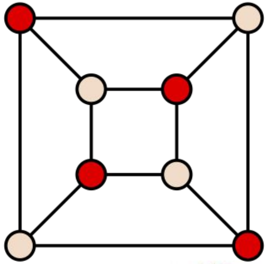
Warm-up: Brute Force Solutions for Hard Problems

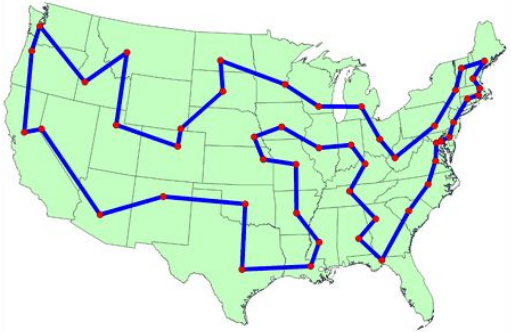
One reliable fall-back strategy for solving hard problems is *brute force*:
(1) enumerate all possible solutions and choose the best (optimization problems)
(2) enumerate all possible certificates and accept if any are correct (decision problems).

How long does it take to solve these problems by brute force?

Maximum Independent Set

Input: graph G with n nodes
Output: largest possible set of nodes such that no two nodes are connected by an edge





Traveling Salesperson Problem (TSP)

Input: weighted graph G with n nodes
Output: Hamiltonian cycle (cycle touching every vertex exactly once) of minimum weight

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^() × poly())

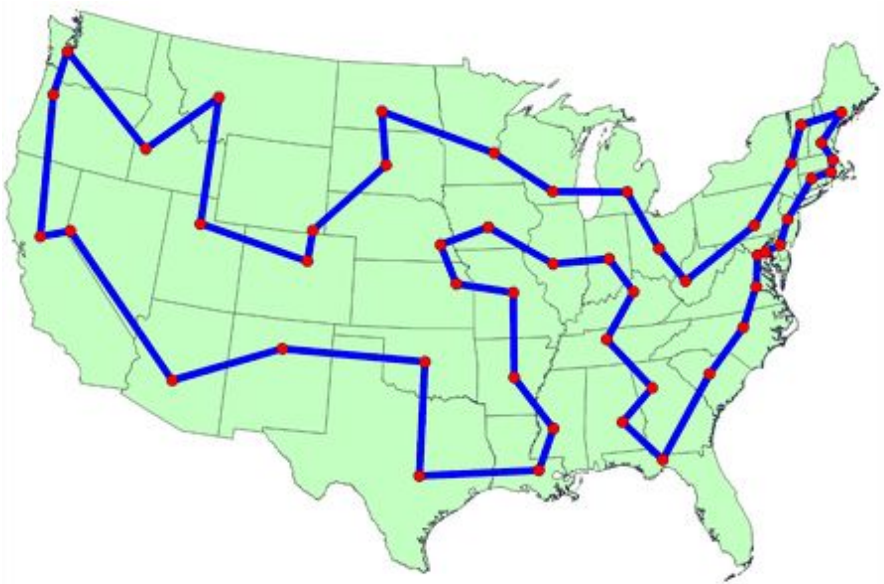
How do we deal with (NP-)hard problems?

- Give up 🤷
- **Adopt a heuristic:** use an algorithm that appears to work well empirically
- **Find a pretty good solution:** design a fast algorithm guaranteed to get an "approximately" correct answer (**Approximation Algorithms)**
- **Restrict the problem:** solve in a tractable subclass / setting
- **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**
  - Other general-purpose ways to express a constraint satisfaction problem

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 *really small*, then my algorithm is efficient".

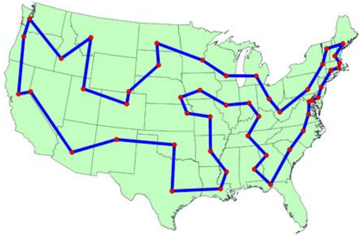
I. The Traveling Salesperson Problem



Input: A weighted graph G with n nodes

Output: Least-weight (shortest) cycle that visits every node exactly once (a “Hamiltonian” cycle)

I. The Traveling Salesperson Problem



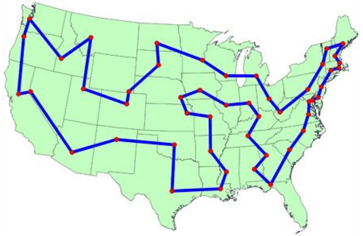
$O^*(n!) \sim= O^*(n^n / e^n)$

Way, *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

I. The Traveling Salesperson Problem



Input: A weighted graph G with n nodes

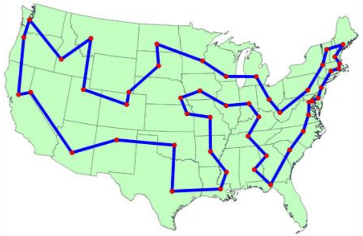
Output: Least-weight (shortest) cycle that visits every node exactly once

Idea: pick an arbitrary start node a.

