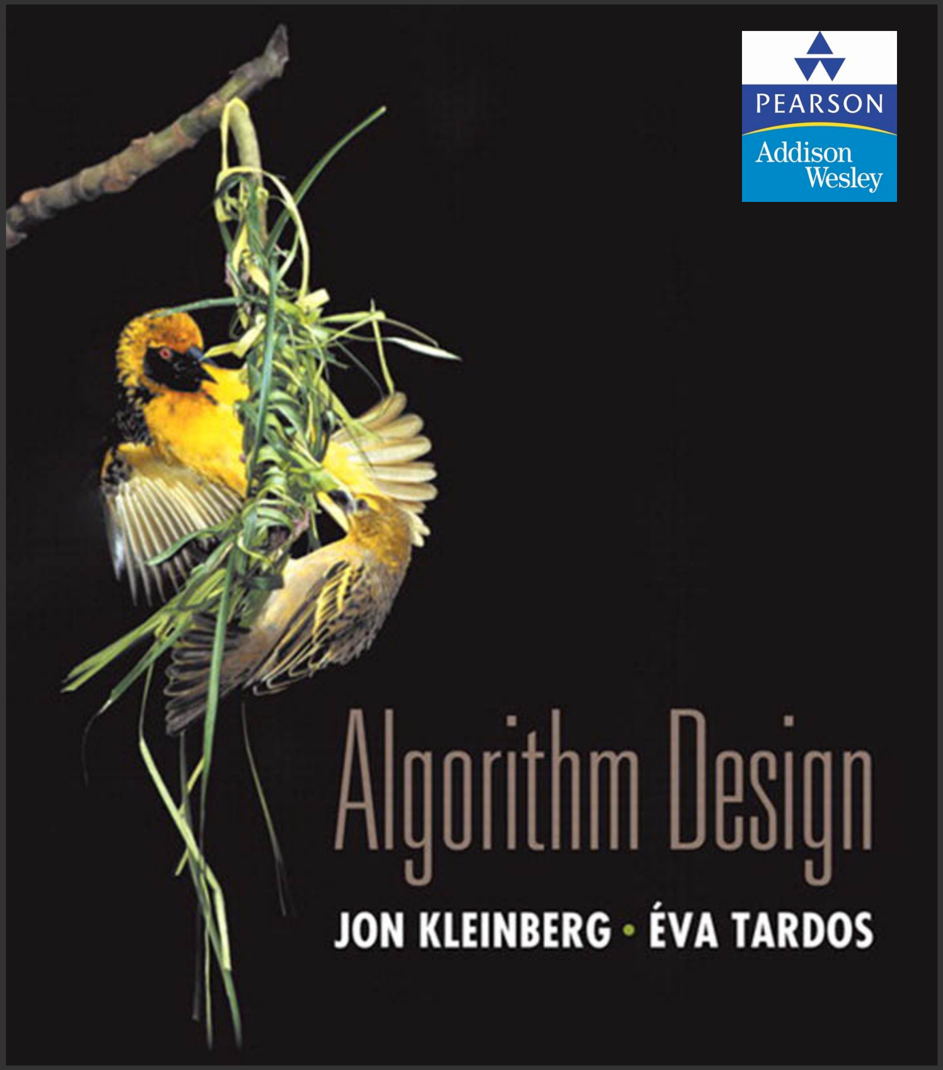



Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

# INTRACTABILITY III

- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

Last updated on 5/5/18 5:15 AM

# Coping with NP-completeness

Q. Suppose I need to solve an **NP**-hard problem. What should I do?

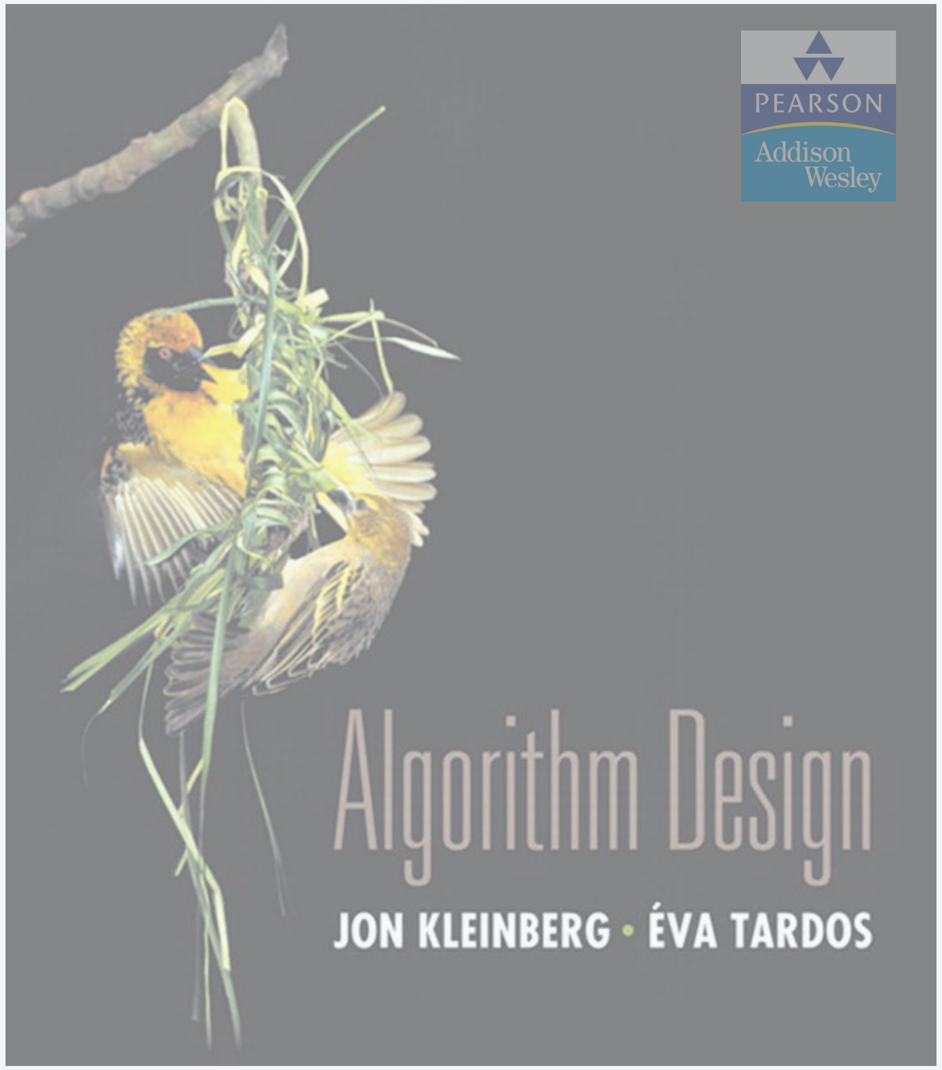
A. Sacrifice one of three desired features.

i. Solve arbitrary instances of the problem.
ii. Solve problem to optimality.
iii. Solve problem in polynomial time.

Coping strategies.

i. Design algorithms for special cases of the problem.
ii. Design approximation algorithms or heuristics.
iii. Design algorithms that may take exponential time.

using greedy, dynamic programming, divide-and-conquer, and network flow algorithms!




SECTION 10.2

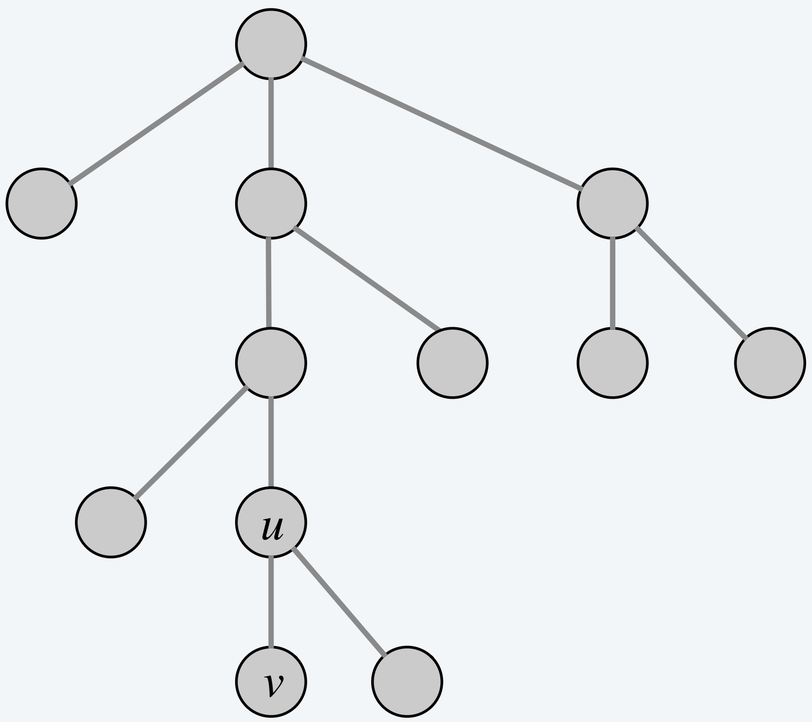
# INTRACTABILITY III

- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

# Independent set on trees

Independent set on trees. Given a tree, find a max-cardinality subset of nodes such that no two are adjacent.

Fact. A tree has at least one node that is a leaf (degree = 1).




Key observation. If node $v$ is a leaf, there exists a max-cardinality independent set containing $v$.

Pf. [exchange argument]

- Consider a max-cardinality independent set $S$.
- If $v \in S$, we're done.
- Otherwise, let $(u, v)$ denote the lone edge incident to $v$.
  - if $u \notin S$ and $v \notin S$, then $S \cup \{ v \}$ is independent $\Rightarrow$ $S$ not maximum
  - if $u \in S$ and $v \notin S$, then $S \cup \{ v \} - \{ u \}$ is independent ▪

# Independent set on trees: greedy algorithm

**Theorem.** The greedy algorithm finds a max-cardinality independent set in forests (and hence trees).

**Pf.** Correctness follows from the previous key observation. ▪

INDEPENDENT-SET-IN-A-FOREST($F$)

---

$S \leftarrow \varnothing$.

WHILE ($F$ has at least 1 edge)

  Let $v$ be a leaf node and let $(u, v)$ be the lone edge incident to $v$.

  $S \leftarrow S \cup \{ v \}$.

  $F \leftarrow F - \{ u, v \}$. ⟵ delete both $u$ and $v$ (including all incident edges)

RETURN $S \cup \{$ nodes remaining in $F$ $\}$.

**Remark.** Can implement in $O(n)$ time by maintaining nodes of degree 1.

**How might the greedy algorithm fail if the graph is not a tree/forest?**

**A.** Might get stuck.

**B.** Might take exponential time.

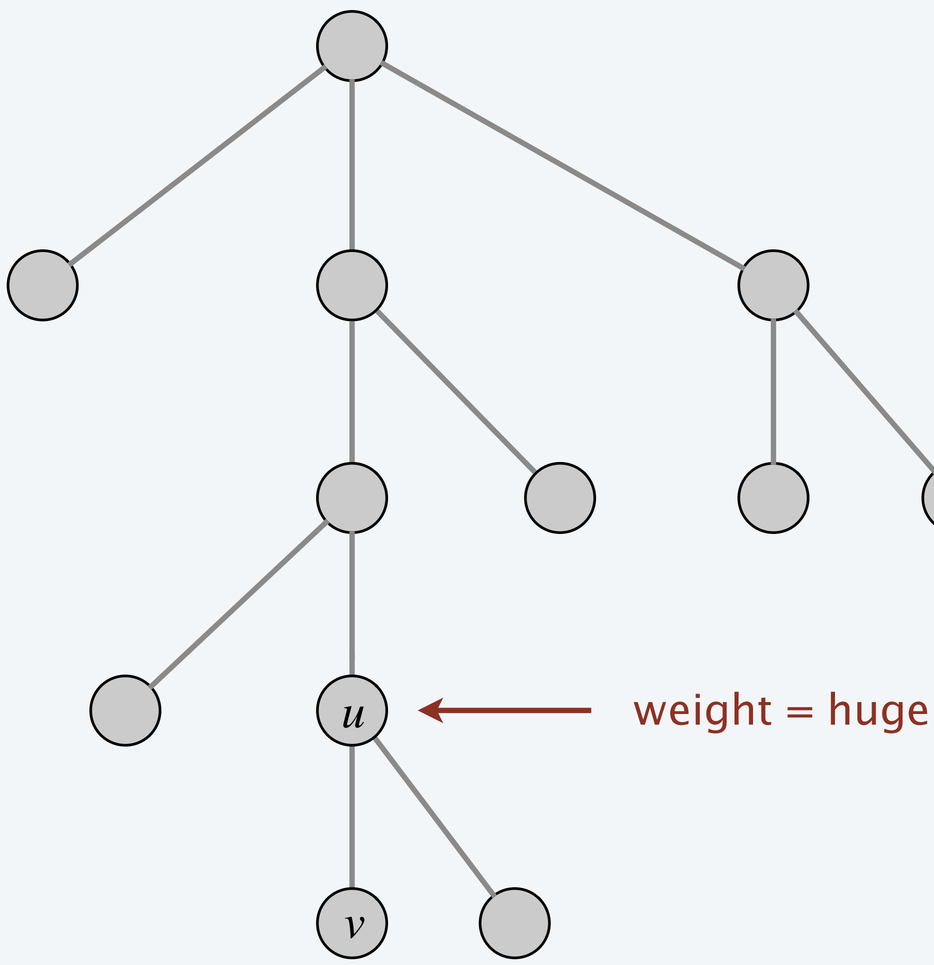
**C.** Might produce a suboptimal independent set.

**D.** Any of the above.

# Weighted independent set on trees

Weighted independent set on trees. Given a tree and node weights $w_v \geq 0$, find an independent set $S$ that maximizes $\Sigma_{v \in S}\, w_v$.

Greedy algorithm can fail spectacularly.



# Weighted independent set on trees

Weighted independent set on trees. Given a tree and node weights $w_v \geq 0$, find an independent set $S$ that maximizes $\Sigma_{v \in S}\, w_v$.

Dynamic-programming solution. Root tree at some node, say $r$.

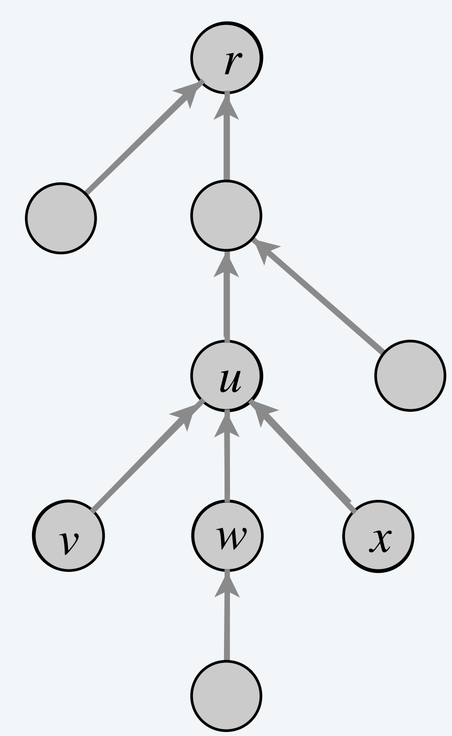
- $OPT_{in}(u)$ = max-weight IS in subtree rooted at $u$, containing $u$.
- $OPT_{out}(u)$ = max-weight IS in subtree rooted at $u$, not containing $u$.
- Goal: $\max \{\, OPT_{in}(r),\ OPT_{out}(r) \,\}$.

overlapping subproblems

Bellman equation.

$$OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$$

$$OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max \left\{ OPT_{in}(v),\ OPT_{out}(v) \right\}$$




children(u) = { v, w, x }

# Intractability III: quiz 2

**In which order to solve the subproblems?**

- **A.** Preorder.
- **B.** Postorder.
- **C.** Level order.
- **D.** Any of the above.

# Weighted independent set on trees: dynamic-programming algorithm

**Theorem.** The DP algorithm computes max weight of an independent set in a tree in $O(n)$ time.

can also find independent set itself (not just value)

WEIGHTED-INDEPENDENT-SET-IN-A-TREE $(T)$

Root the tree $T$ at any node $r$.

$S \leftarrow \varnothing$.

FOREACH (node $u$ of $T$ in postorder/topological order)

ensures a node is processed after all of its descendants

  IF ($u$ is a leaf node)

    $M_{in}[u] = w_u$.

    $M_{out}[u] = 0$.

