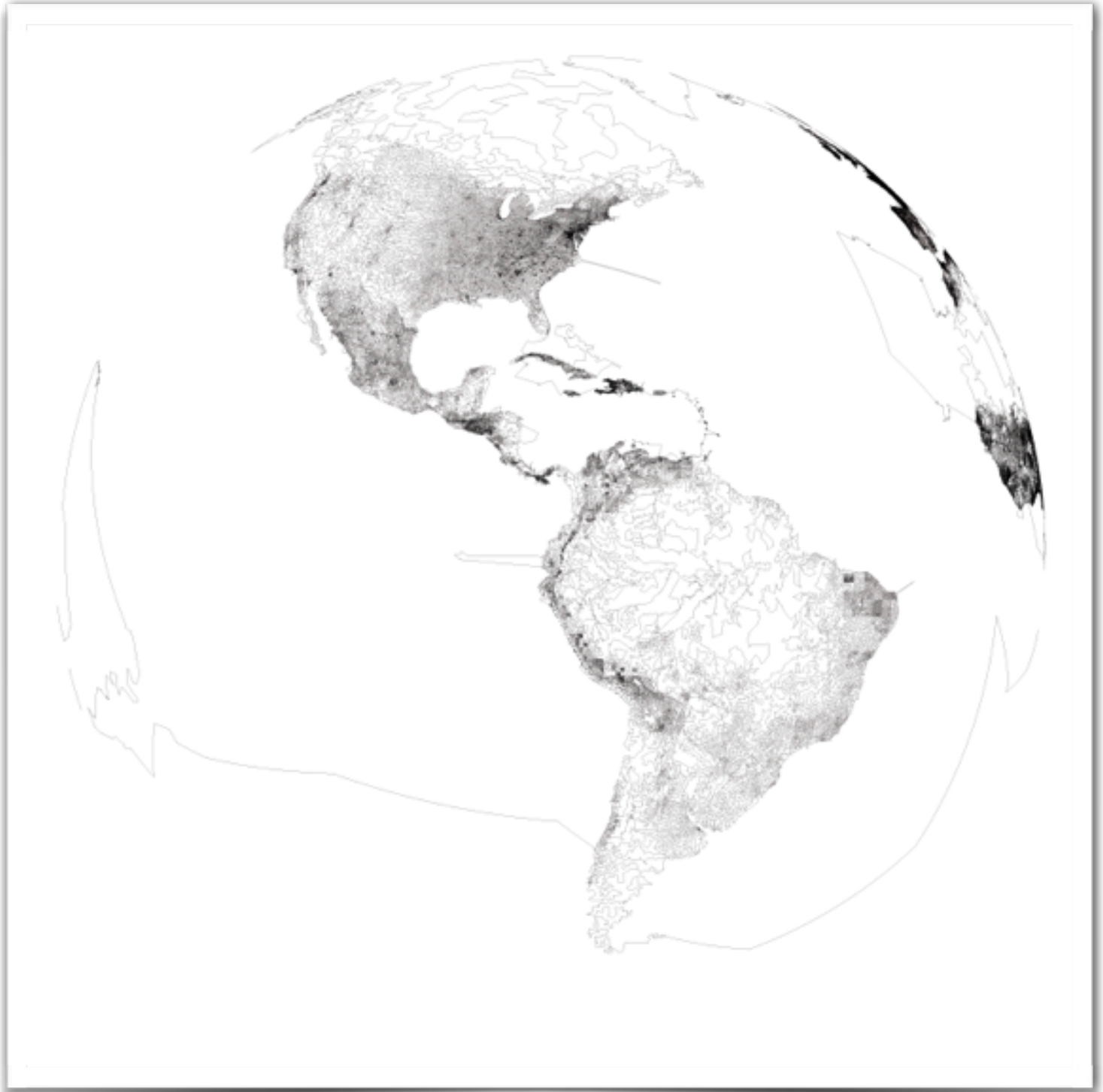
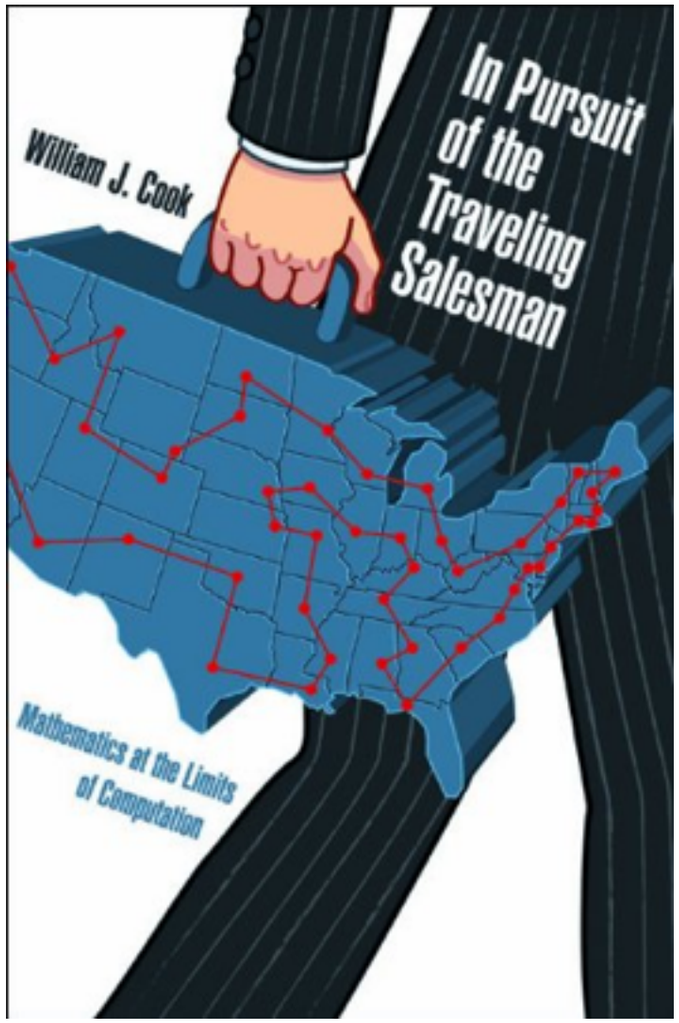


The Traveling Salesman Problem: An Overview

David P. Williamson, Cornell University
Ebay Research
January 21, 2014



(Cook 2012)
A highly readable introduction

Some terminology (imprecise)

- "Problem"
  - Traditional mathematics usage: e.g. Fermat's Last Problem
  - Computational usage: Find an algorithm (computer program) such that given *any* valid input, the desired output is produced.
- A *decision problem*: The output is one of "Yes" or "No".

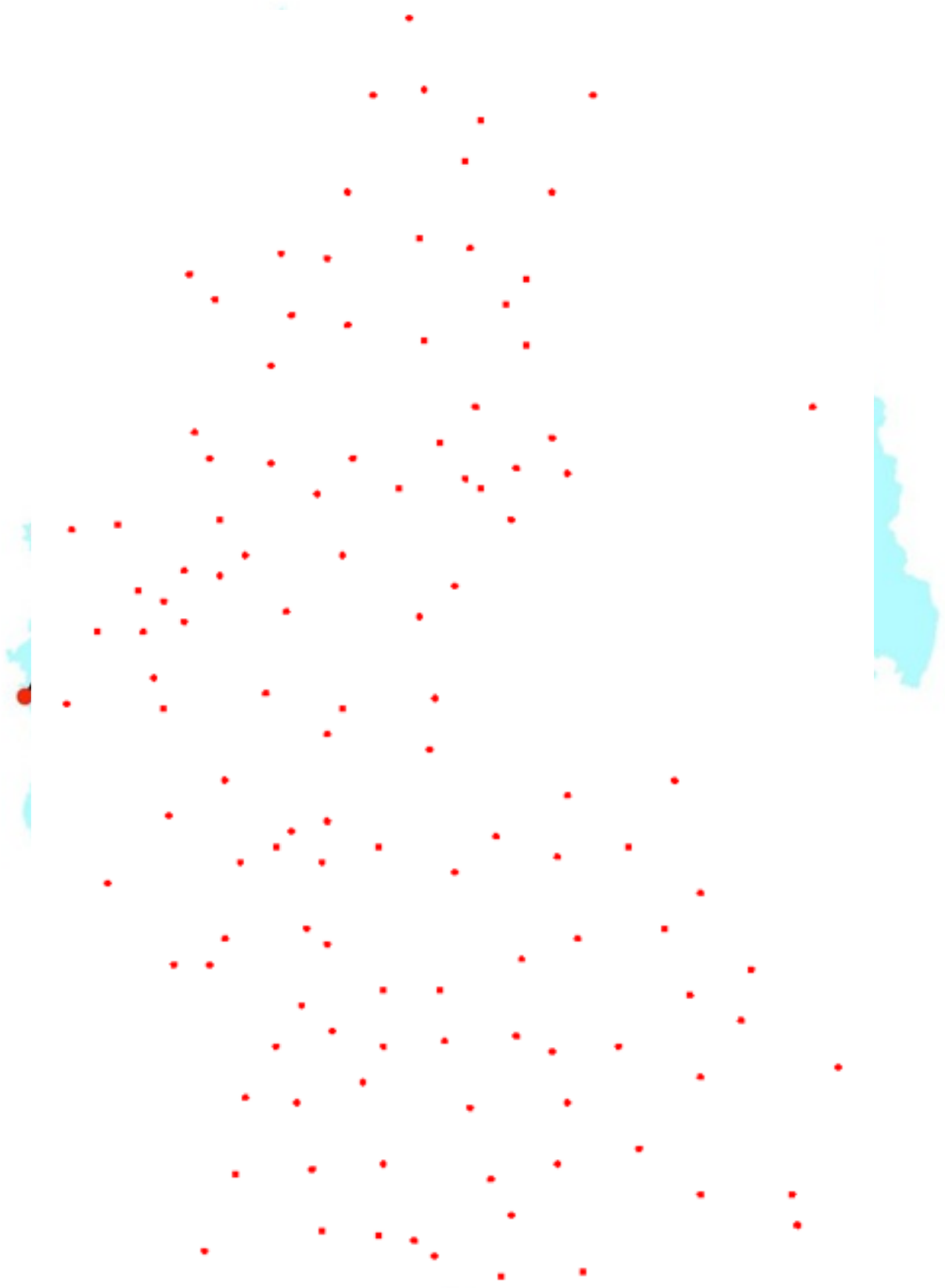
Discrete Optimization Problems

- Appears in many places: scheduling jobs on computers, locating facilities, building networks, stocking inventory,...

- Famous example: *the traveling salesman problem (TSP).*

    - Given *n* cities and the distances between each pair of cities, find the shortest *tour* that visits each city once and returns to the starting point.

- Decision version of TSP: Additional input of a number *C*, “Is the length of the shortest tour at most *C*?”



History

Point of origin of “traveling salesman problem” is unknown. In the US, it seems to have started at Princeton in the 1930s.

