# Advanced Algorithms

April 9th, 2026

# Logistics

- Assignment 3 is out, due April 21
- Keep thinking about final presentation ideas
- Make a time to meet me and discuss before next ~Tuesday / Friday
- April 21 and 23 I will be out of town, check course webpage

# Brute Force Solutions for Hard Problems

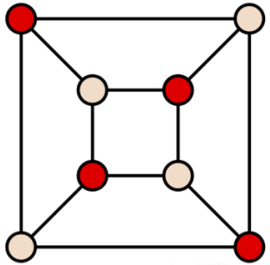
One reliable fall-back strategy for solving hard problems is *brute force*:
(1) enumerate all possible solutions and choose the best (optimization problems)
(2) enumerate all possible solutions and accept if any are correct (decision problems).

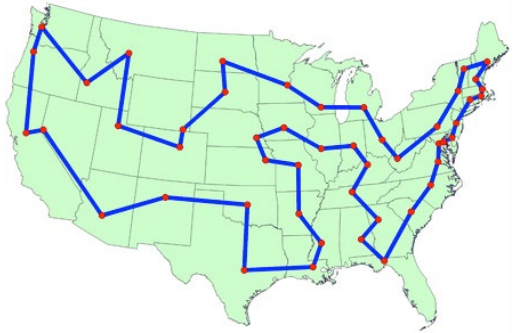
**How long does it take to solve these problem via Brute Force?**

## Minimum Vertex Cover

**Input:** graph G with n nodes

**Output:** smallest possible set of nodes such that every edge is covered by a node we pick





## Traveling Salesperson Problem (TSP)

**Input:** weighted graph G with n nodes

**Output:** Hamiltonian cycle (cycle touching every vertex exactly once) of minimum weight

# How do we deal with (NP-)hard problems?

- Give up
- **Adopt a heuristic:** use an algorithm that appears to work well empirically
- **Approximation Algorithm:** design a fast algorithm guaranteed to get an “approximately” correct answer
- **Restrict the problem:** solve in a tractable subclass / setting of your choosing
- **Formulate as an Integer Program; relax to a Linear Program**
  - In some cases, this amounts to **adopting a heuristic**
  - In other cases, this is provably **correct up to some factor**

**Today:**

- **Exact Exponential Time Algorithms:** Solve the hard problem slowly, but better
- **Parameterized Algorithms:** Measure runtime with respect to some other key parameter. Typical conclusion: “if this structural parameter is very small, then my algorithm is efficient”.
- **Randomized Algorithm** with a small chance of success. Run many times and hope!

# Exact and Parameterized Algorithms for Hard Problems

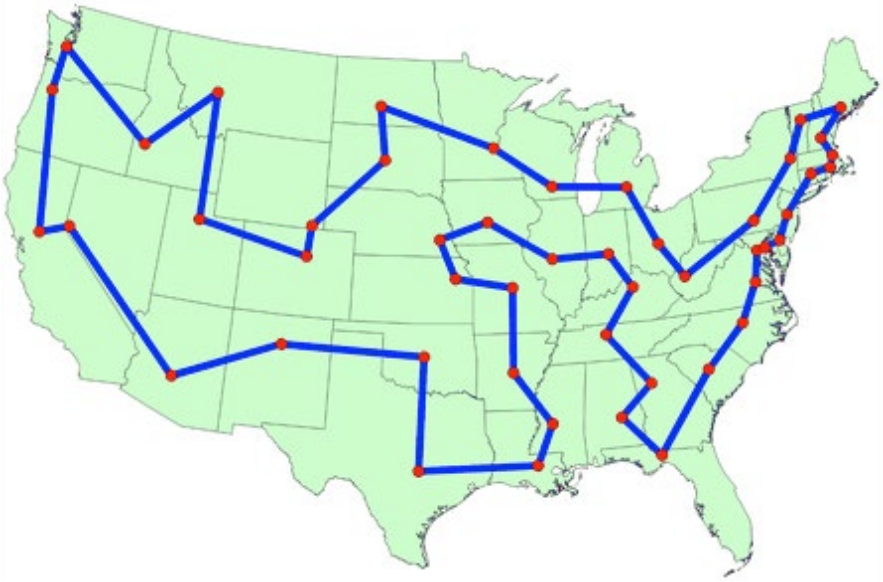
Brute force running time

1. Smarter, smaller "brute force" running time

2. Efficient algorithm *if* a *key parameter* is reasonably sized

$O(2^{(\text{club})} \times \text{poly}(\text{bear}))$

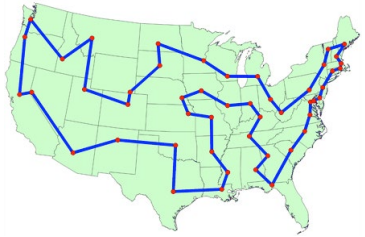
# I. The Traveling Salesperson Problem



**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once, i.e a Hamiltonian cycle

# I. The Traveling Salesperson Problem



$O^*(n!) \approx O^*(n^n / e^n)$

Way bigger than $2^{O(n)}$...

