Approximation Algorithms
<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>Given a Boolean formula $F$, is it satisfiable?</td>
</tr>
<tr>
<td>3SAT</td>
<td>Same, but $F$ is a 3-CNF</td>
</tr>
<tr>
<td>Vertex-Cover</td>
<td>Given $G$ and $k$, are there $k$ vertices which touch all edges?</td>
</tr>
<tr>
<td>Clique</td>
<td>Are there $k$ vertices all connected?</td>
</tr>
<tr>
<td>Max-Cut</td>
<td>Is there a vertex 2-coloring with at least $k$ “cut” edges?</td>
</tr>
<tr>
<td>Hamiltonian-Cycle</td>
<td>Is there a cycle touching each vertex exactly once?</td>
</tr>
</tbody>
</table>
SAT ... is NP-complete
3SAT ... is NP-complete
Vertex-Cover ... is NP-complete
Clique ... is NP-complete
Max-Cut ... is NP-complete
Hamiltonian-Cycle ... is NP-complete
**Decision vs. Optimization/Search**

NP defined to be a class of **decision problems**. Usually there is a natural ‘optimization’ version.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>Given a 3-CNF formula, is it satisfiable?</td>
</tr>
<tr>
<td>Vertex-Cover</td>
<td>Given G and k, are there k vertices which touch all edges?</td>
</tr>
<tr>
<td>Clique</td>
<td>Given G and k, are there k vertices which are all mutually connected?</td>
</tr>
<tr>
<td>Max-Cut</td>
<td>Is there a vertex 2-coloring with at least k “cut” edges?</td>
</tr>
<tr>
<td>Hamiltonian-Cycle</td>
<td>Is there a cycle touching each vertex exactly once?</td>
</tr>
</tbody>
</table>
**Decision vs. Optimization/Search**

NP defined to be a class of **decision problems**.

Usually there is a natural ‘optimization’ version.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td></td>
</tr>
<tr>
<td>Vertex-Cover</td>
<td>Given G, <strong>find the size</strong> of the smallest ( S \subseteq V ) touching all edges.</td>
</tr>
<tr>
<td>Clique</td>
<td>Given G, <strong>find the size</strong> of the largest clique (set of mutually connected vertices).</td>
</tr>
<tr>
<td>Max-Cut</td>
<td>Given G, <strong>find the largest number</strong> of edges ‘cut’ by some vertex 2-coloring.</td>
</tr>
<tr>
<td>Hamiltonian-Cycle</td>
<td></td>
</tr>
</tbody>
</table>
# Decision vs. Optimization/Search

NP defined to be a class of **decision problems**.

Usually there is a natural ‘optimization’ version.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>Given a 3-CNF formula, <strong>find the largest number</strong> of clauses satisfiable by a truth assignment.</td>
</tr>
<tr>
<td>Vertex-Cover</td>
<td>Given G, <strong>find the size</strong> of the smallest $S \subseteq V$ touching all edges.</td>
</tr>
<tr>
<td>Clique</td>
<td>Given G, <strong>find the size</strong> of the largest clique (set of mutually connected vertices).</td>
</tr>
<tr>
<td>Max-Cut</td>
<td>Given G, <strong>find the largest number</strong> of edges ‘cut’ by some vertex 2-coloring.</td>
</tr>
<tr>
<td>Hamiltonian-Cycle</td>
<td></td>
</tr>
</tbody>
</table>
**Decision vs. Optimization/Search**

NP defined to be a class of *decision problems.*

Usually there is a natural ‘optimization’ version.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>Given a 3-CNF formula, find the largest number of clauses satisfiable by a truth assignment.</td>
</tr>
<tr>
<td>Vertex-Cover</td>
<td>Given G, find the size of the smallest $S \subseteq V$ touching all edges.</td>
</tr>
<tr>
<td>Clique</td>
<td>Given G, find the size of the largest clique (set of mutually connected vertices).</td>
</tr>
<tr>
<td>Max-Cut</td>
<td>Given G, find the largest number of edges ‘cut’ by some vertex 2-coloring.</td>
</tr>
<tr>
<td>TSP</td>
<td>Given G with edge costs, find the cost of the cheapest cycle touching each vertex once.</td>
</tr>
</tbody>
</table>
## Decision vs. Optimization/Search

