \ln n$ iterations number number of uncovered elements is at most

$$n(1 - 1/k^*)^{k^* \ln n} \leq ne^{-\ln n} \leq 1.$$

Thus algorithm terminates in at most $k^* \ln n + 1$ iterations. Total number of sets chosen is number of iterations.

Analysis contd

Theorem

Greedy gives a $(\ln n + 1)$-approximation for Set Cover.

- Algorithm generalizes to weighted case easily. Pick sets in each iteration based on ratio of elements covered divided by weight. Analysis a bit harder but also gives a $(\ln n + 1)$-approximation.
- Can show a tighter bound of $(\ln d + 1)$ where d is maximum set size.

Analysis contd

Theorem

Greedy gives a $(\ln n + 1)$*-approximation for Set Cover.*

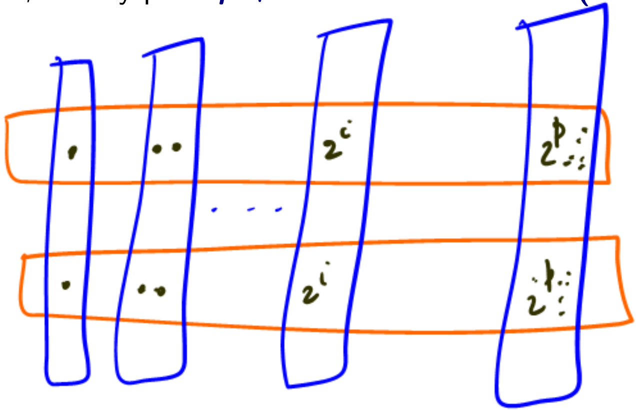
- Algorithm generalizes to weighted case easily. Pick sets in each iteration based on ratio of elements covered divided by weight. Analysis a bit harder but also gives a $(\ln n + 1)$-approximation.
- Can show a tighter bound of $(\ln d + 1)$ where d is maximum set size.

Theorem

Unless $P = NP$ *no* $(\ln n + \epsilon)$*-approximation for Set Cover.*

A bad example for Greedy

$n = 2(1 + 2 + 2^2 + 2^p) = 2(2^{p+1} - 1)$, $m = 2 + 2(p+1)$, $OPT = 2$, Greedy picks $p+1$ and hence ratio is $\Omega(\ln n)$.



Advantage of Greedy

Greedy is a simple algorithm. In several scenarios the set system is *implicit* and exponentially large in n. Nevertheless, the Greedy algorithm can be implemented efficiently if there is an oracle that each step picks the best set efficiently.

Max k-Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ and integer k.

Goal: Pick k subsets to *maximize* number of covered elements.

Max k-Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ and integer k.

Goal: Pick k subsets to *maximize* number of covered elements.

Greedy($\mathcal{U}, S_1, S_2, \ldots, S_m, k$)
```
    Uncovered = U
    for i = 1 to k do
        Pick set S_j that covers max number of uncovered elements
        Add S_j to solution
        Uncovered = Uncovered − S_j
    endWhile
    Output chosen k sets
```

Analysis

Similar to previous analysis.

- Let OPT be max number of covered elements to cover $\mathcal{U}$.
- Let α_i be number of new elements covered in iteration i.
- Let γ_i be number of elements covered by greedy after i iterations.
- Let $\beta_i = OPT - \gamma_i$. Define $\beta_0 = OPT$.

Analysis

Similar to previous analysis.

- Let OPT be max number of covered elements to cover $\mathcal{U}$.
- Let α_i be number of new elements covered in iteration i.
- Let γ_i be number of elements covered by greedy after i iterations.
- Let $\beta_i = OPT - \gamma_i$. Define $\beta_0 = OPT$.

Lemma

$\alpha_i \geq \beta_{i-1}/k$.