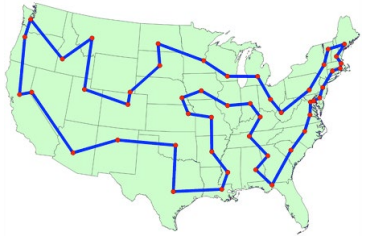
**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once

**Idea:** use dynamic programming!

# I. The Traveling Salesperson Problem





**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once

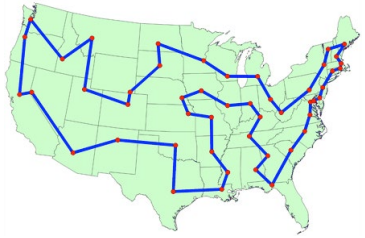
**Idea:** Use dynamic programming

1. Fix arbitrary start node, call it *a*.

2. Define subproblems:

   • For *every* destination *v* and *every* subset S ⊆ *V*, compute the shortest Hamiltonian path from a to v using *exactly* the nodes in S.

# I. The Traveling Salesperson Problem



**Idea:** pick an arbitrary start node *a*.

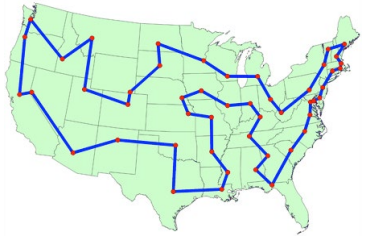
For every destination *v* and every subset S ⊆ *V* with v in S, compute the shortest Hamiltonian path from a to v using *exactly* the nodes in S

**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once

**BHK**(**v**, **S** ⊆ V)

# I. The Traveling Salesperson Problem



**Idea:** pick an arbitrary start node *a*.

For every destination *v* and every subset S ⊆ *V* with v in S, compute the shortest Hamiltonian path from a to v using *exactly* the nodes in S

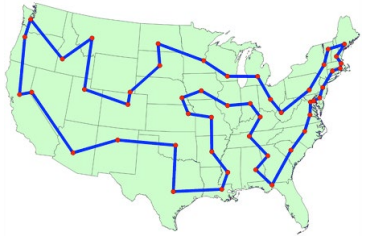
**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once

$$\mathbf{BHK}(v,\ S \subseteq V)$$

$$= \min_{x \in S \setminus v} \{ \mathbf{BHK}(x,\ S \setminus \{v\} + \mathrm{dist}(x,\ v) \}$$

# I. The Traveling Salesperson Problem



**Idea:** pick an arbitrary start node *a*.

For every destination *v* and every subset S ⊆ *V* with v in S, compute the shortest Hamiltonian path from a to v using *exactly* the nodes in S

**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once

$$\mathbf{BHK}(v,\ S \subseteq V)$$

$$= \min_{x \in S \setminus v} \{ \mathbf{BHK}(x,\ S \setminus \{v\} + \mathrm{dist}(x,\ v) \}$$

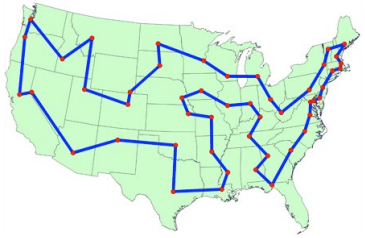
**Base cases?**

**Table-filling?**

**DP table size?**

**Total runtime?**

# I. The Traveling Salesperson Problem



**Idea:** pick an arbitrary start node *a*.

For every destination *v* and every subset S ⊆ *V* with v in S, compute the shortest Hamiltonian path from a to v using exactly the nodes in S

**Input:** A weighted graph *G* with *n* nodes

**Output:** Least-weight (shortest) cycle that visits every node exactly once

$$\mathbf{BHK}(v,\ S \subseteq V) = \min_{x \in S \setminus v} \{ \mathbf{BHK}(x,\ S \setminus \{v\}) + \text{dist}(x,\ v) \}$$

Non-metric!