  ELSE

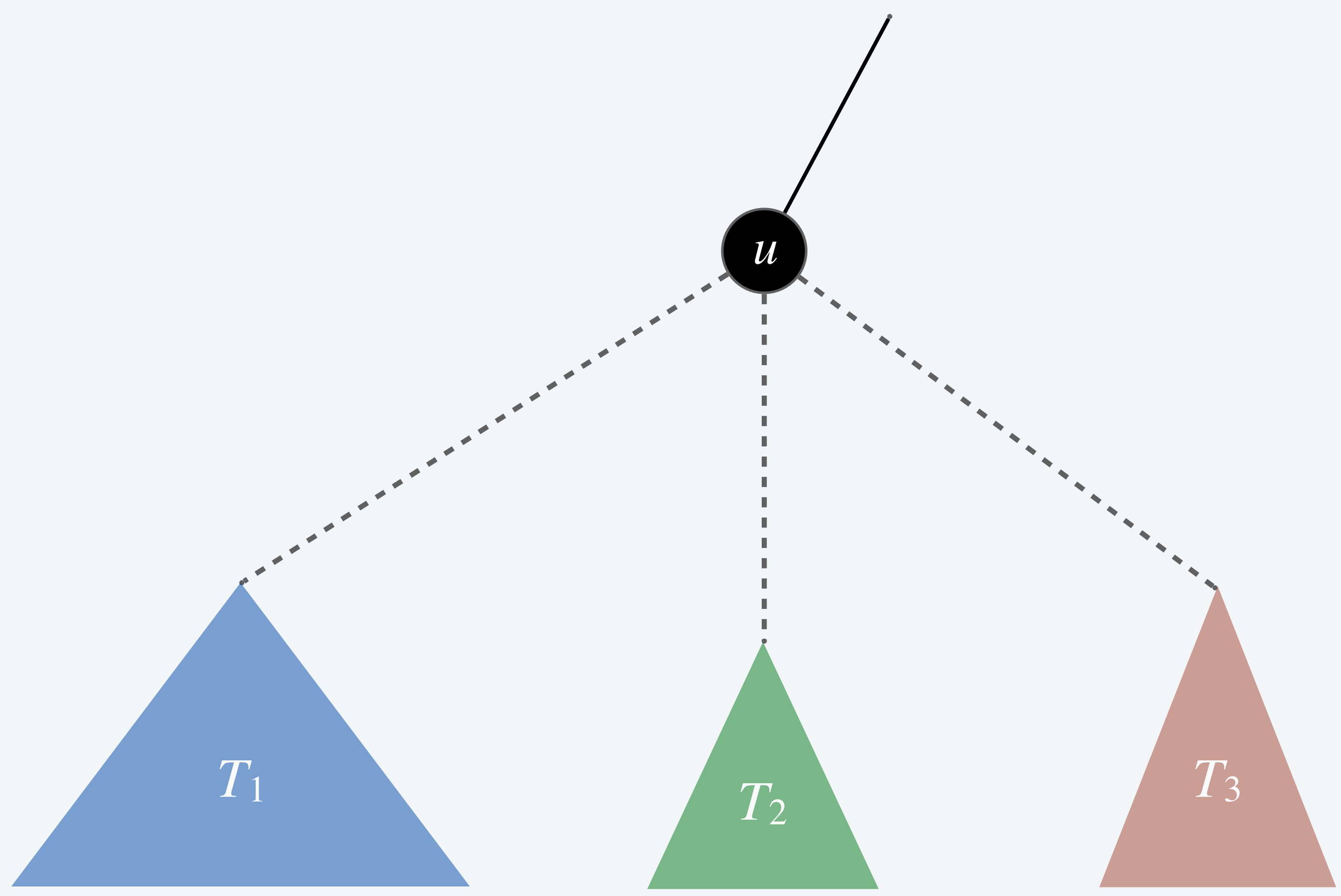
    $M_{in}[u] \ = w_u + \Sigma_{v \in children(u)} M_{out}[v]$.

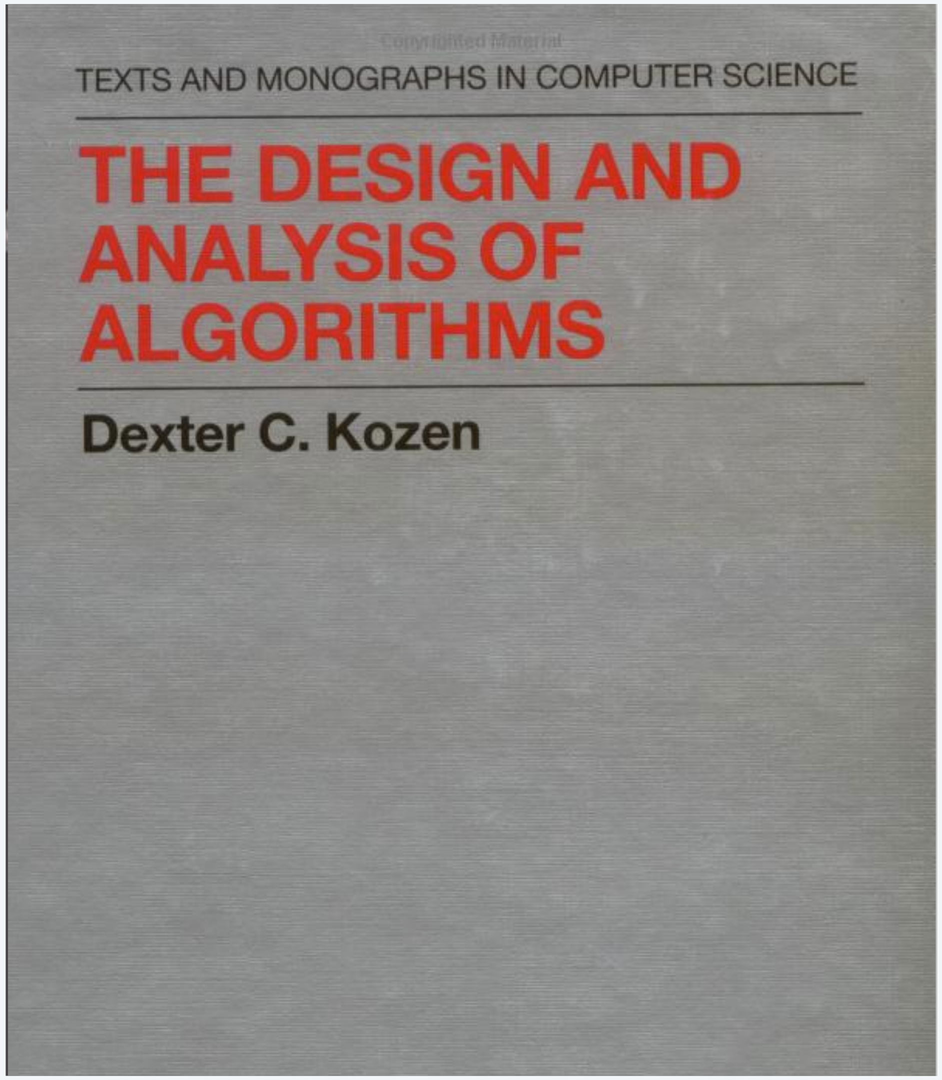
    $M_{out}[u] = \Sigma_{v \in children(u)} \ max \{ M_{in}[v], \ M_{out}[v] \}$.

RETURN $max \{ M_{in}[r], \ M_{out}[r] \}$.

# NP-hard problems on trees: context

Independent set on trees. Tractable because we can find a node that breaks the communication among the subproblems in different subtrees.





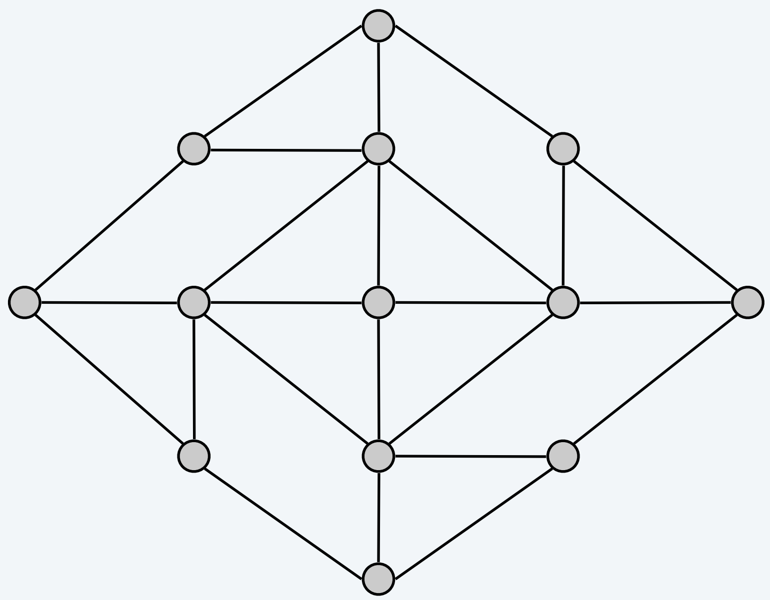

SECTION 23.1

# INTRACTABILITY III

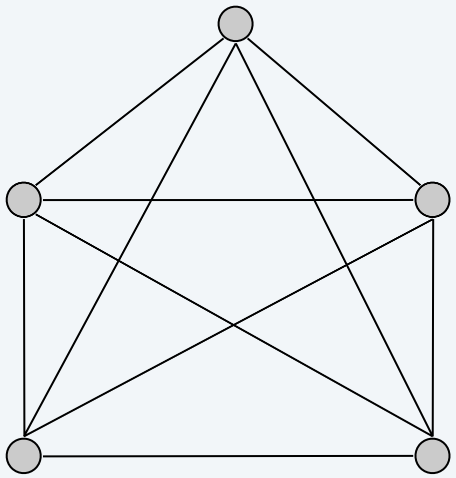
- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

# Planarity

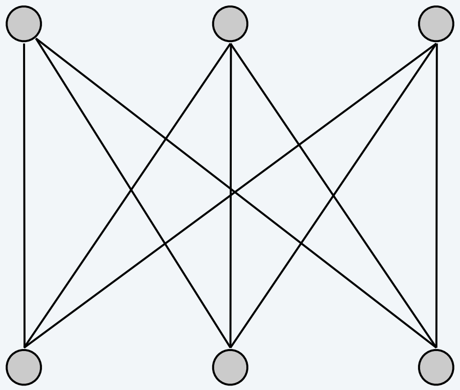
Def. A graph is planar if it can be embedded in the plane in such a way that no two edges cross.



planar



$K_5$ is nonplanar



$K_{3,3}$ is nonplanar

Applications. VLSI circuit design, computer graphics, ...

# Planarity testing

Theorem. [Hopcroft–Tarjan 1974] There exists an $O(n)$ time algorithm to determine whether a graph is planar.

simple planar graph
has at $\leq 3n$ edges

**Efficient Planarity Testing**

JOHN HOPCROFT AND ROBERT TARJAN

*Cornell University, Ithaca, New York*

ABSTRACT. This paper describes an efficient algorithm to determine whether an arbitrary graph $G$ can be embedded in the plane. The algorithm may be viewed as an iterative version of a method originally proposed by Auslander and Parter and correctly formulated by Goldstein. The algorithm uses depth-first search and has $O(V)$ time and space bounds, where $V$ is the number of vertices in $G$. An ALGOL implementation of the algorithm successfully tested graphs with as many as 900 vertices in less than 12 seconds.

# Problems on planar graphs

Fact 0. Many graph problems can be solved faster in planar graphs.
Ex. Shortest paths, max flow, MST, matchings, ...

Fact 1. Some **NP**-complete problems become tractable in planar graphs.
Ex. MAX-CUT, ISING, CLIQUE, GRAPH-ISOMORPHISM, 4-COLOR, ...

Fact 2. Other **NP**-complete problems become easier in planar graphs.
Ex. INDEPENDENT-SET, VERTEX-COVER, TSP, STEINER-TREE, ...

**An $O(n \log n)$ Algorithm for Maximum *st*-Flow in a Directed Planar Graph**

GLENCORA BORRADAILE AND PHILIP KLEIN

*Brown University, Providence, Rhode Island*

Abstract. We give the first correct $O(n \log n)$ algorithm for finding a maximum $st$-flow in a directed planar graph. After a preprocessing step that consists in finding single-source shortest-path distances in the dual, the algorithm consists of repeatedly saturating the leftmost residual $s$-to-$t$ path.

SIAM J. COMPUT.
Vol. 9, No. 3, August 1980


0097-5397/80/0903-0013 $01.00/0

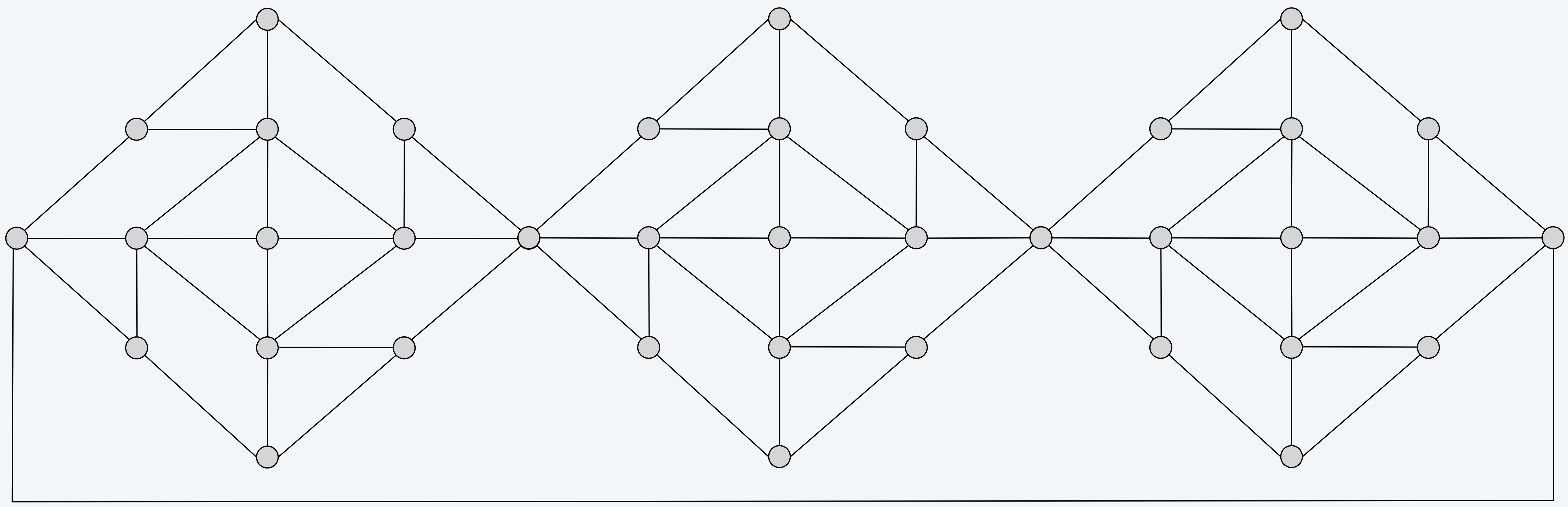
**APPLICATIONS OF A PLANAR SEPARATOR THEOREM***

RICHARD J. LIPTON† AND ROBERT ENDRE TARJAN‡

**Abstract.** Any $n$-vertex planar graph has the property that it can be divided into components of roughly equal size by removing only $O(\sqrt{n})$ vertices. This separator theorem, in combination with a divide-and-conquer strategy, leads to many new complexity results for planar graph problems. This paper describes some of these results.

