Discrete models

Different kinds of discrete representations/models

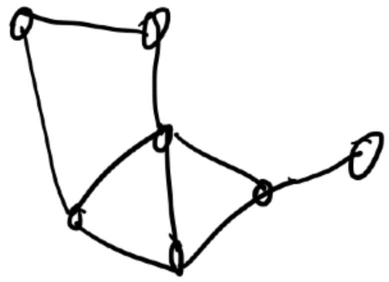
More difficult to structure that the classical mathematical models.

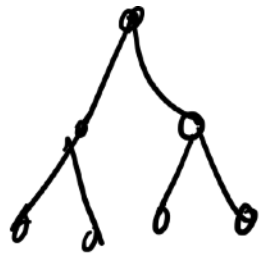
Often concerned with <u>modelling different kinds of information</u>.

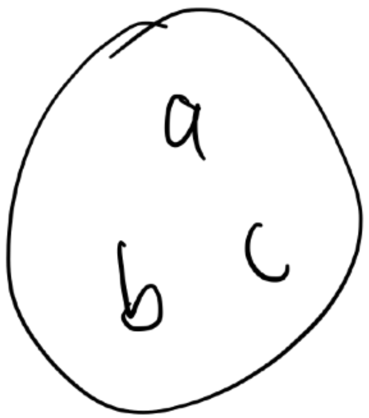
Some model types mostly studied within computer science, also some overlap with mathematics.

Basic discrete representations/model types

17





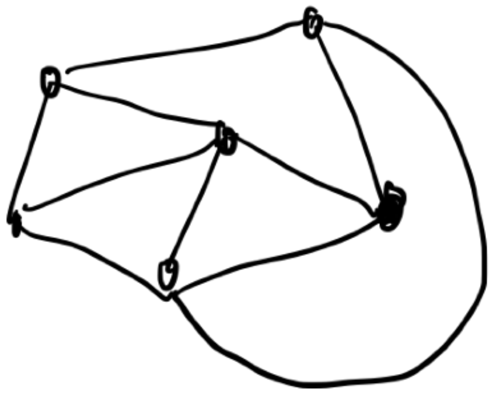


[a, b, c]

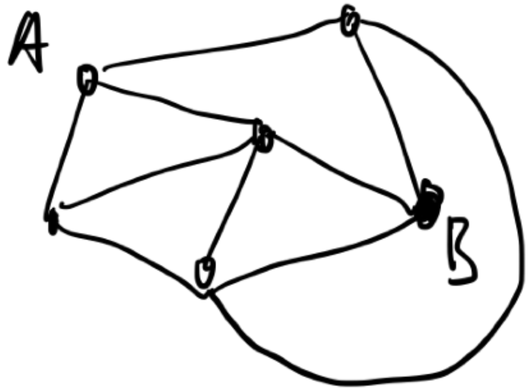
integers, sets, sequences, trees, graphs, networks, …

Model or problem?

is there a path from A to B?



A graph can model things, but it is not a problem



In a problem you have also added a question!

The common language is sometimes a bit confused…

Other discrete representations/model types

General discrete models of representation
(relation, grammar, finite automaton, logic)