Analysis contd

Lemma

$\alpha_i \geq \beta_{i-1}/k^*$.

$$\beta_i = \beta_{i-1} - \alpha_i \leq \beta_{i-1} - \beta_{i-1}/k = (1 - 1/k)\beta_{i-1}.$$

Hence by induction,

$$\beta_i \leq \beta_0 (1 - 1/k)^i = OPT(1 - 1/k)^i.$$

Thus, after k iterations,

$$\beta_k \leq OPT(1 - 1/k)^k \leq OPT/e.$$

Thus $\gamma_k = OPT - \beta_k \geq (1 - 1/e)OPT$.

Analysis contd

Theorem

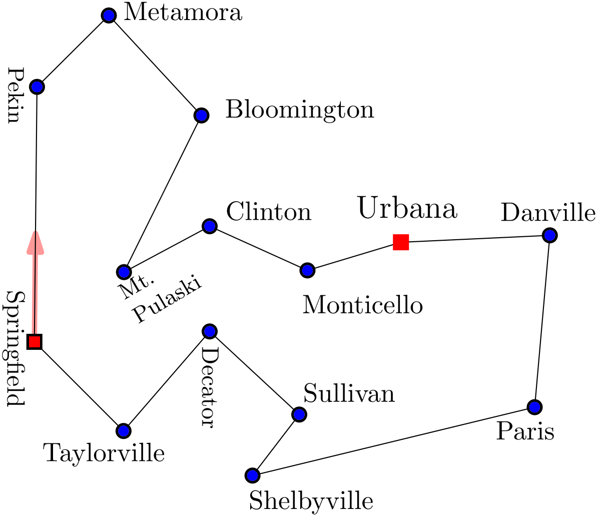
Greedy gives a $(1 - 1/e)$-approximation for Max k-Coverage.

Above theorem generalizes to submodular function maximization and has *many* applications.

Theorem (Feige 1998)

Unless $P = NP$ there is no $(1 - 1/e - \epsilon)$-approximation for Max k-Coverage for any fixed $\epsilon > 0$.

Lincoln's Circuit Court Tour



Traveling Salesman/Salesperson Problem (TSP)

Perhaps the most famous discrete optimization problem

Input: A graph $G = (V, E)$ with edge costs $c : E \rightarrow \mathbb{R}_+$.
Goal: Find a Hamiltonian Cycle of minimum total edge cost

Graph can be undirected or directed. Problem differs substantially.
We will first focus on undirected graphs.

Traveling Salesman/Salesperson Problem (TSP)

Perhaps the most famous discrete optimization problem

Input: A graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}_+$.
Goal: Find a Hamiltonian Cycle of minimum total edge cost

Graph can be undirected or directed. Problem differs substantially. We will first focus on undirected graphs.

Assumption for simplicity: Graph $G = (V, E)$ is a complete graph. Can add missing edges with infinite cost to make graph complete.

Observation: Once graph is complete there is always a Hamiltonian cycle but only Hamiltonian cycles of finite cost are Hamiltonian cycles in the original graph.

Important Special Cases

Metric-TSP: $G = (V, E)$ is a complete graph and c defines a metric space. $c(u, v) = c(v, u)$ for all u, v and $c(u, w) \leq c(u, v) + c(v, w)$ for all u, v, w.

Geometric-TSP: V is a set of points in some Euclidean d-dimensional space $\mathbb{R}^d$ and the distance between points is defined by some norm such as standard Euclidean distance, L_1/Manhatta distance etc.

Another interpretation of Metric-TSP: Given $G = (V, E)$ with edges costs c, find a tour of minimum cost that visits all vertices but can visit a vertex more than once.

Inapproximability of TSP

Observation: In the general setting TSP does not admit any bounded approximation.

- Finding or even deciding whether a graph $G = (V, E)$ has Hamiltonian Cycle is NP-Hard
- Alternatively, suppose $G = (V, E)$ is a simple graph that we complete with infinite cost edges. If G has a Hamilton Cycle then there is a TSP tour of cost n else it is cost ∞.

Metric-TSP

Metric-TSP is simpler and perhaps a more natural problem in some settings.

Theorem

Metric-TSP is NP-Hard.

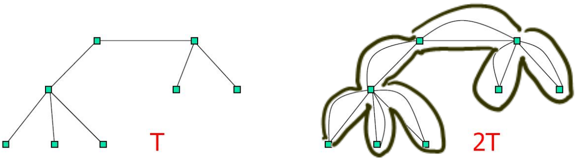
Proof.