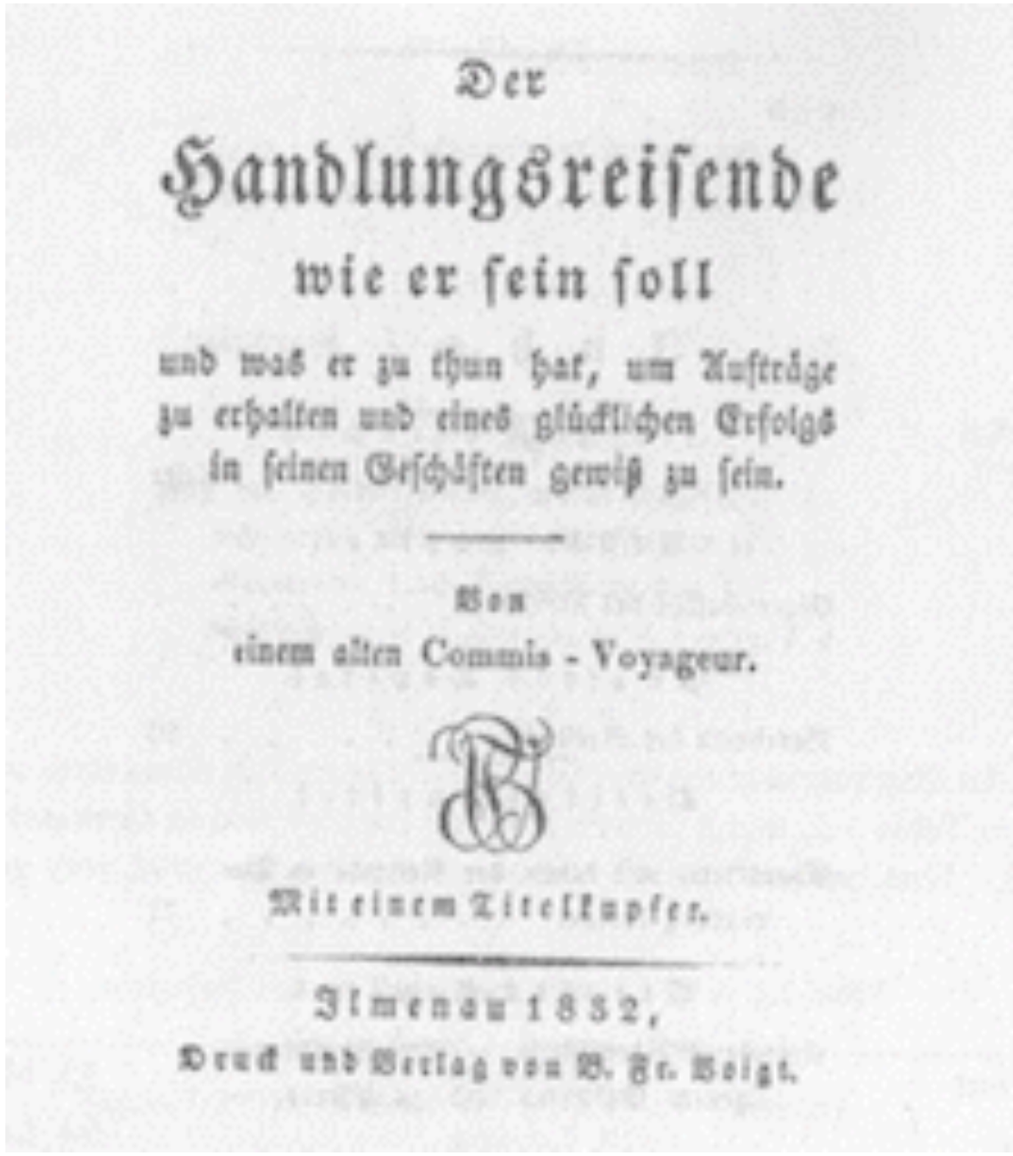
> Koopmans first became interested in the “48 States Problem” of Hassler Whitney when he was with me in the Princeton Surveys, as I tried to solve the problem in connection with the work of Bob Singleton and me on school bus routing for the State of West Virginia. I don’t know who coined the peppier name “Traveling Salesman Problem” for Whitney’s problem, but that name certainly caught on (Flood, 1984 interview)

RAND offers a prize for TSP work in the late 1940s. Paper mentions it in 1949 by name.

History

Of course the problem was faced by real salesman, who realized the difficulty.

> “Business leads the traveling salesman here and there, and there is not a good tour for all occurring cases; but through an expedient choice and division of the tour so much time can be won that we feel compelled to give guidelines about this. Everyone should use as much of the advice as he thinks useful for his application. We believe we can ensure as much that it will not be possible to plan the tours through Germany in consideration of the distances and the traveling back and forth, which deserves the traveler’s special attention, with more economy. The main thing to remember is always to visit as many localities as possible without having to touch them twice.”



Der

Handlungsreisende

wie er sein soll

und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein.

Von

einem alten Commis - Voyageur.

Mit einem Titelkupfer.

Ilmenau 1832,

Druck und Verlag von B. Fr. Voigt.

HELP! WE'RE LOST!

HELP "CAR 54"...AND WIN CASH

54...$1,000 PRIZES
ONE...$10,000 GRAND PRIZE

Map by Rand McNally

Help Toody and Muldoon find the shortest round trip route to visit all 33 locations shown on the map.

All you do is draw connecting straight lines from location to location to show the shortest round trip route.

HERE'S THE CORRECT START...

Begin at Chicago, Illinois. From there, lines show correct route as far as Erie, Pennsylvania. Next, do you go to Carlisle, Pennsylvania or Wana, West Virginia? Check the easy instructions on back of this entry blank for details.

OFFICIAL RULES ON REVERSE SIDE

The Obvious Finite Algorithm

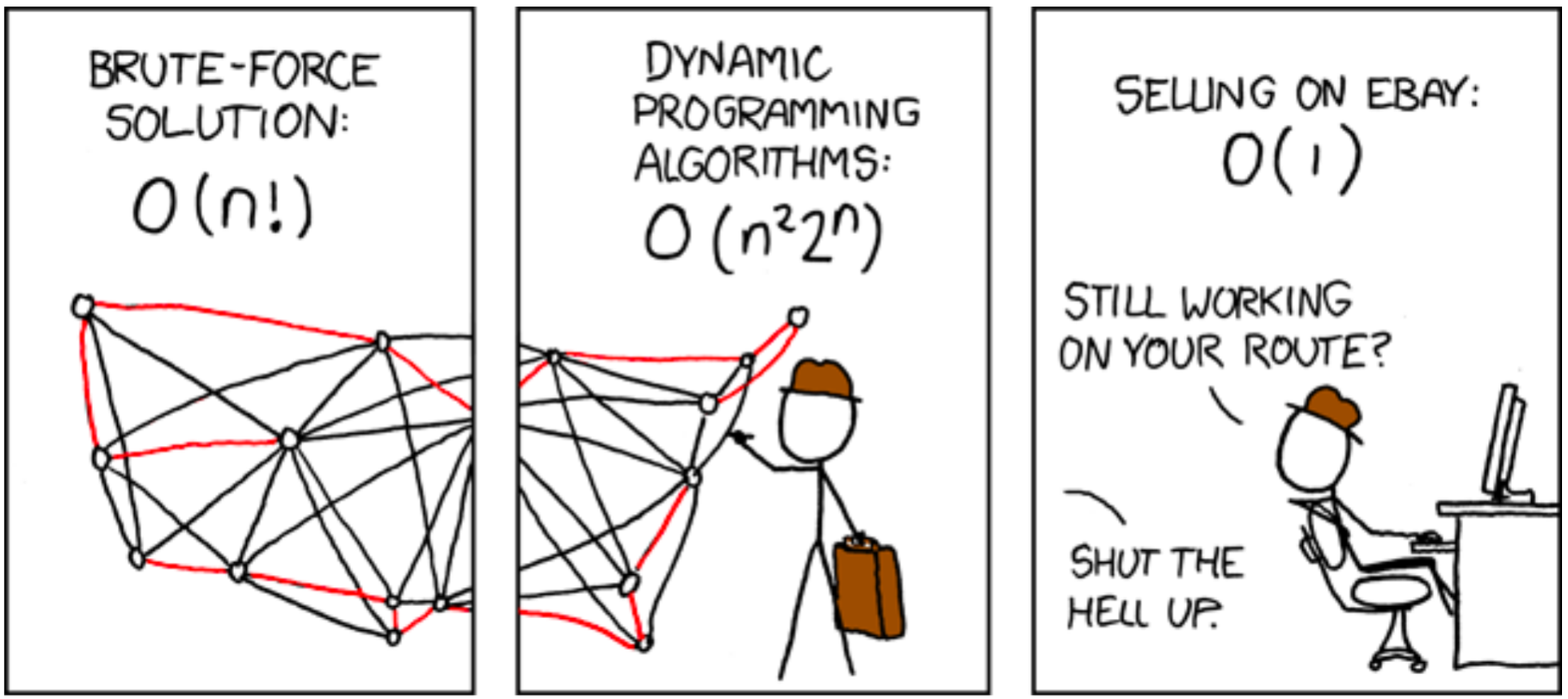
- Consider all *n!* possible orderings of the cities and compute the length of the tour for that ordering, where *n! = n x (n-1) x (n-2) x … x 1.* Keep track of the shortest one found.

- Problem: *n!* grows pretty quickly with *n.* 120! is about 6×10^{198}. 1 tour/ns still is about 10^{182} years.

- Can do better than this algorithm by an algorithm using on the order of $n^2 \times 2^n$ operations. But still on the order of 2×10^{40} operations; at 1 op/ns still about 10^{23} years.

XKCD

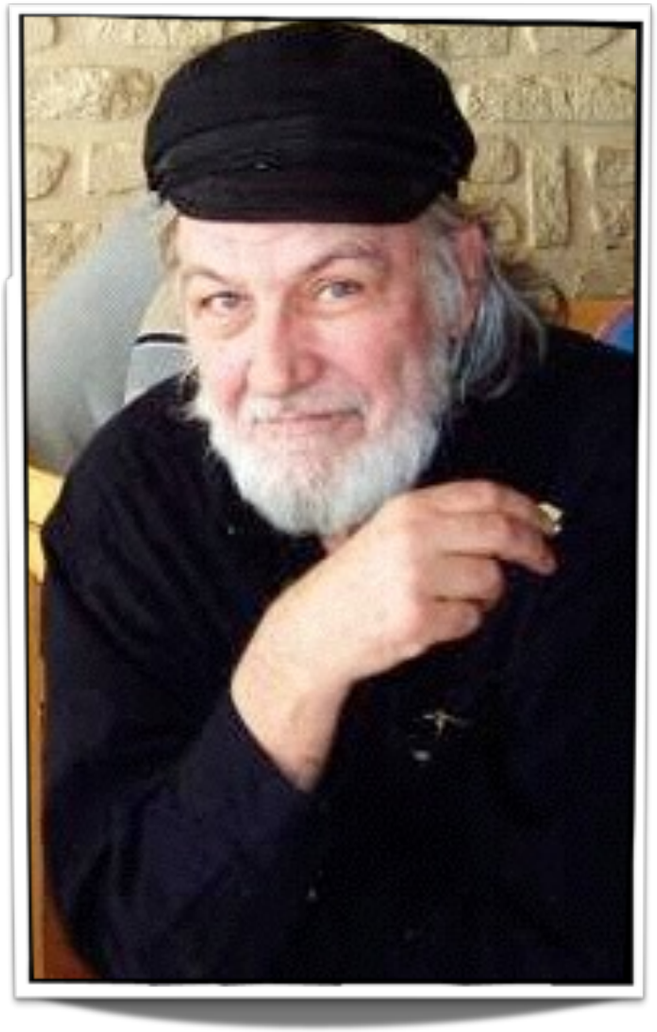
- or you can just sell stuff on Ebay.



“Good” algorithms

Edmonds (1965):

One can find many classes of problems, besides maximum matching and its generalizations, which have algorithms of exponential order but seemingly none better. An example known to organic chemists is that of deciding whether two given graphs are isomorphic. For practical purposes the difference between algebraic and exponential order is often more crucial than the difference between finite and non-finite.



Edmonds (1967):

We say an algorithm is *good* if there is a polynomial function $f(n)$ which, for every positive-integer valued n, is an upper bound on the “amount of work” the algorithm does for any input of “size” n. The concept

raveling saleman problem [cf. 4]. I conjecture that there is no good algorithm for the traveling saleman problem. My reasons are the same as for any mathematical conjecture: (1) It is a legitimate mathematical possibility, and (2) I do not know.

P vs. NP

- Today: *polynomial-time algorithms* are considered the theoretical measure of a good, efficient algorithm.

- *P* is the class of all decision problems solvable by a polynomial-time algorithm.

- *NP* is (roughly) the set of all decision problems for which we can “check” in polynomial time whether the answer is “Yes” (or “No”) if someone gives us a “proof”.

- (Cook, Levin 1971, Karp 1972) Given a polynomial-time algorithm for the decision version of the TSP, we can get a polynomial-time algorithm for any problem in *NP.*

$$P = NP?$$

DOI:10.1145/1562164.1562186

It's one of the fundamental mathematical problems of our time, and its importance grows with the rise of powerful computers.

BY LANCE FORTNOW

The Status of the P versus NP Problem

The software written for this illustration makes a stylized version of a network graph that draws connections between elements based on proximity. The graph constantly changes as the elements sort themselves.