**NP** defined to be a class of **decision problems**.

Usually there is a natural ‘optimization’ version and a natural ‘search’ version.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3SAT</strong></td>
<td>Given a 3-CNF formula, find a truth assignment with the largest number of satisfied clauses.</td>
</tr>
<tr>
<td><strong>Vertex-Cover</strong></td>
<td>Given G, find the smallest $S \subseteq V$ touching all edges.</td>
</tr>
<tr>
<td><strong>Clique</strong></td>
<td>Given G, find the largest clique (set of mutually connected vertices).</td>
</tr>
<tr>
<td><strong>Max-Cut</strong></td>
<td>Given G, find the vertex 2-coloring which ‘cuts’ the largest number of edges.</td>
</tr>
<tr>
<td><strong>TSP</strong></td>
<td>Given G with edge costs, find the cheapest cycle touching each vertex once.</td>
</tr>
</tbody>
</table>
Decision vs. Optimization/Search

NP defined to be a class of **decision problems**.

Usually there is a natural ‘optimization’ version and a natural ‘search’ version.

Technically, the ‘optimization’ or ‘search’ versions cannot be in **NP**, since they’re not languages.

We often still say they are **NP-hard**.

This means: if you could solve them in poly-time, then you could solve any NP problem in poly-time.

Why???
Decision vs. Optimization/Search

More interestingly the opposite is usually true too: Given an efficient solution to the decision problem we can solve the ‘optimization’ and ‘search’ versions efficiently, too.

Find the number (e.g., of satisfiable clauses) via binary search.

Find a solution (e.g., satisfying assignment) by setting variables one by one and, testing each time if there is still a good assignment.
<table>
<thead>
<tr>
<th>Problem</th>
<th>... is NP-complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td></td>
</tr>
<tr>
<td>3SAT</td>
<td></td>
</tr>
<tr>
<td>Vertex-Cover</td>
<td></td>
</tr>
<tr>
<td>Clique</td>
<td></td>
</tr>
<tr>
<td>Max-Cut</td>
<td></td>
</tr>
<tr>
<td>Hamiltonian-Cycle</td>
<td></td>
</tr>
</tbody>
</table>
INVENTS BEAUTIFUL THEORY
OF ALGORITHMIC COMPLEXITY

EVERYTHING IS NP-COMPLETE
There is only one idea in this lecture:

NEVER GIVE UP!
Vertex-Cover

Given graph $G = (V,E)$ and number $k$, is there a size-$k$ “vertex-cover” for $G$?

($S \subseteq V$ is a “vertex-cover” if it touches all edges.)

$G$ has a vertex-cover of size 3.
Vertex-Cover

Given graph $G = (V,E)$ and number $k$, is there a size-$k$ “vertex-cover” for $G$?

($S \subseteq V$ is a “vertex-cover” if it touches all edges.)

$G$ has no vertex-cover of size 2.

(Because you need $\geq 1$ vertex per yellow edge.)
Vertex-Cover

Given graph $G = (V,E)$ and number $k$, is there a size-$k$ “vertex-cover” for $G$?

($S \subseteq V$ is a “vertex-cover” if it touches all edges.)

The Vertex-Cover problem is NP-complete. 😞

✦ assuming “$P \neq NP$”, there is no algorithm running in polynomial time which, for all graphs $G$, finds the minimum-size vertex-cover.
Never Give Up

Subexponential-time algorithms:
Brute-force tries all $2^n$ subsets of $n$ vertices.
Maybe there’s an $O(1.5^n)$-time algorithm.
Or $O(1.1^n)$ time, or $O(2^{n-1})$ time, or...
Could be quite okay if $n = 100$, say.

