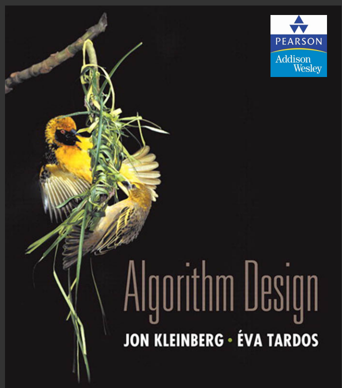



# Intractability III

- *special cases: trees*
- *approximation algorithms: vertex cover*
- *exponential algorithms: TSP*

Lecture slides by Kevin Wayne

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

Last updated on 5/2/18 2:37 PM

## Coping with NP-completeness

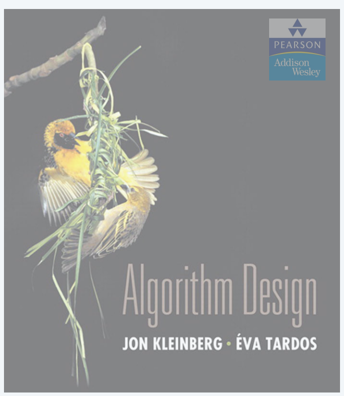
Q. Suppose I need to solve an **NP**-hard problem. What should I do?

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

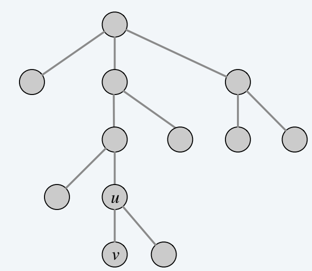
# Intractability III

- *special cases: trees*
- *approximation algorithms: vertex cover*
- *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.



## Intractability III: quiz 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.

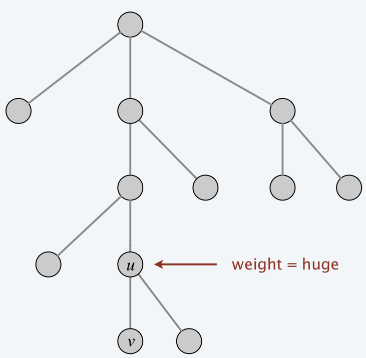
**D.** Any of the above.



## 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 $\Sigma_{v \in S}\, w_v$.

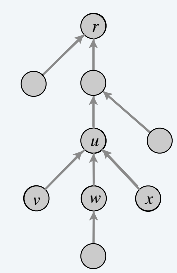
Dynamic-programming solution. Root tree at some node, say $r$.

- $OPT_{in}(u)$ = 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: $\max \{ OPT_{in}(r),\ OPT_{out}(r) \}$.

Bellman equation.

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

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




**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)

  IF ($u$ is a leaf)

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

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

  ELSE

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

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

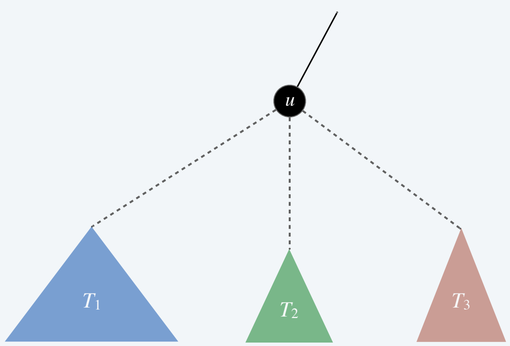
RETURN $\max \{ M_{in}[r],\ M_{out}[r] \}$.

ensures a node is processed after all of its descendants



## NP-hard problems on trees: context

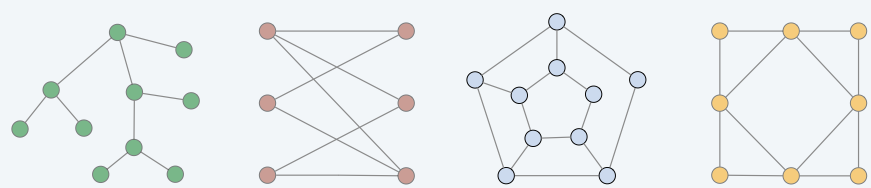
Independent set on trees. Tractable because we can find a node that breaks the communication among the subproblems in different subtrees.