Using dynamic programming:

For *every* destination t and *every* subset $S \subseteq V$, compute the shortest path using *exactly* the nodes in S.

I. The Traveling Salesperson Problem



Idea: pick an arbitrary start node *a*.

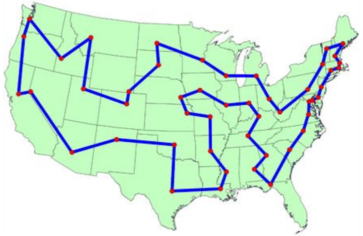
For *every* destination *t* and *every* subset S ⊆ *V*, compute the shortest path using *exactly* the nodes in S.

BHK(a, j, S ⊆ V)

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

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

I. The Traveling Salesperson Problem



Idea: pick an arbitrary start node *a*.

For *every* destination *t* and *every* subset S ⊆ *V*, compute the shortest path 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

$$\mathtt{BHK}(a,\ j,\ S \subseteq V)$$

$$= \min_{v \in S} \left\{ \mathtt{BHK}(a,\ v,\ S \setminus \{v\}) + \mathtt{dist}(v,\ j) \right\}$$

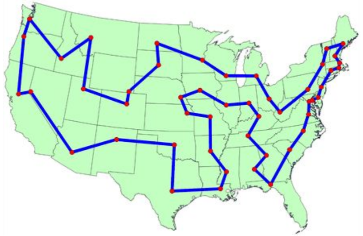
Base cases? S = {}; return dist(a, j)

Table-filling? Increasing |S|

DP table size? $O(n\ 2^n)$

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

I. The Traveling Salesperson Problem



Idea: pick an arbitrary start node *a*.
For *every* destination *t* and *every* subset S ⊆ *V*, compute the shortest path using *exactly* the nodes in S.

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

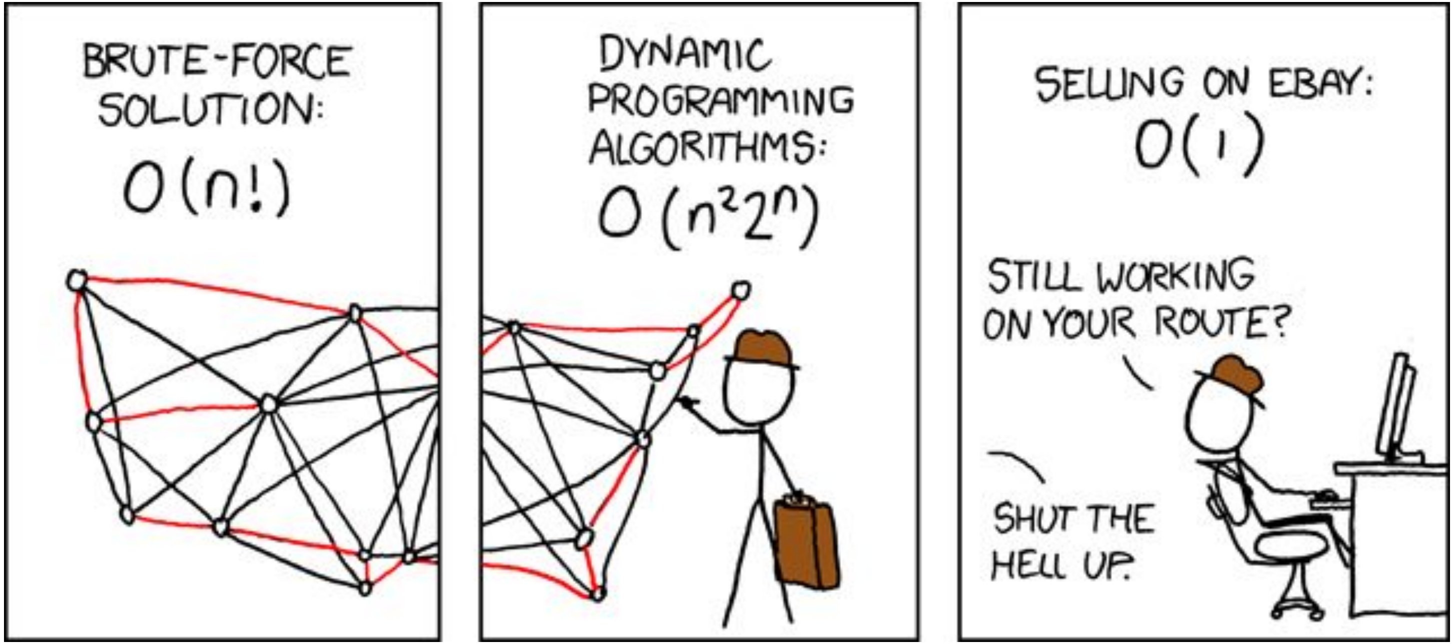
Output: Least-weight (shortest) cycle that visits every node exactly once

