Lecture 26: Dealing with NP-completeness: Approximation Algorithms
Some other NP-complete examples you should know

Hamiltonian-Cycle: Given a directed graph $G = (V, E)$. Is there a cycle in $G$ that visits each vertex in $V$ exactly once?

Hamiltonian-Path: Given a directed graph $G = (V, E)$. Is there a path $p$ in $G$ of length $n - 1$ that visits each vertex in $V$ exactly once?

Same problems are also NP-complete for undirected graphs

Note: If we asked about visiting each edge exactly once instead of each vertex, the corresponding problems are called Euler Tour, Eulerian-Path and are polynomial-time solvable.
Travelling-Salesperson Problem (TSP)

Travelling-Salesperson Problem (TSP):

**Given:** a set of $n$ cities $v_1, \ldots, v_n$ and distance function $d$ that gives distance $d(v_i, v_j)$ between each pair of cities

Find the shortest tour that visits all $n$ cities.

DecisionTSP:

**Given:** a set of $n$ cities $v_1, \ldots, v_n$ and distance function $d$ that gives distance $d(v_i, v_j)$ between each pair of cities and a distance $D$

Is there a tour of total length at most $D$ that visits all $n$ cities?
NP-complete problems we’ve discussed

3SAT → Independent-Set → Clique

↓

Vertex-Cover → 01-Programming → Integer-Programming

↓

Set-Cover

→ 3Color

→ Subset-Sum

→ Hamiltonian-Cycle → DecisionTSP

→ Hamiltonian-Path
Some intermediate problems

Problems reducible to \(\textbf{NP}\) problems not known to be polytime:

Basis for the security of current cryptography:

- **Factoring:** Given an integer \(N\) in binary, find its prime factorization.
- **Discrete logarithm:** Given prime \(p\) in binary, and \(g\) and \(x\) modulo \(p\). Find \(y\) such that \(x \equiv g^y (\text{mod } p)\) if it exists.

Best algorithms known are \(2^{\tilde{O}(n^{1/3})}\) time.

Other famous ones:

- **Graph Isomorphism:** Given graphs \(G\) and \(H\), can they be relabelled to be the same? Best algorithm now \(n^{\Theta(\log^2 n)}\) (recently improved from \(2^{\tilde{O}(n^{1/3})}\) time.

- **Nash equilibrium:** Given a multiplayer game, find randomized strategies for each player so that no player could do better by deviating.
What to do if the problem you want to solve is NP-hard

1st thing to try:

• You might have phrased your problem too generally
  • e.g., In practice, the graphs that actually arise are far from arbitrary
    • Maybe they have some special characteristic that allows you to solve the problem in your special case
      • For example the Independent-Set problem is easy on “interval graphs”
        • Exactly the case for the Interval Scheduling problem!
  • Search the literature to see if special cases already solved
What to do if the problem you want to solve is NP-hard

2nd thing to try if your problem is a minimization or maximization problem

• Try to find a polynomial-time worst-case approximation algorithm
  • For a minimization problem
    • Find a solution with value $\leq K$ times the optimum
  • For a maximization problem
    • Find a solution with value $\geq 1/K$ times the optimum

Want $K$ to be as close to 1 as possible.
Greedy Approximation for Vertex-Cover

On input $G = (V, E)$

$W \leftarrow \emptyset$

$E' \leftarrow E$

while $E' \neq \emptyset$

\[\text{select any } e = (u, v) \in E'\]

$W \leftarrow W \cup \{u, v\}$

$E' \leftarrow E' \setminus \{\text{edges } e \in E' \text{ that touch } u \text{ or } v\}$

Claim: At most a factor 2 larger than the optimal vertex-cover size.

Proof: Edges selected don’t share any vertices so any vertex-cover must choose at least one of $u$ or $v$ each time.

This actually a better approximation factor than the greedy algorithm that repeatedly chooses the highest degree vertex remaining that you considered on Homework 3.
Set-Cover

Find smallest collection of sets containing every point
Set-Cover

Find smallest collection of sets containing every point

Set cover size 4
Set-Cover

Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements

Find smallest collection of sets containing every point
Set-Cover

Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements

Find smallest collection of sets containing every point
Set-Cover

Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements

Find smallest collection of sets containing every point
**Set-Cover**

Greedy Set Cover: Repeatedly choose the set that covers the most # of new elements

Find smallest collection of sets containing every point

**Theorem:** Greedy finds best cover up to a factor of $\ln n$. 
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered
Greedy Set Cover: Repeatedly choose the set that maximizes \# new elements covered

Greedy solution: 5 sets

Greedy solution: $\sim \log_2 n$ sets
Greedy Set Cover: Repeatedly choose the set that maximizes # new elements covered

Optimal: 2 sets
Greedy Approximation to Set-Cover

**Theorem:** If there is a set cover of size $k$ then the greedy set cover has size $\leq k \ln n$.

**Proof:** Suppose that there is a set cover of size $k$. At each step all elements remaining are covered by these $k$ sets. So always a set available covering $\geq \frac{1}{k}$ fraction of remaining els. So # of uncovered els after $i$ sets $\leq \left(1 - \frac{1}{k}\right) \times \# \text{ after } i - 1 \text{ sets}$.

Total after $t$ sets $\leq n \left(1 - \frac{1}{k}\right)^t < n \cdot e^{-t/k} = 1$ for $t = k \ln n$. □

$1 - x < e^{-x}$ for $x > 0$
Travelling-Salesperson Problem (TSP):

Given: a set of $n$ cities $v_1, \ldots, v_n$ and distance function $d$ that gives distance $d(v_i, v_j)$ between each pair of cities

Find the shortest tour that visits all $n$ cities.

