Recall Many practical problems are NP-complete — no one knows a polynomial time algorithm, nor can we prove that none exists.

This lecture: What to do with NP-hard optimization problems.

1. Efficient exhaustive search (backtracking, branch-and-bound). Exponential time in the worst case, but can be useful.

2. Heuristics
   - there might be no guarantee on run-time nor on quality of solution.
   - local search — start with some solution and try to improve it via small “local” changes. hill climbing, simulated annealing
   - particle swarm, evolutionary algorithms

3. Approximation algorithms — today’s topic
   polynomial time and a guarantee on the quality of the solution e.g. for a minimization problem, might guarantee a solution $\leq 2 \cdot \min$
Approximation algorithms for Vertex Cover

Recall A vertex cover is a set \( S \subseteq V \) such that every edge \((u,v) \in E\) has \( u \) or \( v \) (or both) in \( S \).

Optimization problem: find a minimum size vertex cover.

Recall that the decision version is NP-complete.

Greedy Algorithm 1

\[
C := \emptyset \\
\text{repeat} \\
\quad C := C \cup \{\text{vertex of maximum degree}\} \\
\quad \text{remove covered edges} \\
\text{until no edges remain}
\]

Example

\[
C = \{v_1, v_4, v_3\} \\
|C| = 3 \text{ seems optimum (min. size)}
\]

Note: Alg. runs in polynomial time
Approximation algorithms for Vertex Cover

Greedy Algorithm 2

C := \emptyset  \quad F := E \quad // F is uncovered edges

while F \neq \emptyset

pick e = (u,v) from F
add u and v to C
remove edges incident to u from F
remove edges incident to v from F

Example

C = \{u_2, u_5, u_3, u_4\}

|C| = 4  \quad not optimum.

Which is better, Algorithm 1 or Algorithm 2?

On this example, Alg. 1 is better.

Ex. Find an example where Alg. 2 is better.
Approximation algorithms for Vertex Cover

Greedy Algorithm 2

\[ \begin{align*}
C & := \emptyset \quad F := E \quad // \text{F is uncovered edges} \\
\text{while } F \neq \emptyset & \\
\quad & \text{pick } e = (u,v) \text{ from } F \\
\quad & \text{add } u \text{ and } v \text{ to } C \\
\quad & \text{remove edges incident to } u \text{ from } F \\
\quad & \text{remove edges incident to } v \text{ from } F
\end{align*} \]

Analysis of approximation factor

Let \( C = \) vertex cover found by Algorithm 2.
Let \( C_{\text{OPT}} = \) a minimum vertex cover.

Claim. \( |C| \leq 2 \cdot |C_{\text{OPT}}| \)

Proof. Note that the edges chosen form a matching \( M \) (no two edges are incident).

\[ |C| = 2 |M| \]

Any vertex cover must have at least one vertex from each edge in \( M \).

\[ |M| \leq |C_{\text{OPT}}| \quad \text{so} \quad |C| \leq 2 \cdot |C_{\text{OPT}}| \]
Approximation algorithms for Vertex Cover

We say that Algorithm 2 has *approximation factor* $2$ because it produces a vertex cover of size $\leq 2 \cdot \text{optimum}$.

**FACT:** Algorithm 1 has approximation factor $\Theta(\log n)$. It is worse than Algorithm 2.

Recall that Vertex Cover and Independent Set are closely related. However:

**FACT:** Independent Set has no good approximation algorithm unless $P = NP$. 

CS 466 covers this.
Summary of Lecture 22, Part 1

Approximation algorithms for Vertex Cover

What you should know

- what is an approximation algorithm
- what does approximation factor mean
- some NP-complete problems have good approximation algorithms and some do not (unless P = NP)

Next:

Approximation algorithm for Travelling Salesman Problem in the Plane
Travelling Salesman Problem

Given a graph $G$, weights on edges, number $k$, does $G$ have a TSP tour of length $\leq k$?

Euclidean TSP. For the complete graph on points in the plane, with weight = Euclidean distance.

FACT: even Euclidean TSP is NP-complete.

key property of Euclidean case: triangle inequality

$$w(a, c) \leq w(a, b) + w(b, c)$$
Approximation algorithm for Euclidean TSP

compute MST (min. spanning tree)  \( \text{black} \)

take a tour by walking around it  \( \text{blue} \)
(we visit every vertex but maybe more than once)

take shortcuts to avoid revisiting vertices  \( \text{red} \)

note: by the triangle inequality, the short-cuts are shorter

This algorithm takes poly time.
Approximation algorithm for Euclidean TSP

Let $t = \text{length of tour found by this algorithm.}$
Let $t_{\text{TSP}} = \text{length of minimum TSP tour}$

**Claim.** $t \leq 2 \, t_{\text{TSP}}$

This means that in polynomial time, we can find a tour within 2 times the optimum.

**Proof of Claim.**
Let $t_{\text{MST}} = \text{length of MST}$

$t_{\text{MST}} \leq t_{\text{TSP}}$ because deleting one edge of TSP tour (decreasing length) gives some spanning tree, so MST is even less in length.

$t \leq 2 \, t_{\text{MST}}$ because blue tour has length $2 \, t_{\text{MST}}$ and then we take shortcuts (using triangle inequality)

Thus $t \leq 2 \, t_{\text{TSP}}$
We say that the algorithm has **approximation factor** 2 because it finds a tour of length at most 2 times the optimum, i.e. $t \leq 2 \cdot t_{\text{TSP}}$

**FACT:** the factor of 2 can be improved for this problem. For any $\varepsilon > 0$ there is an algorithm that finds a tour of length $\leq (1+\varepsilon) \cdot t_{\text{TSP}}$

But as $\varepsilon \to 0$, the run time becomes exponential.

CS 466 covers this
Summary of Lecture 22

good approximation algorithms for Vertex Cover and TSP.

What you should know

- what is an approximation algorithm

- what does approximation factor mean

- some NP-complete problems have good approximation algorithms and some do not (unless P = NP)