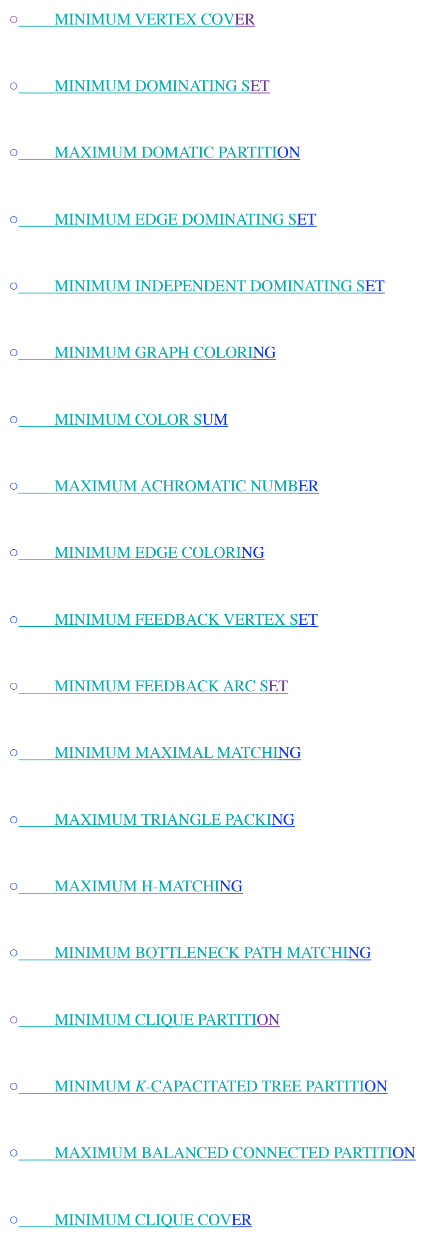
Standard discrete (optimization) problems
(shortest path, minimum spanning tree, knapsack, LP/ILP …)

Models of computation
(combinatorial and sequential networks, Turing machine, lambda calculus... Programming languages)

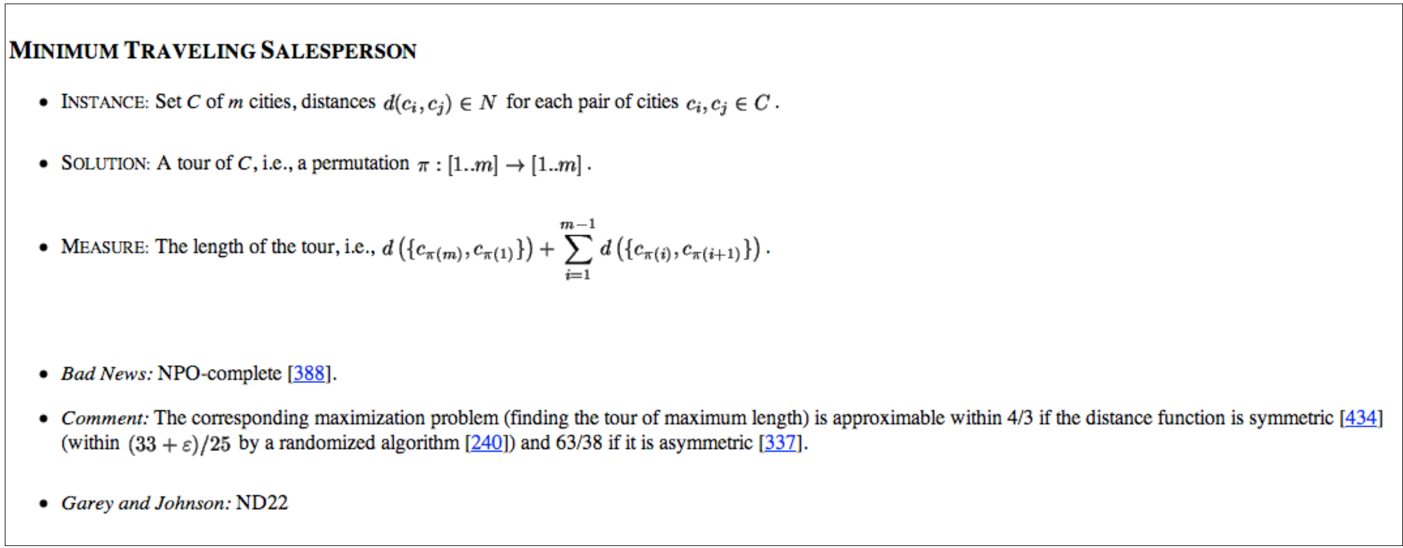
Different specialised models
(e.g. rule based expert system, modelling languages, file formats,…)

There are systems and algorithms for all these model types and problems!