$$\mathrm{BHK}(a,\ j,\ S \subseteq V)$$
$$= \min_{v \in S} \left\{ \mathrm{BHK}(a,\ v,\ S \setminus \{v\}) + \mathrm{dist}(v,\ j) \right\}$$

Base cases? S = {}; return dist(a, j)
Table-filling? Increasing |S|
DP table size? $O(n\,2^n)$
Total runtime? $O(n^2\,2^n)$

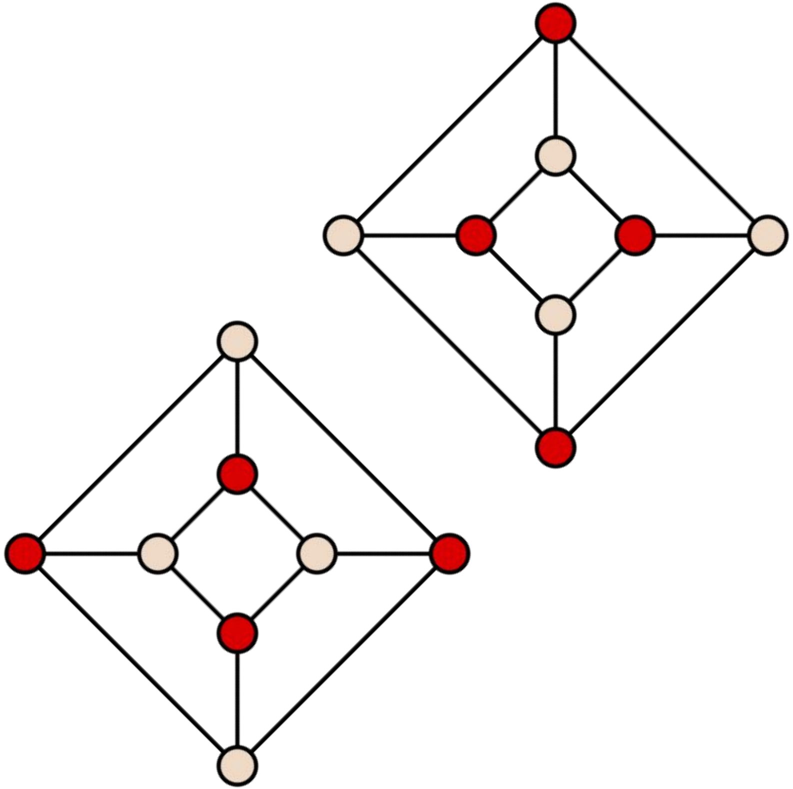
How should we feel about this?

I. The Traveling Salesperson Problem



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

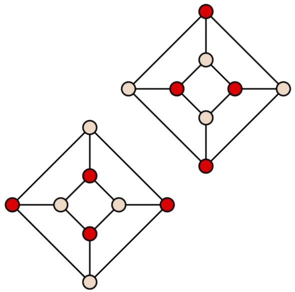
II. Maximum Independent Set



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

Output: Largest set of nodes such that no two nodes are connected by an edge

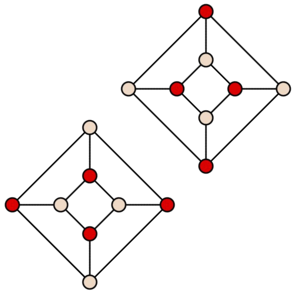
Branch and Bound for Independent Set



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

"Guess and check" all vertex subsets

Branch and Bound for Independent Set

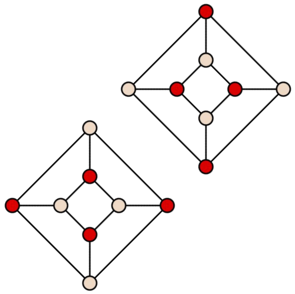


Brute force: $O^*(2^n)$
"Guess and check" all vertex subsets

Equivalent Branching Algorithm:

- Select a node *v*
  - **Include** *v*, exclude v's neighbors, recurse
  - **Exclude** v, recurse

Branch and Bound for Independent Set



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

Equivalent Branching Algorithm:

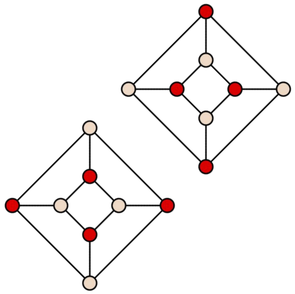
- Select a node *v*
  - **Include** *v*, exclude v's neighbors, recurse
  - **Exclude** v, recurse

Runtime recurrence:
$T(n) < 2\ T(n-1) + O(1)$

Not tight. Can we do better?