Given $G = (V, E)$ we create a new complete graph $G' = (V, E')$ with the following costs. If $e \in E$ cost $c(e) = 1$. If $e \in E' - E$ cost $c(e) = 2$. Easy to verify that c satisfies metric properties. Moreover, G' has TSP tour of cost n iff G has a Hamiltonian Cycle. □

Approximation for Metric-TSP

MST-Heuristic$(G = (V, E), c)$

Compute an minimum spanning tree (MST) T in G
Obtain an Eulerian graph $H = 2T$ by doubling edges of T
An Eulerian tour of H gives a tour of G
Obtain Hamiltonian cycle by shortcutting the tour



Analyzing MST-Heuristic

Lemma

Let $c(T) = \sum_{e \in T} c(e)$ *be cost of MST. We have* $c(T) \leq OPT$.

Analyzing MST-Heuristic

Lemma

Let $c(T) = \sum_{e \in T} c(e)$ *be cost of MST. We have* $c(T) \leq OPT$.

Proof.

A TSP tour is a connected subgraph of G and MST is the cheapest connected subgraph of G. $\square$

Analyzing MST-Heuristic

Lemma

Let $c(T) = \sum_{e \in T} c(e)$ *be cost of MST. We have* $c(T) \leq OPT$*.*

Proof.

A TSP tour is a connected subgraph of G and MST is the cheapest connected subgraph of G. $\square$

Theorem

MST-Heuristic gives a 2*-approximation for Metric-TSP.*

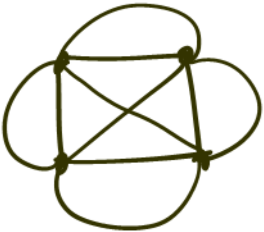
Proof.

Cost of tour is at most $2c(T)$ and hence MST-Heuristic gives a 2-approximation. $\square$

Background on Eulerian graphs

Definition

An *Euler tour* of an undirected multigraph $G = (V, E)$ is a closed walk that visits each edge exactly once. A graph is Eulerian if it has an Euler tour.



Background on Eulerian graphs

Definition

An *Euler tour* of an undirected multigraph $G = (V, E)$ is a closed walk that visits each edge exactly once. A graph is Eulerian if it has an Euler tour.

Theorem (Euler)

An undirected multigraph $G = (V, E)$ *is Eulerian iff* G *is connected and every vertex degree is even.*

Background on Eulerian graphs

Definition

An *Euler tour* of an undirected multigraph $G = (V, E)$ is a closed walk that visits each edge exactly once. A graph is Eulerian if it has an Euler tour.

Theorem (Euler)

An undirected multigraph $G = (V, E)$ *is Eulerian iff* G *is connected and every vertex degree is even.*

Theorem

A directed multigraph $G = (V, E)$ *is Eulerian iff* G *is weakly connected and for each vertex* v, $\mathit{indeg}(v) = \mathit{outdeg}(v)$.

Improved approximation for Metric-TSP

How can we improve the MST-heuristic?

Observation: Finding optimum TSP tour in G is same as finding minimum cost Eulerian subgraph of G (allowing duplicate copies of edges).

Improved approximation for Metric-TSP

How can we improve the MST-heuristic?

Observation: Finding optimum TSP tour in G is same as finding minimum cost Eulerian subgraph of G (allowing duplicate copies of edges).

Christofides-Heuristic$(G = (V, E), c)$

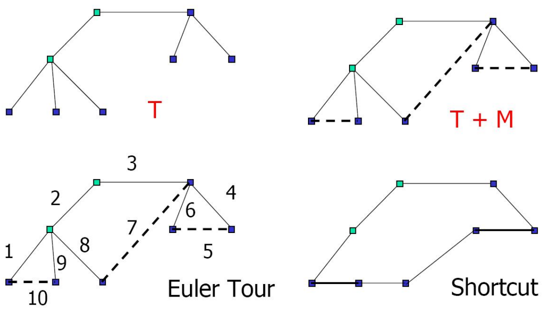
```
    Compute an minimum spanning tree (MST) T in G
    Add edges to T to make Eulerian graph H
    An Eulerian tour of H gives a tour of G
    Obtain Hamiltonian cycle by shortcutting the tour
```

How do we edges to make T Eulerian?