Many standard problems are known and well documented

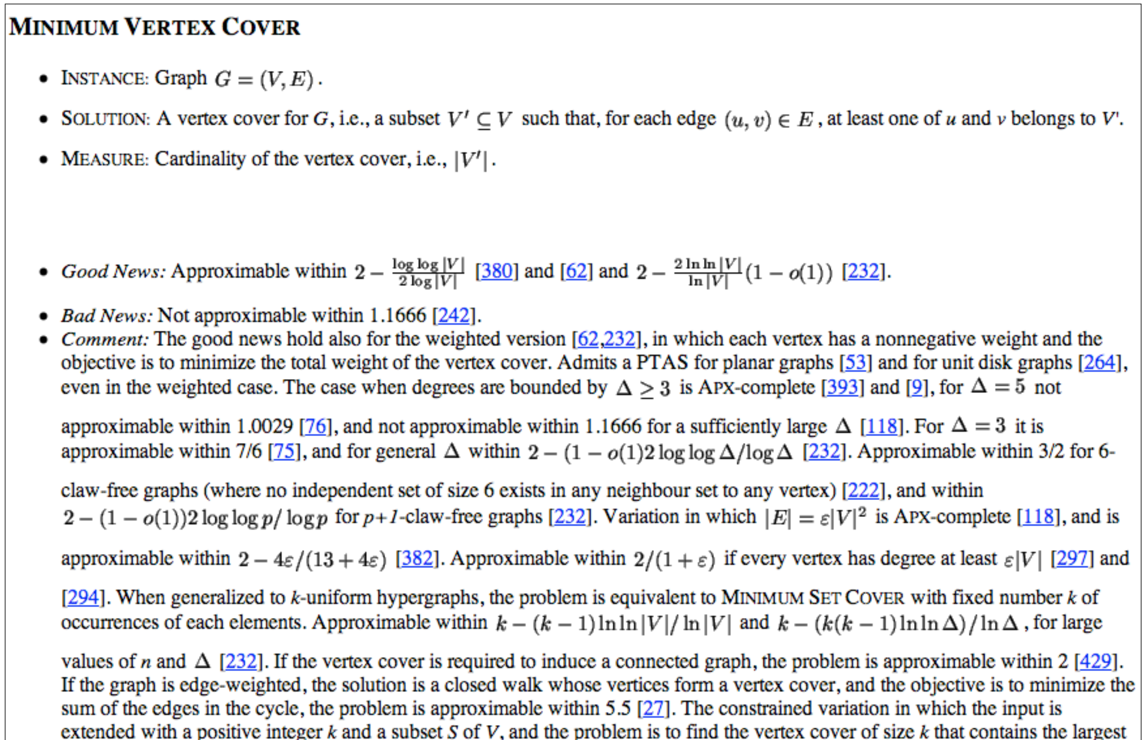


- MINIMUM VERTEX COVER
- MINIMUM DOMINATING SET
- MAXIMUM DOMATIC PARTITION
- MINIMUM EDGE DOMINATING SET
- MINIMUM INDEPENDENT DOMINATING SET
- MINIMUM GRAPH COLORING
- MINIMUM COLOR SUM
- MAXIMUM ACHROMATIC NUMBER
- MINIMUM EDGE COLORING
- MINIMUM FEEDBACK VERTEX SET
- MINIMUM FEEDBACK ARC SET
- MINIMUM MAXIMAL MATCHING
- MAXIMUM TRIANGLE PACKING
- MAXIMUM H-MATCHING
- MINIMUM BOTTLENECK PATH MATCHING
- MINIMUM CLIQUE PARTITION
- MINIMUM *K*-CAPACITATED TREE PARTITION
- MAXIMUM BALANCED CONNECTED PARTITION
- MINIMUM CLIQUE COVER