As of 2010: there is an $O(1.28^n)$-time algorithm.

assuming “P $\neq$ NP”, there is no algorithm running in polynomial time
which, for all graphs $G$, finds the minimum-size vertex-cover.
Never Give Up

Special cases:

Solvable in poly-time for...

**tree** graphs,

**bipartite** graphs,

“**series-parallel**” graphs...

Perhaps for “graphs encountered in practice”?

→ assuming “P ≠ NP”, there is no algorithm running in **polynomial time** which, for all graphs \( G \), finds the **minimum**-size vertex-cover.
Approximation algorithms:

Try to find pretty small vertex-covers.

Still want polynomial time, and for all graphs.

assuming “P ≠ NP”, there is no algorithm running in polynomial time which, for all graphs G, finds the minimum-size vertex-cover.
Gavril’s Approximation Algorithm

Easy Theorem (from 1976):

There is a **polynomial-time** algorithm that, given any graph \( G = (V,E) \), outputs a vertex-cover \( S \subseteq V \) such that

\[
|S| \leq 2|S^*|,
\]

where \( S^* \) is the **smallest** vertex-cover.

“A factor 2-approximation for Vertex-Cover.”
Let’s recall a similar situation from Lecture 10:

My favorite problem, Max-Cut.
**Max-Cut**

**Input:** A graph $G=(V,E)$. 

**Output:** A "2-coloring" of $V$: each vertex designated yellow or blue.

**Goal:** Have as many cut edges as possible. An edge is cut if its endpoints have different colors.
Max-Cut

Input: A graph \( G = (V, E) \).

Output: A “2-coloring” of \( V \): each vertex designated yellow or blue.

Goal: Have as many cut edges as possible. An edge is cut if its endpoints have different colors.
Max-Cut

On one hand:
Finding the **MAX**-Cut is **NP-hard**.

On the other hand:
Polynomial-time “Local Search” algorithm guarantees cutting \( \geq \frac{1}{2}|E| \) edges.

In particular:
\((# \text{ cut by Local Search}) \geq \frac{1}{2} (\text{max # cuttable})\)

“A factor \(\frac{1}{2}\)-approximation for Max-Cut.”
Max-Cut

By the way:

Goemans and Williamson (1994) gave a polynomial-time

0.87856-approximation

for Max-Cut.

It is very beautiful, but pretty difficult!
Not all NP-hard problems created equal!

3SAT, Vertex-Cover, Clique, Max-Cut, TSP, ...

All of these problems are equally NP-hard.

(There’s no poly-time algorithm to find the optimal solution unless P = NP.)

But from the point of view of finding approximately optimal solutions, there is an intricate, fascinating, and wide range of possibilities...
Today: A case study of approximation algorithms

1. A somewhat good approximation algorithm for Vertex-Cover.

2. A pretty good approximation algorithm for the “k-Coverage Problem”.

3. Some very good approximation algorithms for TSP.
Today: A case study of approximation algorithms

1. A somewhat good approximation algorithm for Vertex-Cover.

2. A pretty good approximation algorithm for the “k-Coverage Problem”.

3. Some very good approximation algorithms for TSP.
Vertex-Cover

Given graph $G = (V, E)$ try to find the smallest “vertex-cover” for $G$.

($S \subseteq V$ is a “vertex-cover” if it touches all edges.)
A possible Vertex-Cover algorithm

Simplest heuristic you might think of:

\[
\text{GreedyVC}(G)\\
S \leftarrow \emptyset \\
\text{while not all edges marked as “covered”} \\
\text{find } v \in V \text{ touching most unmarked edges} \\
S \leftarrow S \cup \{v\} \\
\text{mark all edges } v \text{ touches}
\]
GreedyVC example
GreedyVC example

(Break ties arbitrarily.)
GreedyVC example
GreedyVC example

Done. Vertex-cover size 3 (optimal) 😊.
GreedyVC analysis

Correctness:
✓ Always outputs a valid vertex-cover.