Christofides Heuristic: 3/2 approximation

Christofides-Heuristic$(G = (V, E), c)$

Compute an minimum spanning tree (MST) T in G
Let S be vertices of odd degree in T (Note: $|S|$ is even)
Find a minimum cost matching M on S in G
Add M to T to obtain Eulerian graph H
An Eulerian tour of H gives a tour of G
Obtain Hamiltonian cycle by shortcutting the tour



Analysis of Christofides Heuristic

Main lemma:

Lemma

$c(M) \leq OPT/2$.

Assuming lemma:

Theorem

Christofides heuristic returns a tour of cost at most $3OPT/2$.

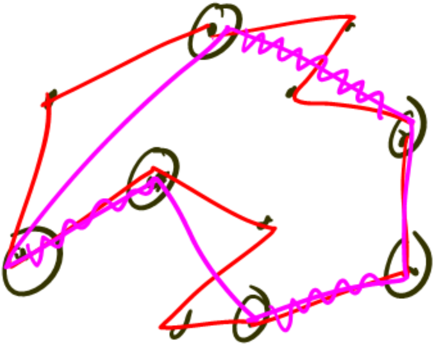
Proof.

$c(H) = c(T) + c(M) \leq OPT + OPT/2 \leq 3OPT/2$. Cost of tour is at most cost of H. □

Analysis of Christofides Heuristic

Lemma

Suppse $G = (V, E)$ *is a metric and* $S \subset V$ *be a subset of vertices. Then there is a TSP tour in* $G[S]$ *(the graph induced on* S*) of cost at most* OPT.



Analysis of Christofides Heuristic

Lemma

Suppse $G = (V, E)$ *is a metric and* $S \subset V$ *be a subset of vertices. Then there is a TSP tour in* $G[S]$ *(the graph induced on* S*) of cost at most* OPT.

Proof.

Let $C = v_1, v_2, \ldots, v_n, v_1$ be an optimum tour of cost OPT in G and let $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ where, without loss of generality $i_1 < i_2 \ldots < i_k$. Then consider the tour $C' = v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1}$ in $G[S]$. The cost of this tour is at most cost of C by shortcutting. $\square$

Proof of lemma for Christofides heuristic

Lemma

$c(M) \leq OPT/2$.

Recall that M is a matching on S the set of odd degree nodes in T.
Recall that $|S|$ is even.

Proof.

From previous lemma, there is tour of cost OPT for S in $G[S]$.
Wlog let this tour be $v_1, v_2, \ldots, v_{2k}, v_1$ where
$S = \{v_1, v_2, \ldots, v_{2k}\}$. Consider two matchings M_a and M_b where
$M_a = \{(v_1, v_2), (v_3, v_4), \ldots, (v_{2k-1}, v_{2k})$ and
$M_b = \{(v_2, v_3), (v_4, v_5), \ldots, (v_{2k}, v_1)$.
$M_a \cup M_b$ is set of edges of tour so $c(M_a) + c(M_b) \leq OPT$ and
hence one of them has cost less than $OPT/2$. $\square$

Other comments

Christofides heuristic has not been improved since 1976!
Major open problem in approximation algorithms.

For points in any fixed dimension d there is a polynomial-time approximation scheme. For any fixed $\epsilon > 0$ a tour of cost $(1 + \epsilon)OPT$ can be computed in polynomial time. [Arora 1996, Mitchell 1996].

Excellent practical code exists for solving large scale instances of TSP that arise in several applications. See Concorde TSP Solver by Applegate, Bixby, Chvatal, Cook.

Directed Graphs and Asymmetric TSP (ATSP)

Question: What about directed graphs?

