15-251: Great Theoretical Ideas in Computer Science

Fall 2016, Lecture 15

October 18, 2016

Approximation Algorithms

SAT	given a Boolean formula F, is it satisfiable?
3SAT	same, but F is a 3-CNF
Vertex-Cover	given G and k… are there k vertices which touch all edges?
Clique	are there k vertices all connected?
Max-Cut	is there a vertex 2-coloring with at least k “cut” edges?
Hamiltonian-Cycle	is there a cycle touching each vertex exactly once?

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**

INVENTS BEAUTIFUL THEORY
OF ALGORITHMIC COMPLEXITY

EVERYTHING IS NP-COMPLETE

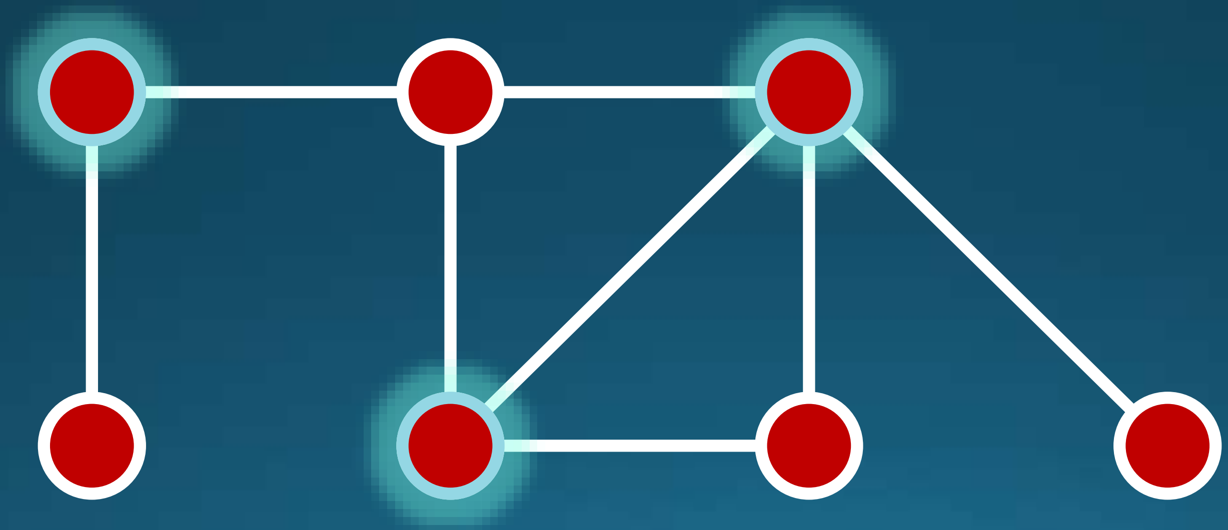
There is only one idea in this lecture:

Don’t Give Up

Vertex-Cover

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

S ⊆ V is a “vertex-cover” if it touches all edges.
(The “popular sets” on HW 5)



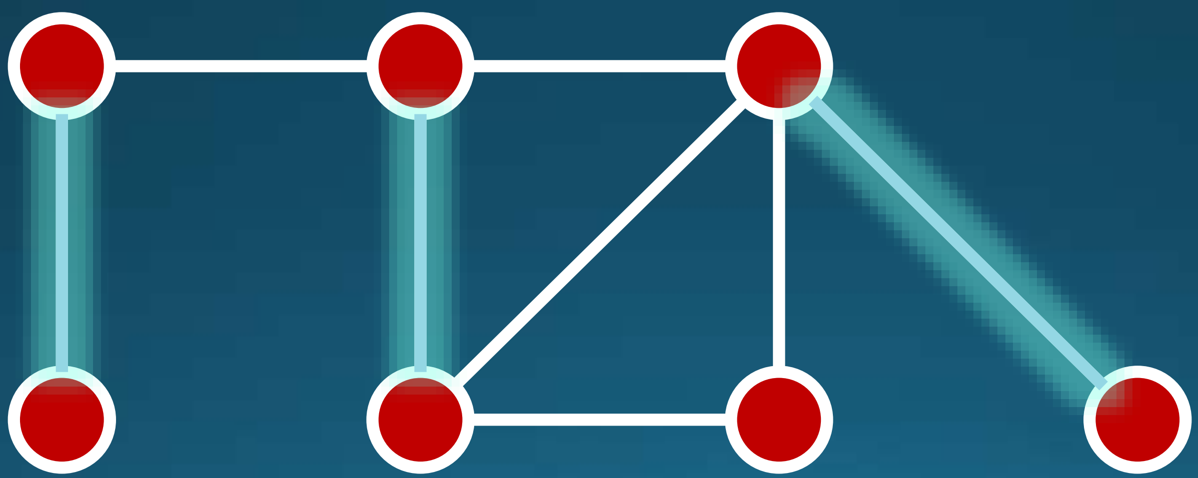
G has a vertex-cover of size 3.

Vertex-Cover

Given graph G = (V,E) and number k,

is there a size-k "vertex-cover" for G?

S ⊆ V is a "vertex-cover" if it touches all edges.



G has no vertex-cover of size 2.

(Because you need ≥ 1 vertex per matching edge.)

Vertex-Cover

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

(S ⊆ V is a "vertex-cover" if it touches all edges.)

The Vertex-Cover problem is NP-complete. ☹

∴ assuming "P ≠ NP", there is **no** algorithm
running in **polynomial time**
which, for **all graphs** G,
finds the **minimum**-size vertex-cover.

Don't 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 ≠ NP", there is **no** algorithm

running in **polynomial time**

which, for **all graphs** G,

finds the **minimum**-size vertex-cover.

Don't 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.

Don’t Give Up

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

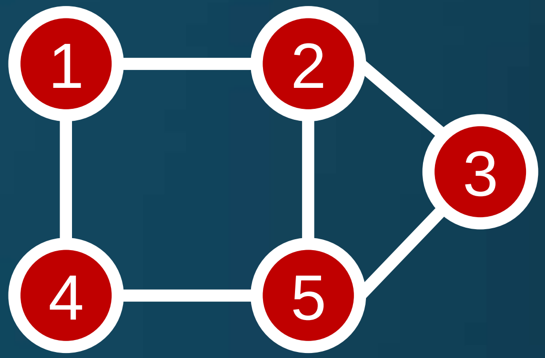
$$|S| \leq 2|S^*|$$

where S^* is the **smallest** vertex-cover.

"A factor 2-approximation for Vertex-Cover."

Another one of my favorite graph problems:

Max-Cut

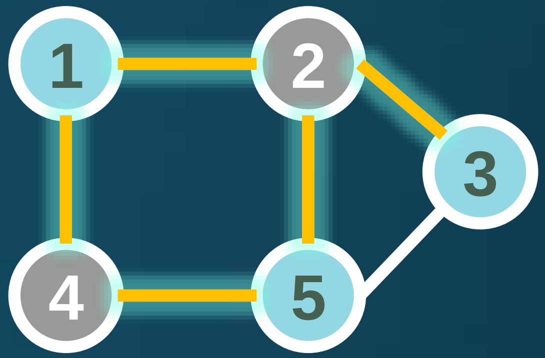


Input: A graph G=(V,E).

Output: A "**2-coloring**" of V:
each vertex designated blue or gray

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 blue or gray.

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.

(Start with arbitrary 2-coloring and repeatedly switch color of a vertex if it improves cut value, till there is no such vertex.)

In particular:

(# cut by Local Search) $\geq \frac{1}{2}$ (max # cuttable)

"A factor ½-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 requires some machinery (semidefinite programming).

A technicality: **Optimization vs. Decision**

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

This is for technical convenience.

Usually have natural 'optimization' version.

3SAT	Given a 3-CNF formula, is it satisfiable?
Vertex-Cover	Given G and k, are there k vertices which touch all edges?
Clique	Given G and k, are there k vertices which are all mutually connected?
Max-Cut	Is there a vertex 2-coloring with at least k "cut" edges?
Hamiltonian-Cycle	Is there a cycle touching each vertex exactly once?

A technicality: **Optimization vs. Decision**

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

This is for technical convenience.

Usually have natural ‘optimization’ version.

3SAT	
Vertex-Cover	Given G, find the smallest S ⊆ V touching all edges.
Clique	Given G, find the largest clique (set of mutually connected vertices).
Max-Cut	Given G, find the largest number of edges ‘cut’ by some vertex 2-coloring.
Hamiltonian-Cycle	

A technicality: **Optimization vs. Decision**

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

This is for technical convenience.

Usually have natural ‘optimization’ version.

Max-3SAT	Given a 3-CNF formula, find the largest number of clauses satisfiable by a truth assignment.
Vertex-Cover	Given G, find the smallest $S \subseteq V$ touching all edges.
Clique	Given G, find the largest clique (set of mutually connected vertices).
Max-Cut	Given G, find the largest number of edges ‘cut’ by some vertex 2-coloring.
Hamiltonian-Cycle	

A technicality: **Optimization vs. Decision**

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

This is for technical convenience.

Usually have natural ‘optimization’ version.