MINIMUM TRAVELING SALESPERSON

- INSTANCE: Set C of m cities, distances $d(c_i,c_j) \in N$ for each pair of cities $c_i, c_j \in C$.
- SOLUTION: A tour of C, i.e., a permutation $\pi : [1..m] \to [1..m]$.
- MEASURE: The length of the tour, i.e., $d\left(\{c_{\pi(m)}, c_{\pi(1)}\}\right) + \sum_{i=1}^{m-1} d\left(\{c_{\pi(i)}, c_{\pi(i+1)}\}\right)$.
- *Bad News:* NPO-complete [388].
- *Comment:* The corresponding maximization problem (finding the tour of maximum length) is approximable within 4/3 if the distance function is symmetric [434] (within $(33+\varepsilon)/25$ by a randomized algorithm [240]) and 63/38 if it is asymmetric [337].
- *Garey and Johnson:* ND22



MINIMUM VERTEX COVER

- INSTANCE: Graph $G = (V, E)$.
- SOLUTION: A vertex cover for G, i.e., a subset $V' \subseteq V$ such that, for each edge $(u, v) \in E$, at least one of u and v belongs to V'.
- MEASURE: Cardinality of the vertex cover, i.e., $|V'|$.
- *Good News:* Approximable within $2 - \frac{\log\log|V|}{2\log|V|}$ [380] and [62] and $2 - \frac{2\ln\ln|V|}{\ln|V|}(1-o(1))$ [232].
- *Bad News:* Not approximable within 1.1666 [242].
- *Comment:* The good news hold also for the weighted version [62,232], in which each vertex has a nonnegative weight and the objective is to minimize the total weight of the vertex cover. Admits a PTAS for planar graphs [53] and for unit disk graphs [264], even in the weighted case. The case when degrees are bounded by $\Delta \geq 3$ is APX-complete [393] and [9], for $\Delta = 5$ not approximable within 1.0029 [76], and not approximable within 1.1666 for a sufficiently large Δ [118]. For $\Delta = 3$ it is approximable within 7/6 [75], and for general Δ within $2-(1-o(1)2\log\log\Delta/\log\Delta$ [232]. Approximable within 3/2 for 6-claw-free graphs (where no independent set of size 6 exists in any neighbour set to any vertex) [222], and within $2-(1-o(1))2\log\log p/\log p$ for *p+1*-claw-free graphs [232]. Variation in which $|E| = \varepsilon|V|^2$ is APX-complete [118], and is approximable within $2-4\varepsilon/(13+4\varepsilon)$ [382]. Approximable within $2/(1+\varepsilon)$ if every vertex has degree at least $\varepsilon|V|$ [297] and [294]. When generalized to *k*-uniform hypergraphs, the problem is equivalent to MINIMUM SET COVER with fixed number *k* of occurrences of each elements. Approximable within $k-(k-1)\ln\ln|V|/\ln|V|$ and $k-(k(k-1)\ln\ln\Delta)/\ln\Delta$, for large values of *n* and Δ [232]. If the vertex cover is required to induce a connected graph, the problem is approximable within 2 [429]. If the graph is edge-weighted, the solution is a closed walk whose vertices form a vertex cover, and the objective is to minimize the sum of the edges in the cycle, the problem is approximable within 5.5 [27]. The constrained variation in which the input is extended with a positive integer *k* and a subset *S* of *V*, and the problem is to find the vertex cover of size *k* that contains the largest

Two ways to solve problems

$$\text{problem} \longrightarrow \text{standard problem} \longrightarrow \text{answer}$$

$$\text{problem} \xrightarrow{\qquad\qquad} \text{answer}$$

Two ways to solve problems

$$\begin{tikzcd}\text{problem} \arrow[r] & \text{standard problem} \arrow[r] & \text{answer}\end{tikzcd}$$

you have to model/translate!

standard algorithm often available!

main approach in this course!

$$\begin{tikzcd}\text{problem} \arrow[r] & \text{answer}\end{tikzcd}$$

create your own algorithm

(take algorithms course!)