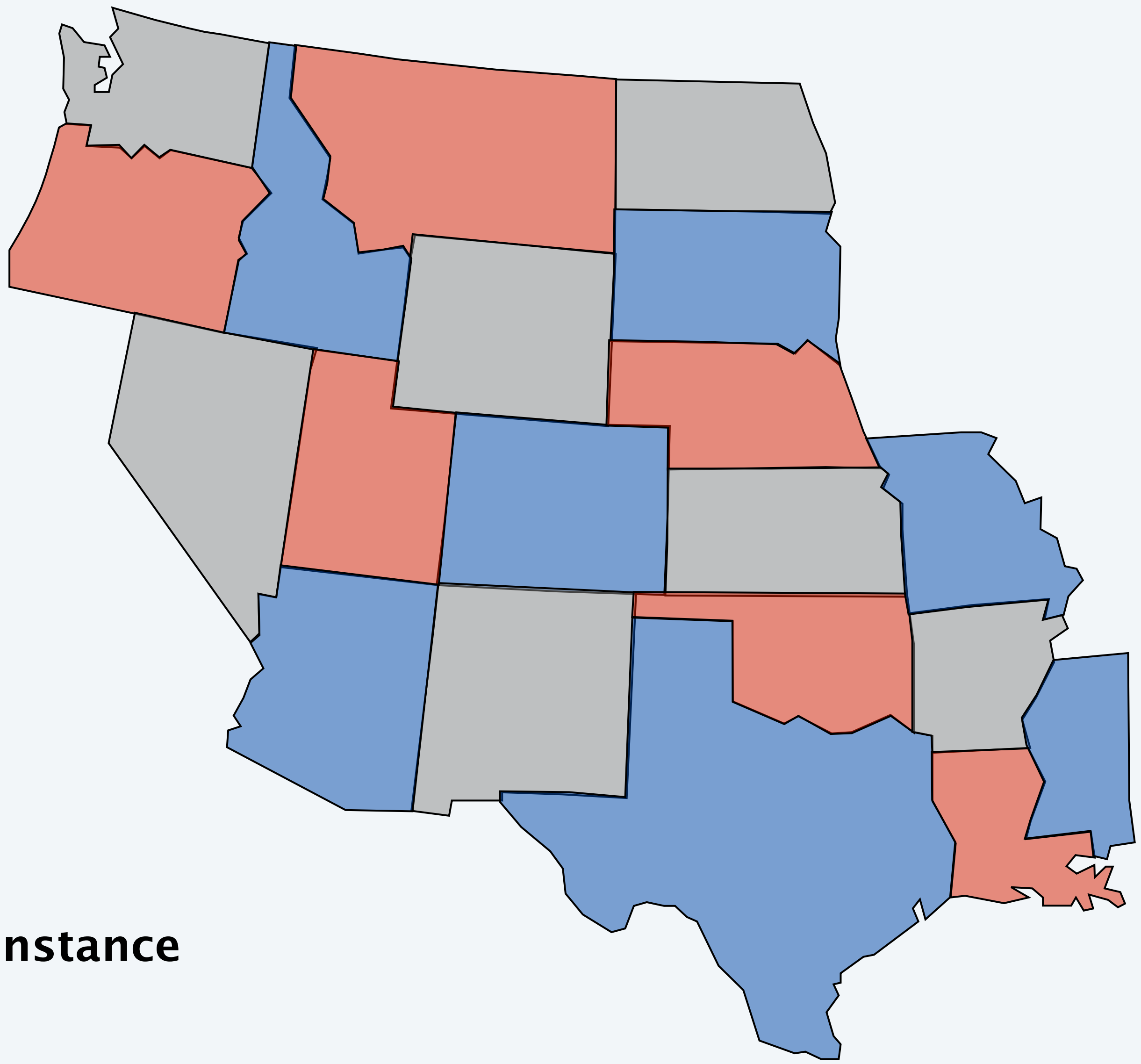
# Planar graph 3-colorability

PLANAR-3-COLOR. Given a planar graph, can it be colored using 3 colors so that no two adjacent nodes have the same color?



# Planar map 3-colorability

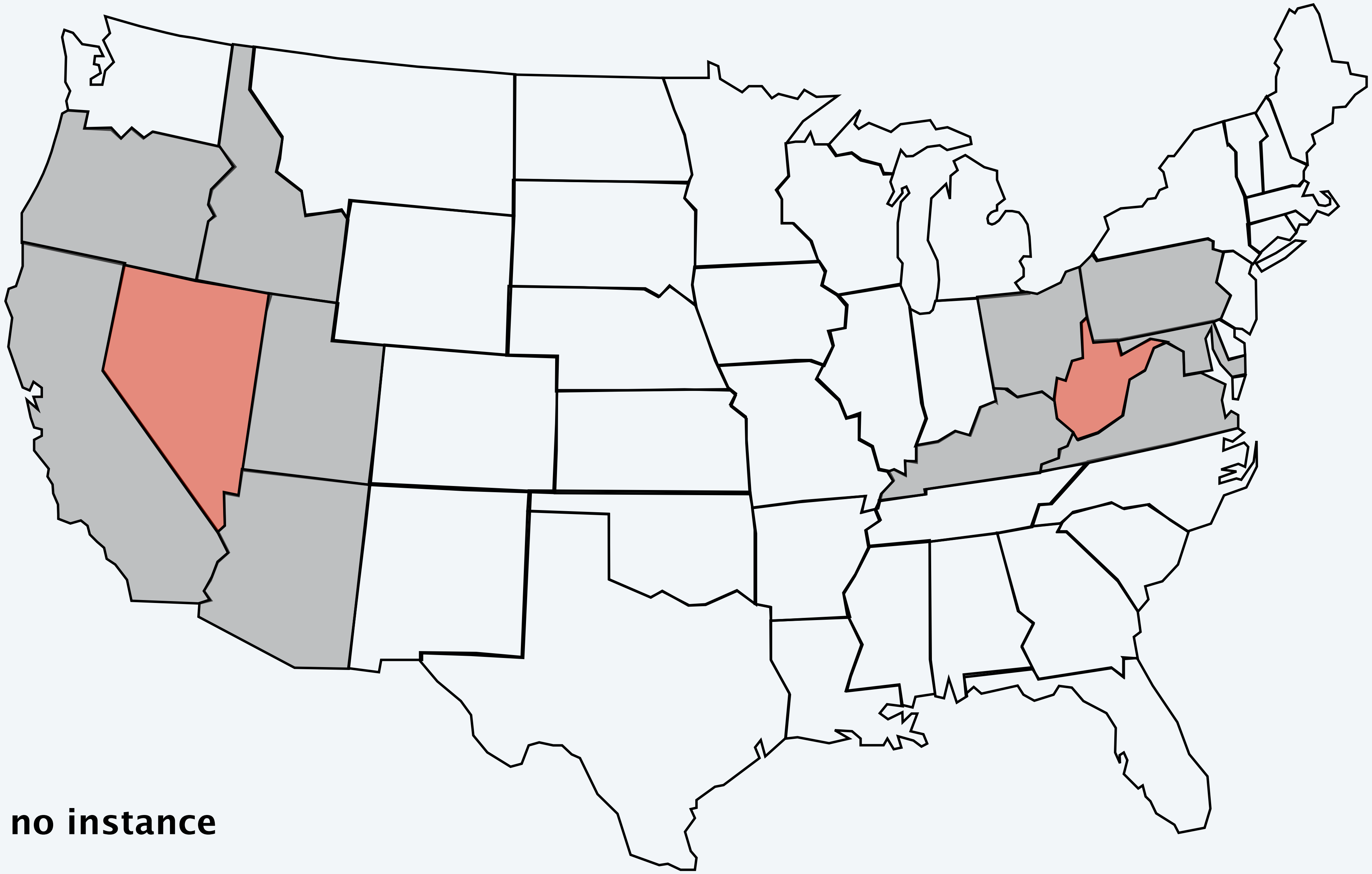
PLANAR-MAP-3-COLOR. Given a planar map, can it be colored using 3 colors so that no two adjacent regions have the same color?



**yes instance**

# Planar map 3-colorability

PLANAR-MAP-3-COLOR. Given a planar map, can it be colored using 3 colors so that no two adjacent regions have the same color?



**no instance**

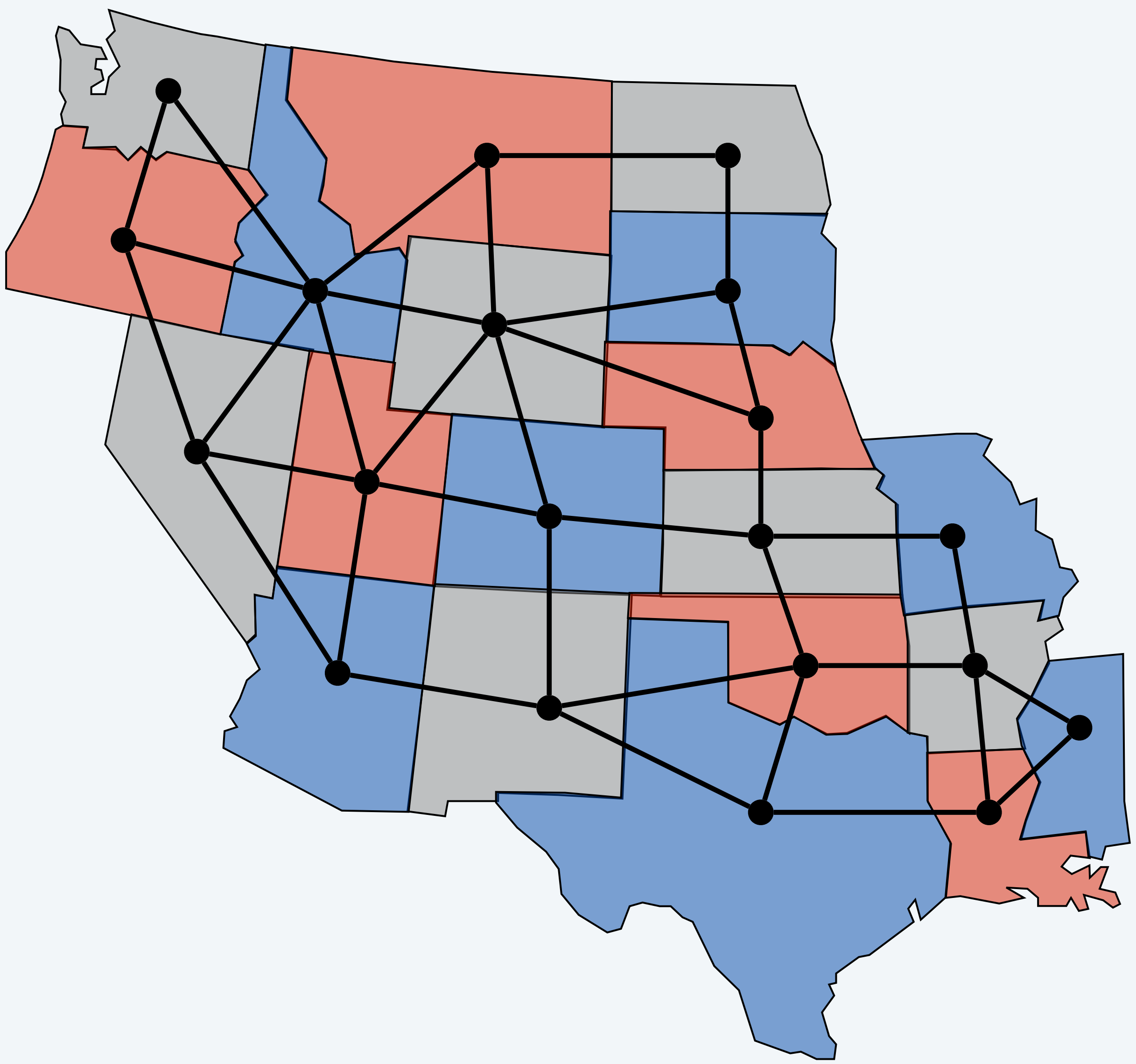
# Planar graph and map 3-colorability reduce to one another

**Theorem.** PLANAR-3-COLOR $\equiv_P$ PLANAR-MAP-3-COLOR.

**Pf sketch.**

- Nodes correspond to regions.
- Two nodes are adjacent iff they share a nontrivial border.

e.g., not Arizona and Colorado



# Planar 3-colorability is NP-complete

Theorem. PLANAR-3-COLOR $\in$ **NP**-complete.

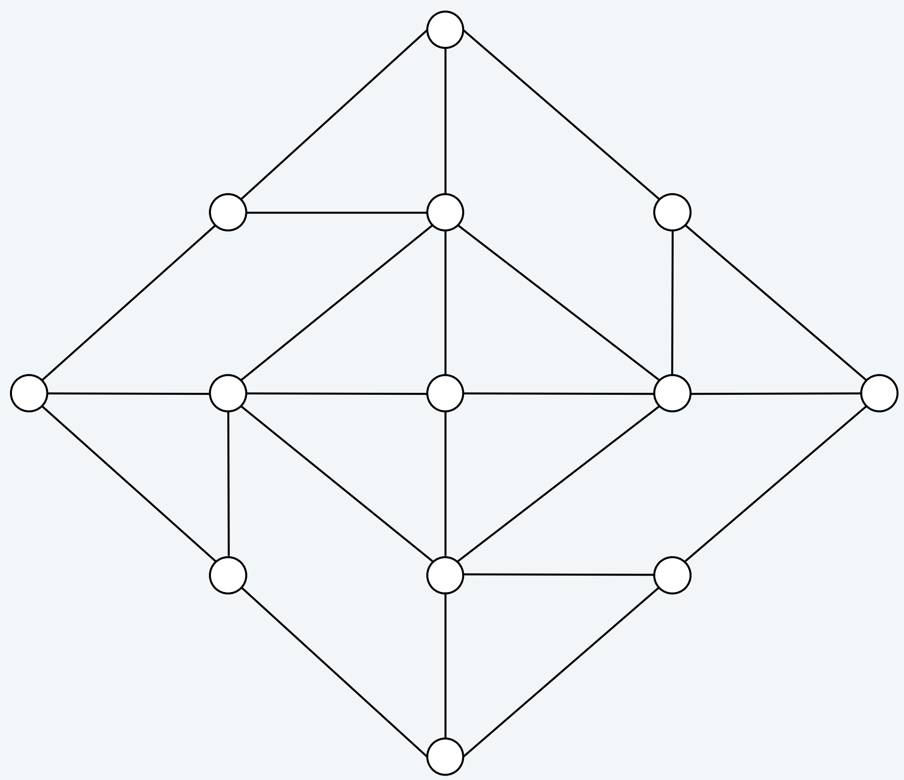
Pf.

- Easy to see that PLANAR-3-COLOR $\in$ **NP**.
- We show 3-COLOR $\leq_P$ PLANAR-3-COLOR.
- Given 3-COLOR instance $G$, we construct an instance of PLANAR-3-COLOR that is 3-colorable iff $G$ is 3-colorable.

# Planar 3-colorability is NP-complete

Lemma. $W$ is a planar graph such that:

- In any 3-coloring of $W$, opposite corners have the same color.
- Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of $W$.



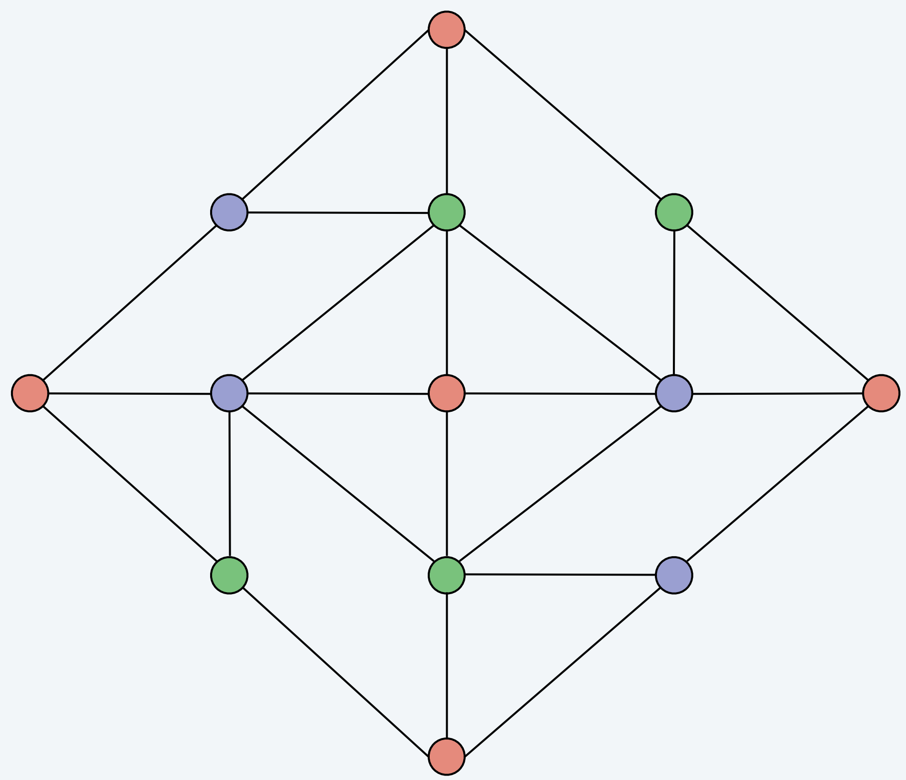
**planar gadget W**

# Planar 3-colorability is NP-complete

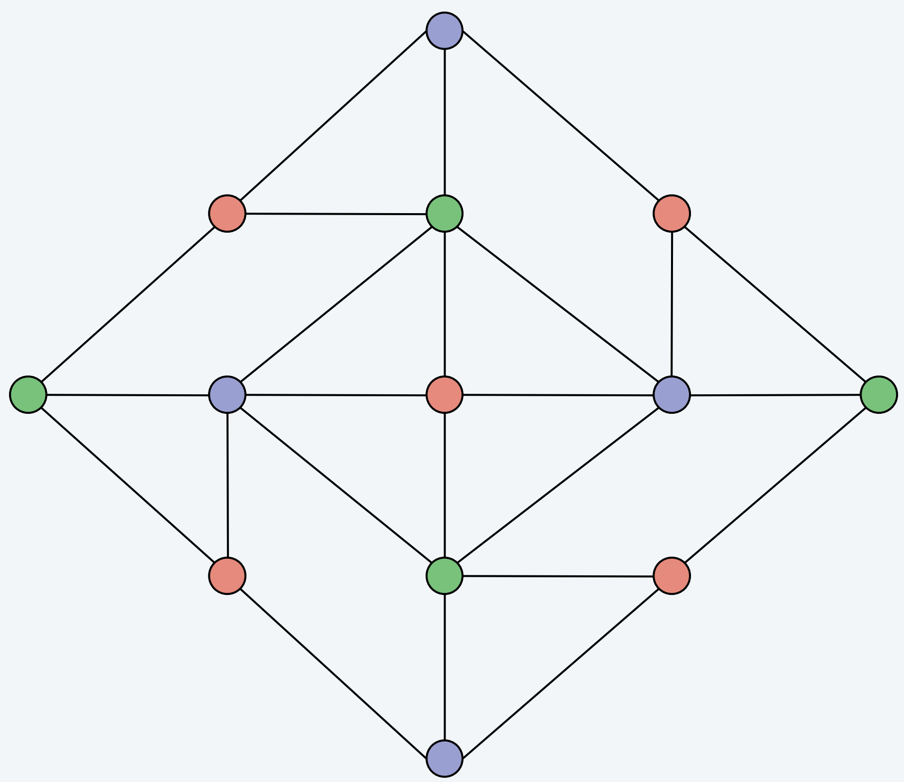
Lemma. $W$ is a planar graph such that:

- In any 3-coloring of $W$, opposite corners have the same color.
- Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of $W$.