Max-3SAT	Given a 3-CNF formula, find the largest number of clauses satisfiable by a truth assignment.
Vertex-Cover	Given G, find the smallest $S \subseteq V$ touching all edges.
Clique	Given G, find the largest clique (set of mutually connected vertices).
Max-Cut	Given G, find the largest number of edges ‘cut’ by some vertex 2-coloring.
TSP	Given G with edge costs, find the cheapest cycle touching each vertex exactly once.

A technicality: **Optimization vs. Decision**

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

This is for technical convenience.

Usually have natural ‘optimization’ version.

Technically, the ‘optimization’ versions can’t be in NP, since they’re not decision problems.

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.

Let’s not worry about this terminology technicality!

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.

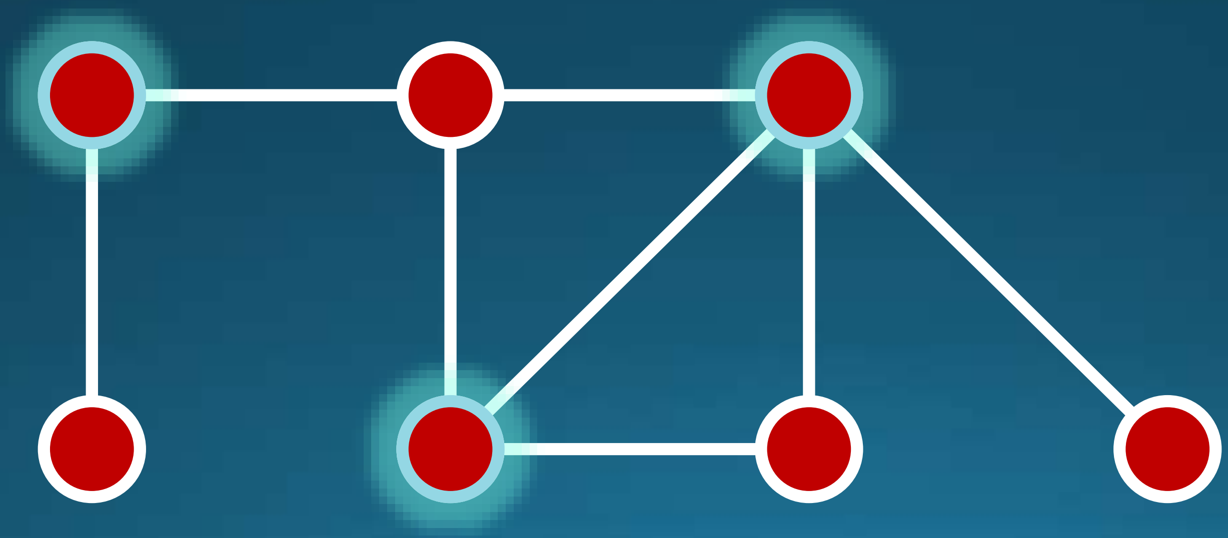
2. A pretty good approximation algorithm
   for the “k-Coverage Problem”.

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 ⊆ V is a "vertex-cover" if it touches all edges.)

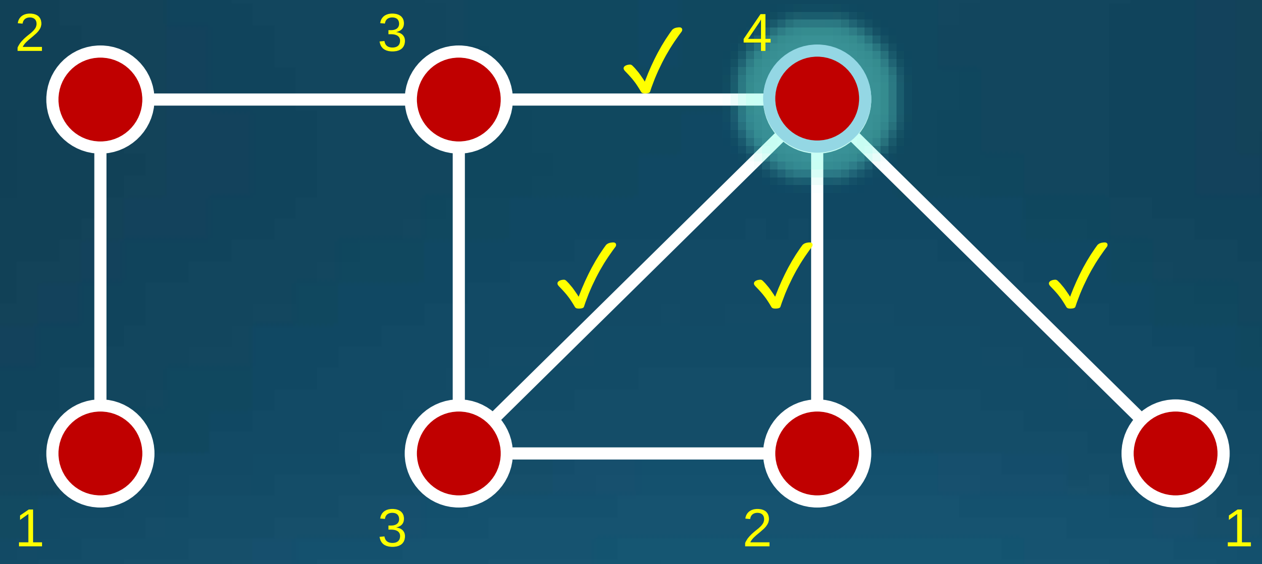


A possible Vertex-Cover algorithm

Simplest heuristic you might think of:

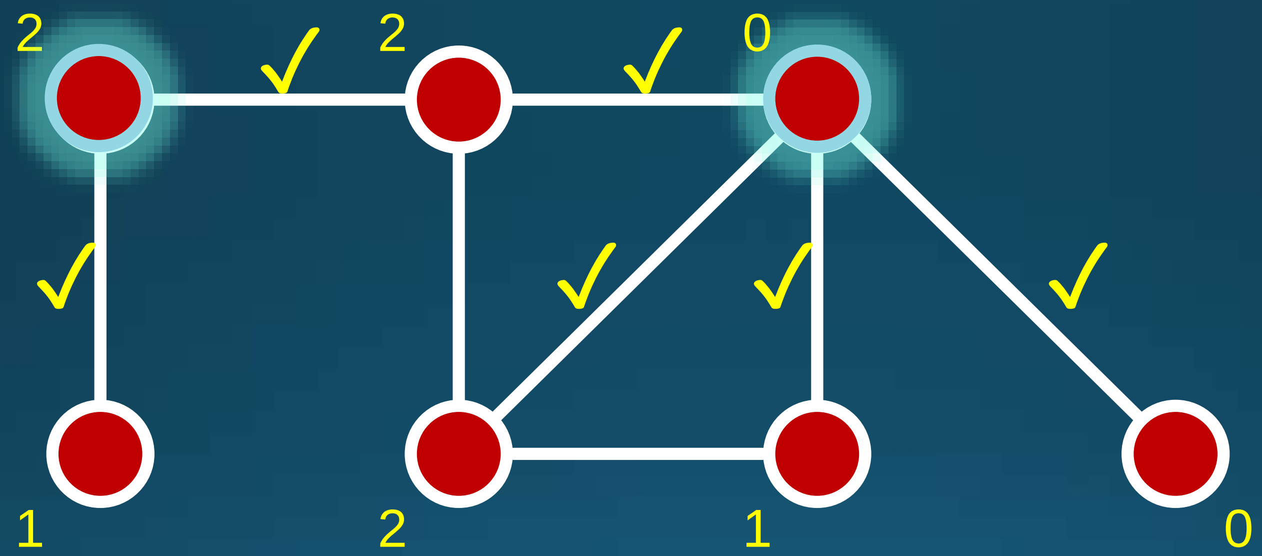
```
GreedyVC(G)
    S ← ∅
    while not all edges marked as "covered"
        find v∈V touching most unmarked edges
        S ← S ∪ {v}
        mark all edges v touches
```