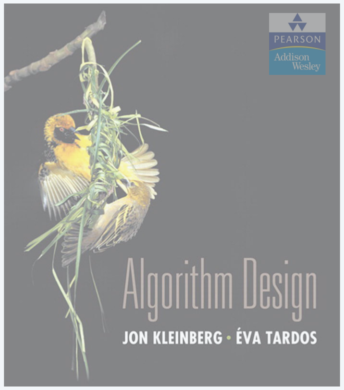



## Poly-time special cases of NP-hard problems

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
- approximation algorithms: vertex cover
- exponential algorithms: TSP

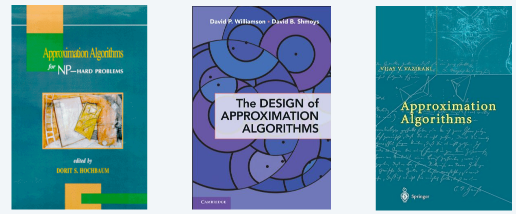
## Approximation algorithms

ρ-approximation algorithm.

- Runs in polynomial time.
- Applies to arbitrary instances of the problem.
- Guaranteed to find a solution within ratio ρ 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!

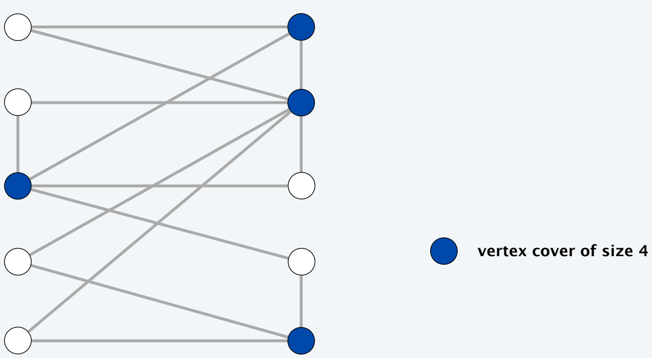





## Vertex cover

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.

## Intractability III: quiz 3

**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 $|S| \leq 2\,|S^*|$. ⟵ 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
- $|S| \underset{\text{design}}{=} 2\,|M| \underset{\text{weak duality}}{\leq} 2\,|S^*|$. ▪

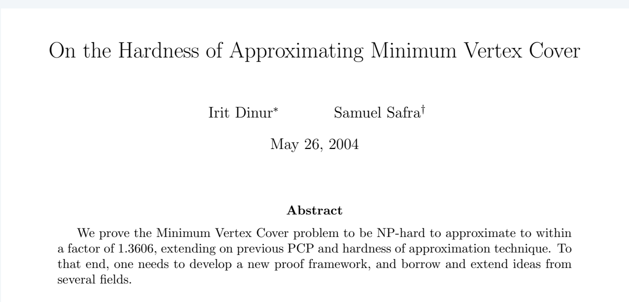
Corollary. Let $M^*$ be a maximum matching. Then, greedy algorithm computes a matching $M$ with $|M| \geq \frac{1}{2}\,|M^*|$.

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



## 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 $\rho < 2$.

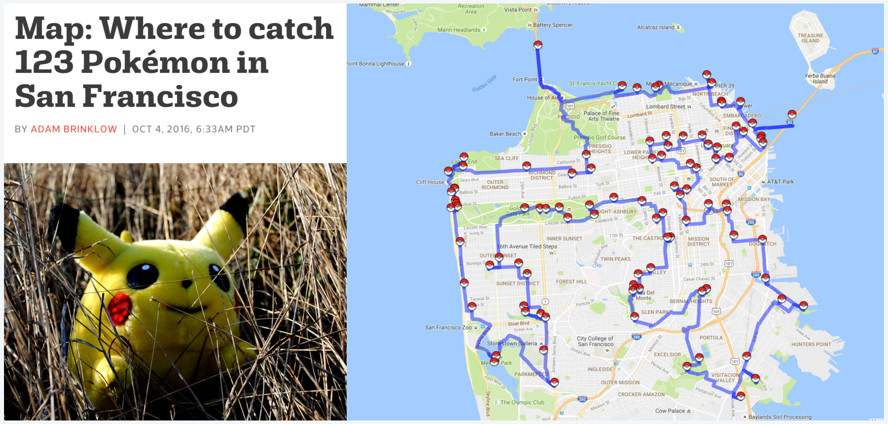


## INTRACTABILITY III

- special cases: trees
- approximation algorithms: vertex cover
- *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 $\leq D$ ?