Running time:
✓ Polynomial time.

Solution quality:
This is the interesting question. There must be some graph \( G \) where it doesn’t find the smallest vertex-cover. Because otherwise... \( P = NP! \)
A bad graph for GreedyVC

Smallest?  3
A bad graph for GreedyVC

Smallest? 3
GreedyVC? 4

So GreedyVC is not a 1.33-approximation. (Because 1.33 < 4/3.)
A worse graph for GreedyVC

Smallest? ???
GreedyVC? 21

So GreedyVC is **not** a 1.74-approximation.
(Because 1.74 < 21/12.)
Even worse graph for GreedyVC

Well... it’s a good homework problem.

We know GreedyVC is **not** a 1.74-approximation.

**Fact:** GreedyVC is **not** a 2.08-approximation.

**Fact:** GreedyVC is **not** a 3.14-approximation.

**Fact:** GreedyVC is **not** a 42-approximation.

**Fact:** GreedyVC is **not** a 999-approximation.
Greed is Bad (for Vertex-Cover)

**Theorem:** $\forall C$, GreedyVC is **not** a $C$-approximation.

In other words:

For any constant $C$,
there is a graph $G$ such that

$$|\text{GreedyVC}(G)| > C \cdot |\text{Min-Vertex-Cover}(G)|.$$
GavrilVC(G)

S ← ∅

while not all edges marked as “covered”

let \{v,w\} be any unmarked edge

S ← S ∪ \{v,w\}

mark all edges v,w touch
GavriliVC example
GavrilVC example
GavrilVC example

Smallest: 3
GavrilVC: 6

So GavrilVC is at best a 2-approximation.
Theorem:
GavrilVC is a 2-approximation for Vertex-Cover.

Proof:
Say GavrilVC(G) does $T$ iterations. So its $|S| = 2T$.
Say it picked edges $e_1, e_2, ..., e_T \in E$.
Key claim: $\{e_1, e_2, ..., e_T\}$ is a matching.
Because... when $e_j$ is picked, it’s unmarked, so its endpoints are not among $e_1, ..., e_{j-1}$.
So any vertex-cover must have $\geq 1$ vertex from each $e_j$. 
**Theorem:**
GavriliVC is a 2-approximation for Vertex-Cover.

**Proof:**
Say GavriliVC(G) does $T$ iterations. So its $|S| = 2T$.
Say it picked edges $e_1, e_2, \ldots, e_T \in E$.

**Key claim:** $\{e_1, e_2, \ldots, e_T\}$ is a matching.
Because... when $e_j$ is picked, it’s unmarked, so its endpoints are not among $e_1, \ldots, e_{j-1}$.

So any vertex-cover must have $\geq 1$ vertex from each $e_j$. Including the **minimum** vertex-cover $S^*$, whatever it is.

Thus $|S^*| \geq T$.

So for Gavril’s final vertex-cover $S$,

$$|S| = 2T \leq 2|S^*|.$$
Today: A case study of approximation algorithms


2. A pretty good approximation algorithm for the “k-Coverage Problem”.

3. Some very good approximation algorithms for TSP.
Today: A case study of approximation algorithms


2. A pretty good approximation algorithm for the “k-Coverage Problem”.

3. Some very good approximation algorithms for TSP.
“k-Coverage” problem
“Pokémon-Coverage” problem

Let’s say you have some Pokémon, and some trainers, each having a subset of Pokémon.

Given k, choose a team of k trainers to maximize the # of distinct Pokémon.
“Pokémon-Coverage” problem

This problem is **NP-hard**. 😞

Approximation algorithm?

We could try to be greedy again...

---

**GreedyCoverage()**

for i = 1...k

add to the team the trainer bringing in the most new Pokémon, given the team so far
Example with $k=3$:

Optimum: 27

Greedy Coverage: 21

So Greedy is at best a 77.7%-approximation.
Greed is Pretty Good (for k-Coverage)

Theorem:

GreedyCoverage is a $63\%$-approximation for k-Coverage.