WHEN EDITOR-IN-CHIEF MOSHE Vardi asked me to write this piece for *Communications*, my first reaction was the article could be written in two words:

Still open.

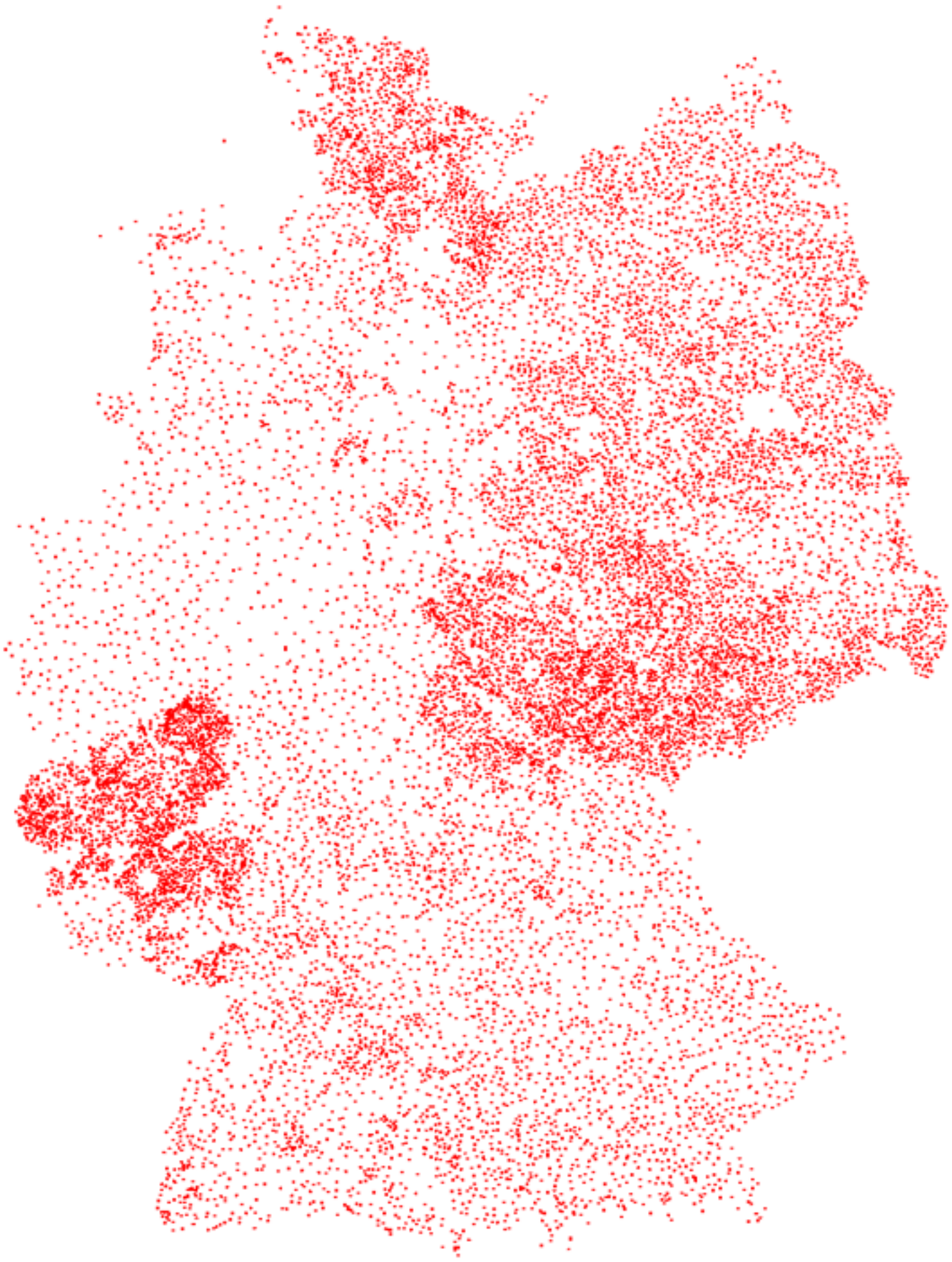
When I started graduate school in the mid-1980s, many believed that the quickly developing area of circuit complexity would soon settle the **P** versus **NP** problem, whether every algorithmic problem with efficiently verifiable solutions have efficiently computable solutions. But circuit complexity and other approaches to the problem have stalled and we have little reason to believe we will see a proof separating **P** from **NP** in the near future.

increased, the cost of computing has dramatically decreased, not to mention the power of the Internet. Computation has become a standard tool in just about every academic field. Whole subfields of biology, chemistry, physics, economics and others are devoted to large-scale computational modeling, simulations, and problem solving.

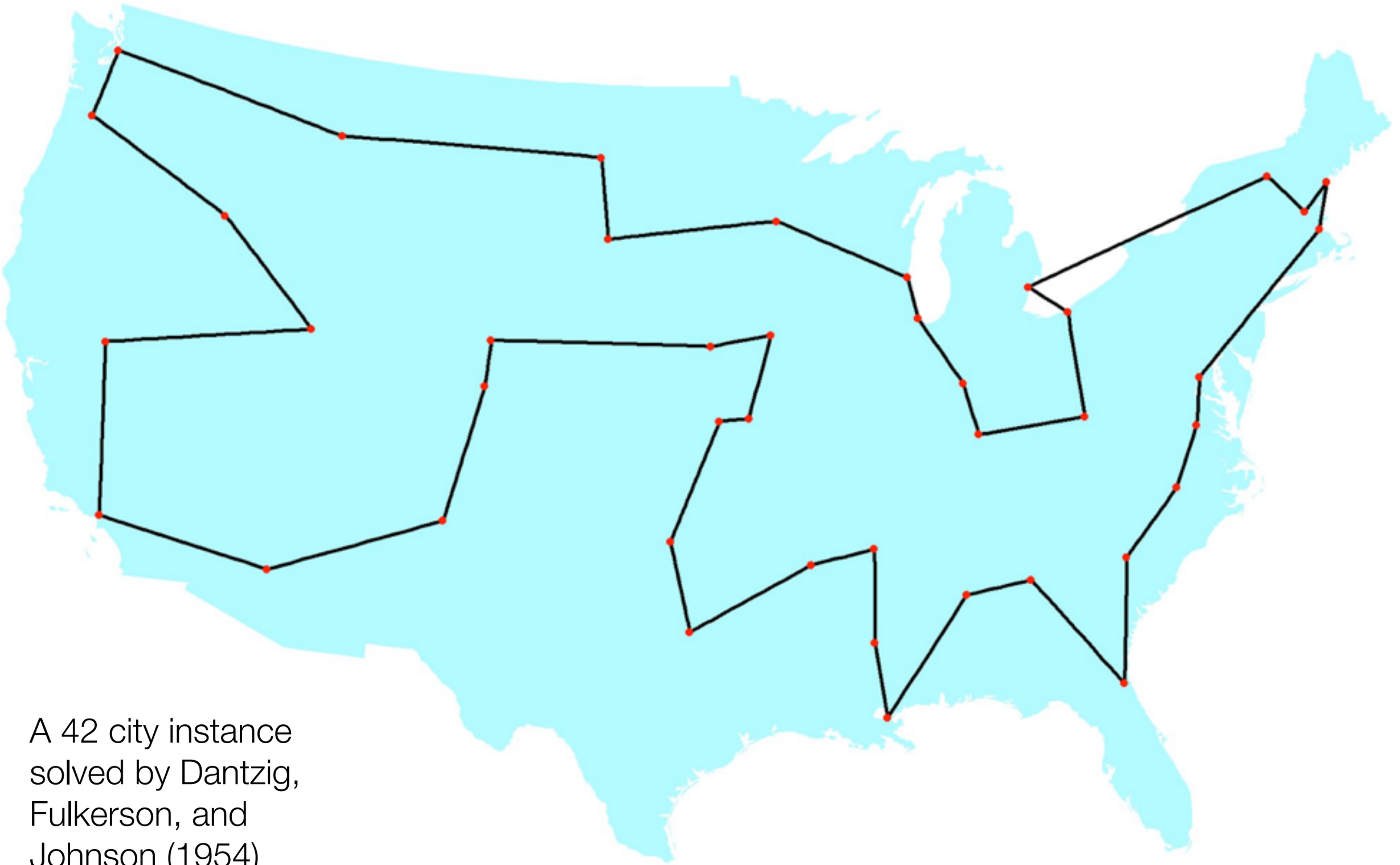
As we solve larger and more complex problems with greater computational power and cleverer algorithms, the problems we cannot tackle begin to stand out. The theory of **NP**-completeness helps us understand these limitations and the **P** versus **NP** prob-

Communications of the ACM, Sept. 2009

A 15112 city instance solved by Applegate, Bixby, Chvátal, and Cook (2001)



A 24978 city instance from Sweden solved by Applegate, Bixby, Chvátal, Cook, and Helsgaun (2004)



A 42 city instance solved by Dantzig, Fulkerson, and Johnson (1954)

George Dantzig (1914-2005) and Linear Programming

- Publishes paper on the Simplex Method for solving linear programs (LPs) in 1947.

- A linear program finds values for variables x_1, x_2, …, x_n that minimizes a linear function (an objective function) subject to linear inequalities or equations on the variables (constraints).

- Example:

$$\begin{aligned} &\text{Maximize } 5x_1 + 5x_2 + 3x_3 \\ &\text{subject to:} \\ x_1 + 3x_2 + x_3 &\leq 3 \\ -x_1 + 3x_3 &\leq 2 \\ 2x_1 - x_2 + 2x_3 &\leq 4 \\ 2x_1 + 3x_2 - x_3 &\leq 2 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

Any solution x that obeys all the constraints is a *feasible* solution. Any feasible solution that maximizes the objective function is an *optimal* solution.

The Dantzig-Fulkerson-Johnson Method

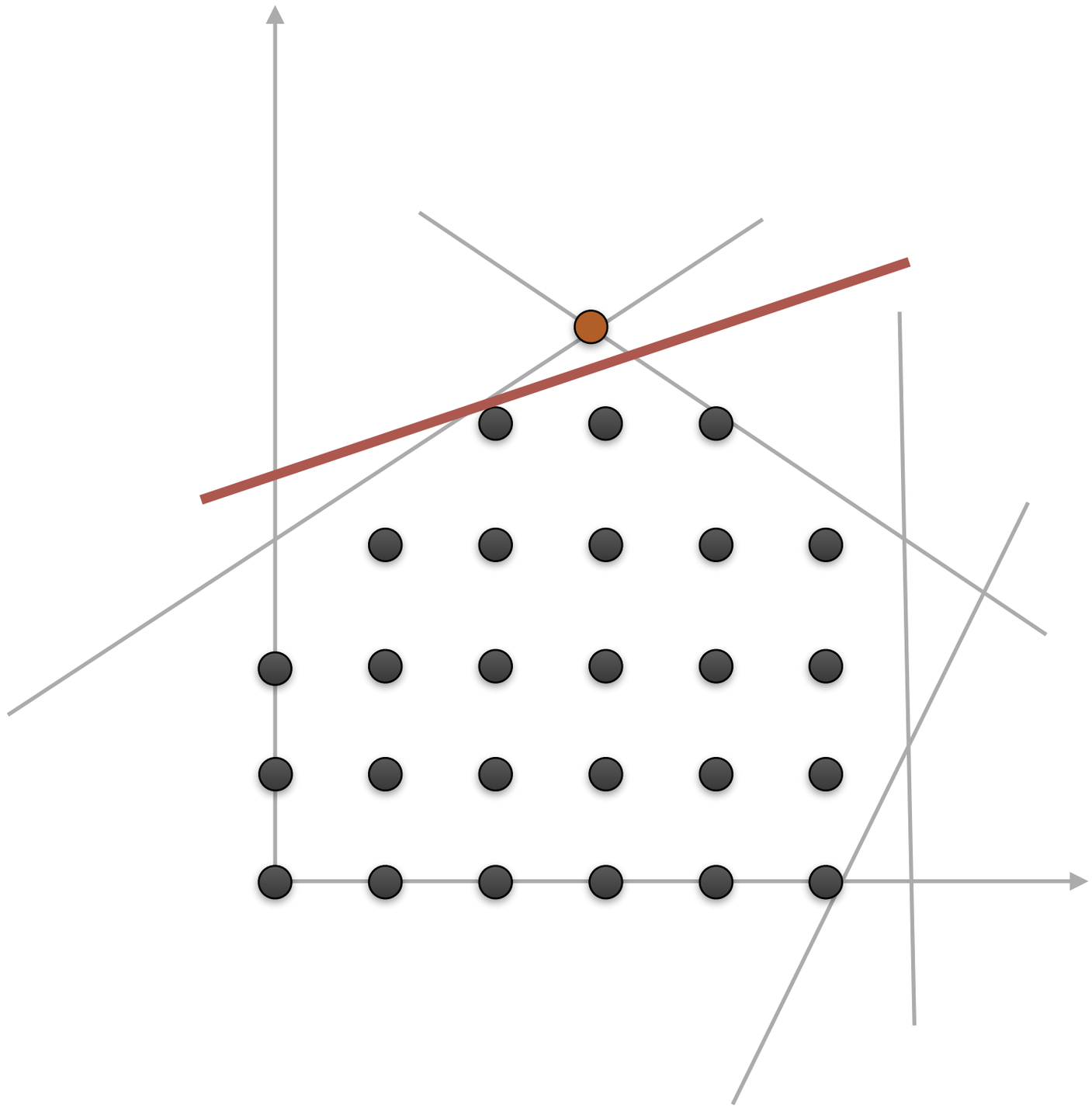
- Set up a linear program! Create a variable x(i,j) for each pair of cities i and j. Want x(i,j) = 1 if salesman travels between i and j, x(i,j) = 0 if he doesn't.