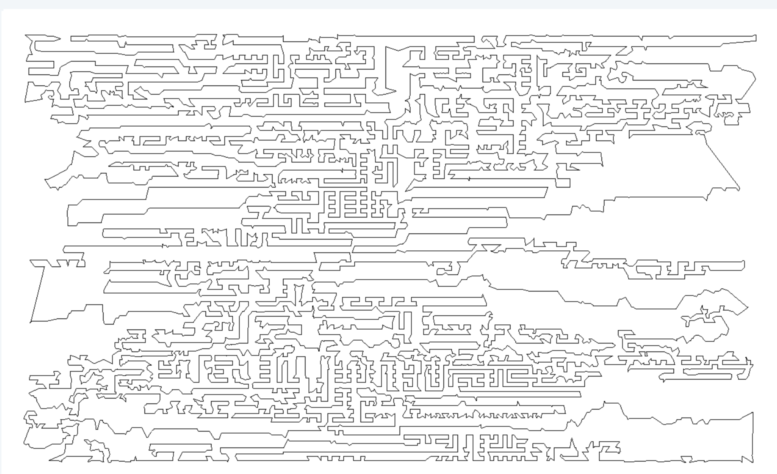
can view as a complete graph

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



## Traveling salesperson problem

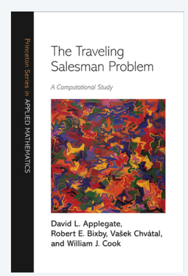
TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?

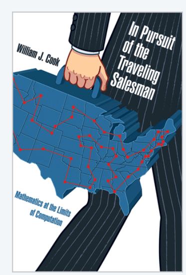


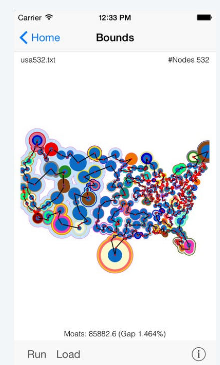
11,849 holes to drill in a programmed logic array
http://www.math.uwaterloo.ca/tsp

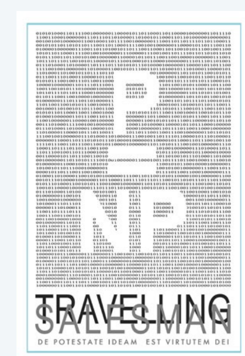


## TSP books, apps, and movies



## 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. ▪



## Intractability III: quiz 4

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

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

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

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

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

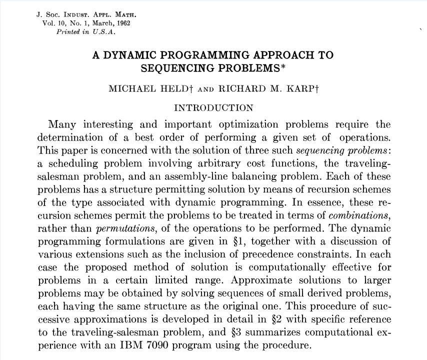
$O^*$ hides $poly(n)$ terms



## 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



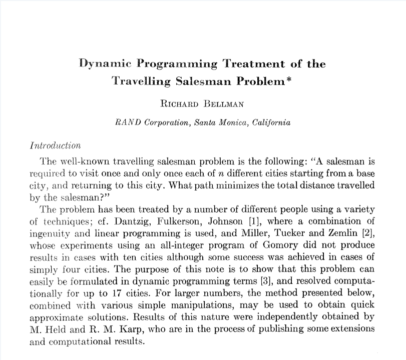
J. Soc. Indust. Appl. Math.
Vol. 10, No. 1, March, 1962
*Printed in U.S.A.*

**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

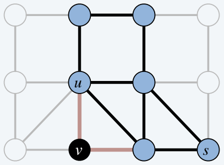
Theorem. [Held–Karp, Bellman 1962] TSP can be solved in $O(n^2\, 2^n)$ time.