Pf. The only 3-colorings (modulo permutations) of $W$ are shown below. ▪



planar gadget W

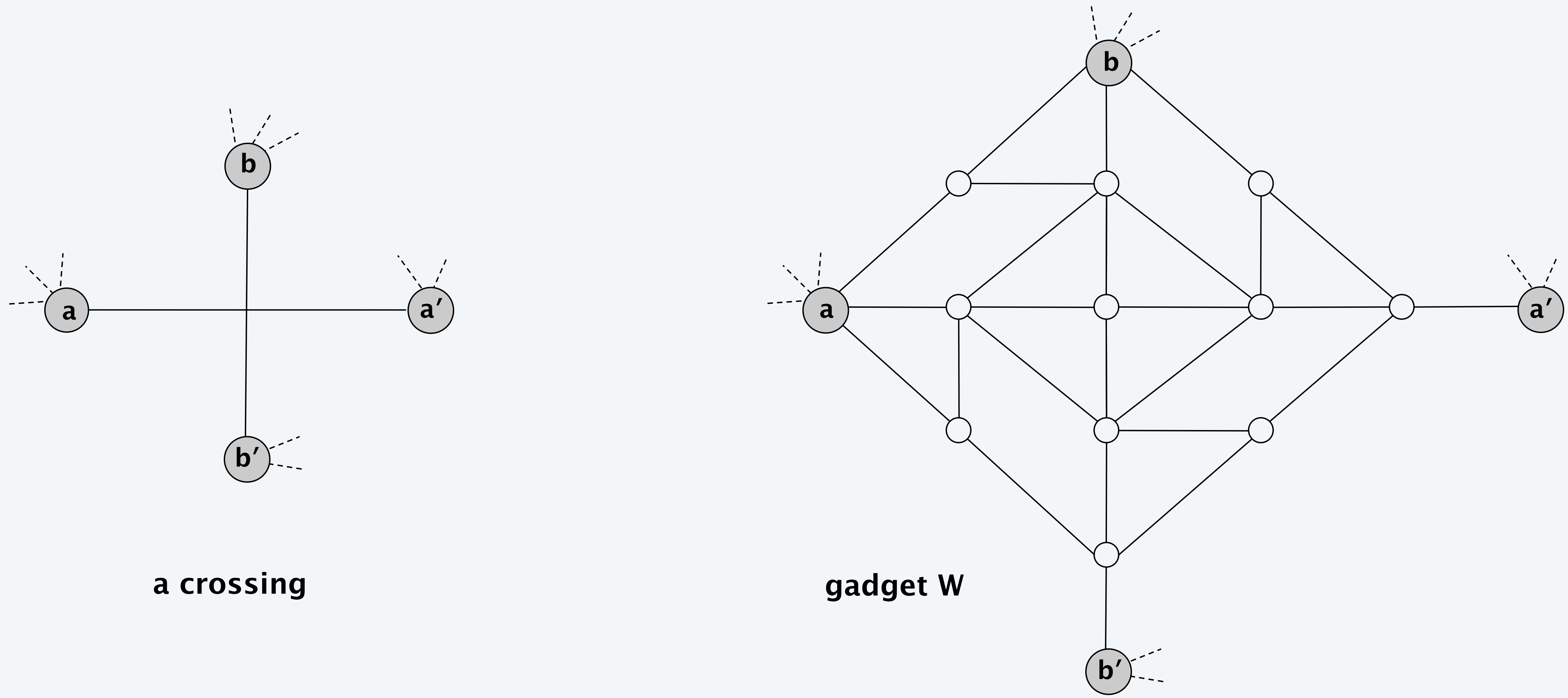


# Planar 3-colorability is NP-complete

Construction. Given instance $G$ of 3-COLOR, draw $G$ in plane, letting edges cross. Form planar $G'$ by replacing each edge crossing with planar gadget $W$.

Lemma. $G$ is 3-colorable iff $G'$ is 3-colorable.

- In any 3-coloring of $W$, $a \neq a'$ and $b \neq b'$.
- If $a \neq a'$ and $b \neq b'$ then can extend to a 3-coloring of $W$.

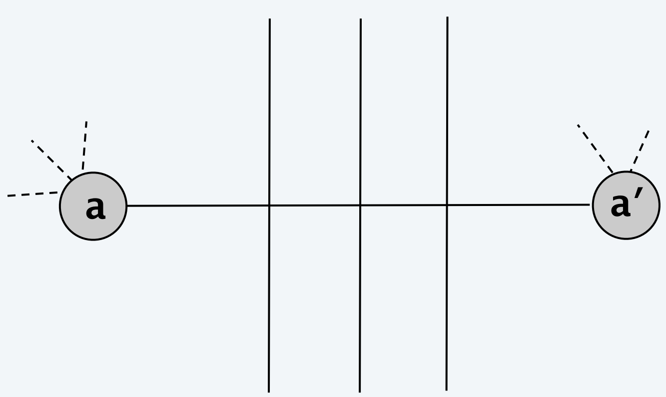


# Planar 3-colorability is NP-complete

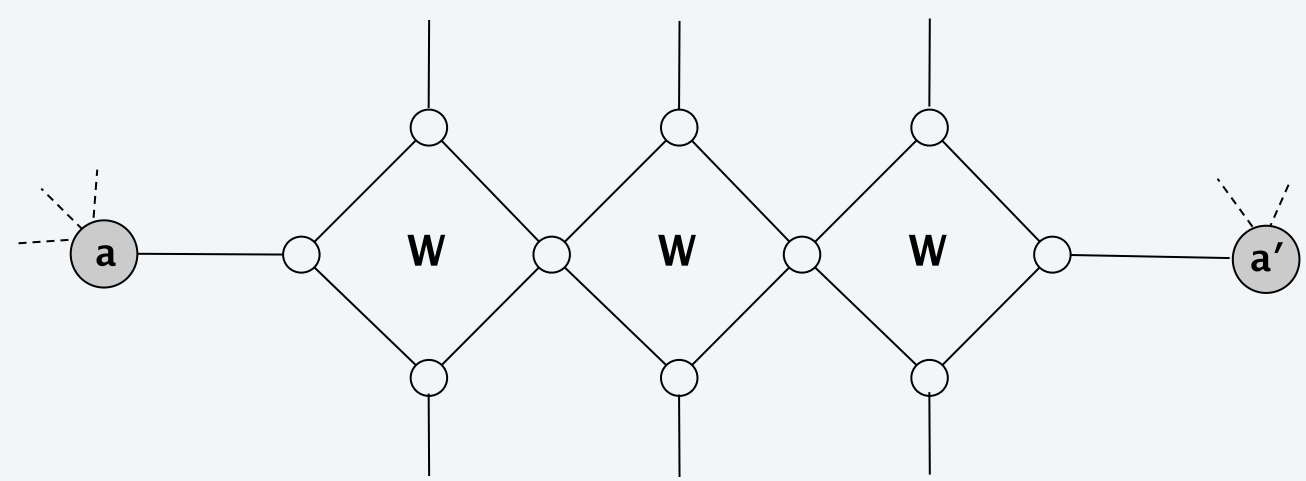
Construction. Given instance $G$ of 3-COLOR, draw $G$ in plane, letting edges cross. Form planar $G'$ by replacing each edge crossing with planar gadget $W$.

Lemma. $G$ is 3-colorable iff $G'$ is 3-colorable.

- In any 3-coloring of $W$, $a \neq a'$ and $b \neq b'$.
- If $a \neq a'$ and $b \neq b'$ then can extend to a 3-coloring of $W$.




**multiple crossings**

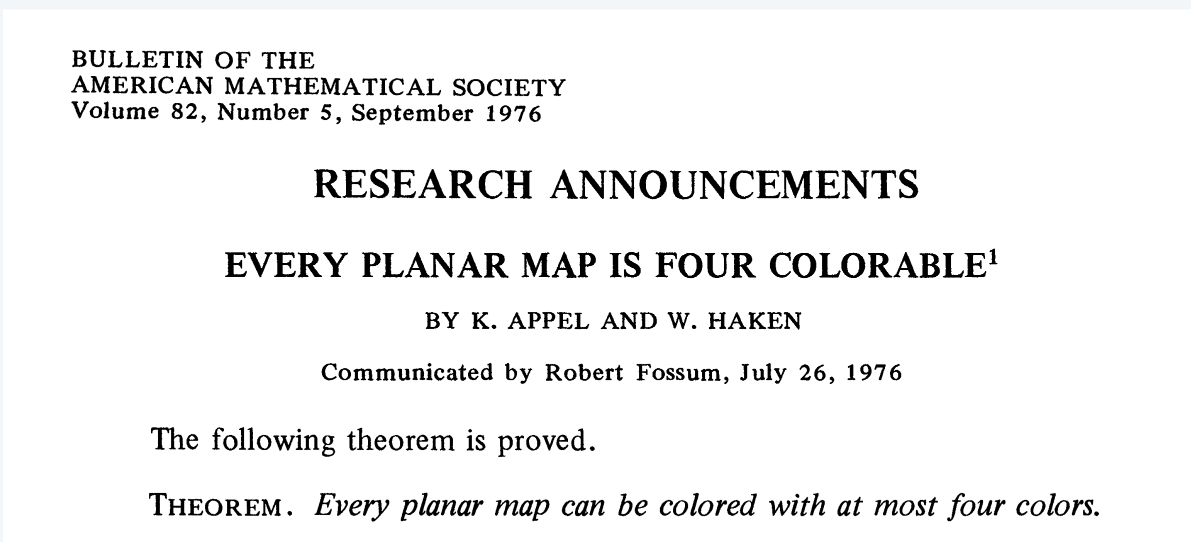



**concatenate copies of gadget W**

# Planar map k-colorability

Theorem. [Appel–Haken 1976] Every planar map is 4-colorable.

- Resolved century-old open problem.
- Used 50 days of computer time to deal with many special cases.
- First major theorem to be proved using computer.



BULLETIN OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 82, Number 5, September 1976

RESEARCH ANNOUNCEMENTS

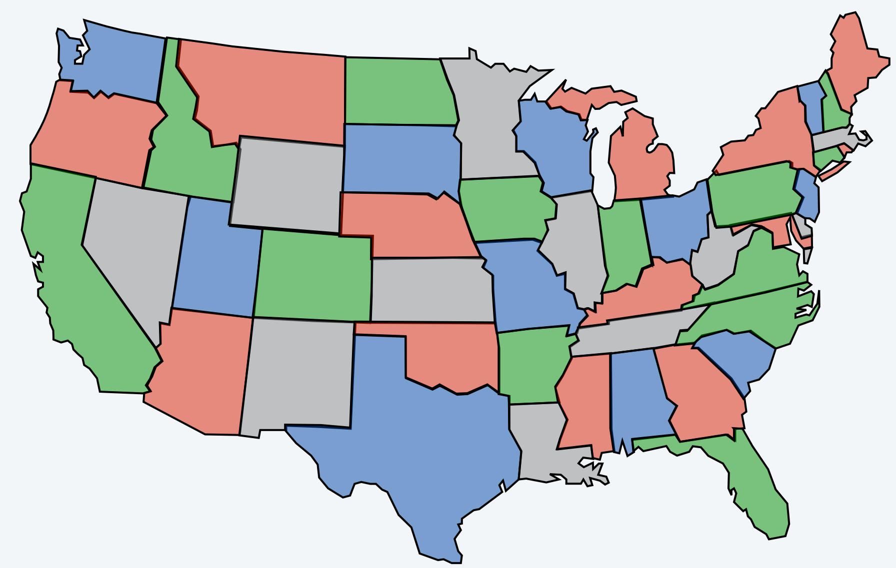
EVERY PLANAR MAP IS FOUR COLORABLE[1]

BY K. APPEL AND W. HAKEN

Communicated by Robert Fossum, July 26, 1976

The following theorem is proved.

THEOREM. *Every planar map can be colored with at most four colors.*



Remarks.

- Appel–Haken yields $O(n^4)$ algorithm to 4-color of a planar map.
- Best known: $O(n^2)$ to 4-color; $O(n)$ to 5-color.
- Determining whether 3 colors suffice is **NP**-complete.

# Beyond planarity

Graph minor theorem. [Robertson–Seymour 1983–2004]
Pf of theorem. Tour de force.

Corollary. There exist an $O(n^3)$ algorithm to determine if a graph can be embedded in the torus in such a way that no two edges cross.

$$2 \uparrow k \;=\; \underbrace{2^{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}}_{k}$$

more than $2 \uparrow 2 \uparrow 2 \uparrow (n/2)$

Mind boggling fact 1. The proof is highly nonconstructive!
Mind boggling fact 2. The constant of proportionality is enormous!

> " *Unfortunately, for any instance G = (V, E) that one could fit into the known universe, one would easily prefer* $n^{70}$ *to even constant time, if that constant had to be one of Robertson and Seymour's.* " — *David Johnson*

Theorem. There exists an explicit $O(n)$ algorithm.
Practice. LEDA implementation guarantees $O(n^3)$.

# Poly-time special cases of NP-hard problems

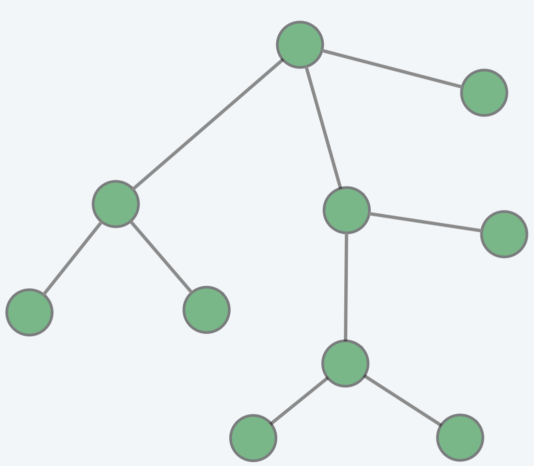
Trees. Vertex-Cover, Independent-Set, Longest-Path, Graph-Isomorphism, ...

Bipartite graphs. Vertex-Cover, Independent-Set, 3-Color, Edge-Color, ...

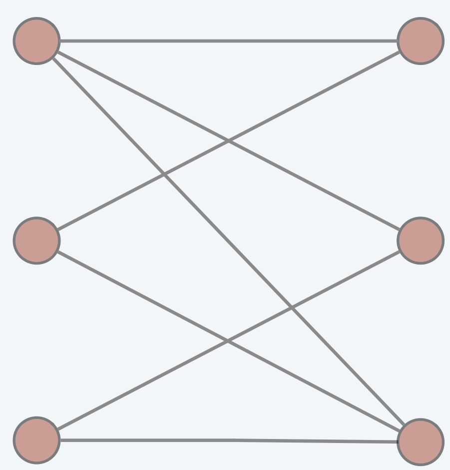
Planar graphs. Max-Cut, Ising, Clique, Graph-Isomorphism, 4-Color, ...

Bounded treewidth. Ham-Cycle, Independent-Set, Graph-Isomorphism, ...

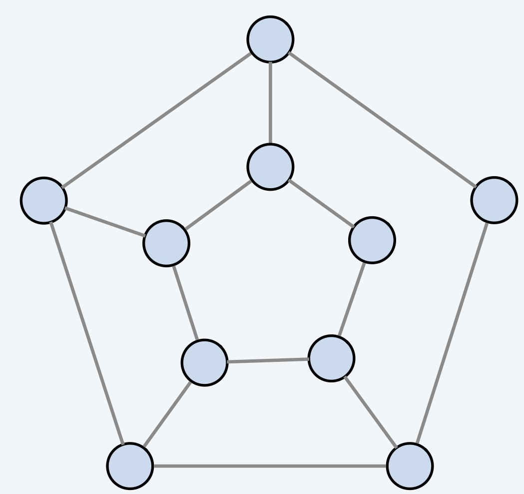
Small integers. Subset-Sum, Knapsack, Partition, ...