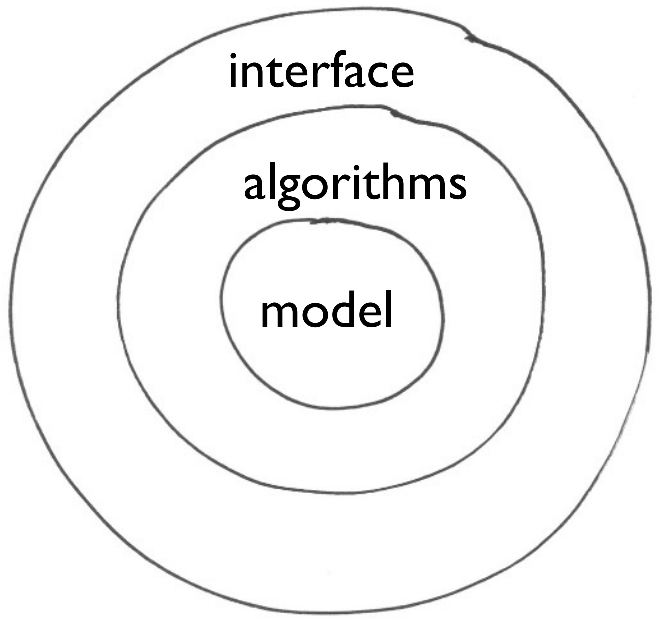
” Model as the problem…”

1. First be clear about what problem you have.

2. Then think about how you can solve it by using a subroutine for another problem.

This is not about solving the problem yourself. Just translate it!

Models, algorithms and software



word processing, databases, graphics programs, communication, security, control, AI, decision support, games, image and sound formats, …

A good software system has a good model (or problem) at its core, and powerful algorithms for that model.

The model itself determines to a considerable extent what you can do with the system.

If you want to understand or design a software system this is how you should think!

Often when a computer program is difficult to understand or use, it is because the underlying model is messy and not mathematically clean and precise.

(To avoid misunderstanding: depending on the application, a software system can be based on any mathematical model, eg. splines, optimization model, probability distribution etc., and not only discrete models.)