GreedyVC example

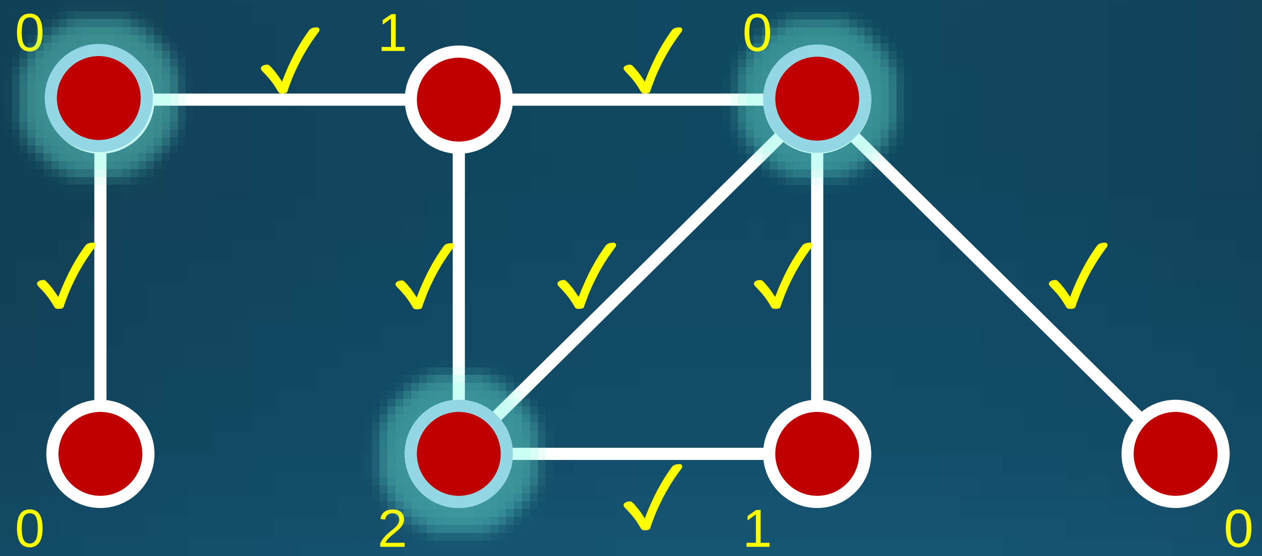


GreedyVC example

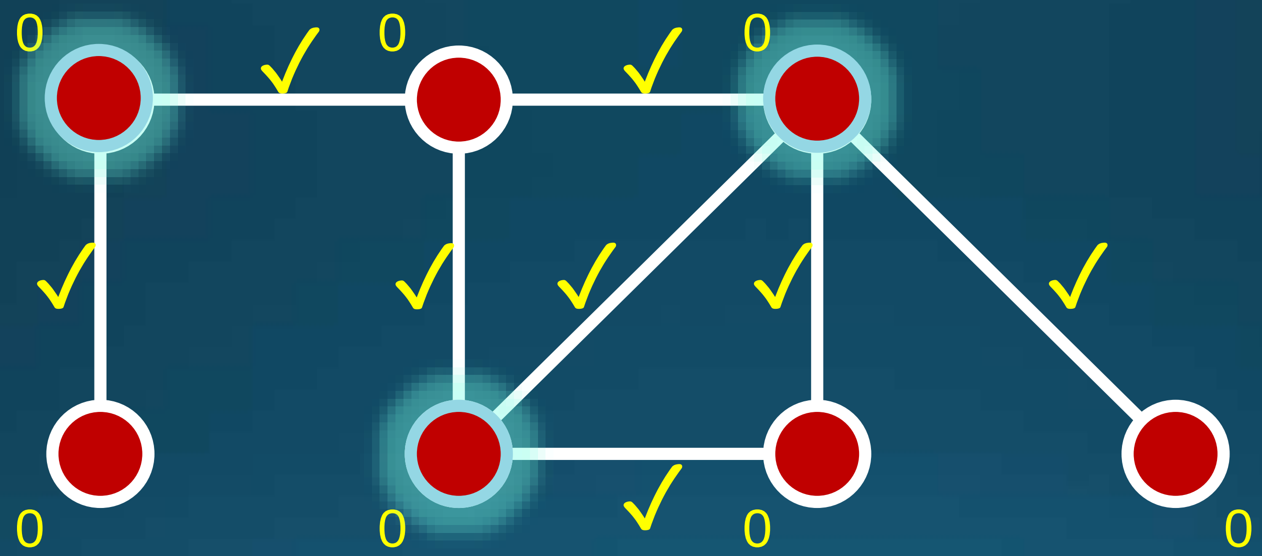
(Break ties arbitrarily.)



GreedyVC example



GreedyVC example



Done. Vertex-cover size 3 (optimal) ☺.

GreedyVC analysis

Correctness:

✓ Always outputs a **valid** vertex-cover.

Running time:

✓ Polynomial time (good enough).

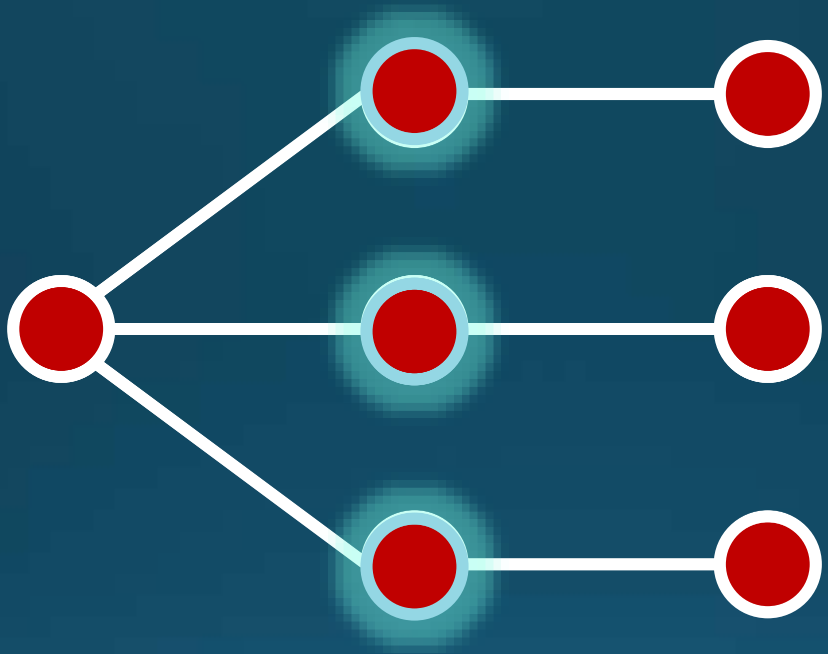
Solution quality:

This is the interesting question.

There must be some graph G where it doesn't find the **smallest** vertex-cover.

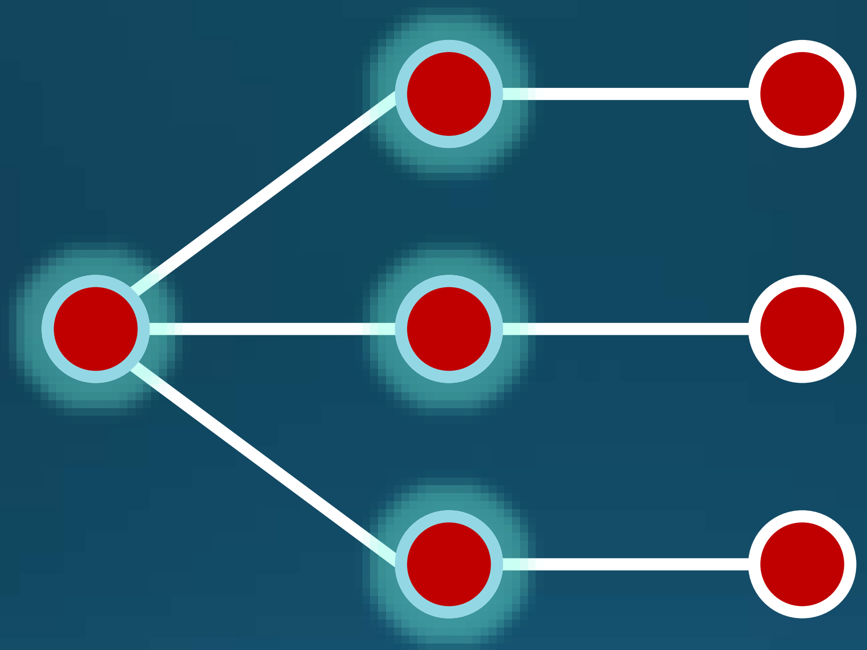
Because otherwise… P = NP!