More precisely, $1 - 1/e$

where $e \approx 2.718281828\ldots$
Proof: (Don’t read if you don’t want to.)

Let $P^*$ be the Pokémon covered by the best $k$ trainers. Define $r_i = |P^*| - \# \text{ Pokémon covered after } i \text{ steps of Greedy}$. We’ll prove by induction that $r_i \leq (1 - 1/k)^i \cdot |P^*|$. The base case $i=0$ is clear, as $r_0 = |P^*|$. For the inductive step, suppose Greedy enters its $i$th step. At this point, the number of uncovered Pokémon in $P^*$ must be $\geq r_{i-1}$. We know there are some $k$ trainers covering all these Pokémon. Thus one of these trainers must cover at least $r_{i-1}/k$ of them. Therefore the trainer chosen in Greedy’s $i$th step will cover $\geq r_{i-1}/k$ Pokémon. Thus $r_i \leq r_{i-1} - r_{i-1}/k = (1 - 1/k) \cdot r_{i-1} \leq (1 - 1/k) \cdot (1 - 1/k)^{i-1} \cdot |P^*|$ by induction. Thus we have completed the inductive proof that $r_i \leq (1 - 1/k)^i \cdot |P^*|$. Therefore the Greedy algorithm terminates with $r_k \leq (1 - 1/k)^k \cdot |P^*|$. Since $1 - 1/k \leq e^{-1/k}$ (Taylor expansion), we get $r_k \leq e^{-1} \cdot |P^*|$. Thus Greedy covers at least $|P^*| - e^{-1} \cdot |P^*| = (1 - 1/e) \cdot |P^*|$ Pokémon. This completes the proof that Greedy is a $(1 - 1/e)$-approximation algorithm.
Today: A case study of approximation algorithms


2. A 63% $(1-1/e)$ approximation algorithm for the “k-Coverage Problem”.

3. Some very good approximation algorithms for TSP.
Today: A case study of approximation algorithms


2. A 63% \((1-1/e)\) approximation algorithm for the “k-Coverage Problem”.

3. Some very good approximation algorithms for TSP.
TSP
(Traveling Salesperson Problem)

Many variants. Most common is “Metric-TSP”:

Input: A graph $G=(V,E)$ with edge costs.
Output: A “tour”: i.e., a walk that visits each vertex at least once, and starts and ends at the same vertex.
Goal: Minimize total cost of tour.
TSP example

Cheapest tour:

\[ 3 + 5 + 5 + 16 + 26 + 4 + 12 + 2 + 2 = 71 \]
TSP is probably the most famous NP-complete problem.

It has inspired many things...
Textbooks

- The Traveling Salesman Problem
  A Computational Study
  David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook

- The Traveling Salesman: Computational Solutions for TSP Applications

- Combinatorial Optimization
  The Traveling Salesman Problem and Its Variations
“Popular” books
Museum exhibits
Movies

TRAVELLING SALESMAN
A CEREBRAL THRILLER. COMING SOON
TRAVELLINGSALESMANMOVIE.COM @TRAVSALEM
’60s sitcom-themed household-goods conglomerate ad/contests
People genuinely want to solve large instances.

Applications in:
- Schoolbus routing
- Moving farm equipment
- Package delivery
- Space interferometer scheduling
- Circuit board drilling
- Genome sequencing
- ...

Basic Approximation Algorithm: The MST Heuristic

Given $G$ with edge costs...
1. Compute an MST $T$ for $G$, rooted at any $s \in V$.
2. Visit the vertices via DFS from $s$. 
MST Heuristic example

Step 1: MST
Step 2: DFS

Valid tour? ✓
Poly-time? ✓
Cost?

2 \times \text{MST Cost}

(84 in this case)
MST Heuristic

Theorem: MST Heuristic is factor-$2$ approximation.

Key Claim: Optimal TSP cost $\geq$ MST Cost always.

This implies the Theorem, since

MST Heuristic Cost $= 2 \times$ MST Cost.