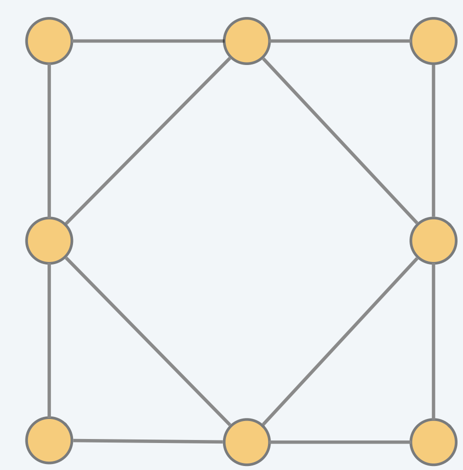
**tree**



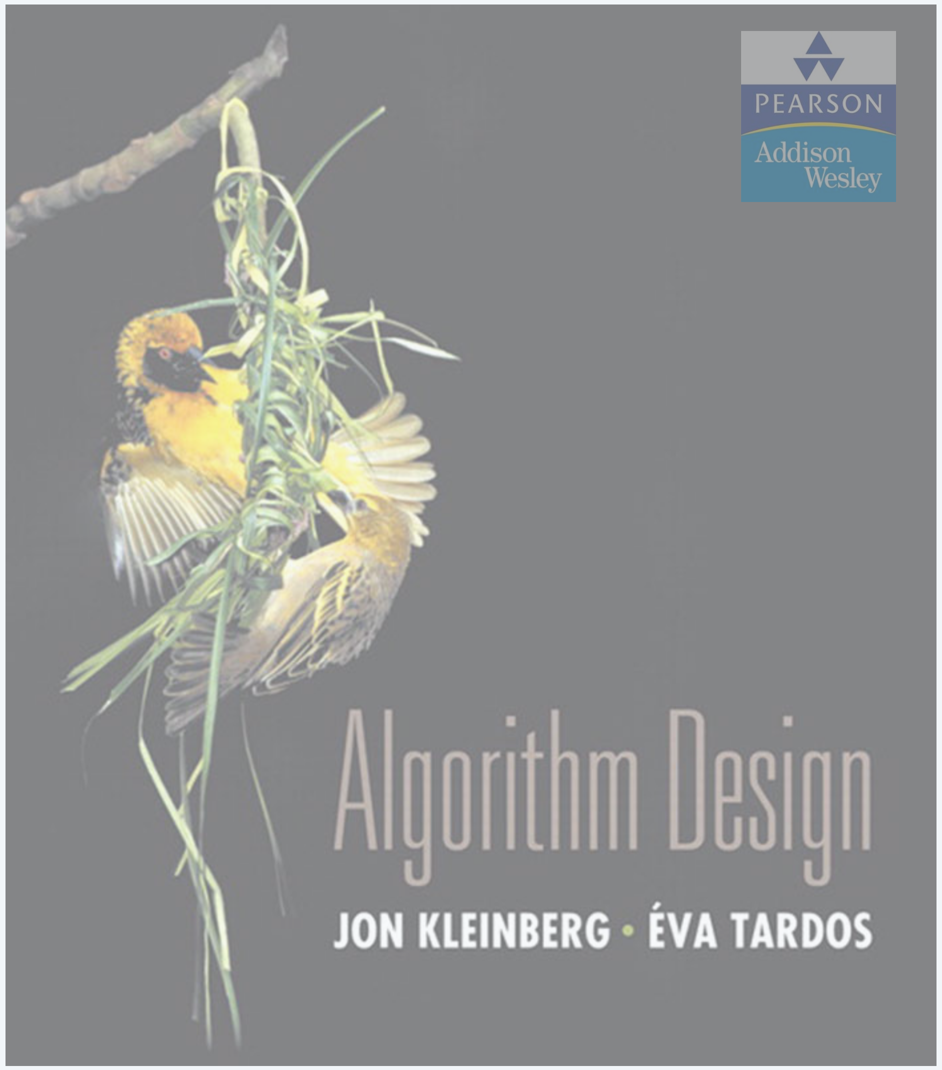
**bipartite**



**planar**



**bounded treewidth**




SECTION 11.8

# INTRACTABILITY III

- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

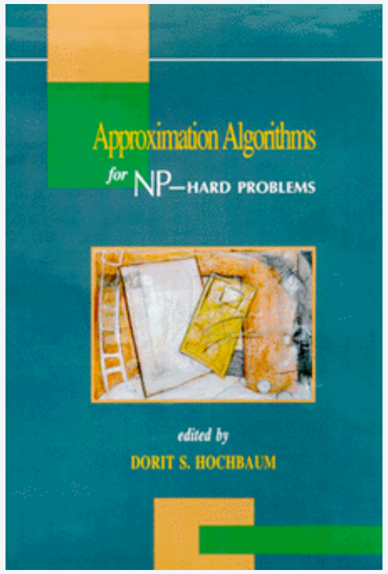
# Approximation algorithms

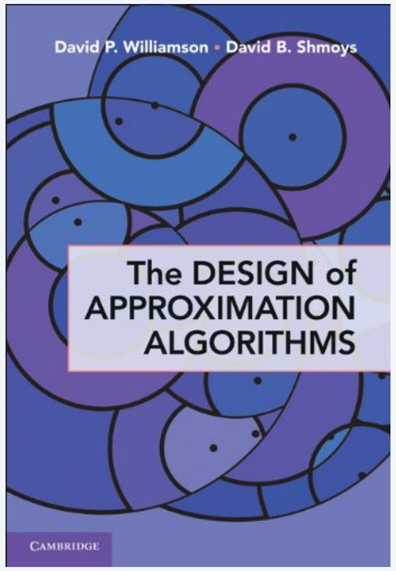
ρ-approximation algorithm.

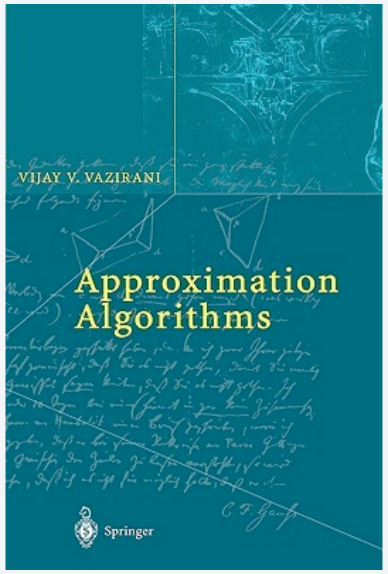
- Runs in polynomial time.
- Applies to arbitrary instances of the problem.
- Guaranteed to find a solution within ratio $\rho$ of true optimum.

Ex. Given a graph $G$, can find a vertex cover that uses $\leq 2\, OPT(G)$ vertices in $O(m + n)$ time.

Challenge. Need to prove a solution's value is close to optimum value, without even knowing what optimum value is!



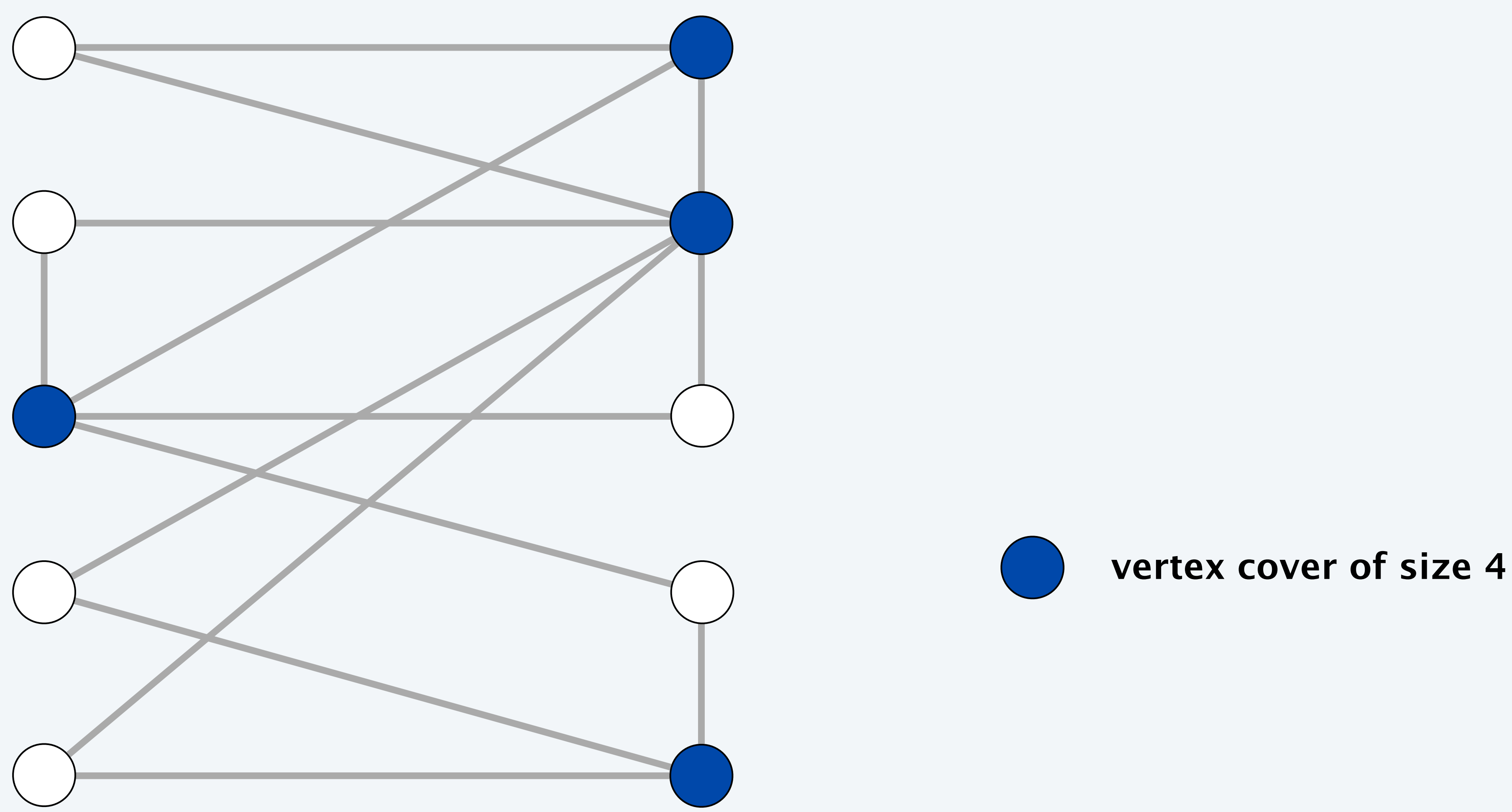




# Vertex cover

VERTEX-COVER. Given a graph $G = (V, E)$, find a min-size vertex cover.

for each edge $(u, v) \in E$:
either $u \in S$, $v \in S$, or both



# Vertex cover: greedy algorithm

VERTEX-COVER. Given a graph $G = (V, E)$, find a min-size vertex cover.

GREEDY-VERTEX-COVER($G$)

$S \leftarrow \varnothing$.

$E' \leftarrow E$.

WHILE ($E' \neq \varnothing$)

  Let $(u, v) \in E'$ be an arbitrary edge.

  $M \leftarrow M \cup \{(u, v)\}$. ← $M$ is a matching

  $S \leftarrow S \cup \{u\} \cup \{v\}$. ← every vertex cover must take at least one of these; we take both

  Delete from $E'$ all edges incident to either $u$ or $v$.

RETURN $S$.

Running time. Can be implemented in $O(m + n)$ time.

**Given a graph $G$, let $M$ be any matching and let $S$ be any vertex cover. Which of the following must be true?**

**A.** $|M| \leq |S|$

**B.** $|S| \leq |M|$

**C.** $|S| = |M|$

**D.** None of the above.

# Vertex cover: greedy algorithm is a 2-approximation algorithm

**Theorem.** Let $S^*$ be a minimum vertex cover. Then, greedy algorithm computes a vertex cover $S$ with $|S| \le 2|S^*|$. ← 2-approximation algorithm

**Pf.**

- $S$ is a vertex cover. ← delete edge only after it's already covered
- $M$ is a matching. ← when $(u, v)$ added to $M$, all edges incident to either $u$ or $v$ are deleted
- $|S| \underset{\text{design}}{=} 2|M| \underset{\text{weak duality}}{\le} 2|S^*|$. ▪

**Corollary.** Let $M^*$ be a maximum matching. Then, greedy algorithm computes a matching $M$ with $|M| \ge \frac{1}{2}|M^*|$.

**Pf.** $|M| = \frac{1}{2}|S| \underset{\text{weak duality}}{\ge} \frac{1}{2}|M^*|$. ▪

# Vertex cover inapproximability

**Theorem.** [Dinur–Safra 2004] If **P** ≠ **NP**, then no $\rho$-approximation for VERTEX-COVER for any $\rho < 1.3606$.

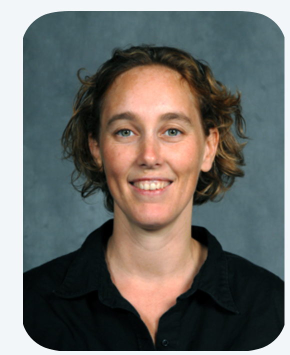
On the Hardness of Approximating Minimum Vertex Cover

Irit Dinur* Samuel Safra†

May 26, 2004

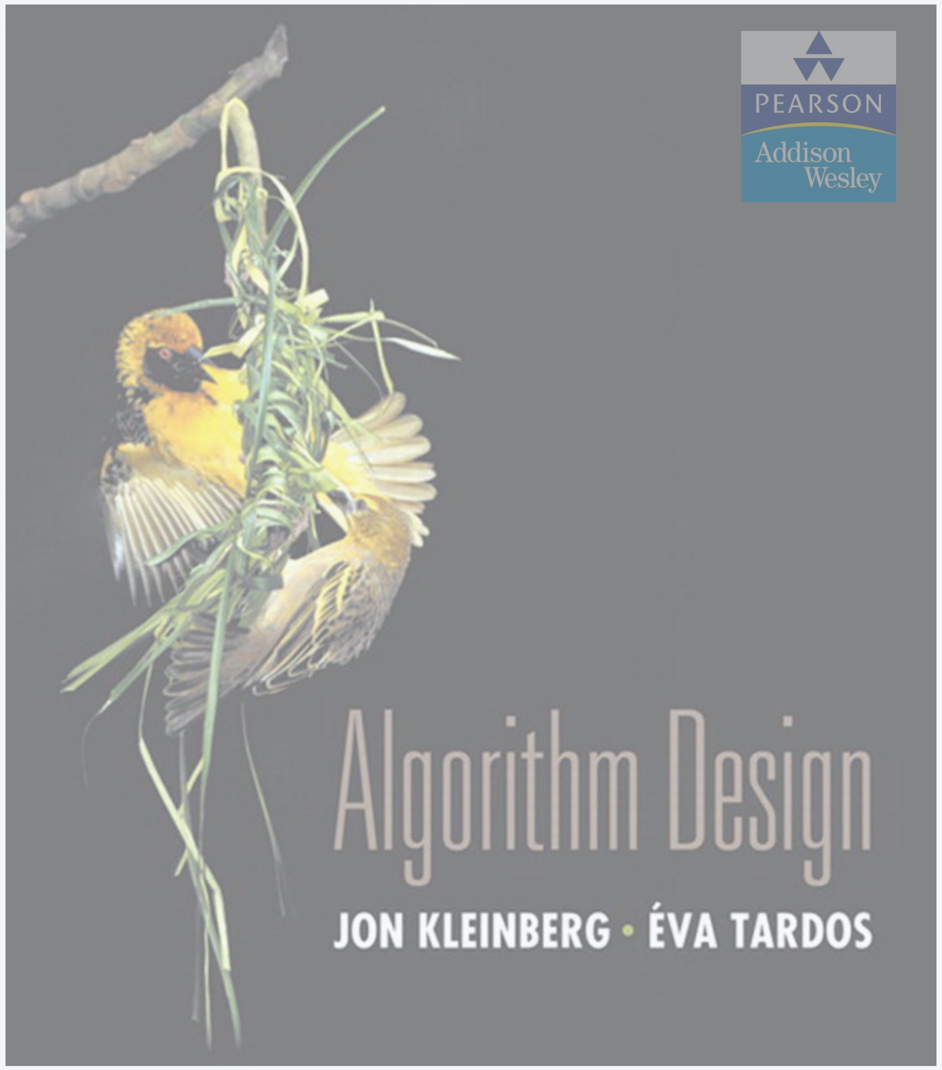
**Abstract**

We prove the Minimum Vertex Cover problem to be NP-hard to approximate to within a factor of 1.3606, extending on previous PCP and hardness of approximation technique. To that end, one needs to develop a new proof framework, and borrow and extend ideas from several fields.