A bad graph for GreedyVC



Smallest? 3

A bad graph for GreedyVC

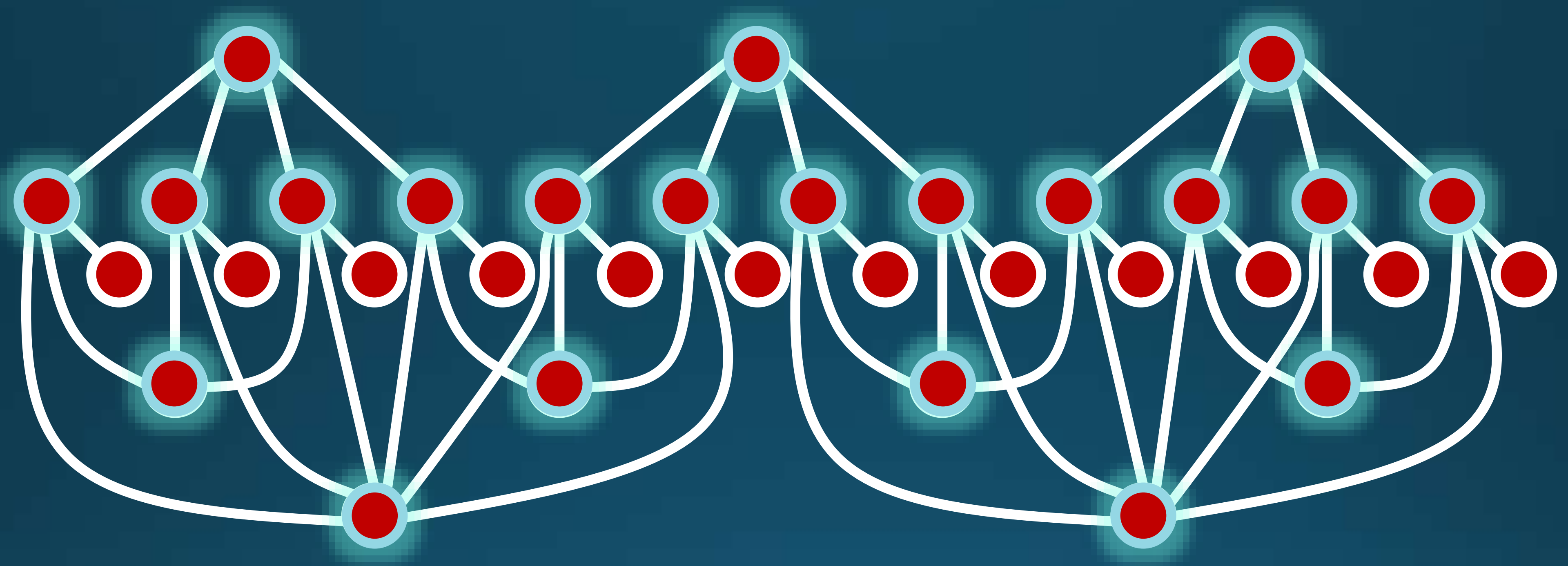


Smallest? 3

GreedyVC? 4

So GreedyVC is **not** even a 1.33-approximation.
(Because 1.33 < 4/3.)

A worse graph for GreedyVC



Smallest?

Poll

GreedyVC? 21

So GreedyVC is **not** even 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-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

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

Gavril's simple algorithm

GavrilVC(G)

S ← ∅

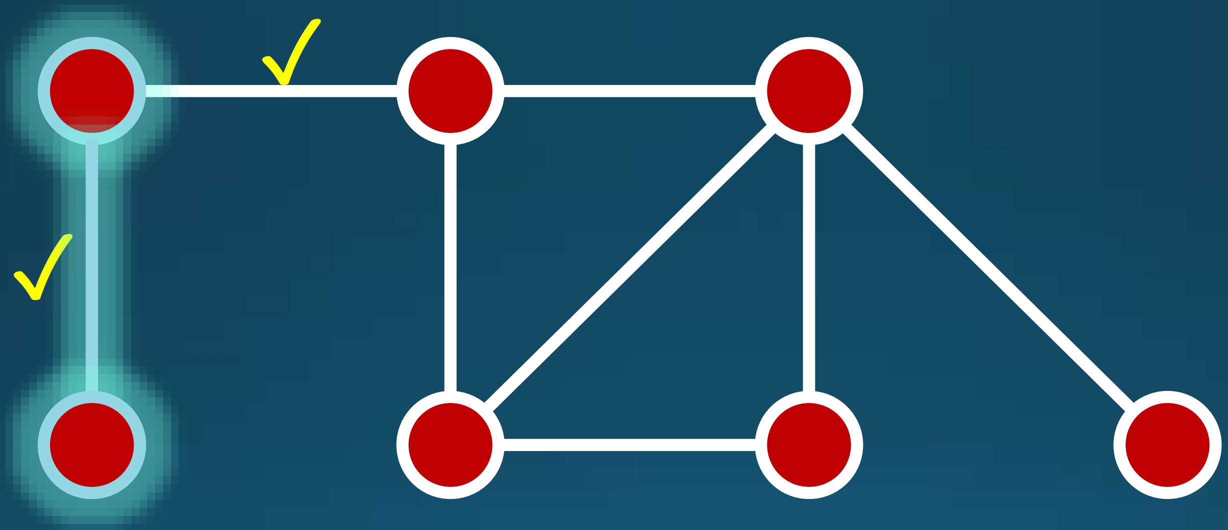
while **not** all edges marked as "covered"

let {v,w} be any unmarked edge

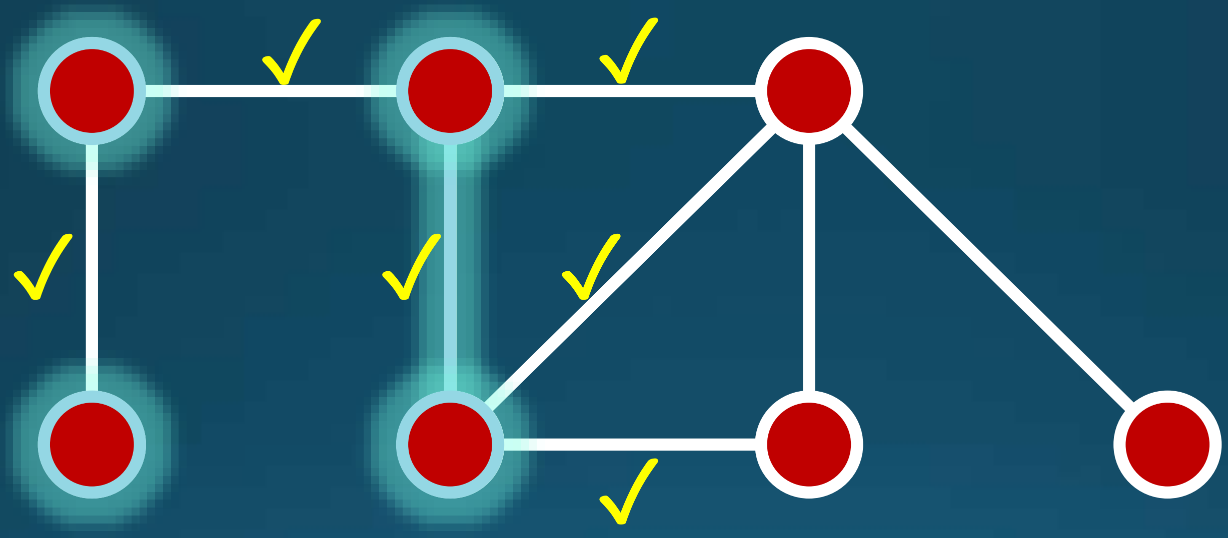
S ← S ∪ {v,w}

mark all edges v,w touch as covered

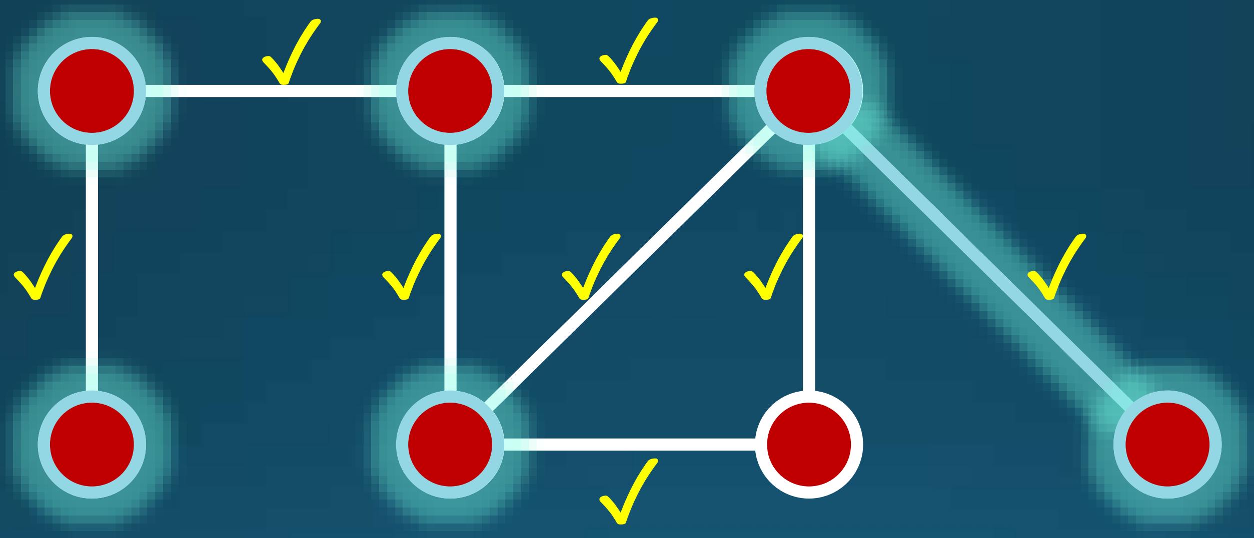
GavrilVC example



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.