- Let c(i,j) be the cost of traveling between cities i and j.

- Let V denote the set of all cities. Let E denote the set of all (i,j) pairs. Get a *complete* graph G=(V,E).

- Idea: Solve the linear program. If the x(i,j) are integers and form a tour, stop. Otherwise, find a new constraint to add (a *cutting plane*).

Cutting planes



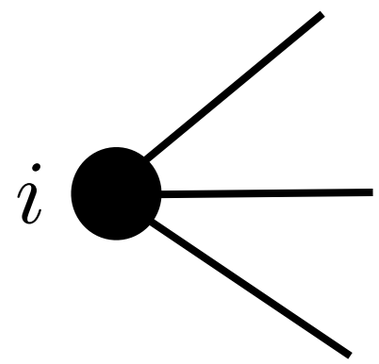
The linear program

$$\text{Minimize} \sum_{(i,j)\in E} c(i,j)x(i,j)$$

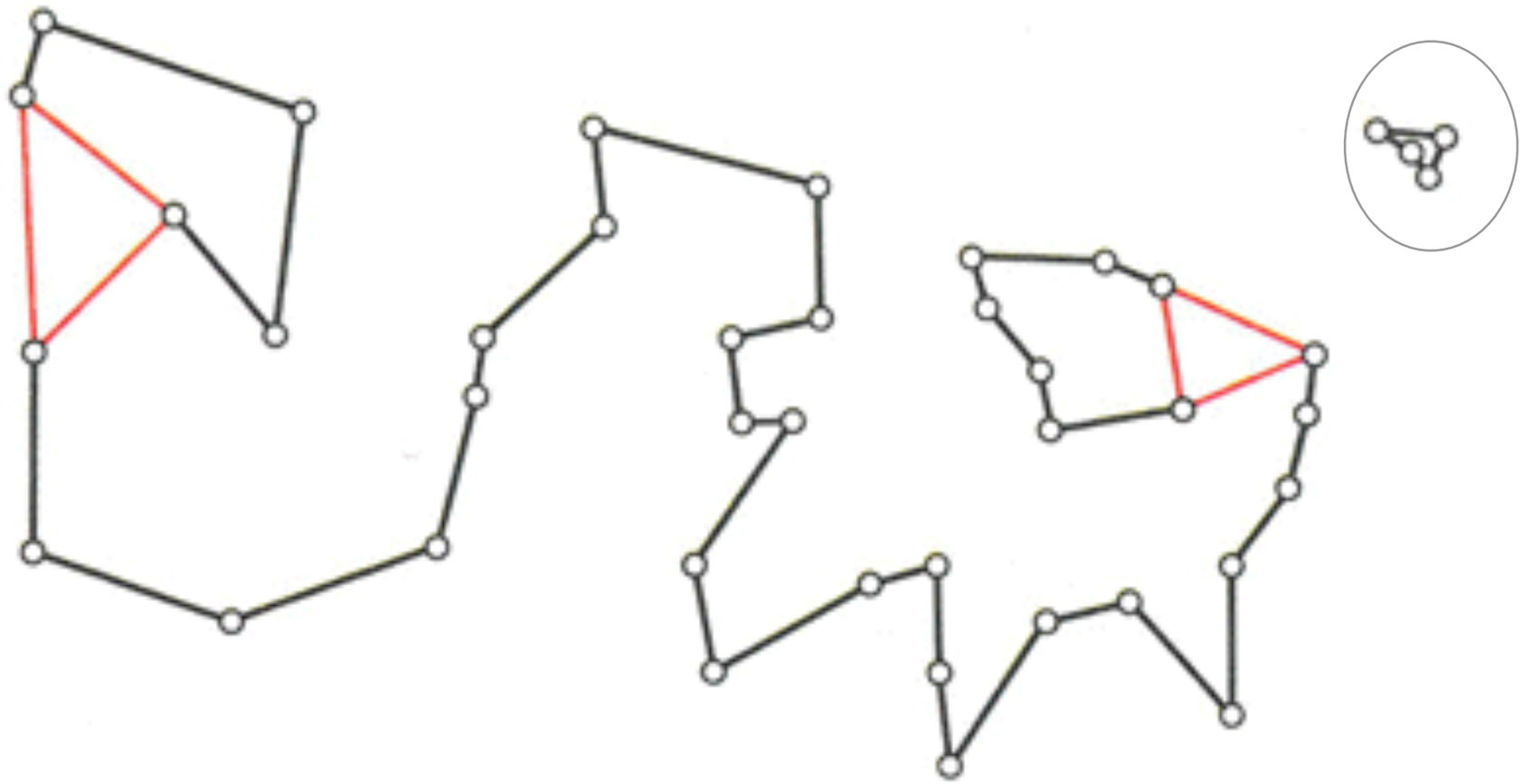
$$\text{subject to:}$$

$$\sum_{j:(i,j)\in E} x(i,j) = 2 \qquad \text{for all } i \in V$$

$$0 \leq x(i,j) \leq 1 \qquad \text{for all } (i,j) \in E$$



Initial solution to 42 city problem

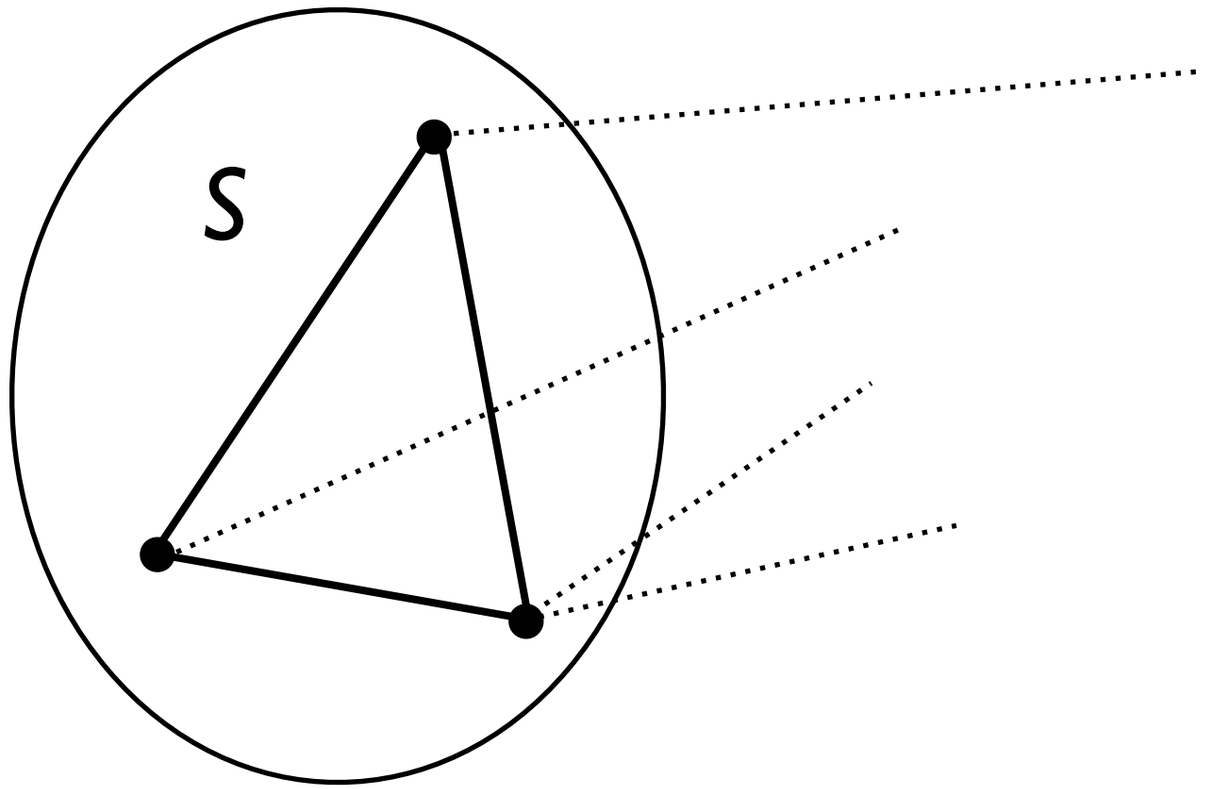


From Cook, p. 128

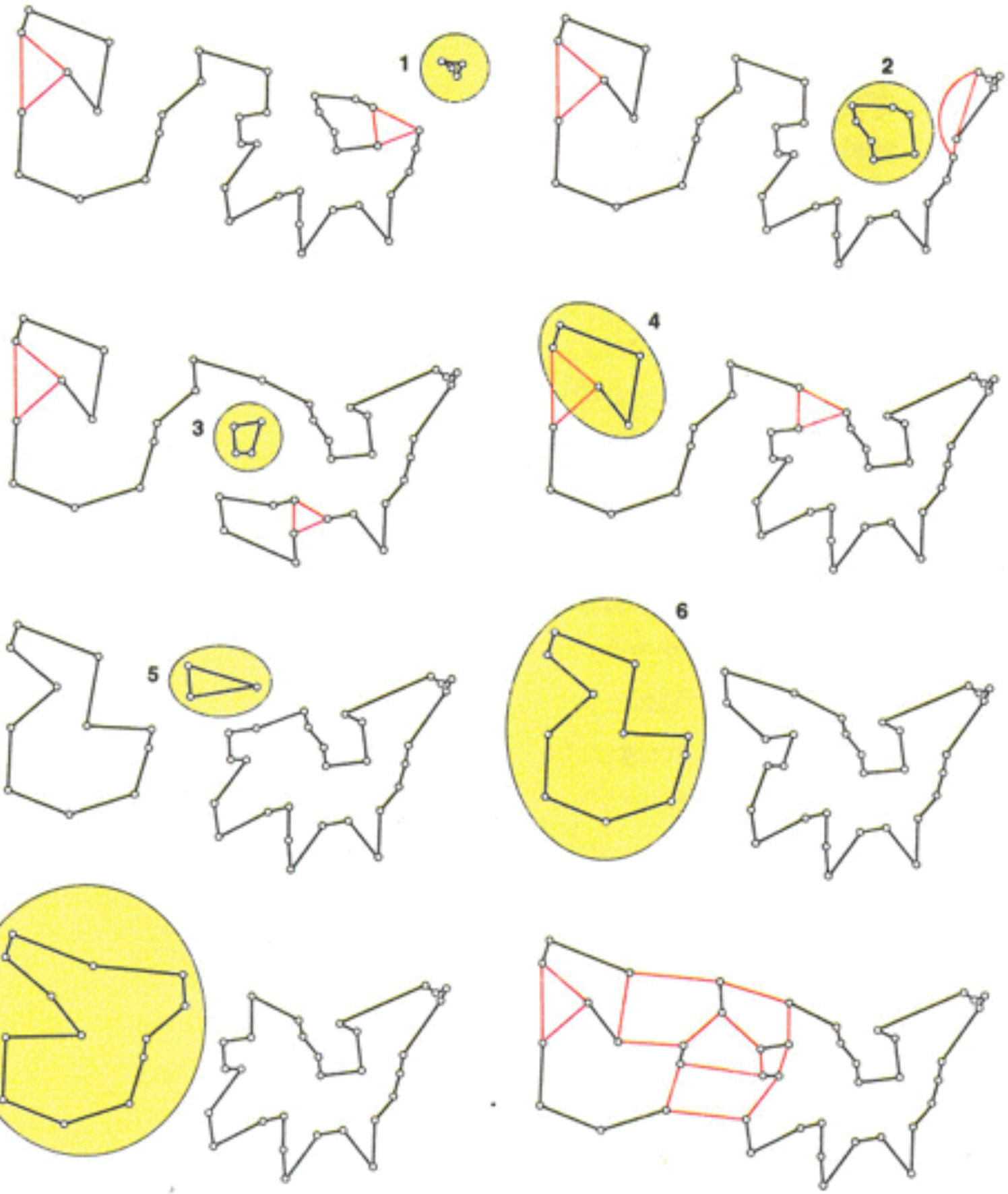
“Loop conditions”

- For any subset S of cities, the tour must enter the set S at least once and leave the set S at least once.

$$\sum_{(i,j)\in E: i\in S, j\notin S} x(i,j) \geq 2 \qquad \text{for all } S \subset V.$$



Constraints are sometimes called “subtour elimination constraints”.