Equivalent of Metric-TSP is Asymmetric-TSP (ATSP)

- Input is a complete directed graph $G = (V, E)$ with edge costs $c : E \rightarrow \mathbb{R}_+$.
- Edge costs are not necessarily symmetric. That is $c(u, v)$ can be different from $c(v, u)$
- Edge costs satisfy assymetric triangle inequality: $c(u, w) \leq c(u, v) + c(v, w)$ for all $u, v, w \in V$.

Directed Graphs and Asymmetric TSP (ATSP)

Question: What about directed graphs?

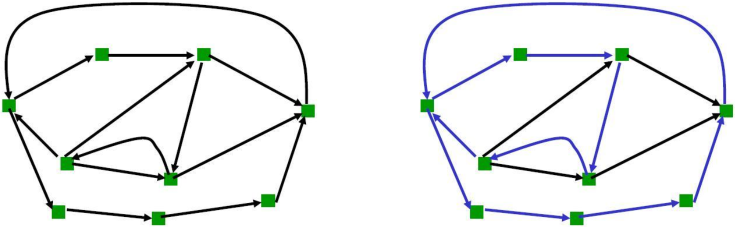
Equivalent of Metric-TSP is Asymmetric-TSP (ATSP)

- Input is a complete directed graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}_+$.
- Edge costs are not necessarily symmetric. That is $c(u, v)$ can be different from $c(v, u)$
- Edge costs satisfy assymetric triangle inequality: $c(u, w) \leq c(u, v) + c(v, w)$ for all $u, v, w \in V$.

Alternate interpretation: given directed graph $G = (V, E)$ find a closed walk that visits all vertices (can visit a vertex more than once).

ATSP

Alternate interpretation: given directed graph $\boldsymbol{G = (V, E)}$ find a closed walk that visits all vertices (can visit a vertex more than once).



Same as finding a minimum cost connected Eulerian subgraph of $\boldsymbol{G}$.

Approximation for ATSP

Harder than Metric-TSP

- Simple $\log_2 n$ approximation from 1980.
- Improved to $O(\log n / \log \log n)$-approximation in 2010.
- Further improved to $O((\log \log n)^c)$-approximation in 2015.

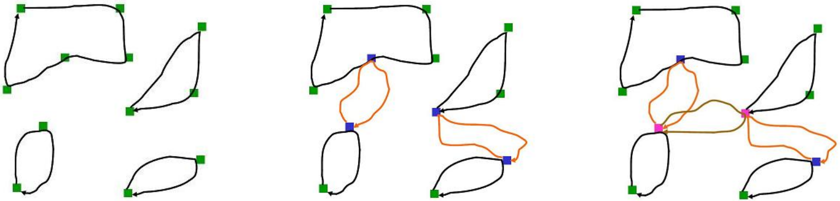
Believed that a constant factor approximation exists via a natural LP relaxation.

The O(log n) Approximation

Recall that a cycle cover is a collection of node disjoint cycles that contain all nodes.

CycleShrinkingAlgorithm($G(V, A), c : A \to \mathcal{R}^+$):
    If $|V| = 1$ output the trivial cycle consisting of V
    Find a *minimum cost cycle cover* with cycles $C_1, \ldots, C_k$
    From each C_i pick an arbitrary proxy node v_i
    Let $S = \{v_1, v_2, \ldots, v_k\}$
    Recursively solve problem on $G[S]$ to obtain a solution C
    $C' = C \cup C_1 \cup C_2 \ldots C_k$ is a Eulerian graph.
    Shortcut C' to obtain a cycle on V and output C'.

Illustration



Analysis

Lemma

Cost of a cycle cover is at most OPT.

Analysis

Lemma

Cost of a cycle cover is at most OPT.

Lemma

Suppse $G = (V, E)$ is a directed graph with edge costs that satisfies asymmetric triangle inequality and $S \subset V$ be a subset of vertices. Then there is a TSP tour in $G[S]$ (the graph induced on S) of cost at most OPT.

Lemma

The number of vertices shrinks by half in each iteration and hence total of at most $\lceil \log n \rceil$ cycle covers.

Hence total cost of all cycle covers is at most $\lceil \log n \rceil \cdot OPT$.