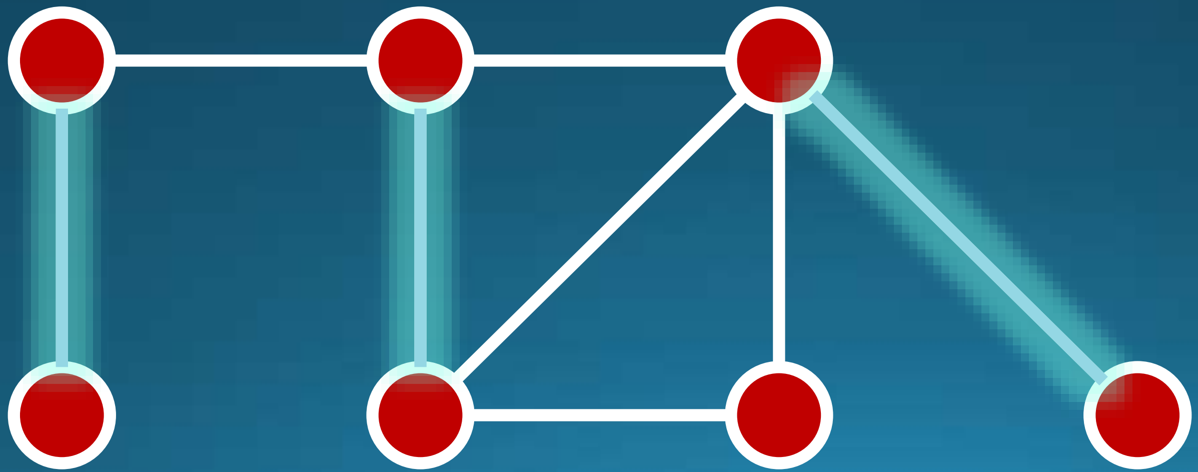
Say it picked edges $e_1, e_2, \ldots, e_T \in E$.

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 ≥ 1 vertex from each e_j.



Theorem:

GavrilVC is a **2**-approximation for Vertex-Cover.

Proof:

Say GavrilVC(G) does T iterations. So its $|S| =$ 2T.

Say it picked edges $e_1, e_2, \ldots, e_T \in E$.

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 ≥ 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,

$$|S| = 2T \leq 2|S^*|. \qquad \blacksquare$$

Today: A case study of approximation algorithms

1. A 2-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 2-approximation algorithm for Vertex-Cover.

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?

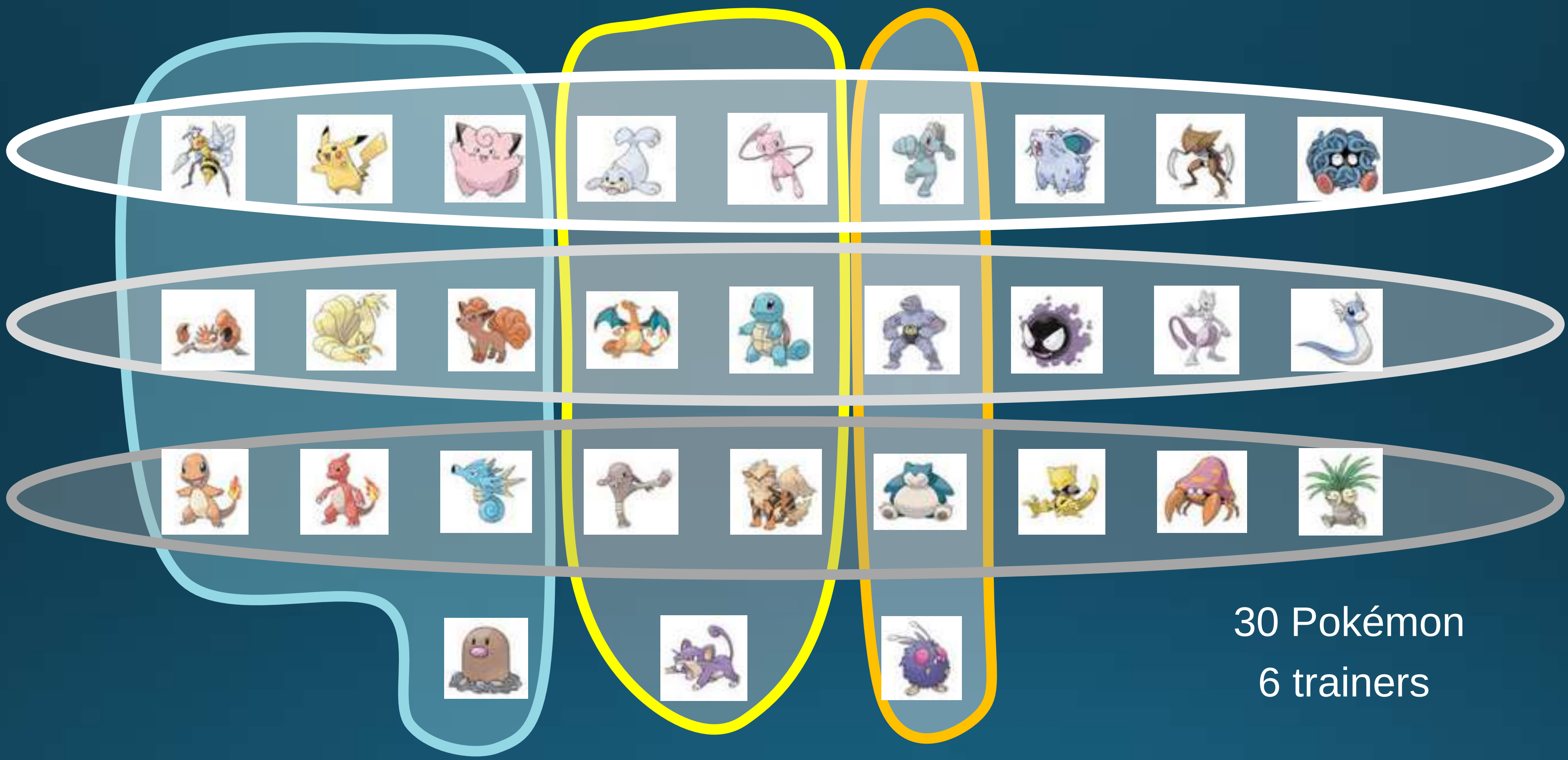
We could try to be greedy again…

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

GreedyCoverage: 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 $r_i = |P^*| -$ # Pokémon covered after i 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 ith 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 $(1-1/k)^k \leq 1/e$, we get $r_k \leq |P^*|/e$

Thus Greedy covers at least $|P^*| - |P^*|/e = (1-1/e) \cdot |P^*|$ Pokémon.

This completes the proof that Greedy is a $(1-1/e)$-approximation algorithm.

Today: A case study of approximation algorithms

1. A 2-approximation algorithm for Vertex-Cover.
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

1. A 2-approximation algorithm for Vertex-Cover.

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":

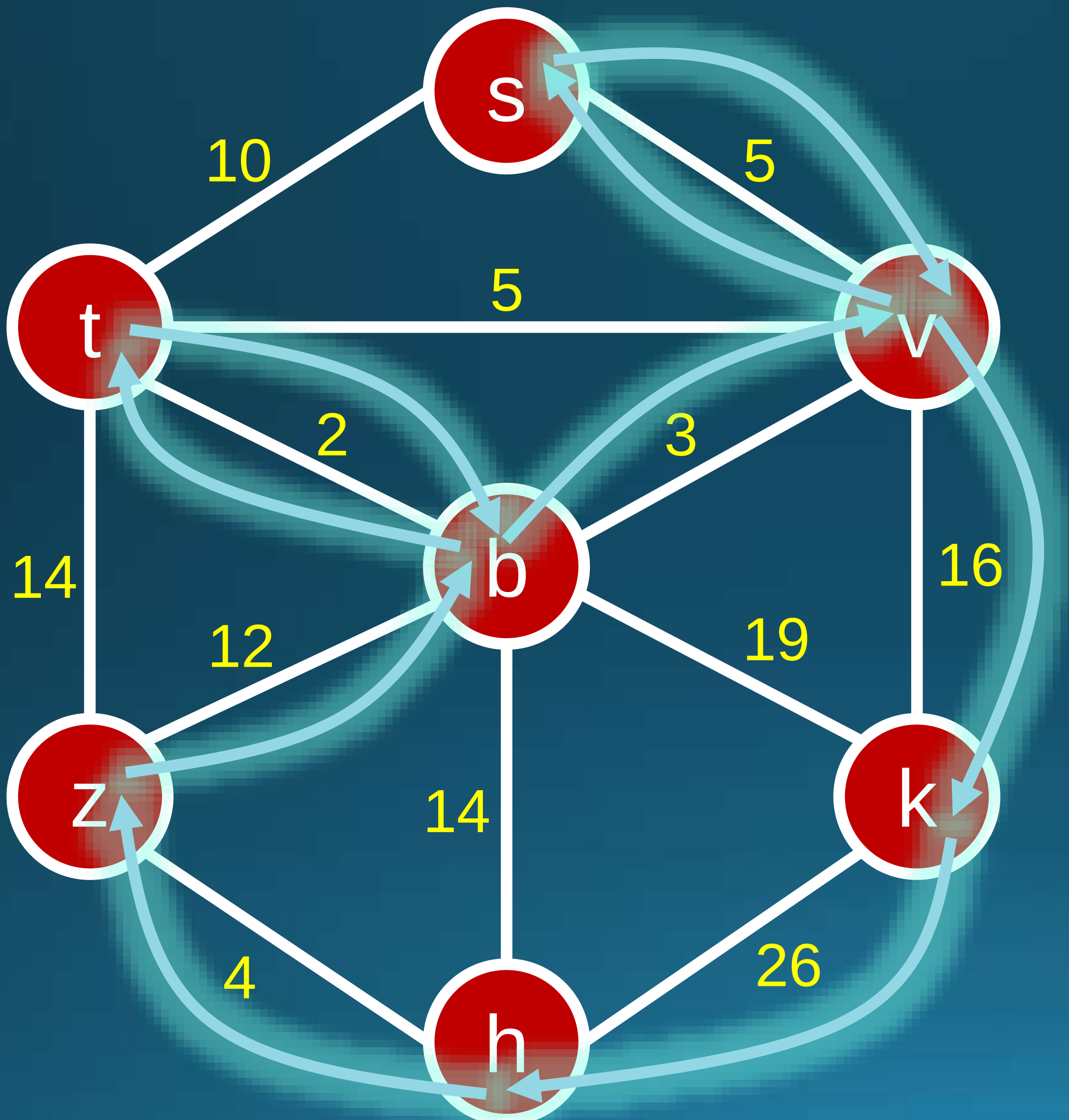
(distance between two nodes is shortest path in graph)