Cook, p. 129

Subtour Linear Program (Subtour LP)

$$\text{Minimize} \sum_{(i,j)\in E} c(i,j)x(i,j)$$

$$\text{subject to:}$$

$$\sum_{j:(i,j)\in E} x(i,j) = 2 \qquad \text{for all } i \in V$$

$$\sum_{(i,j)\in E: i\in S, j\notin S} x(i,j) \geq 2 \qquad \text{for all } S \subset V$$

$$0 \leq x(i,j) \leq 1 \qquad \text{for all } (i,j) \in E$$

Any integer solution to this LP is a tour. Any optimal solution is a lower bound on the cost of the shortest tour.

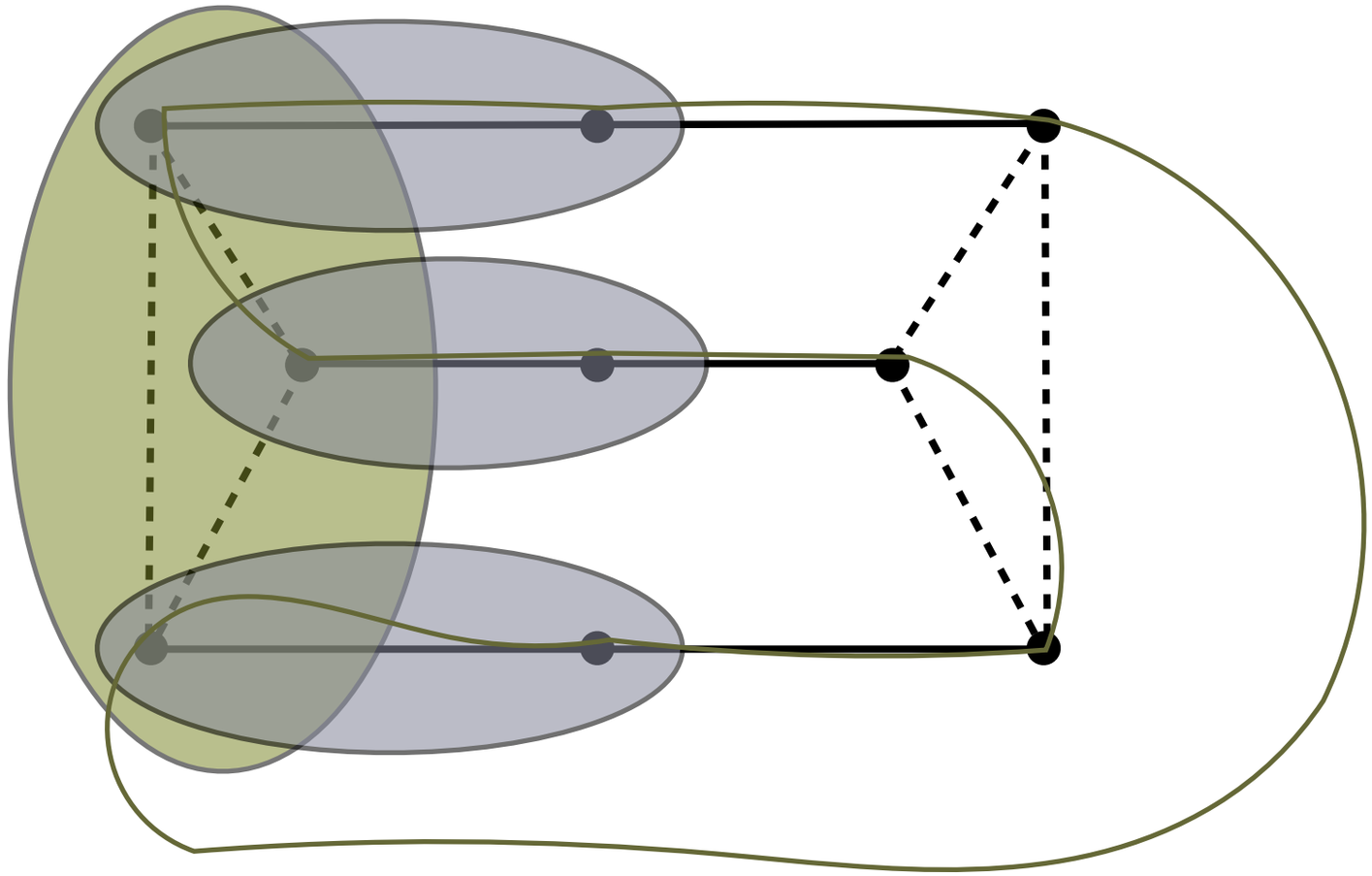
How strong is the Subtour LP bound?

Johnson, McGeoch, and Rothberg (1996) and Johnson and McGeoch (2002) report experimentally that the Subtour LP is very close to the optimal.

Random Uniform Euclidean				TSPLIB			
Name	%Gap	Opttime	HKtime	Name	%Gap	Opttime	HKtime
E1k.0	0.77	1406	2.13	dsj1000	0.61	410	3.68
E1k.1	0.64	3855	2.15	pr1002	0.89	34	2.40
E1k.2	0.72	1211	2.02	si1032	0.08	25	11.32
E1k.3	0.62	956	1.92	u1060	0.65	571	3.62
E1k.4	0.69	330	1.69	vm1084	1.33	605	2.40
E1k.5	0.59	233	2.42	pcb1173	0.96	468	1.70
E1k.6	0.79	2940	1.67	d1291	1.18	27394	4.54
E1k.7	0.94	8003	1.95	rl1304	1.55	189	4.08
E1k.8	1.01	4347	1.65	rl1323	1.65	3742	4.49
E1k.9	0.61	189	2.14	nrw1379	0.43	578	2.40
E3k.0	0.71	533368	9.57	fl1400	1.74	1549	9.83
E3k.1	0.67	425631	10.54	u1432	0.29	224	2.42
E3k.2	0.74	342370	9.41	fl1577	1.66	6705	38.19
E3k.3	0.67	147135	10.30	d1655	0.94	263	6.51
E3k.4	0.73		8.07	vm1748	1.35	2224	4.43
Random Clustered Euclidean				u1817	0.90	449231	5.01
C1k.0	0.54	337	9.83	rl1889	1.55	10023	11.45
C1k.1	0.41	534	10.84	d2103	1.44	–	8.19
C1k.2	0.42	320	8.79	u2152	0.62	45205	8.10
C1k.3	0.53	214	7.63	u2319	0.02	7068	3.16
C1k.4	0.58	768	9.36	pr2392	1.22	117	5.75
C1k.5	0.58	139	9.29	pcb3038	0.81	80829	7.26
C1k.6	0.73	1247	7.07	fl3795	1.04	69886	123.66
C1k.7	0.58	449	13.24	fnl4461	0.55	–	12.47
C1k.8	0.34	140	10.40	rl5915	1.56	–	42.00
C1k.9	0.66	703	9.61	rl5934	1.38		56.15
C3k.0	0.62	16009	53.03	pla7397	0.58	–	55.42
C3k.1	0.61	17754	126.49	rl11849	1.02		102.41
C3k.2	0.70	18237	80.39	usa13509	0.66	–	120.20
C3k.3	0.57	6349	71.57	d15112	0.52		90.13
C3k.4	0.57	4845	44.02				
Random Matrices							
M1k.0	0.01	60	5.47	M3k.0	0.00	612	40.35
M1k.1	0.03	137	5.51	M3k.1	0.01	546	39.52
M1k.2	0.01	151	5.63	M10k.0	0.00	1377	367.84
M1k.3	0.01	169	5.26				

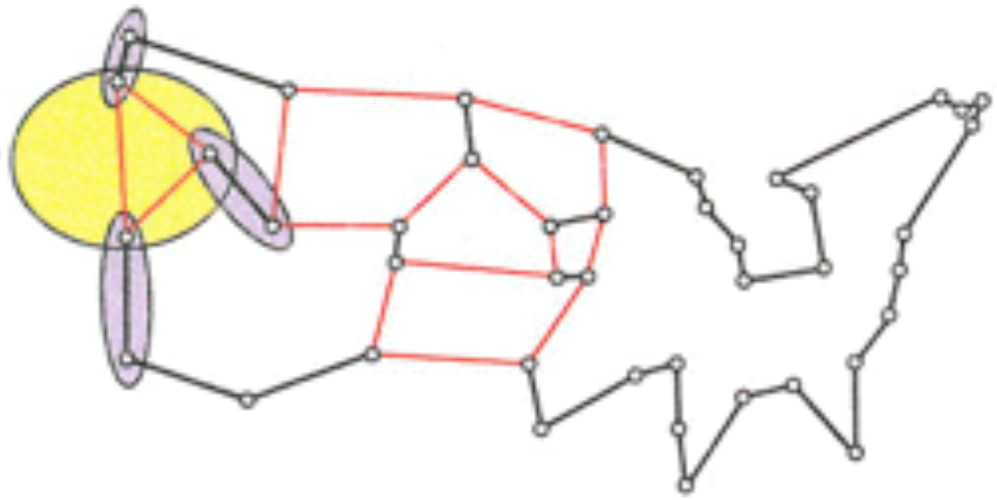
More cutting planes

- It's possible to find cutting planes to help with other types of fractional solutions.