Vienna Symphonic Library

Bösendorfer mic'd for sampling

Example of changing underlying models: electronic pianos

1960's: simple waveform and decay synthesis

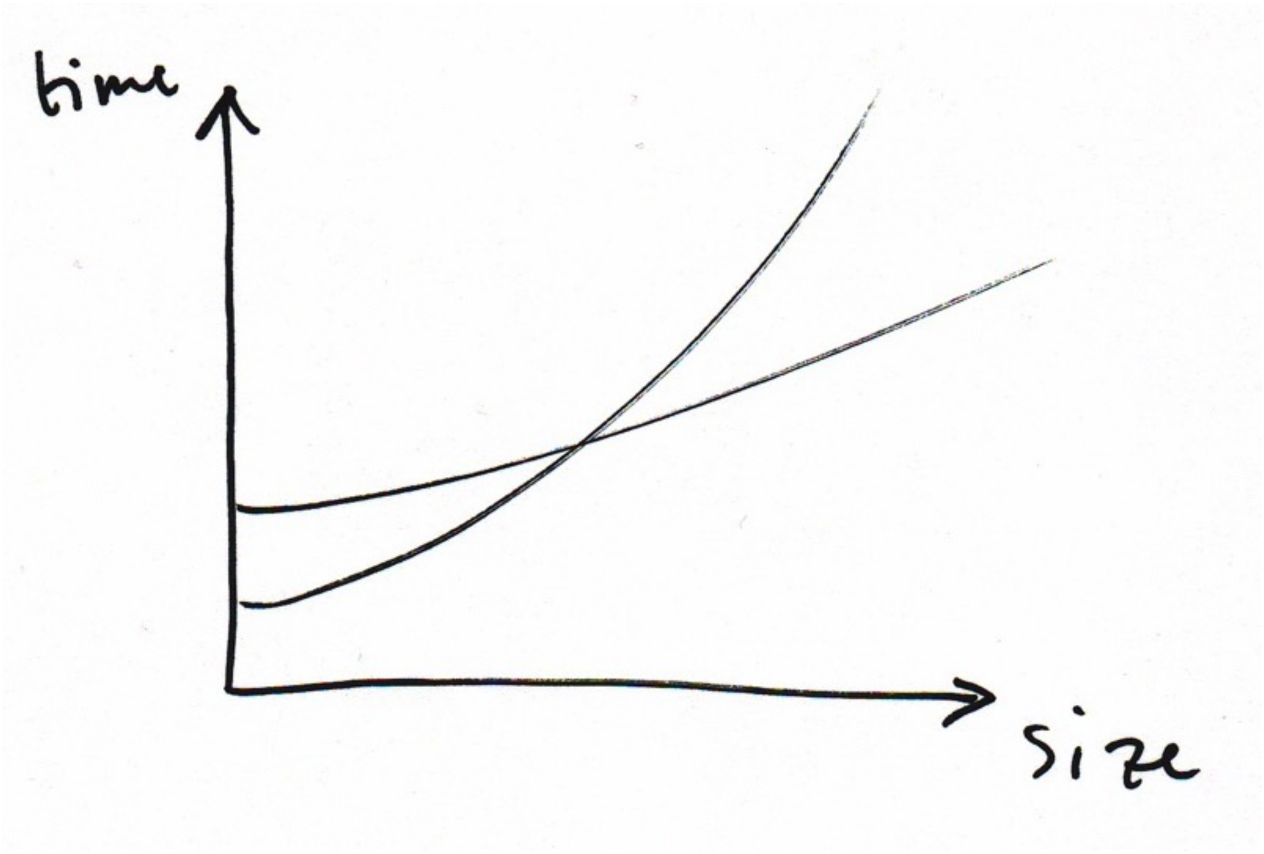
1980's: sampling synthesis

2000- : physical modelling

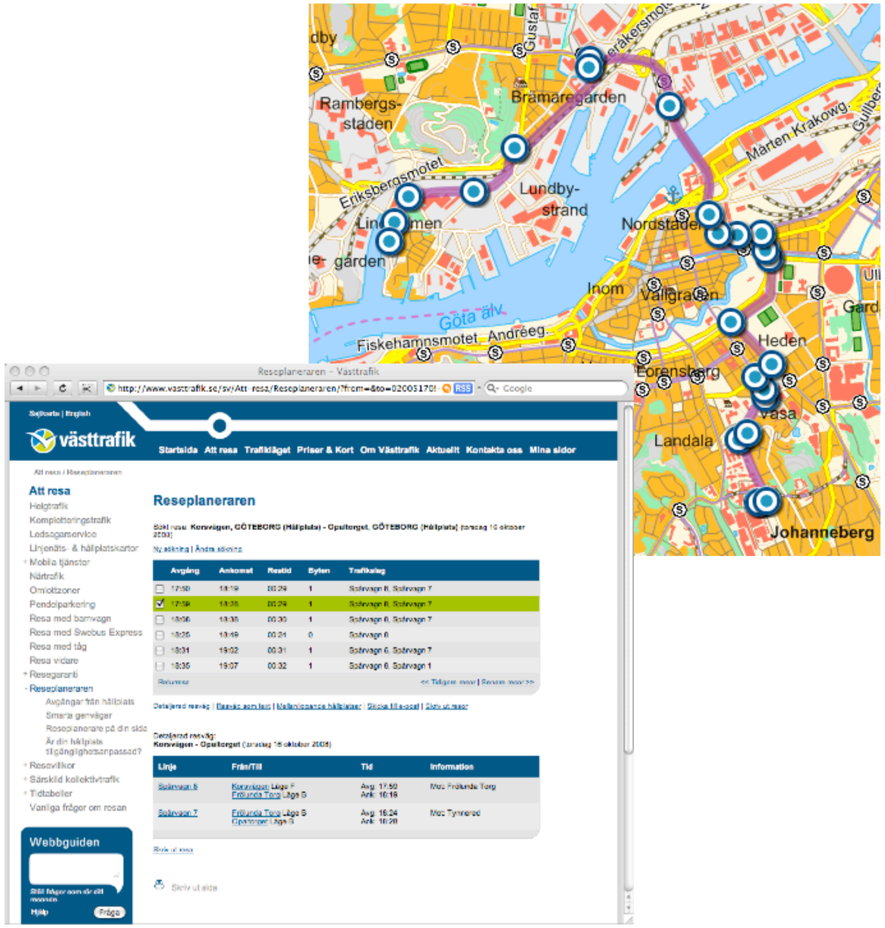
Bösendorfer mic'd for sampling

something about algorithms and algorithm design

Complexity of algorithms



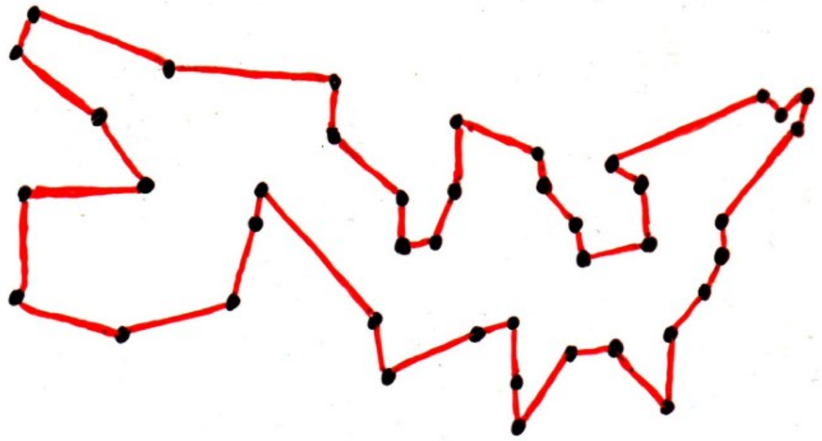
The Shortest Path Problem



polynomial growth

n	$c\ n^2$
10	0,001 s
20	0,004 s
30	0,009 s
40	0,016 s
50	0,025 s
60	0,036 s

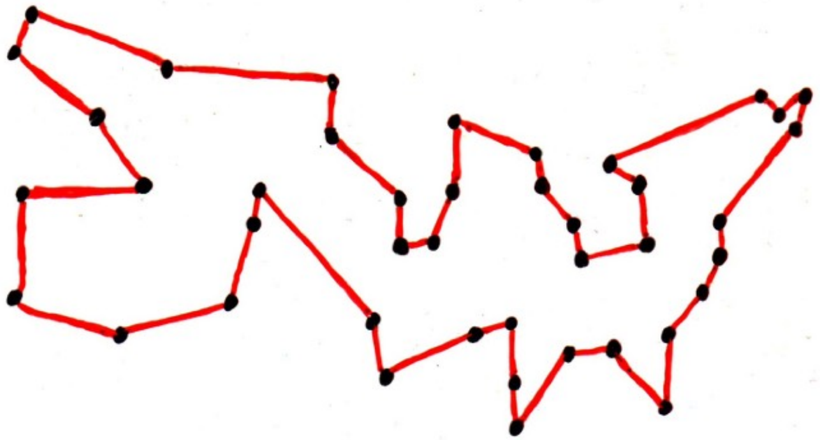
The Travelling Salesperson Problem



exponential growth