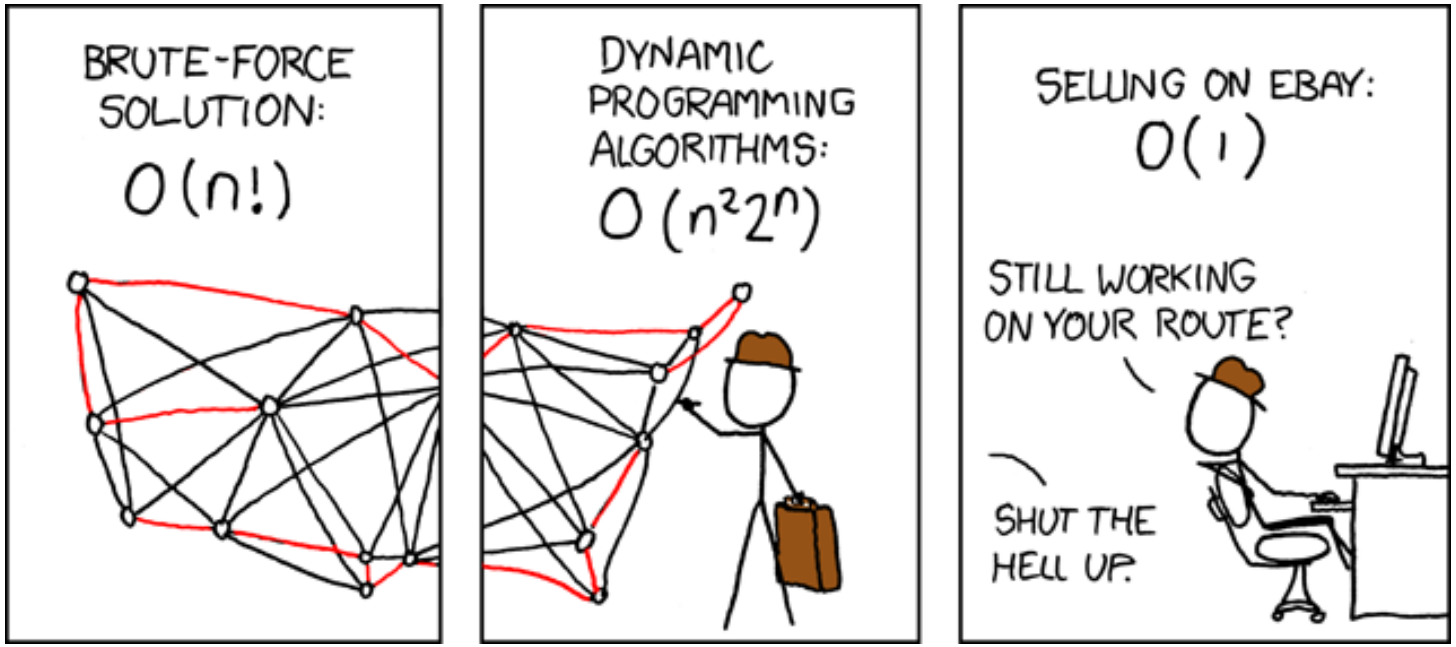
**Base cases?** If |S| = 1, return dist(a, v)

**Table-filling?** Fill in order of increasing |S|

**DP table size?** $n \cdot 2^n$

**Total runtime?** $O(n^2 \cdot 2^n)$

# I. The Traveling Salesperson Problem




Open question: Can we get $2^{0.99999n}$?