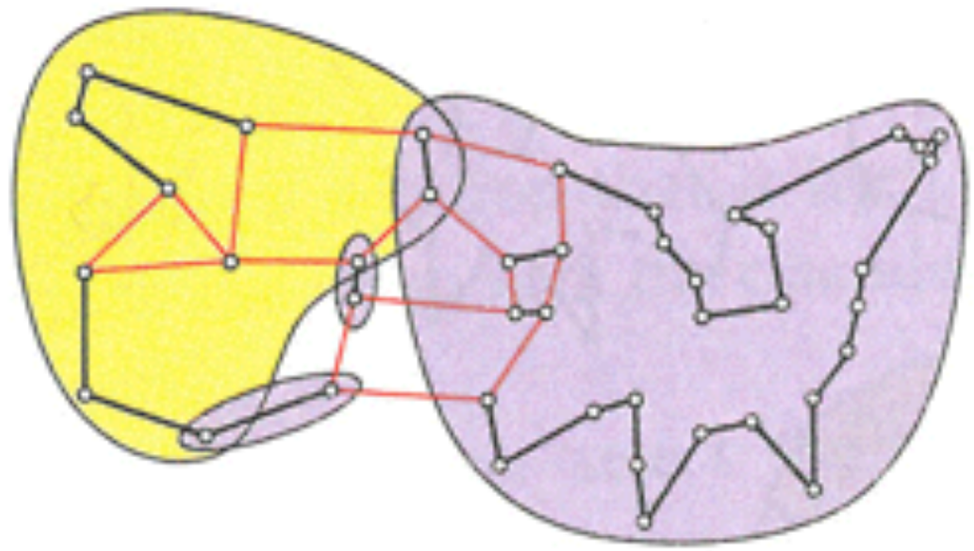


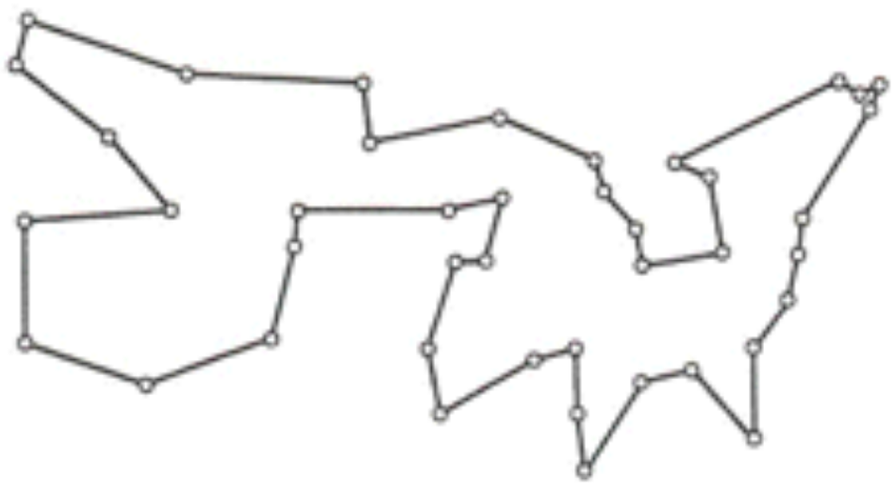
Tour must cross four sets ten times

8



9

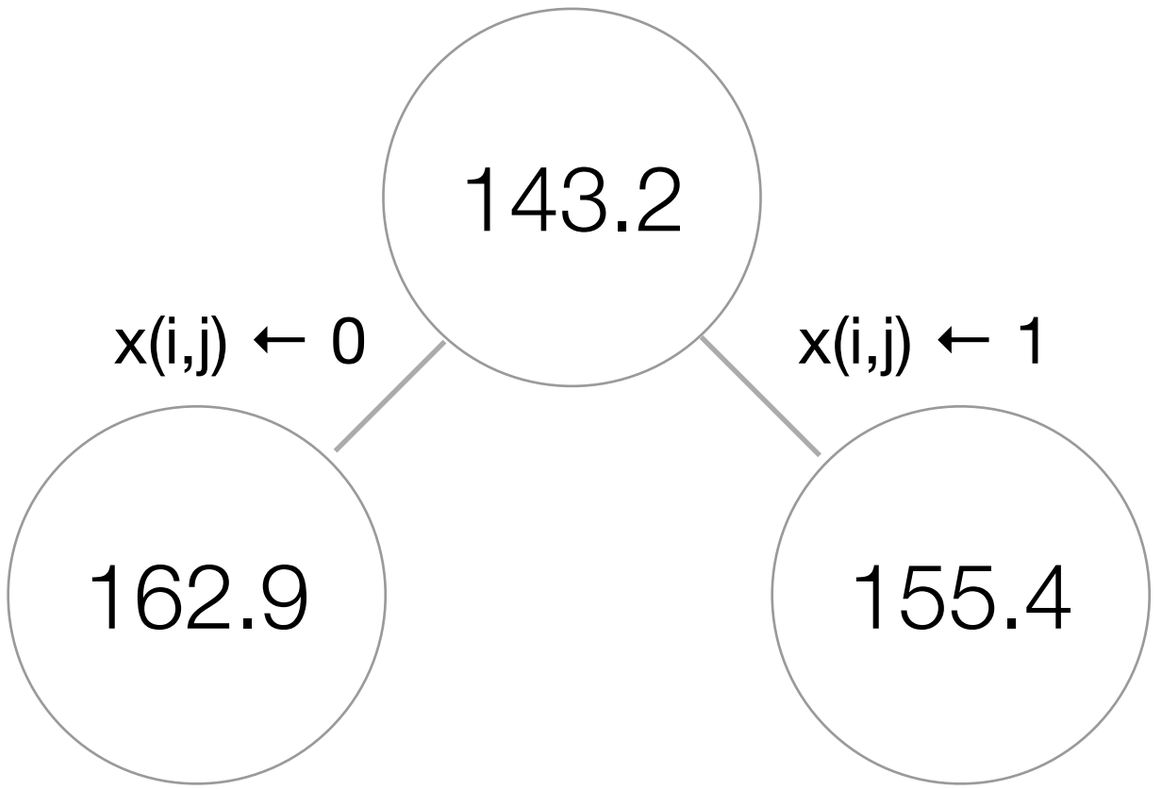




Cook, p. 130

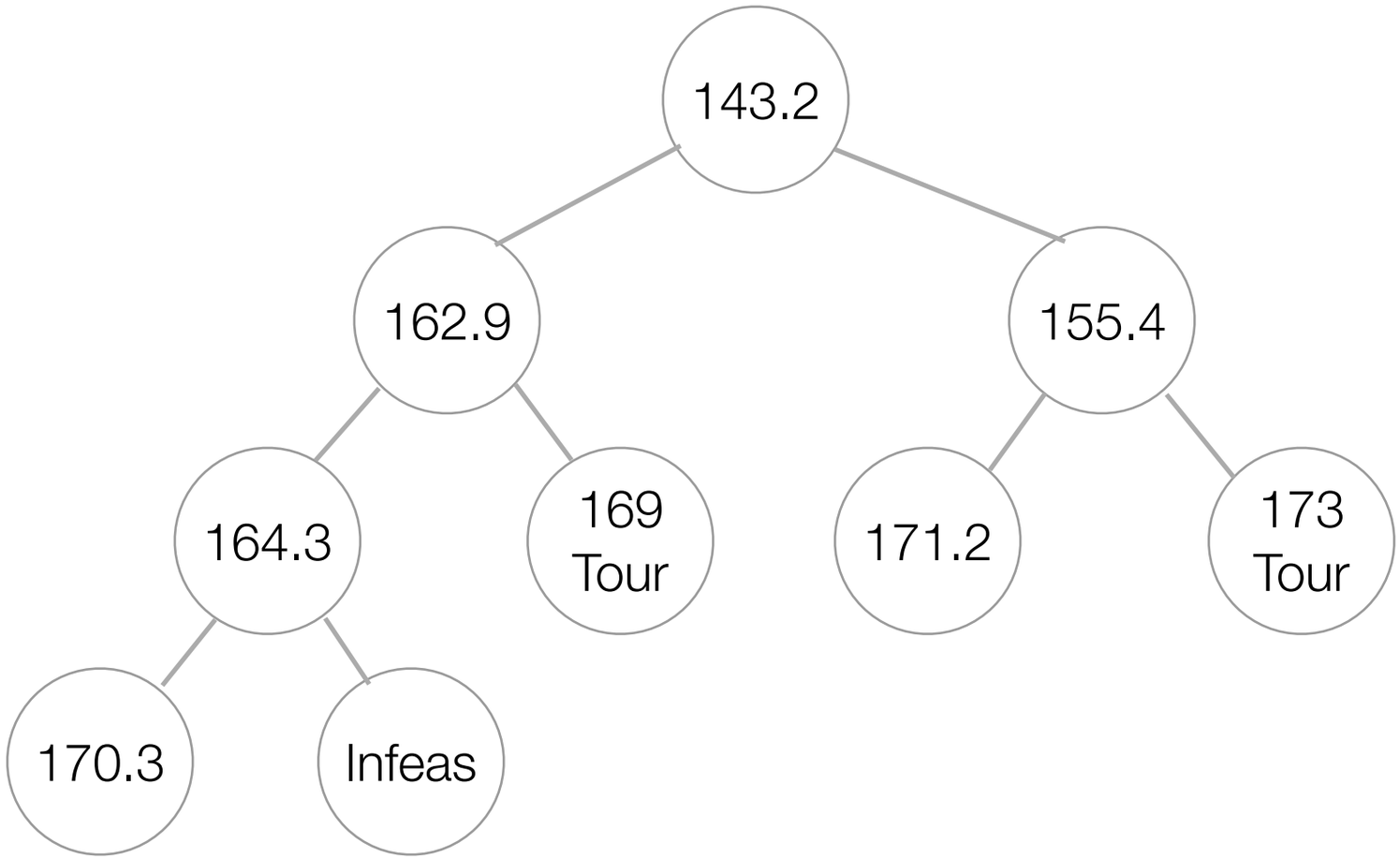
Branch and bound

- Yet another way to make progress is to take a fractional variable x(i,j), and solve two subproblems, one with x(i,j) set to 1, the other with x(i,j) set to 0, take the smaller solution found.



Branch and bound

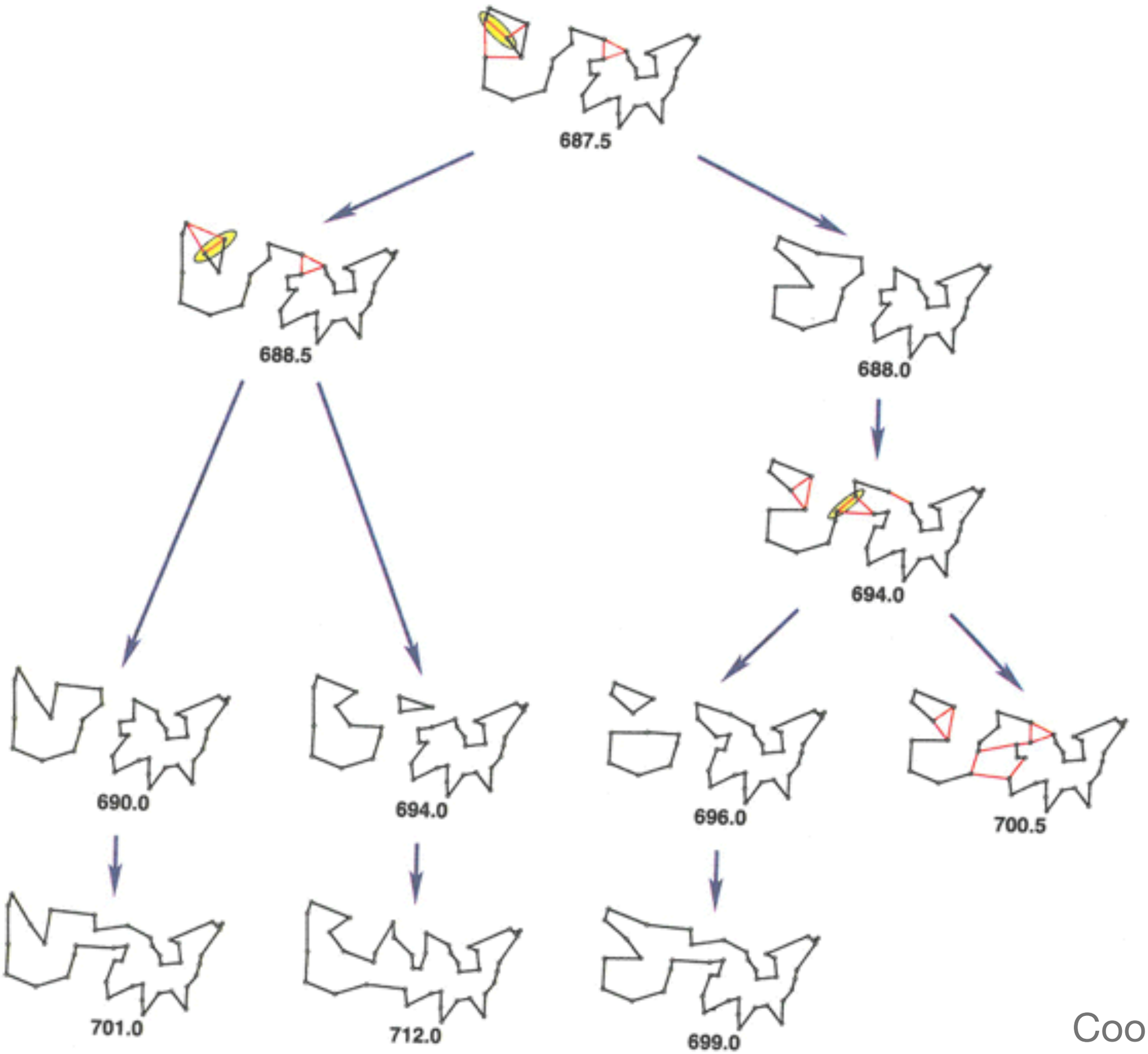
- Solving the subproblems can be done recursively, leading to a branch and bound “tree”.