Board work: Maximum Independent Set (FK 1.2)

Branch and Bound for Independent Set

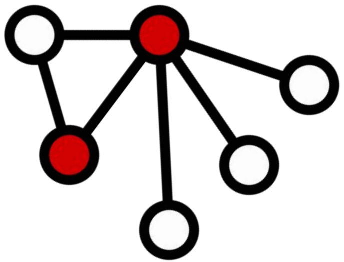


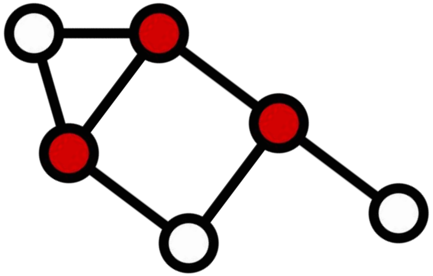
Brute force: $O^*(2^n)$
"Guess and check" all vertex subsets

Many further improvements!

- $O^*(1.286^n)$ [Tarjan '72]
- $O^*(1.260^n)$ [TT '77]
- $O^*(1.232^n)$ [Jian '86]
- $O^*(1.211^n)$ [Robson '86]
- …
- $O^*(1.1996^n)$ [XN '17]

III. Minimum Vertex Cover

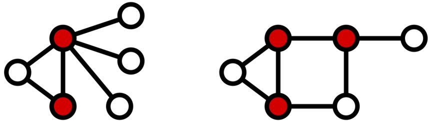




Input: A graph G with n nodes

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

III. Minimum Vertex Cover



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

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

Branching algorithms? Yes

ILPs? Yes

Let’s try something new. What if we want a vertex cover of size at most *k*?

Board work: Vertex Cover (KT 10.1)

Brute force: $O(n^k)$ possible covers * $O(kn)$ to check if the cover is correct.

What does $O(kn^{k+1})$ mean with respect to "polynomial time"?

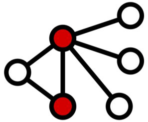
Observation: each vertex can "cover" at most *n* edges. We can reject unless our graph has at most kn edges.

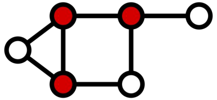
Consider an edge (u, v). Either:

- u is in our cover; thus we can cover G - {u} with k-1 vertices
- v is in our cover; thus we can cover G - {v} with k-1 vertices.

$O(2^k kn)$ time!

III. Minimum Vertex Cover





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

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

Branching algorithms? Yes

ILPs? Yes

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

More improvements!

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

Aside: Parameterized Tractability

“Fixed-Parameter Tractable”: $f(k) \times \text{poly}(n)$

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! So we’ll stick to branch and bound…

IV. Bonus: 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

$$\underbrace{(x_1 \overset{\text{OR}}{\vee} \overline{x}_2 \overset{\text{OR}}{\vee} x_{42})}_{\text{Clause 1}} \underset{\text{AND}}{\wedge} \underbrace{(x_2 \vee x_3 \vee \overline{x}_{17})}_{\text{Clause 2}} \wedge \cdots \underset{\text{AND}}{\wedge} \underbrace{(\overline{x}_3 \vee x_5 \vee x_{17})}_{\text{Clause k}}$$

IV. Bonus: 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)

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

IV. Bonus: An Exact Algorithm for 3SAT

Randomized!

How likely is this procedure to give us a solution?

Our random guesses

$$\begin{aligned} x_1 &= F \\ x_2 &= T \\ x_3 &= F \\ x_4 &= T \\ x_5 &= F \\ x_6 &= T \end{aligned} \qquad \begin{aligned} x_1 &= T \\ x_2 &= T \\ x_3 &= F \\ x_4 &= T \\ x_5 &= F \\ x_6 &= F \end{aligned}$$

Fixed solution *S* (satisfies all clauses)

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

$$\begin{array}{ccccccl} F & & F & & F & & \text{Our random guesses} \\ (x_1 & \vee & \bar{x}_2 & \vee & x_3) & & \\ T & & F & & F & & S \text{ (satisfies all clauses)} \end{array}$$

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!

IV. Bonus: 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**?

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

A Randomized Exact Algorithm for 3-SAT: Analysis

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

Final Algorithm:

- Try 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: $O(1.334^n)$!

So you're saying there's a chance…