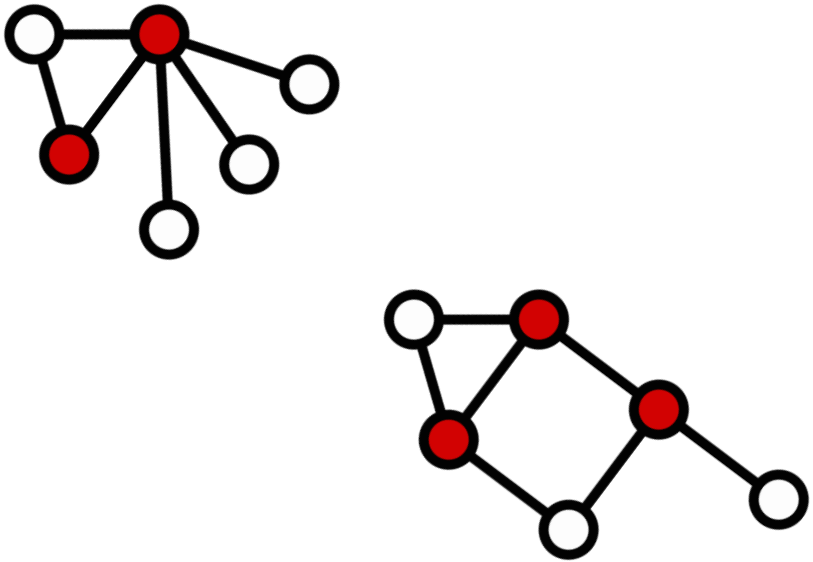
# II. Parameterized Tractability

**“Fixed-Parameter Tractable”:** f(k) $\times$ poly(n)
Read: “efficiently solvable, as long as k is very small”

Typical parameter: solution size

Typical algorithmic idea: find a “kernel” of size $f(k)$

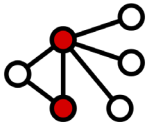
# II. Minimum Vertex Cover

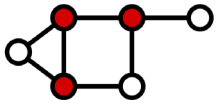


**Input:** A graph *G* with *n* nodes

**Output:** Smallest set of nodes such that every edge is “covered” by an adjacent node in our set

# II. Minimum Vertex Cover





**Brute force:** $O^*(2^n)$
"Guess and check" all vertex subsets

**Input:** graph *G* with *n* nodes
**Output:** Smallest set of nodes such that every edge is "covered" by an adjacent node in our set

**Branching algorithms?** Yes

**ILPs?** Yes

Let's try something new.
What if we want to see if there is a vertex cover of size at most *k?*

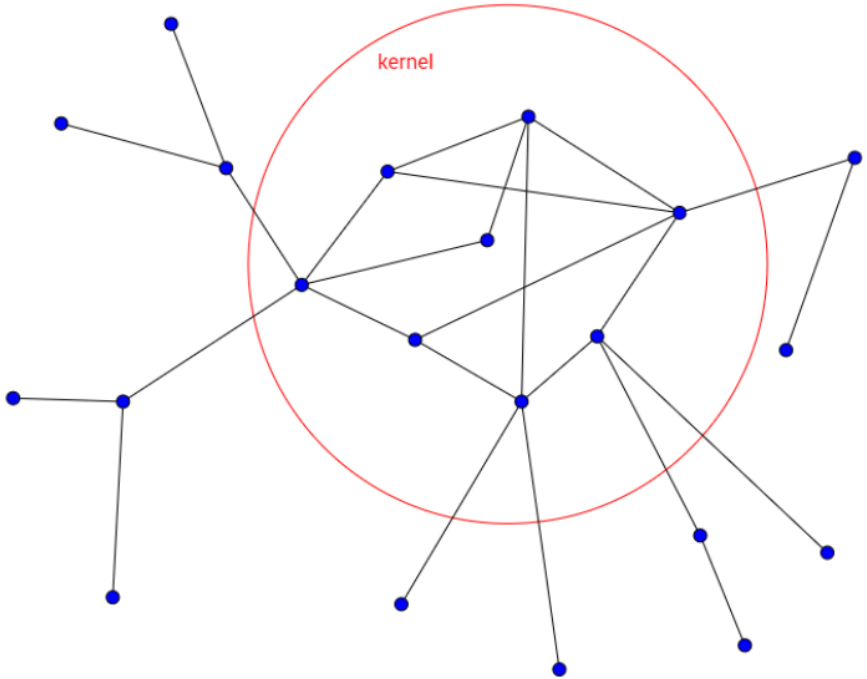
# II. Minimum Vertex Cover

**We want to check if there is a vertex cover of size at most $k$**

Observations:

1. First, notice that we can do brute force in time O*($n^k$). This is not polynomial time!
2. Suppose the graph has a vertex of degree > k. What can I conclude?
3. Hence, we may assume all nodes have degree at most $k$.
4. What if the graph has > $k^2$ edges?
5. Hence, we may assume the graph has at most $k^2$ edges (and hence $\leq 2k^2$ vertices) This is our kernel!
6. Run a brute-force algorithm on this graph. Runtime? $O\left(\binom{2k^2}{k} k^2 + n + m\right)$

# II. Minimum Vertex Cover




The center part of the graph is the kernel, which is “harder” than the rest of the graph.