**Open research problem.** Close the gap.

**Conjecture.** no $\rho$-approximation for VERTEX-COVER for any $\rho < 2$.




SECTION 11.8

# INTRACTABILITY III

- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

# Knapsack problem

Knapsack problem.

- Given $n$ objects and a knapsack.
- Item $i$ has value $v_i > 0$ and weighs $w_i > 0$. ← we assume $w_i \leq W$ for each $i$
- Knapsack has weight limit $W$.
- Goal: fill knapsack so as to maximize total value.

Ex: $\{ 3, 4 \}$ has value $40$.

| item | value | weight |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 6 | 2 |
| 3 | 18 | 5 |
| 4 | 22 | 6 |
| 5 | 28 | 7 |

**original instance (W = 11)**

# Knapsack is NP-complete

SUBSET-SUM. Given a set $X$, values $u_i \geq 0$, and an integer $U$, is there a subset $S \subseteq X$ whose elements sum to exactly $U$?

KNAPSACK. Given a set $X$, weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit $W$, and a target value $V$, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \;\leq\; W$$

$$\sum_{i \in S} v_i \;\leq\; V$$

Theorem. SUBSET-SUM $\leq_P$ KNAPSACK.

Pf. Given instance $(u_1, \ldots, u_n, U)$ of SUBSET-SUM, create KNAPSACK instance:

$$v_i = w_i = u_i \qquad \sum_{i \in S} u_i \;\leq\; U$$

$$V = W = U \qquad \sum_{i \in S} u_i \;\geq\; U$$

# Knapsack problem: dynamic programming I

Def. $OPT(i, w)$ = max value subset of items $1,..., i$ with weight limit $w$.

Case 1. $OPT$ does not select item $i$.

- $OPT$ selects best of $1, \ldots, i-1$ using up to weight limit $w$.

Case 2. $OPT$ selects item $i$.

- New weight limit = $w - w_i$.
- $OPT$ selects best of $1, \ldots, i-1$ using up to weight limit $w - w_i$.

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{\, OPT(i-1,w),\ \ v_i + OPT(i-1, w-w_i)\,\} & \text{otherwise} \end{cases}$$

Theorem. Computes the optimal value in $O(n\, W)$ time.

- Not polynomial in input size.
- Polynomial in input size if weights are small integers.

# Knapsack problem: dynamic programming II

Def. $OPT(i, v)$ = min weight of a knapsack for which we can obtain a solution of value $\geq v$ using a subset of items $1,\ldots, i$.

Note. Optimal value is the largest value $v$ such that $OPT(n, v) \leq W$.

Case 1. $OPT$ does not select item $i$.

- $OPT$ selects best of $1, \ldots, i-1$ that achieves value $\geq v$.

Case 2. $OPT$ selects item $i$.

- Consumes weight $w_i$, need to achieve value $\geq v - v_i$.
- $OPT$ selects best of $1, \ldots, i-1$ that achieves value $\geq v - v_i$.

$$OPT(i, v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \min\left\{OPT(i-1, v),\ w_i + OPT(i-1, v - v_i)\right\} & \text{otherwise} \end{cases}$$

# Knapsack problem: dynamic programming II

Theorem. Dynamic programming algorithm II computes the optimal value in $O(n^2\, v_{\max})$ time, where $v_{\max}$ is the maximum of any value.

Pf.

- The optimal value $V^* \leq\, n\, v_{\max}$.
- There is one subproblem for each item and for each value $v \leq V^*$.
- It takes $O(1)$ time per subproblem. ▪

Remark 1. Not polynomial in input size!

Remark 2. Polynomial time if values are small integers.

# Knapsack problem: poly-time approximation scheme

## Intuition for approximation algorithm.

- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded/scaled instance.
- Return optimal items in rounded instance.

| item | value | weight |
|---|---|---|
| 1 | 934221 | 1 |
| 2 | 5956342 | 2 |
| 3 | 17810013 | 5 |
| 4 | 21217800 | 6 |
| 5 | 27343199 | 7 |

**original instance (W = 11)**

| item | value | weight |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 6 | 2 |
| 3 | 18 | 5 |
| 4 | 22 | 6 |
| 5 | 28 | 7 |

**rounded instance (W = 11)**

# Knapsack problem: poly-time approximation scheme

Round up all values:

- $0 < \varepsilon \leq 1$ = precision parameter.
- $v_{\max}$ = largest value in original instance.
- $\theta$ = scaling factor = $\varepsilon\, v_{\max} / 2n$.

$$\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta\,, \qquad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$$

Observation. Optimal solutions to problem with $\bar{v}$ are equivalent to optimal solutions to problem with $\hat{v}$.

Intuition. $\bar{v}$ close to $v$ so optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm II is fast.

# Knapsack problem: poly-time approximation scheme

Theorem. If $S$ is solution found by rounding algorithm and $S^*$ is any other feasible solution, then $(1+\epsilon)\sum_{i \in S} v_i \;\geq\; \sum_{i \in S^*} v_i$

Pf. Let $S^*$ be any feasible solution satisfying weight constraint.

$$\begin{aligned}
\sum_{i \in S^*} v_i &\leq \sum_{i \in S^*} \bar{v}_i && \text{always round up} \\
&\leq \sum_{i \in S} \bar{v}_i && \text{solve rounded instance optimally} \\
&\leq \sum_{i \in S} (v_i + \theta) && \text{never round up by more than } \theta \\
&\leq \sum_{i \in S} v_i \;+\; n\theta && |S| \leq n \\
&= \sum_{i \in S} v_i \;+\; \tfrac{1}{2}\,\epsilon\, v_{\max} && \theta = \varepsilon\, v_{\max} / 2n \\
&\leq (1+\epsilon)\sum_{i \in S} v_i && v_{\max} \leq 2\, \Sigma_{i \in S}\, v_i
\end{aligned}$$

subset containing only the item of largest value

choosing $S^* = \{ \max \}$

$$\begin{aligned}
v_{\max} &\leq \sum_{i \in S} v_i \;+\; \tfrac{1}{2}\,\epsilon\, v_{\max} \\
&\leq \sum_{i \in S} v_i \;+\; \tfrac{1}{2}\, v_{\max}
\end{aligned}$$

thus

$$v_{\max} \;\leq\; 2 \sum_{i \in S} v_i$$

# Knapsack problem: poly-time approximation scheme

Theorem. For any $\varepsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 + \varepsilon)$ factor of the optimum in $O(n^3 / \varepsilon)$ time.

Pf.

- We have already proved the accuracy bound.
- Dynamic program II running time is $O(n^2\, \hat{v}_{\max})$, where

$$\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \left\lceil \frac{2n}{\epsilon} \right\rceil$$

# INTRACTABILITY III

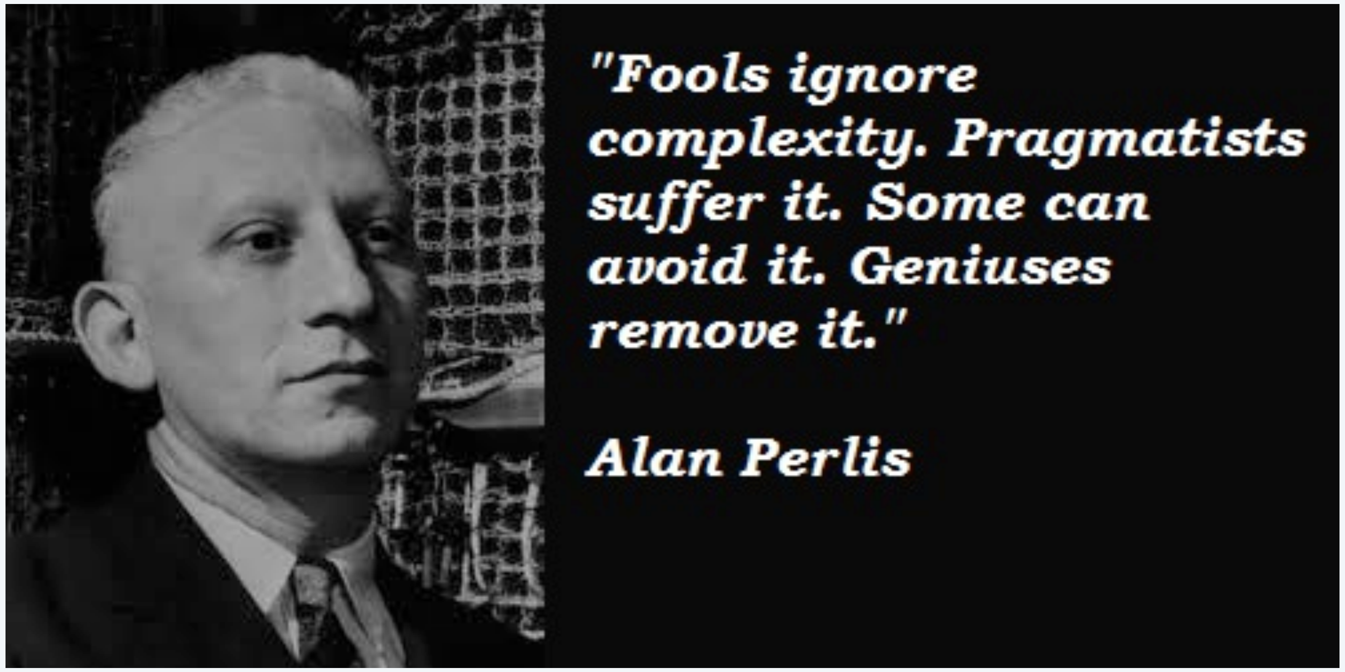
- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

# Exact exponential algorithms

Complexity theory deals with worst-case behavior.

- Instances you want to solve may be "easy."

> " *For every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run.*" — *Alan Perlis*



**What is complexity of 3-SAT? Choose the best answer.**

**A.** $O(n^2)$

**B.** $O^*(1.34^n)$

**C.** $O^*(1.84^n)$

**D.** $O^*(2^n)$

$O^*$ ignores $poly(m, n)$ terms

# Exact algorithms for 3-satisfiability

Brute force. Given a 3-SAT instance with $n$ variables and $m$ clauses, the brute-force algorithm takes $O((m + n)\, 2^n)$ time.

Pf.

- There are $2^n$ possible truth assignments to the $n$ variables.
- We can evaluate a truth assignment in $O(m + n)$ time. ▪

# Exact algorithms for 3-satisfiability

A recursive framework. A 3-SAT formula $\Phi$ is either empty or the disjunction of a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ and a 3-SAT formula $\Phi'$ with one fewer clause.

$$\begin{aligned}\Phi &= (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \\ &= (\ell_1 \wedge \Phi') \vee (\ell_2 \wedge \Phi') \vee (\ell_3 \wedge \Phi') \\ &= (\Phi' \mid \ell_1 = \mathit{true}) \vee (\Phi' \mid \ell_2 = \mathit{true}) \vee (\Phi' \mid \ell_3 = \mathit{true})\end{aligned}$$

Notation. $\Phi \mid x = \mathit{true}$ is the simplification of $\Phi$ by setting $x$ to $\mathit{true}$.

Ex.

- $\Phi \quad = (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (w \vee y \vee \neg z) \wedge (\neg x \vee y \vee z)$.
- $\Phi' \quad = (x \vee \neg y \vee z) \wedge (w \vee y \vee \neg z) \wedge (\neg x \vee y \vee z)$.
- $(\Phi' \mid x = \mathit{true}) \quad = (w \vee y \vee \neg z) \wedge (y \vee z)$.

each clause has ≤ 3 literals

# Exact algorithms for 3-satisfiability

A recursive framework. A 3-SAT formula $\Phi$ is either empty or the disjunction of a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ and a 3-SAT formula $\Phi'$ with one fewer clause.

3-SAT $(\Phi)$

---

IF $\Phi$ is empty RETURN *true*.

$(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \leftarrow \Phi$.

IF 3-SAT $(\Phi' \mid \ell_1 = \mathit{true})$ RETURN *true*.

IF 3-SAT $(\Phi' \mid \ell_2 = \mathit{true})$ RETURN *true*.

IF 3-SAT $(\Phi' \mid \ell_3 = \mathit{true})$ RETURN *true*.

RETURN *false*.

Theorem. The brute-force 3-SAT algorithm takes $O(\text{poly}(n)\ 3^n)$ time.

Pf. $T(n) \le 3T(n-1) + \text{poly}(n)$. ▪

# Exact algorithms for 3-satisfiability

Key observation. The cases are not mutually exclusive. Every satisfiable assignment containing clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ must fall into one of 3 classes:

- $\ell_1$ is *true*.
- $\ell_1$ is *false*; $\ell_2$ is *true*.
- $\ell_1$ is *false*; $\ell_2$ is *false*; $\ell_3$ is *true*.

3-SAT $(\Phi)$

IF $\Phi$ is empty RETURN *true*.

$(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \leftarrow \Phi$.

IF 3-SAT$(\Phi' \mid \ell_1 = true)$ RETURN *true*.

IF 3-SAT$(\Phi' \mid \ell_1 = false, \ell_2 = true)$ RETURN *true*.

IF 3-SAT$(\Phi' \mid \ell_1 = false, \ell_2 = false, \ell_3 = true)$ RETURN *true*.

RETURN *false*.

# Exact algorithms for 3-satisfiability

**Theorem.** The brute-force algorithm takes $O(1.84^n)$ time.

**Pf.** $T(n) \le T(n-1) + T(n-2) + T(n-3) + O(m+n)$. ▪

largest root of $r^3 = r^2 + r + 1$

3-SAT $(\Phi)$

---

IF $\Phi$ is empty RETURN *true*.

$(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \leftarrow \Phi$.

IF 3-SAT$(\Phi' \mid \ell_1 = \mathit{true})$ RETURN *true*.

IF 3-SAT$(\Phi' \mid \ell_1 = \mathit{false}, \ell_2 = \mathit{true})$ RETURN *true*.

IF 3-SAT$(\Phi' \mid \ell_1 = \mathit{false}, \ell_2 = \mathit{false}, \ell_3 = \mathit{true})$ RETURN *true*.

RETURN *false*.

# Exact algorithms for 3-satisfiability

**Theorem.** There exists a $O(1.33334^n)$ deterministic algorithm for 3-SAT.

## A Full Derandomization of Schöning's $k$-SAT Algorithm

Robin A. Moser and Dominik Scheder

Institute for Theoretical Computer Science
Department of Computer Science
ETH Zürich, 8092 Zürich, Switzerland
{robin.moser, dominik.scheder}@inf.ethz.ch

August 25, 2010

**Abstract**

Schöning [7] presents a simple randomized algorithm for $k$-SAT with running time $O(a_k^n \text{poly}(n))$ for $a_k = 2(k-1)/k$. We give a deterministic version of this algorithm running in time $O((a_k + \epsilon)^n \text{poly}(n))$, where $\epsilon > 0$ can be made arbitrarily small.

# Exact algorithms for satisfiability

DPPL algorithm. Highly-effective backtracking procedure.

- Splitting rule: assign truth value to literal; solve both possibilities.
- Unit propagation: clause contains only a single unassigned literal.
- Pure literal elimination: if literal appears only negated or unnegated.

**A Computing Procedure for Quantification Theory***

MARTIN DAVIS

*Rensselaer Polytechnic Institute, Hartford Division, East Windsor Hill, Conn.*

AND

HILARY PUTNAM

*Princeton University, Princeton, New Jersey*

The hope that mathematical methods employed in the investigation of formal logic would lead to purely computational methods for obtaining mathematical theorems goes back to Leibniz and has been revived by Peano around the turn of the century and by Hilbert's school in the 1920's. Hilbert, noting that all of classical mathematics could be formalized within quantification theory, declared that the problem of finding an algorithm for determining whether or not a given formula of quantification theory is valid was the central problem of mathematical logic. And indeed, at one time it seemed as if investigations of this "decision" problem were on the verge of success. However, it was shown by Church and by Turing that such an algorithm can not exist. This result led to considerable pessimism regarding the possibility of using modern digital computers in deciding significant mathematical questions. However, recently there has been a revival of interest in the whole question. Specifically, it has been realized that while no *decision procedure* exists for quantification theory there are many proof procedures available—that is, uniform procedures which will ultimately locate a proof for any formula of quantification theory which is valid but which will usually involve seeking "forever" in the case of a formula which is not valid—and that some of these proof procedures could well turn out to be feasible for use with modern computing machinery.

**A Machine Program for Theorem-Proving†**

Martin Davis, George Logemann, and Donald Loveland

*Institute of Mathematical Sciences, New York University*

The programming of a proof procedure is discussed in connection with trial runs and possible improvements.

In [1] is set forth an algorithm for proving theorems of quantification theory which is an improvement in certain respects over previously available algorithms such as that of [2]. The present paper deals with the programming of the algorithm of [1] for the New York University, Institute of Mathematical Sciences' IBM 704 computer, with some modifications in the algorithm suggested by this work, with the results obtained using the completed algorithm. Familiarity with [1] is assumed throughout.