A better idea: branch and cut

- Combine both ideas: branch and bound, plus cutting planes at each node of the computation.

- Cutting planes generated at a node can be added to a pool for checking by future nodes.



Cook, p. 149

Heuristics

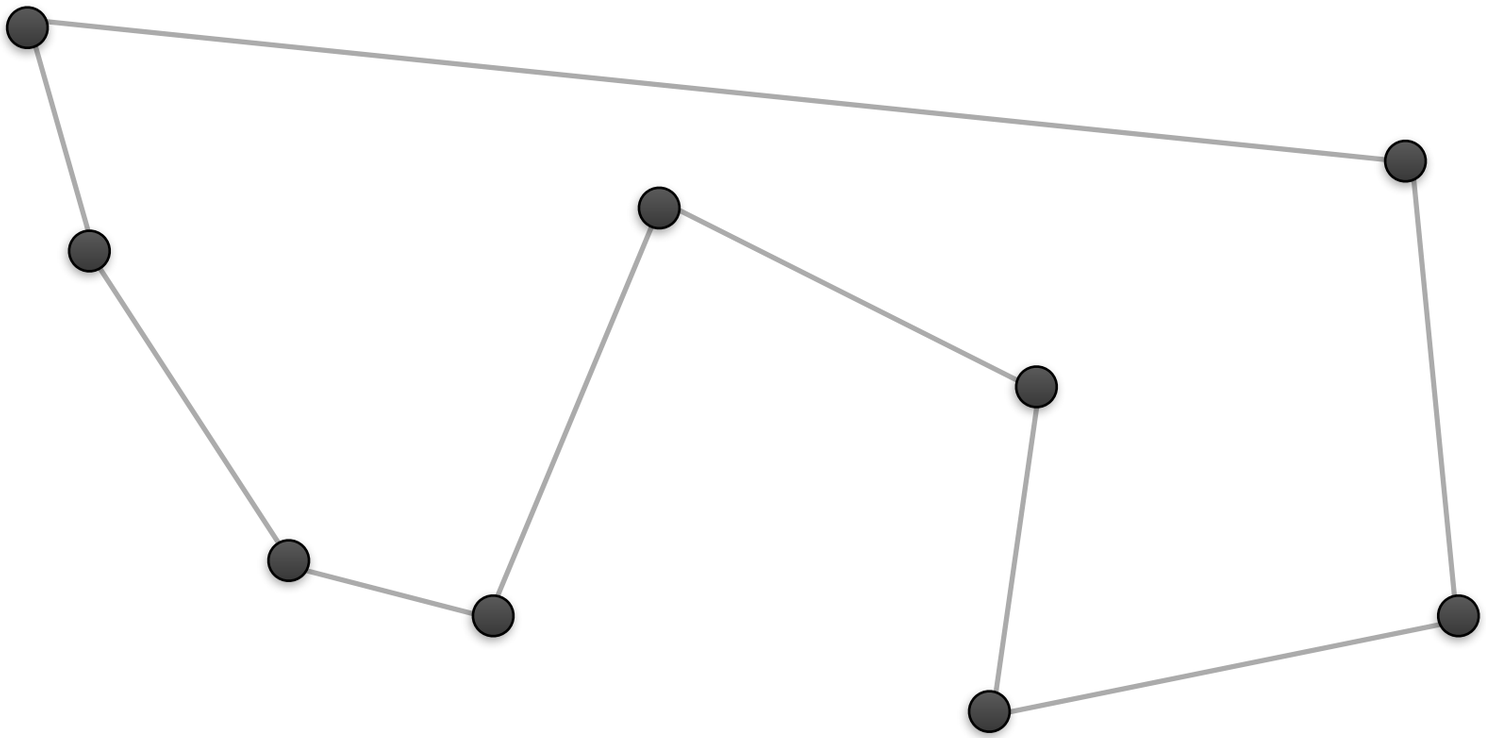
- Note that the cheaper the tour we find in the course of the branch and cut tree, the more of the tree we can avoid searching.

- Don’t need to wait until the tree finds an integer solution: can find some suboptimal one, and use that.

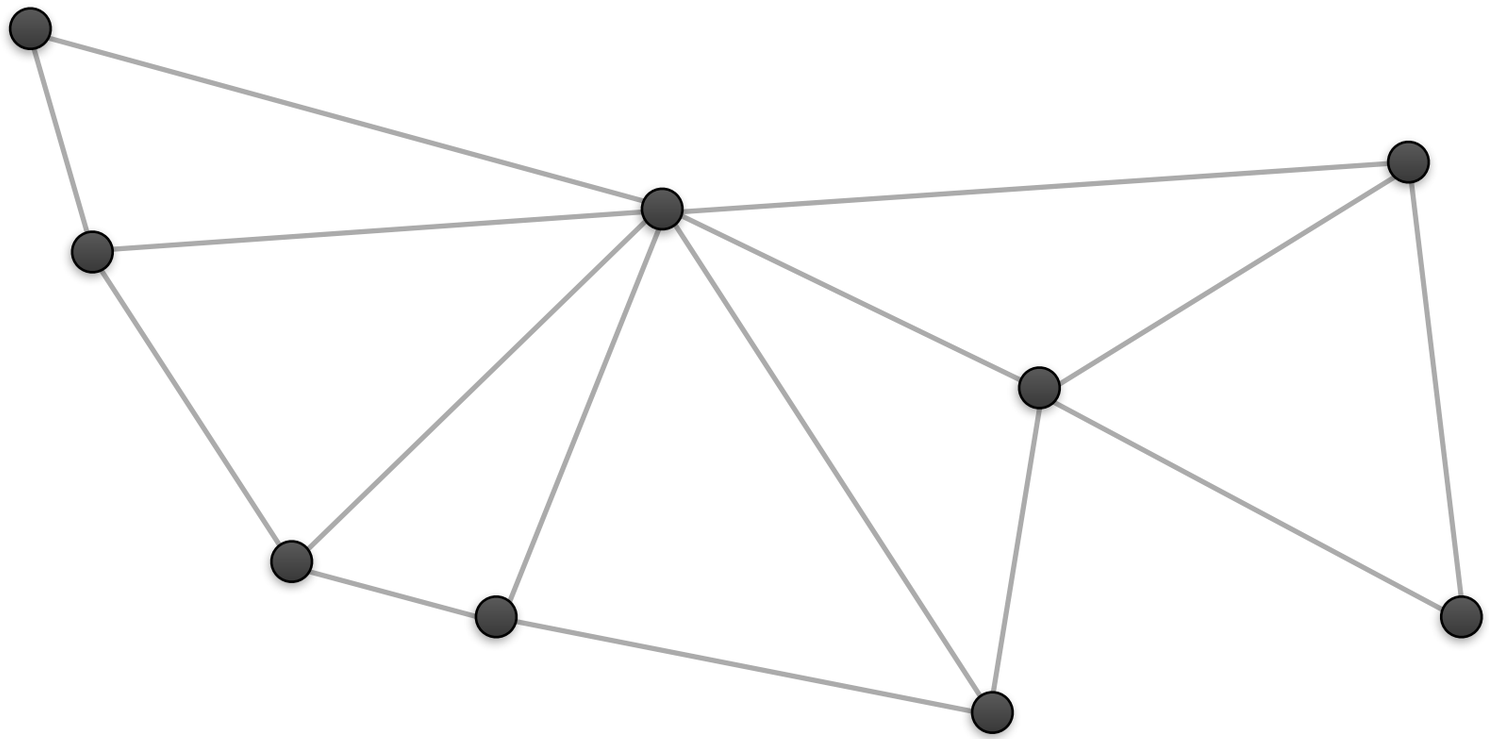
Heuristics: Two types

- Tour construction: Find an initial tour
  - E.g. nearest neighbor, nearest addition
- Tour improvement: Given a starting tour, find a better one
  - E.g. 2-OPT, 3-OPT, k-OPT…

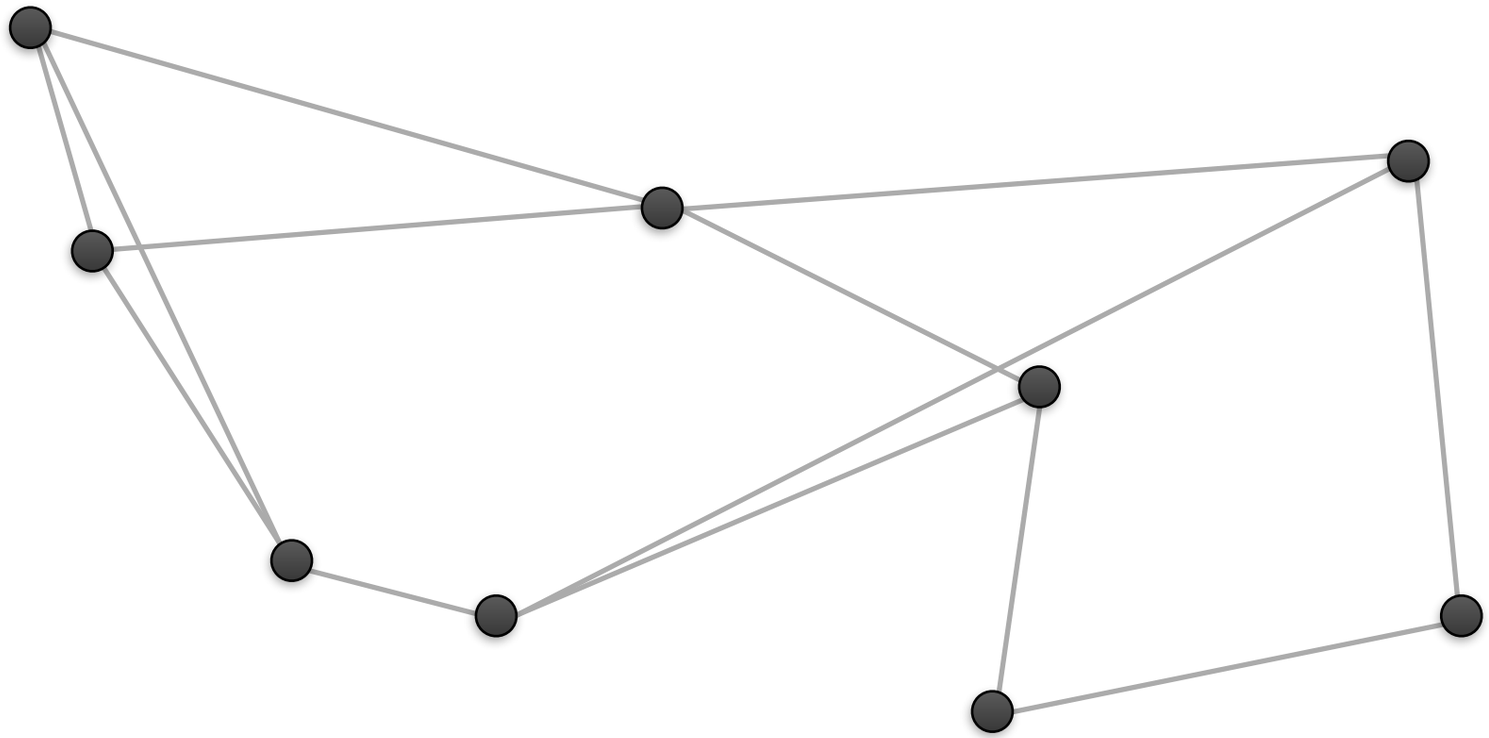
Nearest neighbor



Nearest addition



2-OPT



The Kings

- Lin-Kernighan (LK) (1973) and an extension due to Helsgaun (LKH) (1998): complex way of extending *k* edge changes to *k*+1 edge changes if it helps.