# II. Minimum Vertex Cover

**Brute force:** $O^*(2^n)$

"Guess and check" all vertex subsets

**Input:** graph *G* with *n* nodes

**Output:** Smallest set of nodes such that every edge is "covered" by an adjacent node in our set

**Branching algorithms?** Yes

**ILPs?** Yes

**Kernelization, then brute force:**

$$O\left(\binom{2k^2}{k} k^2 + n + m\right)$$

**More improvements:**

- $O(kn + 2^k k^{2k+2})$ [BG '93]
- $O(kn + 1.274^k)$ [CKX '10]

# II. Parameterized Tractability

**“Fixed-Parameter Tractable”:** $f(k) \times poly(n)$
Read: “efficiently solvable, as long as k is very small”

**“W[1]-hard”:** Analogous to NP-hard
Read: “unlikely to be Fixed-Parameter Tractable (with respect to a certain parameter)”

Maximum Independent Set is W[1]-hard in size of the independent set!

# III. An Exact Algorithm for 3SAT

Randomized

**Input:** Boolean formula with *n* variables in 3-Conjunctive Normal Form (an AND of 3-OR clauses)

**Output:** A variable assignment satisfying all clauses, or “no” if no satisfying assignment exists

$$(x_1 \overset{\text{OR}}{\vee} \overline{x}_2 \overset{\text{OR}}{\vee} x_{42}) \overset{\text{AND}}{\wedge} (x_2 \vee x_3 \vee \overline{x}_{17}) \wedge \cdots \overset{\text{AND}}{\wedge} (\overline{x}_3 \vee x_5 \vee x_{17})$$

Clause 1 Clause 2 Clause k

# III. An Exact Algorithm for 3SAT

Randomized

1. Choose a random assignment of variables.
2. If all clauses aren't satisfied, then…
   - Choose an unsatisfied clause
   - Flip one variable in that clause (choose at random)

$$(\overset{F}{x_1} \vee \overset{F}{\bar{x}_2} \vee \overset{F}{x_3}) \longrightarrow (\overset{F}{x_1} \vee \overset{F}{\bar{x}_2} \vee \overset{T}{x_3})$$

# III. An Exact Algorithm for 3SAT

Randomized

How likely is this procedure to give us a solution?

**Our random guesses**

| | | |
|---|---|---|
| $x_1 = F$ | | $x_1 = T$ |
| $x_2 = T$ | ✓ | $x_2 = T$ |
| $x_3 = F$ | ✓ | $x_3 = F$ |
| $x_4 = T$ | ✓ | $x_4 = T$ |
| $x_5 = F$ | ✓ | $x_5 = F$ |
| $x_6 = T$ | | $x_6 = F$ |

**Fixed solution *S* (satisfies all clauses)**

Let ***t*** be the number of variables where our guesses agree with ***S***.

$$(\underset{T}{\overset{F}{x_1}} \vee \underset{F}{\overset{F}{\bar{x}_2}} \vee \underset{F}{\overset{F}{x_3}})$$

Our random guesses

*S* (satisfies all clauses)

If our assignment fails to satisfy a clause, we disagree with ***S*** in that clause.
Therefore, at every step, ***t*** *increases* to ***t****+1* with probability at least 1/3!

# III. An Exact Algorithm for 3SAT

Randomized

- Suppose our initial guess gets **t** ≥ 0.5n variables matching ***S***. (We'll assume this for simplicity. If we try several times this is likely to be true at least once.)
- At each step, ***t*** increases to ***t+1*** with probability at least 1/3.
- If ***t*** = *n*, we have found our satisfying solution

Chance of making **0.5n "correct" moves in a row**?

$$(\frac{1}{3})^{0.5n} \geq 1.733^{-n}$$

# A Randomized Exact Algorithm for 3-SAT: Analysis

We now have an efficient procedure which returns a satisfying assignment (if it exists) with probability at least:

$$(\frac{1}{3})^{0.5n} \geq 1.733^{-n}$$

**Final Algorithm:**

Roll the dice:

- We run our procedure $(1.733+0.001)^n$ times
- Hope we get lucky!

**This seems silly, but…**

- Succeeds w.h.p
- Faster than brute force
- Better analysis: $O(1.334^n)$

# Recap

- Three case studies in faster exact algorithms:
  - TSP through Dynamic Programming
  - Vertex Cover through Fixed-Parameter Analysis
  - 3-SAT through randomization

But remember: you can always just solve an Integer Program