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



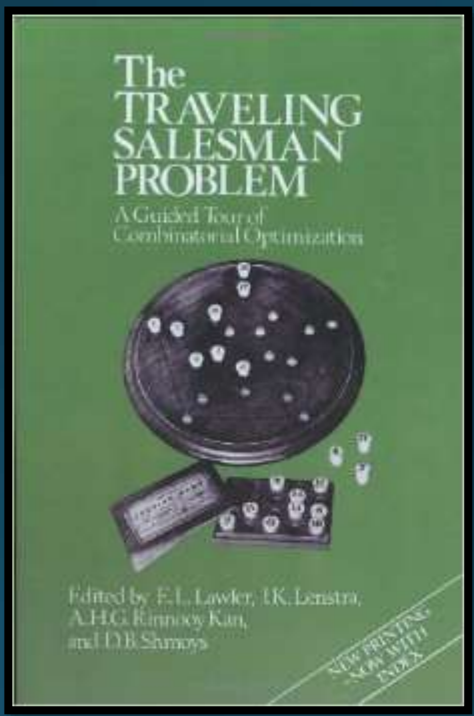
Cheapest tour:

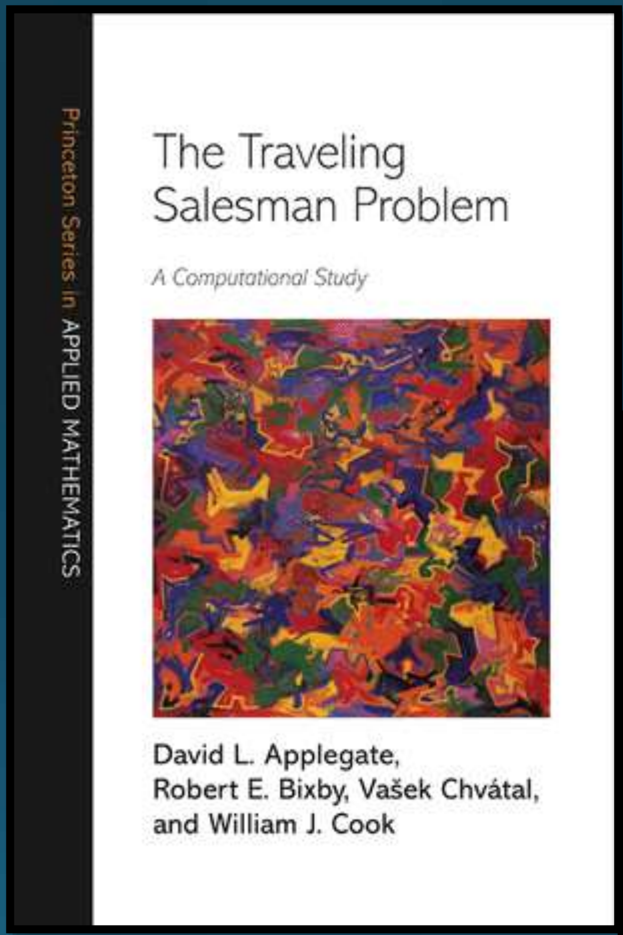
3
\+ 5
\+ 5
\+ 16
\+ 26
\+ 4
\+ 12
\+ 2
\+ 2
= 71

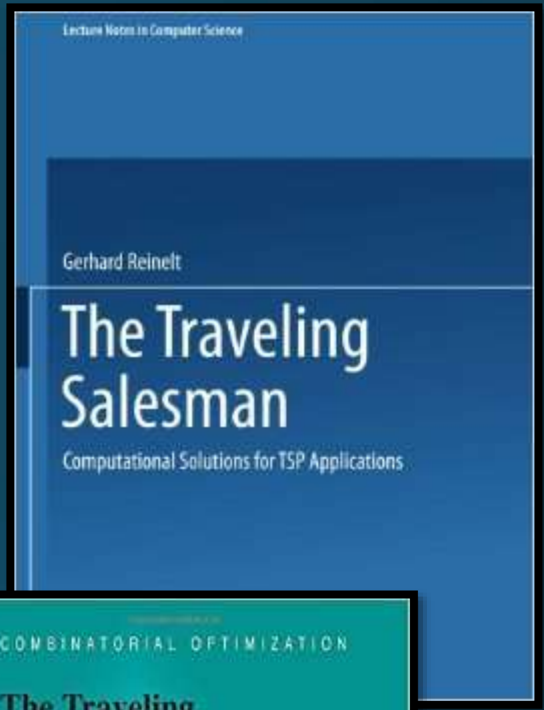
TSP is probably the most famous NP-complete problem.

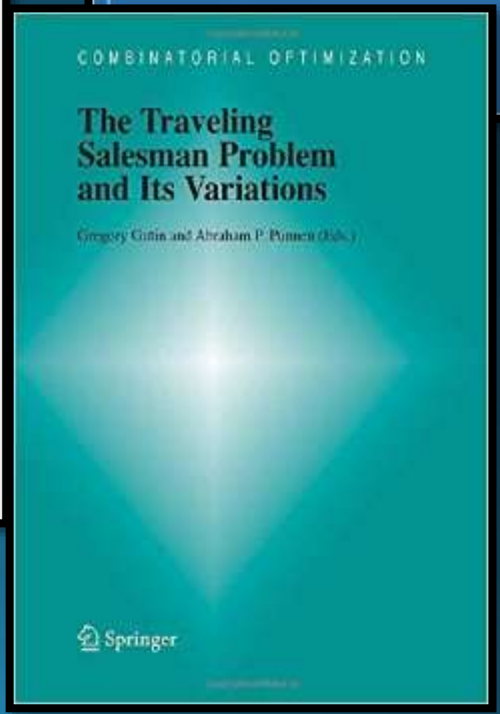
It has inspired many things…

Textbooks

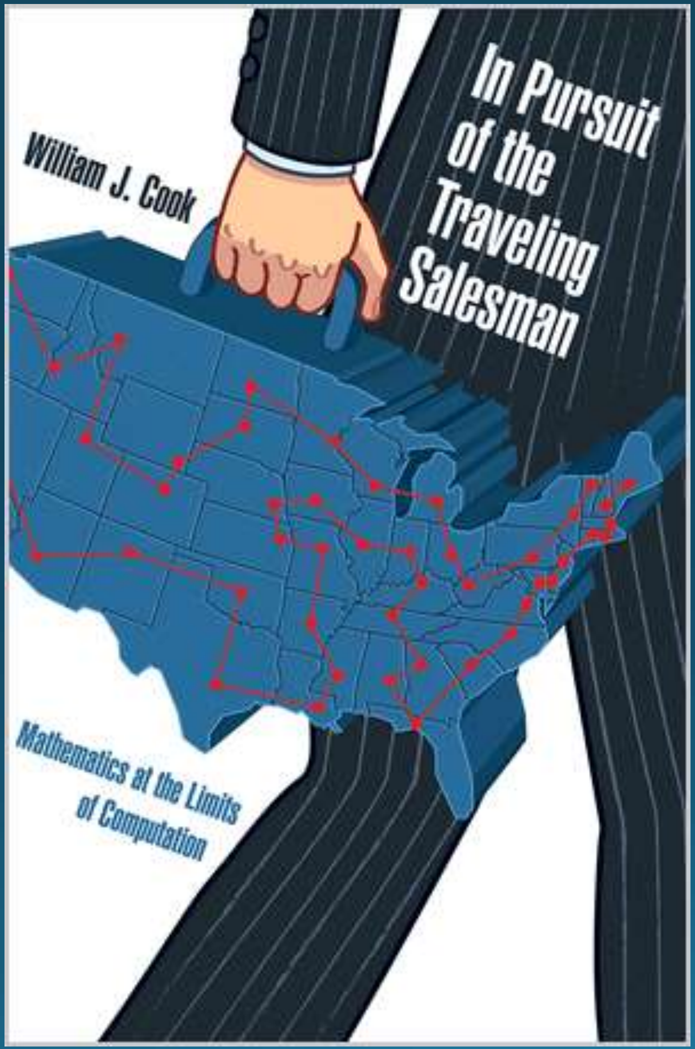

The TRAVELING SALESMAN PROBLEM
A Guided Tour of Combinatorial Optimization
Edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys
NEW PRINTING NOW WITH INDEX


Princeton Series in APPLIED MATHEMATICS
The Traveling Salesman Problem
A Computational Study
David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook

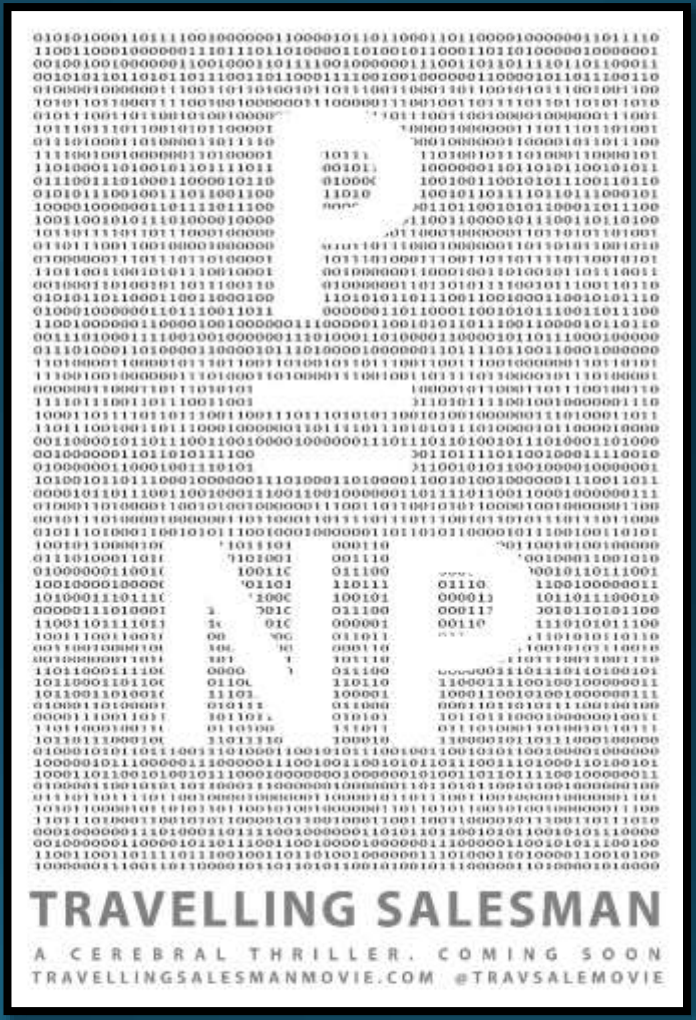

Lecture Notes in Computer Science
Gerhard Reinelt
The Traveling Salesman
Computational Solutions for TSP Applications


COMBINATORIAL OPTIMIZATION
The Traveling Salesman Problem and Its Variations
Springer

“Popular” books

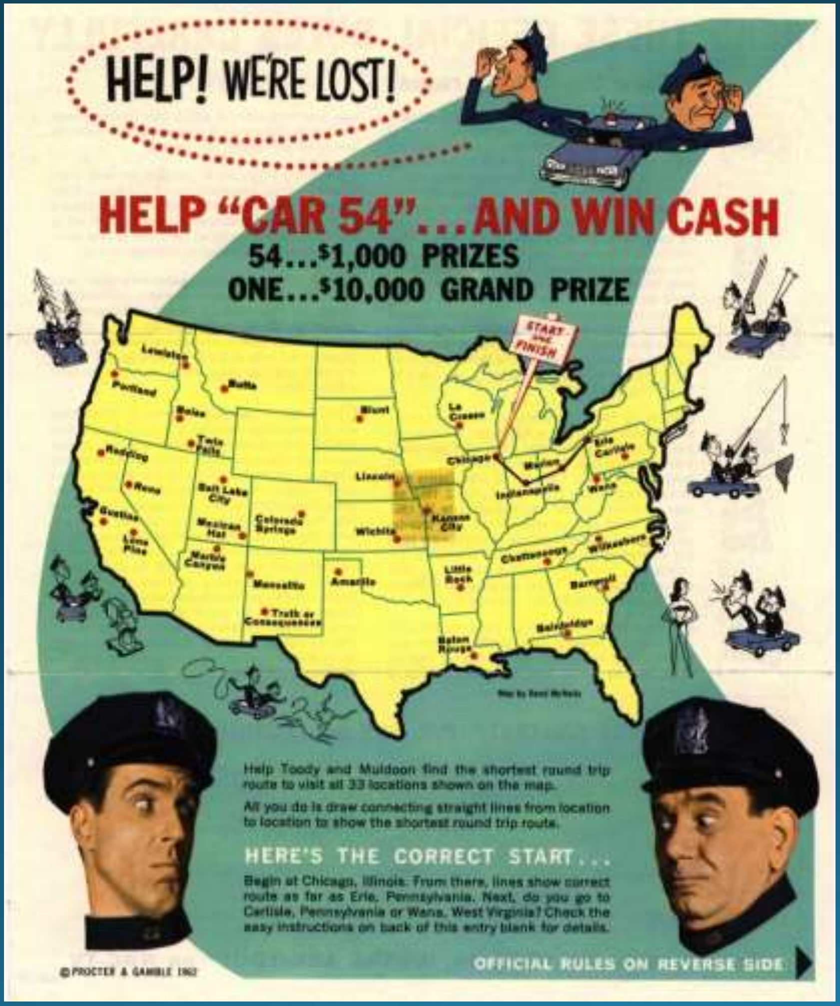


Movies



(Advice: do not watch this movie)

’60s sitcom-themed household-goods conglomerate ad/contests



People genuinely want to solve large instances.

Applications in:

- School bus routing
- Moving farm equipment
- Package delivery
- Space interferometer scheduling
- Circuit board drilling
- Genome sequencing
- …

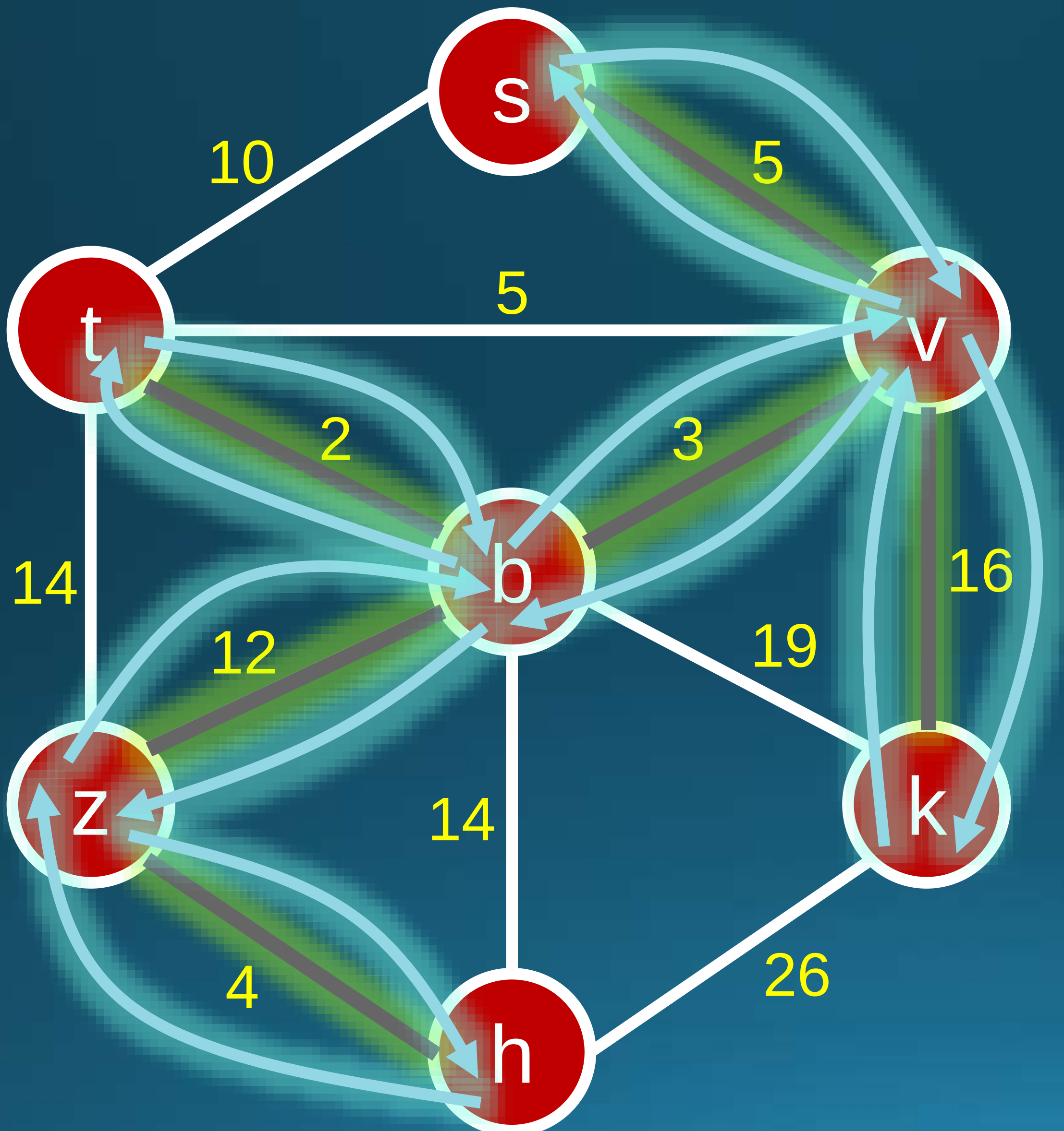
Basic Approximation Algorithm:

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 × MST Cost

(84 in this case)

MST Heuristic

Theorem: MST Heuristic is factor-2 approximation.