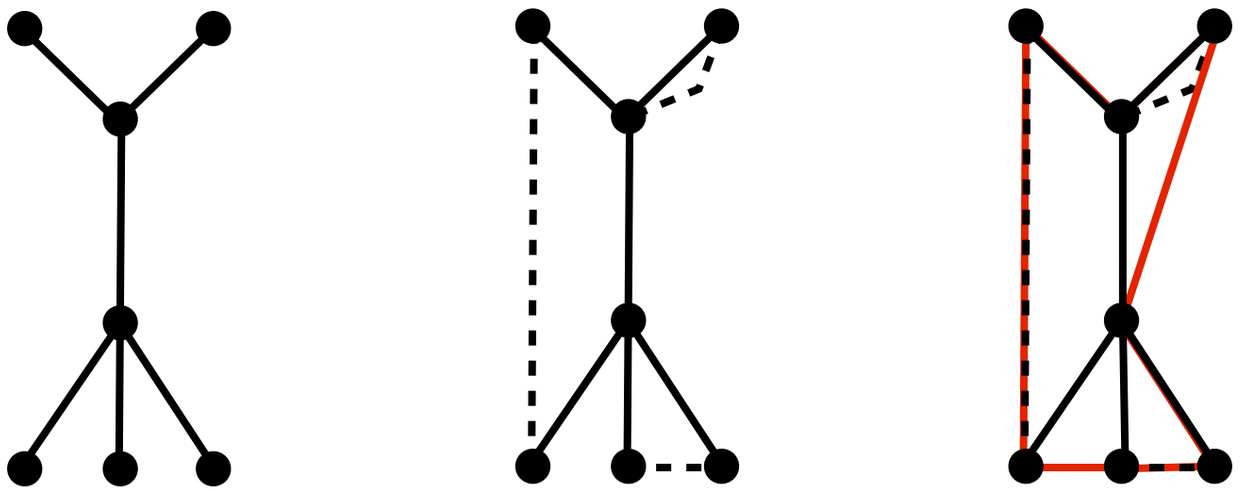
- (Cook 2012) points out that LK is usually enough to find the optimal 42-city tour from a random tour.

Approximation algorithms

- An α-approximation algorithm is a polynomial-time algorithm (a “good” algorithm) that always returns a solution of cost within a factor of α of the optimal solution.

Christofides' algorithm

- Christofides (1976) shows how to compute a tour in polynomial time of cost 3/2 optimal: compute a min-cost spanning tree, compute a matching on the odd-degree vertices, then "shortcut" a traversal of the resulting Eulerian graph.



An open question

- Can we do better than this?
  - For problems from Euclidean plane, approximation scheme possible.
  - For problems of the sort we've seen, not known.

- Wolsey (1980) and Shmoys & W (1990) show that Christofides' algorithm produces a solution within a factor of 3/2 of the Subtour LP bound.

- Examples known where optimal tour is 4/3 times the Subtour LP bound.

$$\frac{4}{3} \leq \frac{\text{Optimal tour}}{\text{Subtour LP}} \leq \frac{3}{2}$$

But how did we get here?

And why should we care?

How we got here

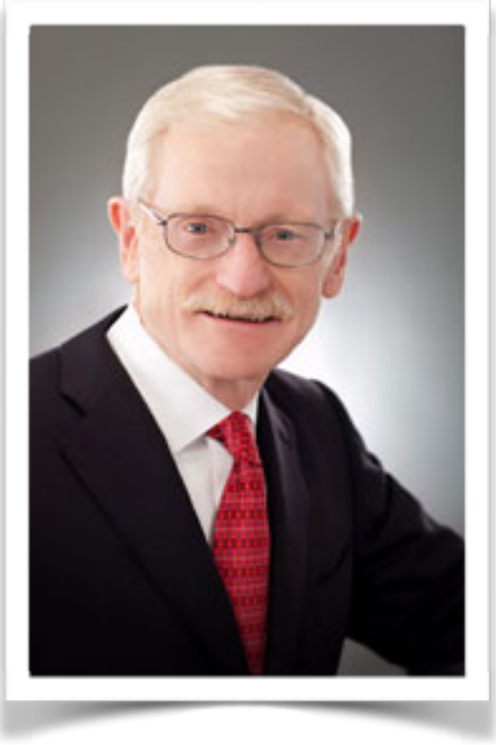
- Moore's law + commodity computing, sure (Sweden computation took 85 CPU years in 2003).

- Huge improvements in LP solver codes.

  - Bixby (2002, 2004) shows experimentally that codes got roughly 3300 times faster between 1988 and 2004 (*machine independent*), while machines got 1600 times faster.

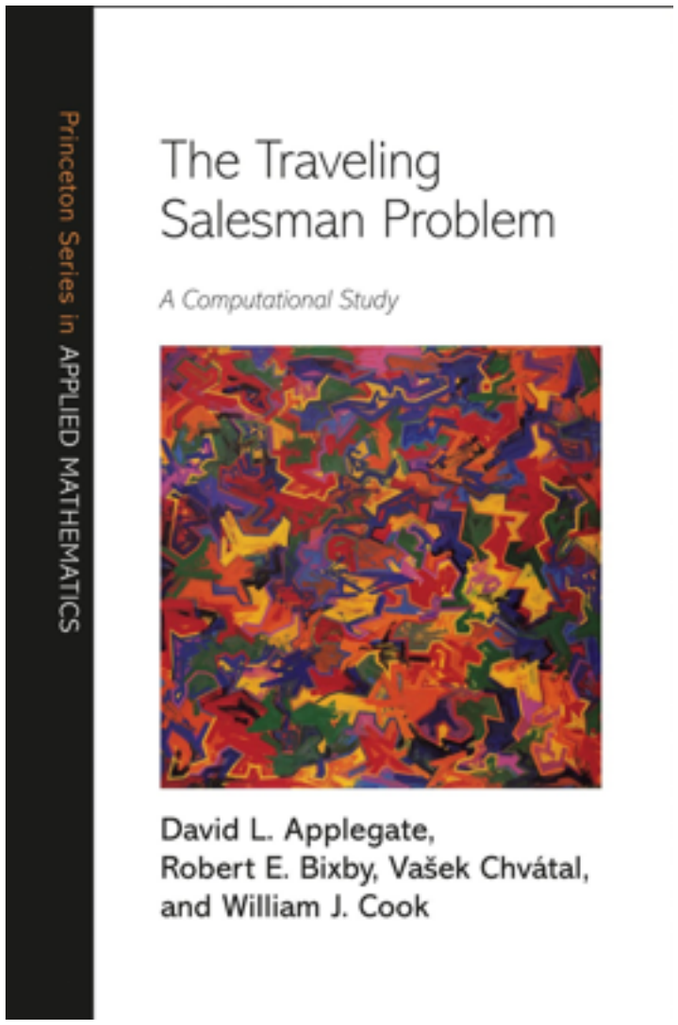
  - So total speedup 3300 x 1600 = 5,300,000.

  - E.g. a problem that took two months to solve in 1988 took 1 second in 2004.



How we got here

- Much better cutting plane generation, node selection, branching rules, heuristics, etc.



(Applegate, Bixby, Chvátal, Cook 2006)
A technical summary of their 20 years of work in 600 pages.

Codes

- Concorde: A TSP code by Cook available at www.math.uwaterloo.ca/tsp. Source code available, binaries for Windows and Linux.

- Also available as an iOS app.

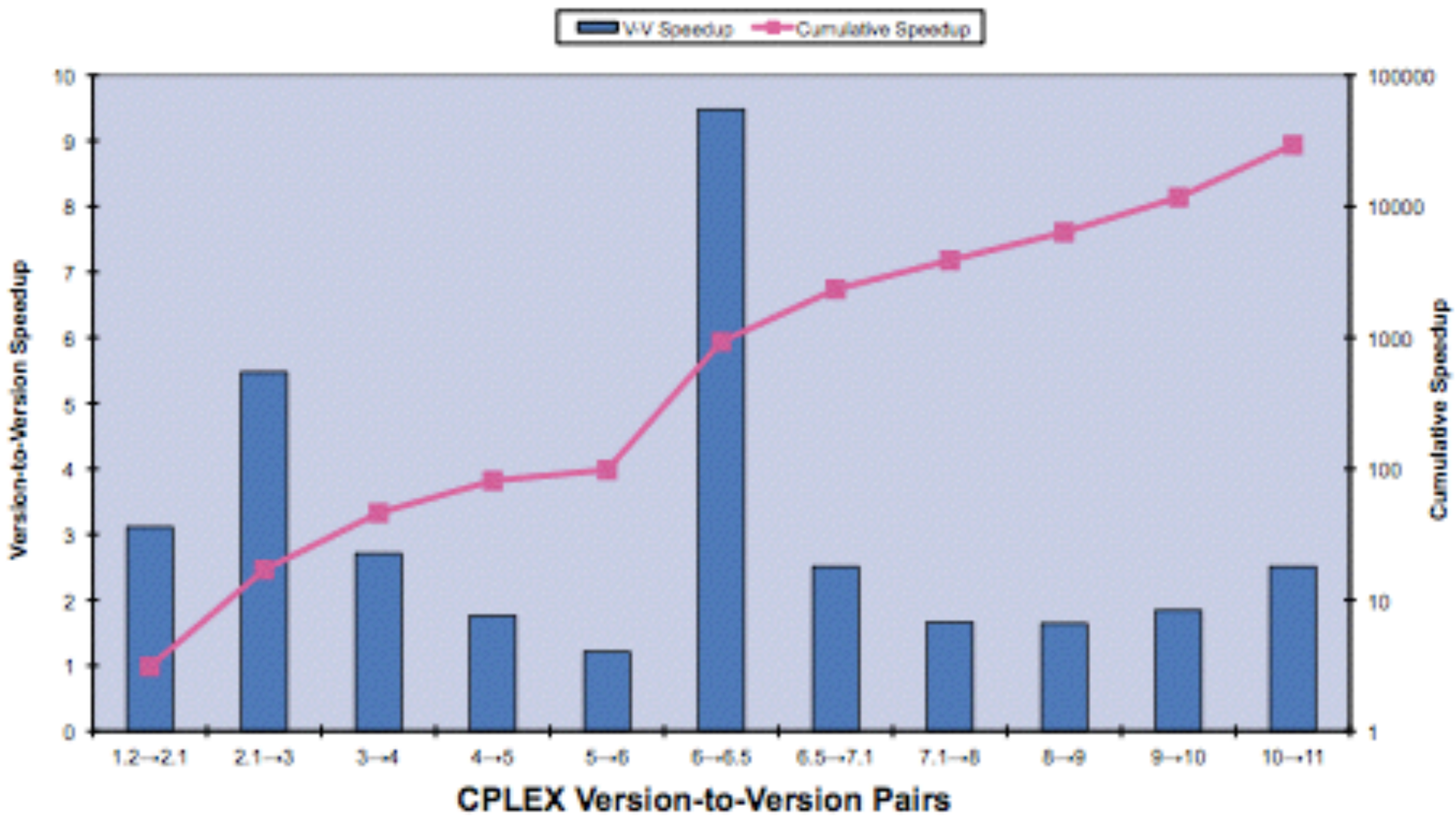
Bob Bixby

- Starts a company which sells CPLEX, a simplex method code written in C, in 1987. Also contains code for solving *mixed-integer programming* (linear programs in which variables can be constrained to be integer).

- The company sold to ILOG in 1997.
- ILOG sold to IBM in 2009 (mostly for business rules management software).
- In 2008, starts alternative LP/MIP code company Gurobi (with two developers from CPLEX, Gu and Rothberg).

Mixed-integer programming (MIP) codes

- Same techniques used in solving the TSP also used in commercial MIP codes … and same sort of speedups seen.



1991 2007 (Bixby 2010)

MIP codes

- (Bixby 2010) reports yet another factor 16 speedup since 2007, so roughly 500,000x speedup (machine independent) since 1991.

- E.g. California Unit Commitment Problem (from power generation)

  - 1989: 8 hours, no progress for 2 day problem; 1 hour to solve initial LP for 7 day problem (“theoretically complicated, computationally cumbersome”).

- 1999: 22 minutes on desktop PC for 7 day problem

- 2008: 43 seconds for 7 day problem

“Christos Papadimitriou told me that the traveling salesman problem is not a problem. It’s an addiction.”

– Jon Bentley 1991

What if P ≠ NP, but it doesn't matter in practice?