# Exact algorithms for satisfiability

Chaff. State-of-the-art SAT solver.

- Solves real-world SAT instances with ~ 10K variable.

  Developed at Princeton by undergrads.

## Chaff: Engineering an Efficient SAT Solver

Matthew W. Moskewicz
Department of EECS
UC Berkeley
moskewcz@alumni.princeton.edu

Conor F. Madigan
Department of EECS
MIT
cmadigan@mit.edu

Ying Zhao, Lintao Zhang, Sharad Malik
Department of Electrical Engineering
Princeton University
{yingzhao, lintaoz, sharad}@ee.princeton.edu

**ABSTRACT**

Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the

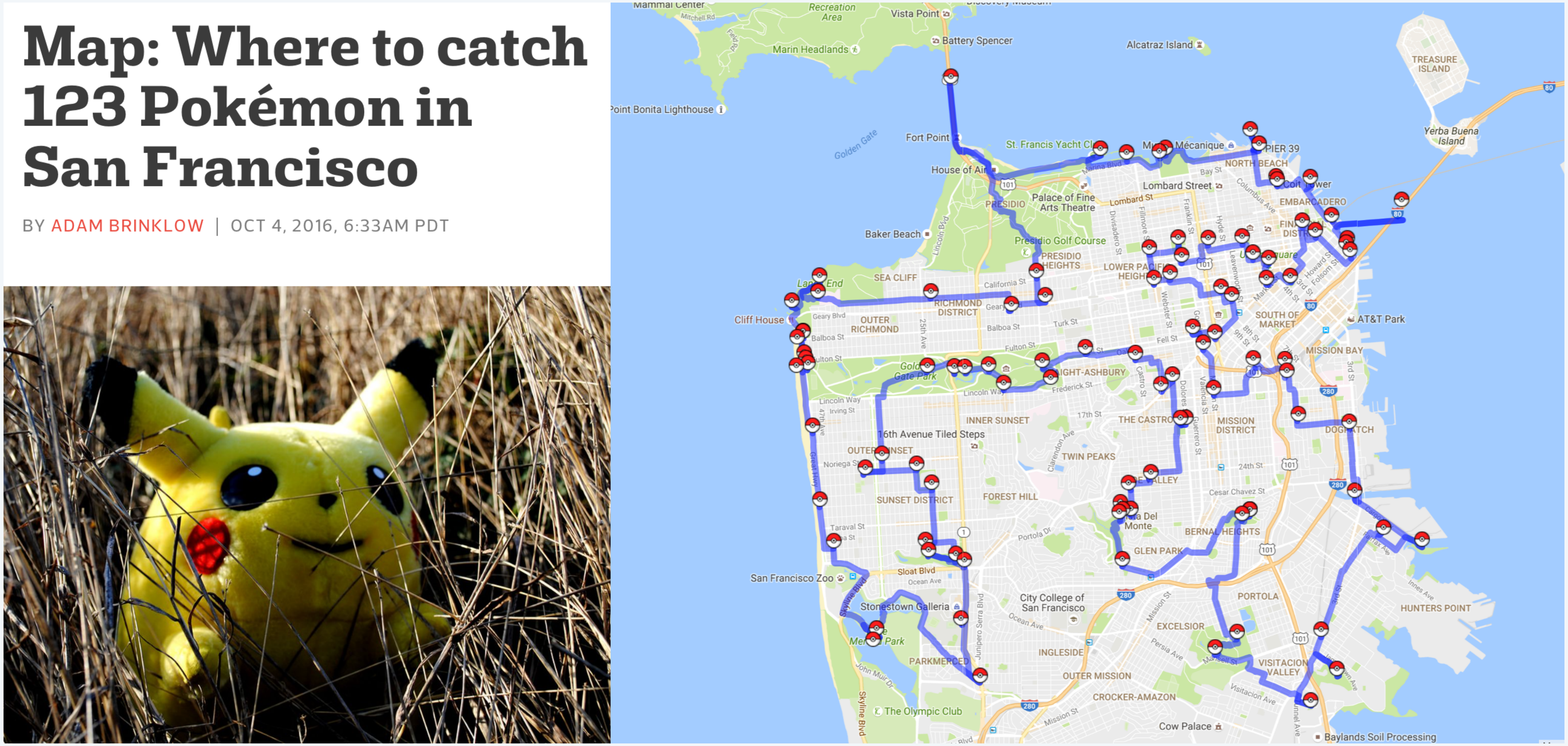
Many publicly available SAT solvers (e.g. GRASP [8], POSIT [5], SATO [13], rel_sat [2], WalkSAT [9]) have been developed, most employing some combination of two main strategies: the Davis-Putnam (DP) backtrack search and heuristic local search. Heuristic local search techniques are not guaranteed to be complete (i.e. they are not guaranteed to find a satisfying assignment if one exists or prove unsatisfiability); as a

# INTRACTABILITY III

- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

# Pokemon Go

Given the locations of $n$ Pokémon, find shortest tour to collect them all.



# Traveling salesperson problem

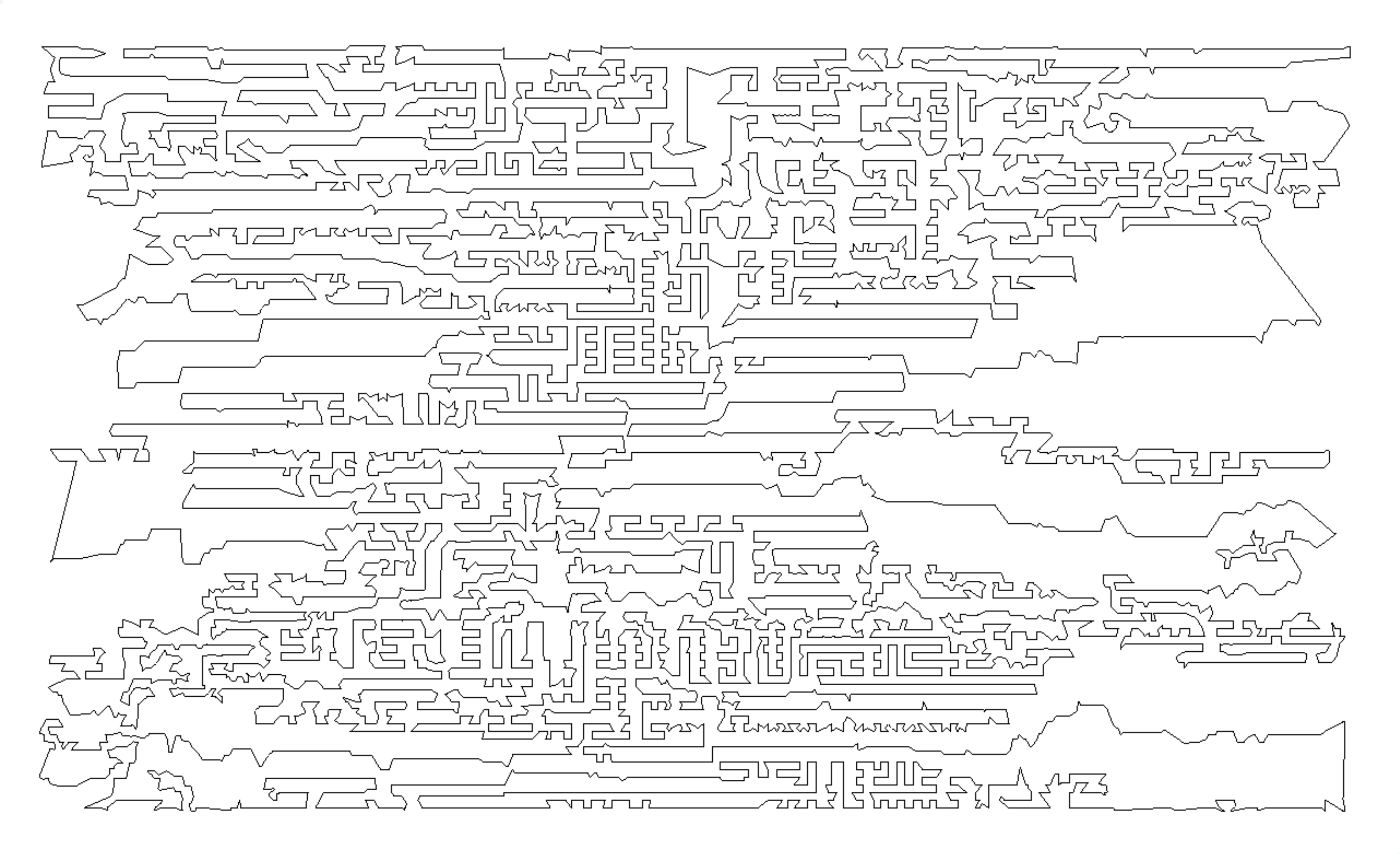
TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?

can view as a complete graph

13,509 cities in the United States
http://www.math.uwaterloo.ca/tsp

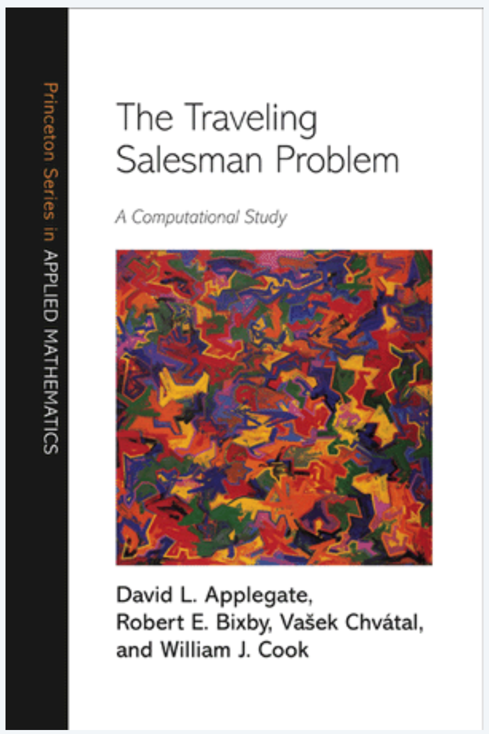
# Traveling salesperson problem

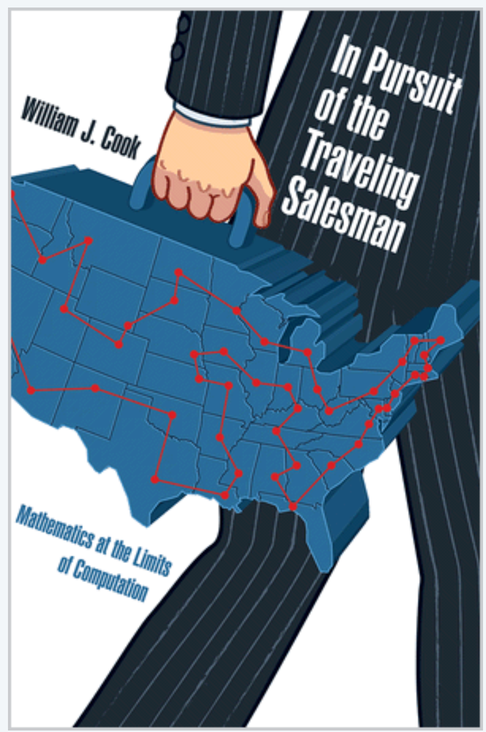
TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?

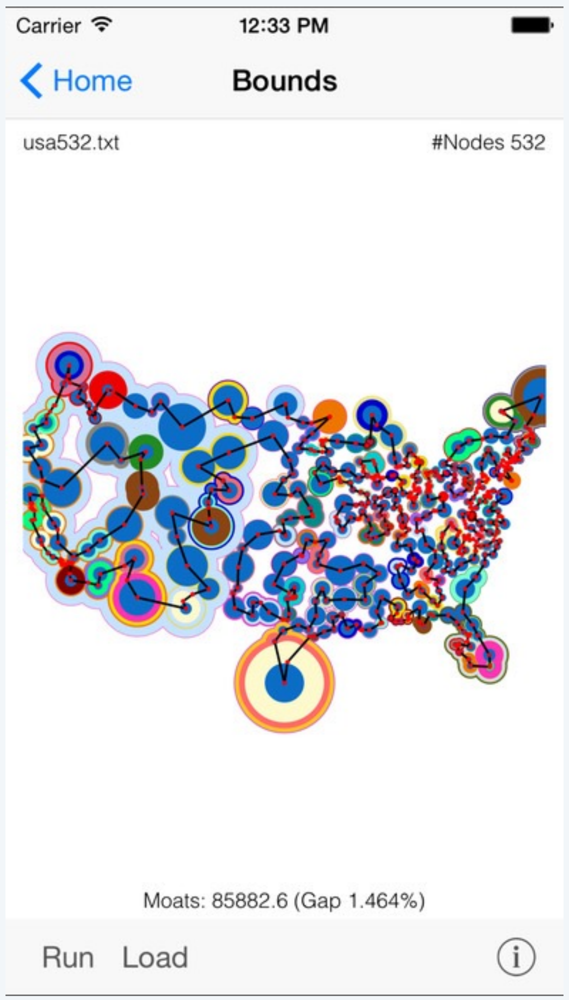


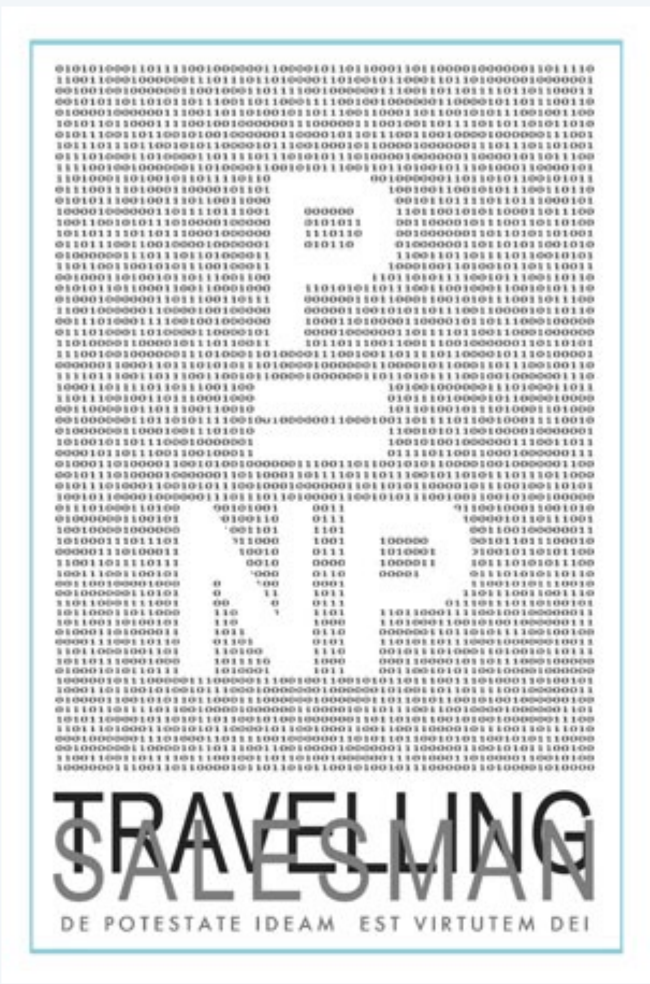
11,849 holes to drill in a programmed logic array
http://www.math.uwaterloo.ca/tsp

# TSP books, apps, and movies



# Hamilton cycle reduces to traveling salesperson problem

TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?

HAMILTON-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a cycle that visits every node exactly once?

Theorem. HAMILTON-CYCLE $\leq_P$ TSP.

Pf.

- Given an instance $G = (V, E)$ of HAMILTON-CYCLE, create $n = |V|$ cities with distance function

$$d(u, v) \; = \; \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

- TSP instance has tour of length $\leq n$ iff $G$ has a Hamilton cycle. ▪

**What is complexity of TSP? Choose the best answer.**

**A.** $O(n^2)$

**B.** $O^*(1.657^n)$

**C.** $O^*(2^n)$

**D.** $O^*(n!)$

$O^*$ hides $poly(n)$ terms

# Exponential algorithm for TSP: dynamic programming

Theorem. [Held–Karp, Bellman 1962] TSP can be solved in $O(n^2\, 2^n)$ time.

HAMILTON-CYCLE is a special case

J. Soc. Indust. Appl. Math.
Vol. 10, No. 1, March, 1962
*Printed in U.S.A.*

**A DYNAMIC PROGRAMMING APPROACH TO SEQUENCING PROBLEMS***

MICHAEL HELD† AND RICHARD M. KARP†

INTRODUCTION

Many interesting and important optimization problems require the determination of a best order of performing a given set of operations. This paper is concerned with the solution of three such *sequencing problems*: a scheduling problem involving arbitrary cost functions, the traveling-salesman problem, and an assembly-line balancing problem. Each of these problems has a structure permitting solution by means of recursion schemes of the type associated with dynamic programming. In essence, these recursion schemes permit the problems to be treated in terms of *combinations*, rather than *permutations*, of the operations to be performed. The dynamic programming formulations are given in §1, together with a discussion of various extensions such as the inclusion of precedence constraints. In each case the proposed method of solution is computationally effective for problems in a certain limited range. Approximate solutions to larger problems may be obtained by solving sequences of small derived problems, each having the same structure as the original one. This procedure of successive approximations is developed in detail in §2 with specific reference to the traveling-salesman problem, and §3 summarizes computational experience with an IBM 7090 program using the procedure.