Key Claim: Optimal TSP cost ≥ MST Cost always.

This implies the Theorem, since

MST Heuristic Cost = **2** × 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 ≥ 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 1.1 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.

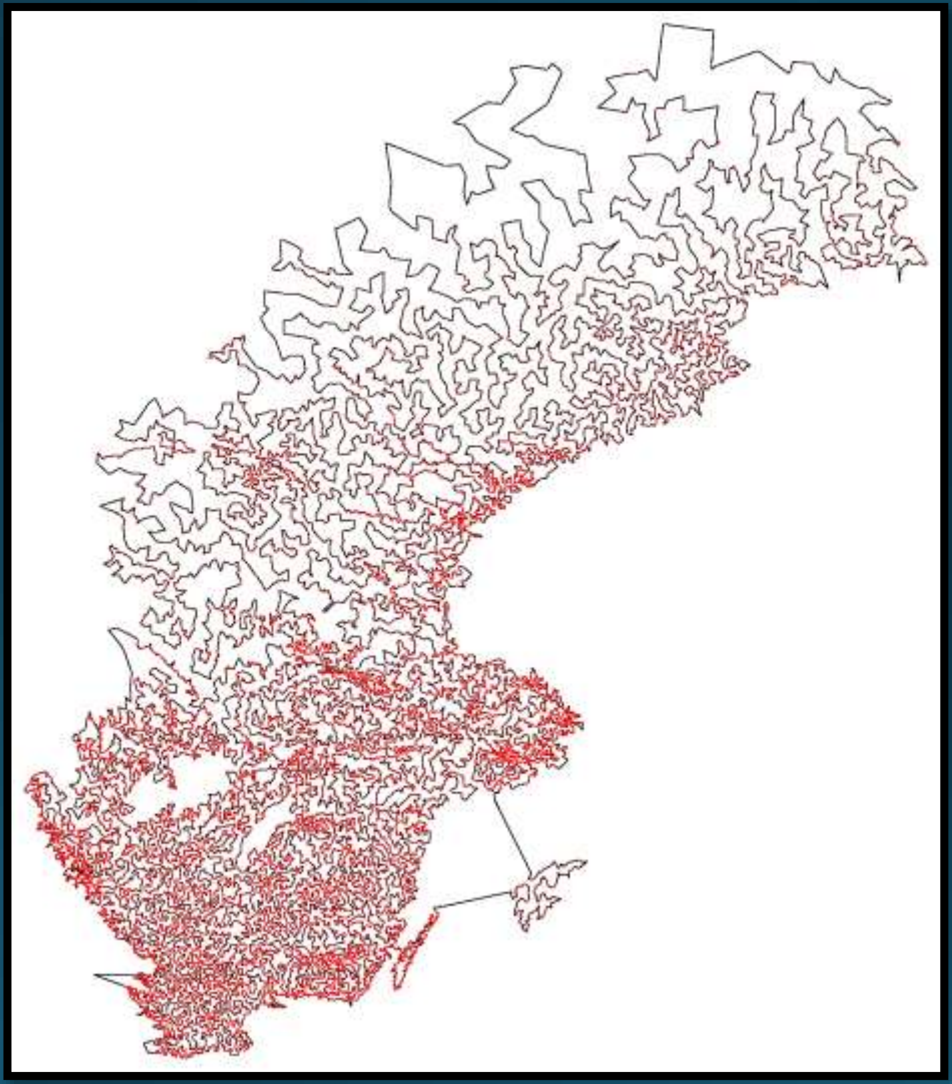
This special case is still NP-hard.

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

(Running time is like $O(n (\log n)^{1/\epsilon})$.)

Euclidean-TSP:

NP-hard, but not **that** hard



n > 10,000
is feasible

Today: A case study of approximation algorithms

1. A 2-approximation algorithm for Vertex-Cover.

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

3. A 1.5-approximation algorithm for Metric-TSP.

4. A $(1+\epsilon)$-approximation alg. for Euclidean-TSP.

Can we do better?

1. A 2-approximation algorithm for Vertex-Cover.
2. A 63% $(1-1/e)$ approximation algorithm for the "k-Coverage Problem".
3. A 1.5-approximation algorithm for Metric-TSP.
4. A $(1+\epsilon)$-approximation alg. for Euclidean-TSP.

Can we do better?

1. A 2-approximation algorithm for Vertex-Cover.

What more do you want?!

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

3. A 1.5-approximation algorithm for Metric-TSP.

4. A $(1+\epsilon)$-approximation alg. for Euclidean-TSP.

Can we do better?

3. A 1.5-approximation algorithm for Metric-TSP.

On one hand:

No improvement in the last 40 years.

On the other hand:

Researchers **strongly** believe we **can** improve the factor of 1.5.

Lots of progress on special cases and related problems in the last 5 years.

I predict an improvement within next 6 years.

QUANTA MAGAZINE
illuminating science

Computer Scientists Take Road Less Traveled

After decades without progress, new shortcuts are discovered in the traveling salesman problem.

Federal Highway Administration

By Erica Klarreich
January 29, 2013

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** β > 1−1/e, it is NP-hard
to factor β-approximate k-Coverage.

Proved in 1998 by Feige,
building on many prior works.

Unwound proof length of reduction: ≈ 100 pages.

Can we do better?

1. A 2-approximation algorithm for Vertex-Cover.

It is open if we can do better.

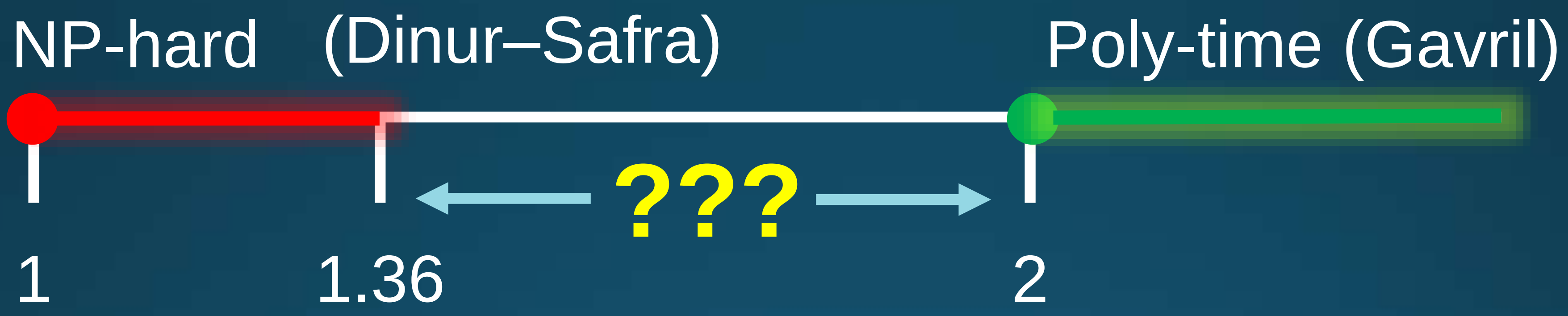
Theorem (Dinur & Safra, 2002, Annals of Math.):

For any $\beta < 10\sqrt{5} - 21 \approx 1.36$,

it is NP-hard to β-approximate Vertex-Cover.

Approximating Vertex-Cover

Approximation Factor



Between 1.36 & 2: unknown.

But a barrier called **“Unique Games Conjecture”** has been identified against improving factor 2 approximation

Unique Games Conjecture

Conjecture made by Subhash Khot in 2002 on intractability of certain approximation problem:

2016 MacArthur Fellow
(among long list of major honors)

Given linear equations of form $x_i - x_j \equiv \alpha_{ij} \pmod{p}$

such that there is an assignment of $x_i's$ with values in $\{0,1,\ldots,p-1\}$ satisfying 0.999^* of the equations,

it is hard to find assignment satisfying γ_p fraction of the equations, for some $\gamma_p \to 0$ as $p \to \infty$

* 0.999 is really (1-ε) for arbitrary ε>0

The Unique Games Conjecture has many striking consequences

No (2-ε)-approximation algo for Vertex Cover [Khot-Regev'03]

No (0.87856+ε)-approx. algo. for Max-Cut! [Khot-Kindler-Mossel-O'Donnell'05]

Single unified algorithm (semidefinite programming) gives optimal approximation for **all** *constraint satisfaction problems* (like Max-Cut, Max-3SAT, etc.) [Raghavendra'08]

And many more implications…

Unlike P vs. NP, no consensus opinion on UGC's validity.

A fascinating chapter in current algorithms & complexity research

Study Guide

Definitions:

Approximation algorithm.

The idea of “greedy” algorithms.

Algorithms and analysis:

Gavril algorithm for Vertex-Cover.

MST Heuristic for TSP.