Pf. [dynamic programming]

pick node $s$ arbitrarily

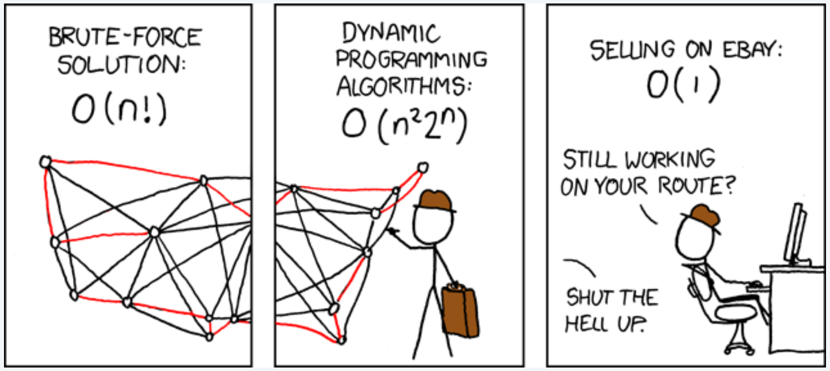
- Subproblems: $c(s, v, X)$ = cost of cheapest path between $s$ and $v \neq s$ that visits every node in $X$ exactly once (and uses only nodes in $X$).
- Goal: $\min_{v \in V} \; c(s, v, V) + c(v, s)$
- There are $\leq n\, 2^n$ subproblems and they satisfy the recurrence:

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




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

## Dynamic programming and the TSP




https://xkcd.com/399

## 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 ~ $10^{21}$



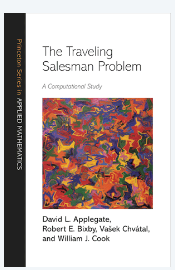
## Concorde TSP solver

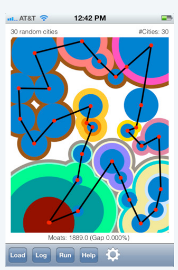
Concorde TSP solver. [Applegate–Bixby–Chvátal–Cook]

- 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!





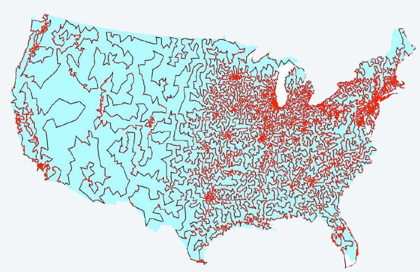

## Euclidean traveling salesperson problem

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 $\le L$?

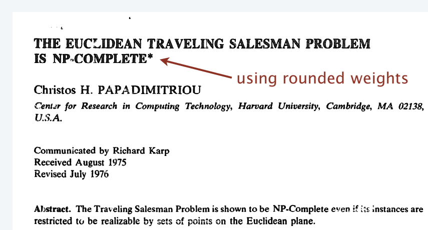
Fact. 3-SAT $\le_P$ EUCLIDEAN-TSP.

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*

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.

using rounded weights

## 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.




## That's all, folks: keep searching!

*Woh-oh-oh-oh, find the longest path!*
*Woh-oh-oh-oh, find the longest path!*

*If you said P is NP tonight,*
*There would still be papers left to write.*
*I have a weakness;*
*I'm addicted to completeness,*
*And I keep searching for the longest path.*

*The algorithm I would like to see*
*Is of polynomial degree.*
*But it's elusive:*
*Nobody has found conclusive*
*Evidence that we can find a longest path.*

*I have been hard working for so long.*
*I swear it's right, and he marks it wrong.*
*Somehow I'll feel sorry when it's done: GPA 2.1*
*Is more than I hope for.*

*Garey, Johnson, Karp and other men (and women)*
*Tried to make it order n log n.*
*Am I a mad fool*
*If I spend my life in grad school,*
*Forever following the longest path?*

*for (int i = 0; i < 3; i++)*
*Woh-oh-oh-oh, find the longest path!*

**Written by Dan Barrett in 1988 while a student at Johns Hopkins during a difficult algorithms take-home final**