**Dynamic Programming Treatment of the Travelling Salesman Problem***

RICHARD BELLMAN

*RAND Corporation, Santa Monica, California*

*Introduction*

The well-known travelling salesman problem is the following: "A salesman is required to visit once and only once each of $n$ different cities starting from a base city, and returning to this city. What path minimizes the total distance travelled by the salesman?"

The problem has been treated by a number of different people using a variety of techniques; cf. Dantzig, Fulkerson, Johnson [1], where a combination of ingenuity and linear programming is used, and Miller, Tucker and Zemlin [2], whose experiments using an all-integer program of Gomory did not produce results in cases with ten cities although some success was achieved in cases of simply four cities. The purpose of this note is to show that this problem can easily be formulated in dynamic programming terms [3], and resolved computationally for up to 17 cities. For larger numbers, the method presented below, combined with various simple manipulations, may be used to obtain quick approximate solutions. Results of this nature were independently obtained by M. Held and R. M. Karp, who are in the process of publishing some extensions and computational results.

# Exponential algorithm for TSP: dynamic programming

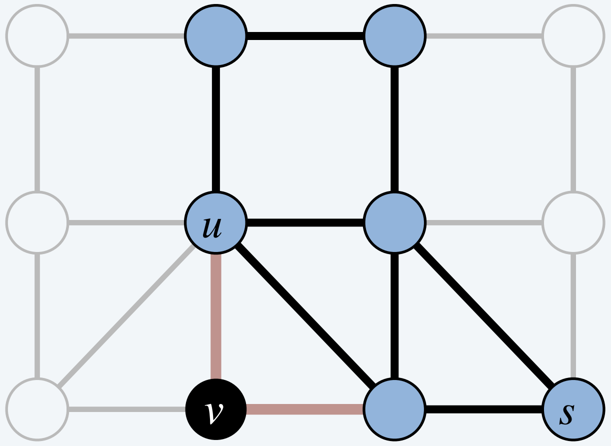
**Theorem.** [Held–Karp, Bellman 1962] TSP can be solved in $O(n^2\, 2^n)$ time.

**Pf.** [dynamic programming]

pick node $s$ arbitrarily

- Subproblems: $c(s, v, X)$ = cost of cheapest path between $s$ and $v \neq s$ that visits every node in $X$ exactly once (and uses only nodes in $X$).
- Goal: $\min_{v \in V} \; c(s, v, V) + c(v, s)$
- There are $\leq n\, 2^n$ subproblems and they satisfy the recurrence:

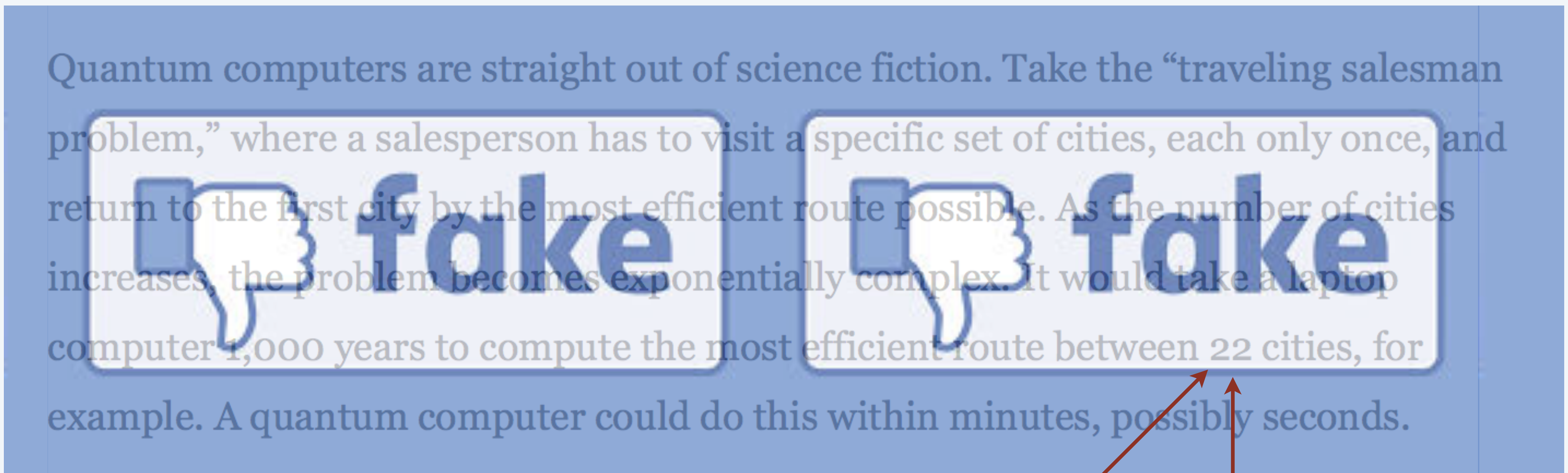
$$c(s,v,X) = \begin{cases} c(s,v) & \text{if } |X| = 2 \\ \min\limits_{u \in X \setminus \{s,v\}} c(s,u,X \setminus \{v\}) + c(u,v) & \text{if } |X| > 2. \end{cases}$$




- The values $c(s, v, X)$ can be computed in increasing order of the cardinality of $X$. ▪

# 22-city TSP instance takes 1,000 years

The Washington Post


Quantum computers are straight out of science fiction. Take the “traveling salesman problem,” where a salesperson has to visit a specific set of cities, each only once, and return to the first city by the most efficient route possible. As the number of cities increases, the problem becomes exponentially complex. It would take a laptop computer 1,000 years to compute the most efficient route between 22 cities, for example. A quantum computer could do this within minutes, possibly seconds.



$2^{22} = 4{,}194{,}304$

$22! = 1{,}124{,}000{,}727{,}777{,}607{,}680{,}000 \sim 10^{21}$

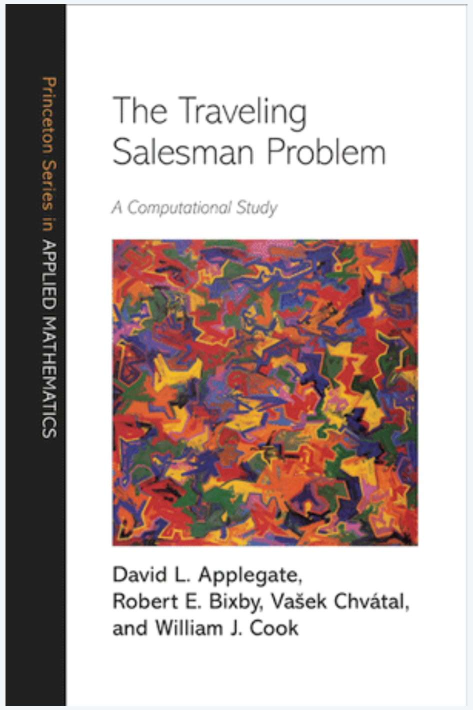
# Concorde TSP solver

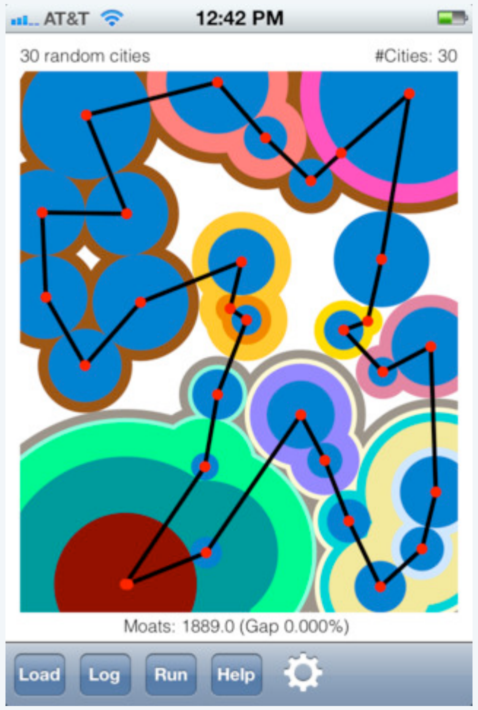
Concorde TSP solver. [Applegate–Bixby–Chvátal–Cook]

- Linear programming + branch-and-bound + polyhedral combinatorics.
- Greedy heuristics, including Lin–Kernighan.
- MST, Delaunay triangulations, fractional $b$-matchings, ...

Remarkable fact. Concorde has solved all 110 TSPLIB instances.

largest instance has 85,900 cities!



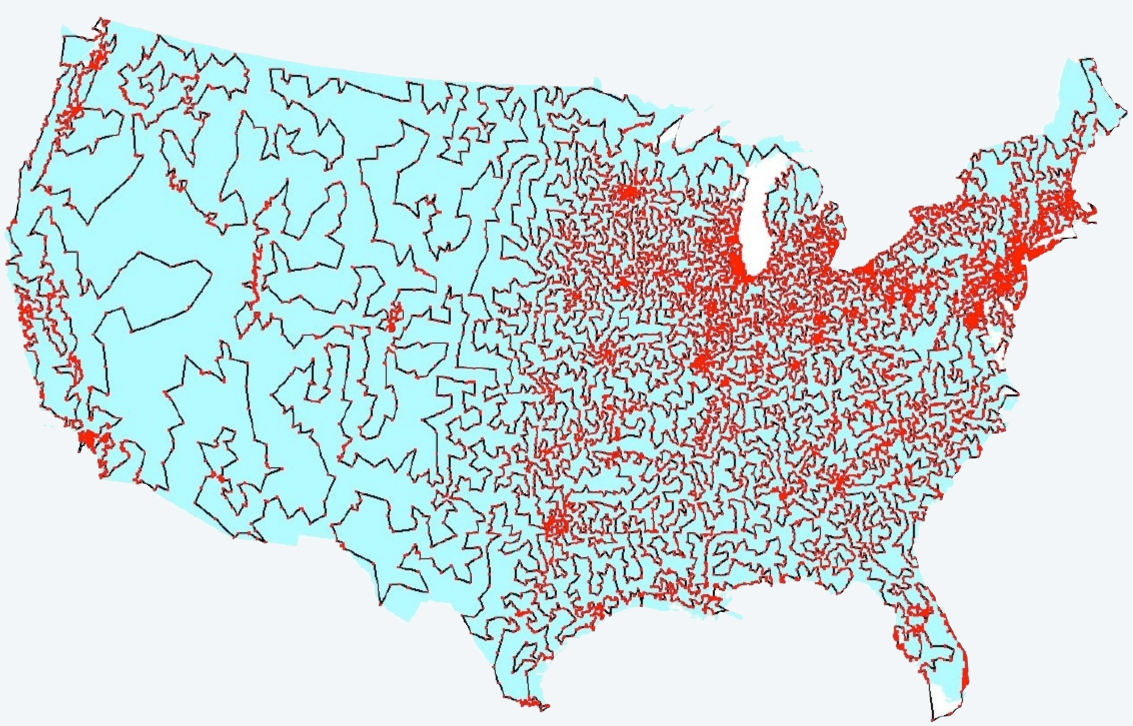



# Euclidean traveling salesperson problem

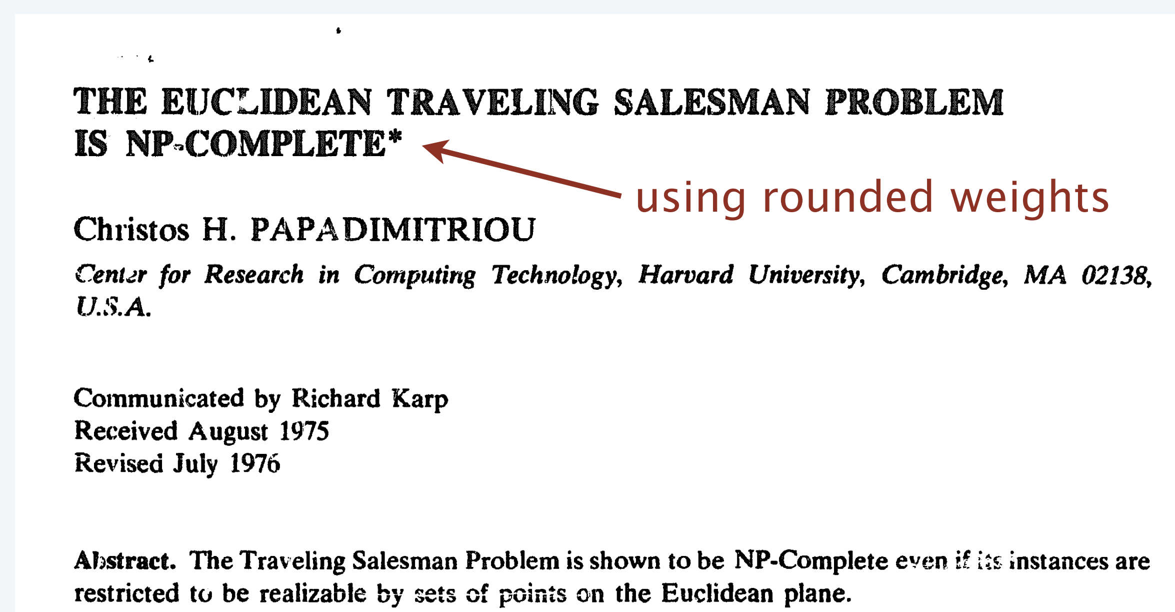
Euclidean TSP. Given $n$ points in the plane and a real number $L$, is there a tour that visit every city exactly once that has distance $\leq L$?

Fact. 3-SAT $\leq_P$ EUCLIDEAN-TSP.
Remark. Not known to be in **NP**.

$$\sqrt{5} + \sqrt{6} + \sqrt{18} \;<\; \sqrt{4} + \sqrt{12} + \sqrt{12}$$
$$8.928198407 \;<\; 8.928203230$$



13509 cities in the USA and an optimal tour



THE EUCLIDEAN TRAVELING SALESMAN PROBLEM IS NP-COMPLETE*

using rounded weights

Christos H. PAPADIMITRIOU
*Center for Research in Computing Technology, Harvard University, Cambridge, MA 02138, U.S.A.*

Communicated by Richard Karp
Received August 1975
Revised July 1976

**Abstract.** The Traveling Salesman Problem is shown to be NP-Complete even if its instances are restricted to be realizable by sets of points on the Euclidean plane.

# Euclidean traveling salesperson problem

Theorem. [Arora 1998, Mitchell 1999] Given $n$ points in the plane, for any constant $\varepsilon > 0$: there exists a poly-time algorithm to find a tour whose length is at most $(1 + \varepsilon)$ times that of the optimal tour.

Pf recipe. Structure theorem + divide-and-conquer + dynamic programming.

Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems

Sanjeev Arora
Princeton University

Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036, USA
Tel: (212) 555-1212; Fax: (212) 555-2000

We present a polynomial time approximation scheme for Euclidean TSP in fixed dimensions. For every fixed $c > 1$ and given any $n$ nodes in $\Re^2$, a randomized version of the scheme finds a $(1 + 1/c)$-approximation to the optimum traveling salesman tour in $O(n(\log n)^{O(c)})$ time. When the nodes are in $\Re^d$, the running time increases to $O(n(\log n)^{(O(\sqrt{d}c))^{d-1}})$. For every fixed $c, d$ the running time is $n \cdot \mathrm{poly}(\log n)$, i.e., *nearly linear* in $n$. The algorithm can be derandomized, but this increases the running time by a factor $O(n^d)$. The previous best approximation algorithm for the problem (due to Christofides) achieves a 3/2-approximation in polynomial time.

GUILLOTINE SUBDIVISIONS APPROXIMATE POLYGONAL SUBDIVISIONS: A SIMPLE POLYNOMIAL-TIME APPROXIMATION SCHEME FOR GEOMETRIC TSP, $K$-MST, AND RELATED PROBLEMS

JOSEPH S. B. MITCHELL*

**Abstract.** We show that any polygonal subdivision in the plane can be converted into an "$m$-guillotine" subdivision whose length is at most $(1 + \frac{c}{m})$ times that of the original subdivision, for a small constant $c$. "$m$-Guillotine" subdivisions have a simple recursive structure that allows one to search for shortest such subdivisions in polynomial time, using dynamic programming. In particular, a consequence of our main theorem is a simple polynomial-time approximation scheme for geometric instances of several network optimization problems, including the Steiner minimum spanning tree, the traveling salesperson problem (TSP), and the $k$-MST problem.