MetricTSP:

The distance function $d$ satisfies the triangle inequality:

$$d(u, w) \leq d(u, v) + d(v, w)$$

Proper tour: visit each city exactly once.
Minimum Spanning Tree Approximation: Factor of 2
TSP: Minimum Spanning Tree Factor 2 Approximation

Euler Tour of doubled MST:

Euler tour covers each edge twice so $\text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G)$

Any tour contains a spanning tree so $\text{MST}(G) \leq \text{TOUR}_{\text{OPT}}(G)$

So $\text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G) \leq 2 \text{TOUR}_{\text{OPT}}(G)$

This visits each node more than once, so not a proper tour.
Why did this work?

• We found an Euler tour on a graph that used the edges of the original graph (possibly repeated).

• The weight of the tour was the total weight of the new graph.

• Suppose now
  • All edges possible
  • Weights satisfy the triangle inequality (MetricTSP)
MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Euler Tour of doubled MST:

Euler tour covers each edge twice so \( \text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G) \)

Any tour contains a spanning tree so \( \text{MST}(G) \leq \text{TOUR}_{\text{OPT}}(G) \)

So \( \text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G) \leq 2 \text{TOUR}_{\text{OPT}}(G) \)

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.
MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Euler tour covers each edge twice so \( \text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G) \)

Any tour contains a spanning tree so \( \text{MST}(G) \leq \text{TOUR}_{\text{OPT}}(G) \)

So \( \text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G) \leq 2 \text{TOUR}_{\text{OPT}}(G) \)

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.
MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Final:

Euler tour covers each edge twice so $\text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G)$

Any tour contains a spanning tree so $\text{MST}(G) \leq \text{TOUR}_{\text{OPT}}(G)$

So $\text{TOUR}_{\text{MST}}(G) = 2 \text{MST}(G) \leq 2 \text{TOUR}_{\text{OPT}}(G)$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.
Christofides Algorithm: A factor 3/2 approximation

Any subgraph of the weighted complete graph that has an Euler Tour will work also!

Fact: To have an Euler Tour it suffices to have all degrees even.

Christofides Algorithm:

• Compute an MST $T$
• Find the set $O$ of odd-degree vertices in $T$
• Add a minimum-weight perfect matching* $M$ on the vertices in $O$ to $T$ to make every vertex have even degree
  • There are an even number of odd-degree vertices!
  • Use an Euler Tour $E$ in $T \cup M$ and then shortcut as before

Theorem: $\text{Cost}(E) \leq 1.5 \text{ TOUR}_{OPT}$

*Requires finding optimal matchings in general graphs, not just bipartite ones
Christofides Approximation

Any tour contains a spanning tree so $MST \leq TOUR_{OPT}$

We just need to show that the matching $M$ has $\text{cost}(M) \leq TOUR_{OPT}/2$
Any tour costs at least the cost of two matchings $M_1$ and $M_2$ on $O$

$$2 \text{cost}(M) \leq \text{cost}(M_1) + \text{cost}(M_2) \leq \text{TOUR}_{OPT}$$
Christofides Approximation Final Tour

Total $\text{cost}(E) \leq 3 \text{ TOUR}_{OPT}/2$
Max-3SAT Approximation

Max-3SAT: Given a 3CNF formula $F$ find a truth assignment that satisfies the maximum possible # of clauses of $F$.

Observation: A single clause on 3 variables only rules out $1/8$ of the possible truth assignments since each literal has to be false to be ruled out.

$\Rightarrow$ a random truth assignment will satisfy the clause with probability $7/8$.

So in expectation, if $F$ has $m$ clauses, a random assignment satisfies $7m/8$ of them.

A greedy algorithm can achieve this: Choose most frequent literal appearing in clauses that are not yet satisfied and set it to true.

If $P \neq NP$ no better approximation is possible
Knapsack Problem

Each item has a value $v_i$ and a weight $w_i$. Maximize $\sum_{i \in S} v_i$ with $\sum_{i \in S} w_i \leq W$.

Theorem: For any $\varepsilon > 0$ there is an algorithm that produces a solution within $(1 + \varepsilon)$ factor of optimal for the Knapsack problem with running time $O(n^2/\varepsilon^2)$

“Polynomial-Time Approximation Scheme” or PTAS

Algorithm: Maintain the high order bits in the dynamic programming solution.
Hardness of Approximation

Polynomial-time approximation algorithms for NP-hard optimization problems can sometimes be ruled out unless P = NP.

Easy example:

**Coloring:** Given a graph $G = (V, E)$ find the smallest $k$ such that $G$ has a $k$-coloring.

Because 3-coloring is NP-hard, no approximation ratio better than $4/3$ is possible unless P = NP because you would have to be able to figure out if a 3-colorable graph can be colored in < 4 colors. i.e. if it can be 3-colored.

• We now know a huge amount about the hardness of approximating NP optimization problems if P ≠ NP.

• Approximation factors are very different even for closely related problems like Vertex-Cover and Independent-Set.
Approximation Algorithms/Hardness of Approximation

Research has classified many problems based on what kinds of polytime approximations are possible if $P \neq NP$

- **Best:** $(1 + \varepsilon)$ factor for any $\varepsilon > 0$. (PTAS)
  - packing and some scheduling problems, TSP in plane
- Some fixed constant factor $> 1$. e.g. $2, 3/2, 8/7, 100$
  - Vertex Cover, Max-3SAT, MetricTSP, other scheduling problems
  - Exact best factors or very close upper/lower bounds known for many problems.
- $\Theta(\log n)$ factor
  - Set Cover, Graph Partitioning problems
- **Worst:** $\Omega(n^{1-\varepsilon})$ factor for every $\varepsilon > 0$.
  - Clique, Independent-Set, Coloring