Proof of Claim:

Take all edges in optimal TSP solution.
They form a connected graph on all $|V|$ vertices.
Take any spanning tree from within these edges.
Its cost is at least the MST Cost.
Therefore the original TSP tour’s cost is $\geq$ MST Cost.
Can we do better?

Nicos Christofides, Tepper faculty, 1976:

There is a polynomial-time, factor 1.5-approximation algorithm for (Metric) TSP.

Proof is not too hard. Ingredients:
- MST Heuristic
- Eulerian Tours
- Cheapest Perfect Matching algorithm
Even better in a special case

In the important special case "Euclidean-TSP", vertices are points in $\mathbb{R}^2$, costs are just the straight-line distances.

This special case is still NP-hard.

**Theorem** (Arora, Mitchell, 1998): For Euclidean-TSP, there is a polynomial-time factor 1.3 approximation algorithm.
Even better in a special case

In the important special case “Euclidean-TSP”, vertices are points in $\mathbb{R}^2$, costs are just the straight-line distances.

This special case is still NP-hard.

**Theorem** (Arora, Mitchell, 1998): For Euclidean-TSP, there is a polynomial-time factor 1.1 approximation algorithm.
Even better in a special case

In the important special case “Euclidean-TSP”, vertices are points in $\mathbb{R}^2$, costs are just the straight-line distances.

This special case is still NP-hard.

**Theorem** (Arora, Mitchell, 1998): For Euclidean-TSP, there is a polynomial-time factor 1.01 approximation algorithm.
Even better in a special case

In the important special case “Euclidean-TSP”, vertices are points in $\mathbb{R}^2$, costs are just the straight-line distances.

This special case is still NP-hard.

**Theorem** (Arora, Mitchell, 1998): For Euclidean-TSP, there is a polynomial-time factor 1.001 approximation algorithm.
Even better in a special case

In the important special case “Euclidean-TSP”, vertices are points in $\mathbb{R}^2$, costs are just the straight-line distances.

This special case is still NP-hard.

**Theorem** (Arora, Mitchell, 1998):
For Euclidean-TSP, there is a polynomial-time factor $1.0001$ approximation algorithm.
Even better in a special case

In the important special case “Euclidean-TSP”, vertices are points in $\mathbb{R}^2$, costs are just the straight-line distances.

This special case is still NP-hard.

**Theorem** (Arora, Mitchell, 1998):
For Euclidean-TSP, there is a polynomial-time factor $1+\varepsilon$ approximation algorithm, for any $\varepsilon > 0$.

(Running time is like $O(n (\log n)^{1/\varepsilon})$.)
Euclidean-TSP:
NP-hard, but not *that* hard

\[ n > 10,000 \] is feasible
Can we do better?


2. A 63% \((1-1/e)\) approximation algorithm for the “k-Coverage Problem”.

3. A \((1+\varepsilon)\)-approximation alg. for Euclidean-TSP.
Can we do better?

2. A 63% \((1-1/e)\) approximation algorithm for the “k-Coverage Problem”.

We cannot do better. (Unless P=NP.)

**Theorem:** For any \(\beta > 1-1/e\), it is NP-hard to factor \(\beta\)-approximate k-Coverage.

Proved in 1998 by Feige, building on many prior works. Proof length of reduction: \(\approx 100\) pages.
Can we do better?


We have no idea if we can do better.

**Theorem** (Dinur & Safra, 2002, Annals of Math.):
For any $\beta > 10\sqrt{5} - 21 \approx 1.36$, it is NP-hard to $\beta$-approximate Vertex-Cover.
Approximating Vertex-Cover

Approximation Factor

NP-hard (Dinur–Safra) 1.36 2 Poly-time (Gavril)

Between 1.36 and 2: totally unknown. Raging controversy.
Study Guide

Definitions:

Approximation algorithm.

The idea of “greedy” algorithms.

Algorithms and analysis:

Gavril algorithm for Vertex-Cover.

MST Heuristic for TSP.