n	c 2^n
10	0,001 s
20	1 s
30	18 min
40	13 days
50	36 years
60	36600 years

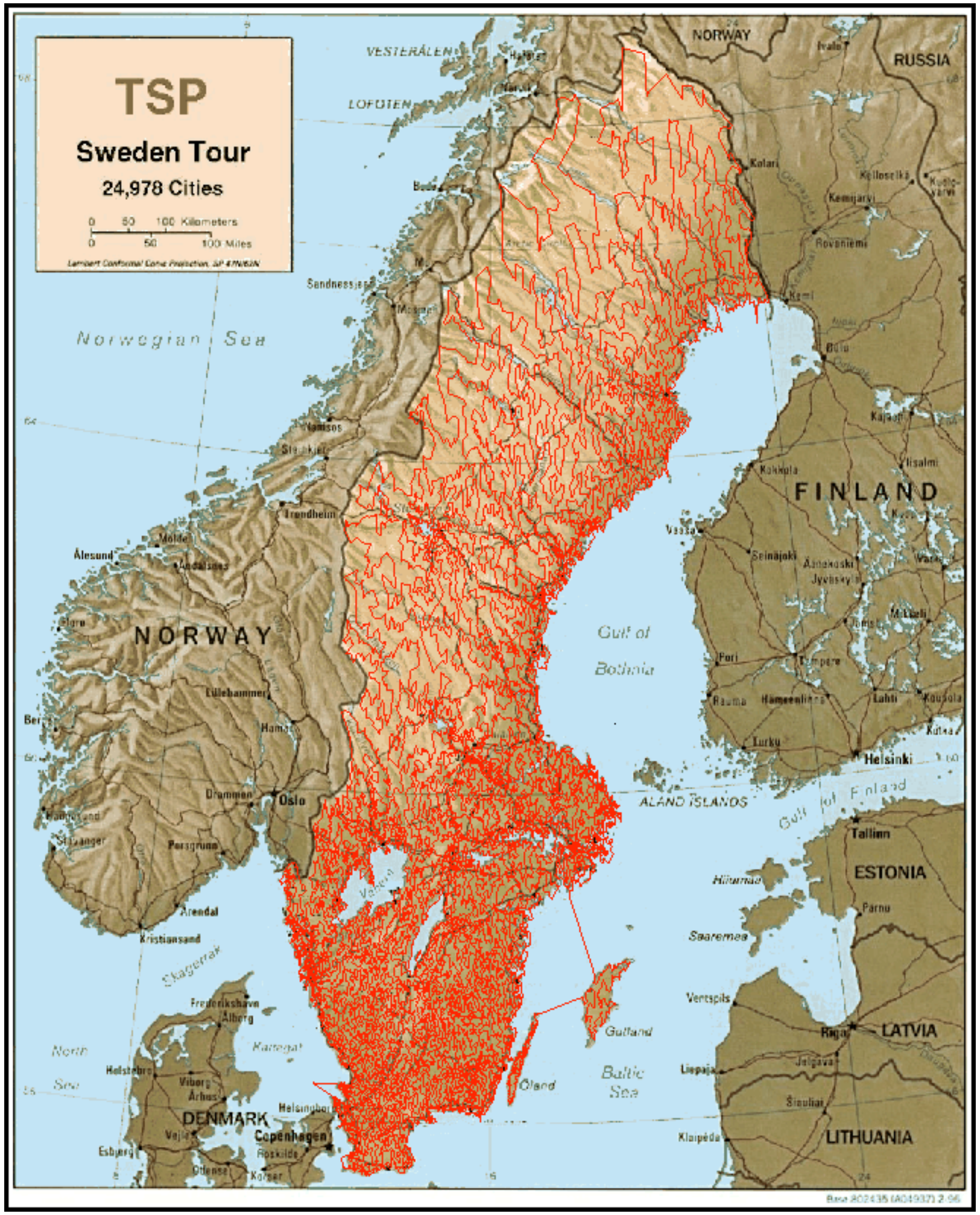
The Travelling Salesperson Problem



no known polynomial algorithm!

exponential growth

n	$c\ 2^n$
10	0,001 s
20	1 s
30	18 min
40	13 days
50	36 years
60	36600 years



Many of the known problems are difficult problems for which no polynomial algorithm is known (so called NP-complete or NP-hard problems).

But even for relatively large instances of such problems the situation is far from hopeless, when sophisticated algorithms are used. This is one of the largest TSP instances currently solved. The solution is proven to be optimal.

See http://www.tsp.gatech.edu/sweden/index.html

Standard discrete problems

Polynomial: searching and sorting, shortest path, minimum spanning tree, …

Exponential (really NP-hard*): TSP, knapsack, …

With exponential problems you may have to compromise with quality to solve large problem instances.

* this means that only exponential algorithms are known for solving the problem to optimality. Probably no polynomial algorithms exist.

General algorithm design

Be sure to *define* the problem clearly before thinking about algorithms.
Any special properties of your problem could be useful when solving it.

Then:

- See if you can translate it (or a part of it) into a problem that is well known

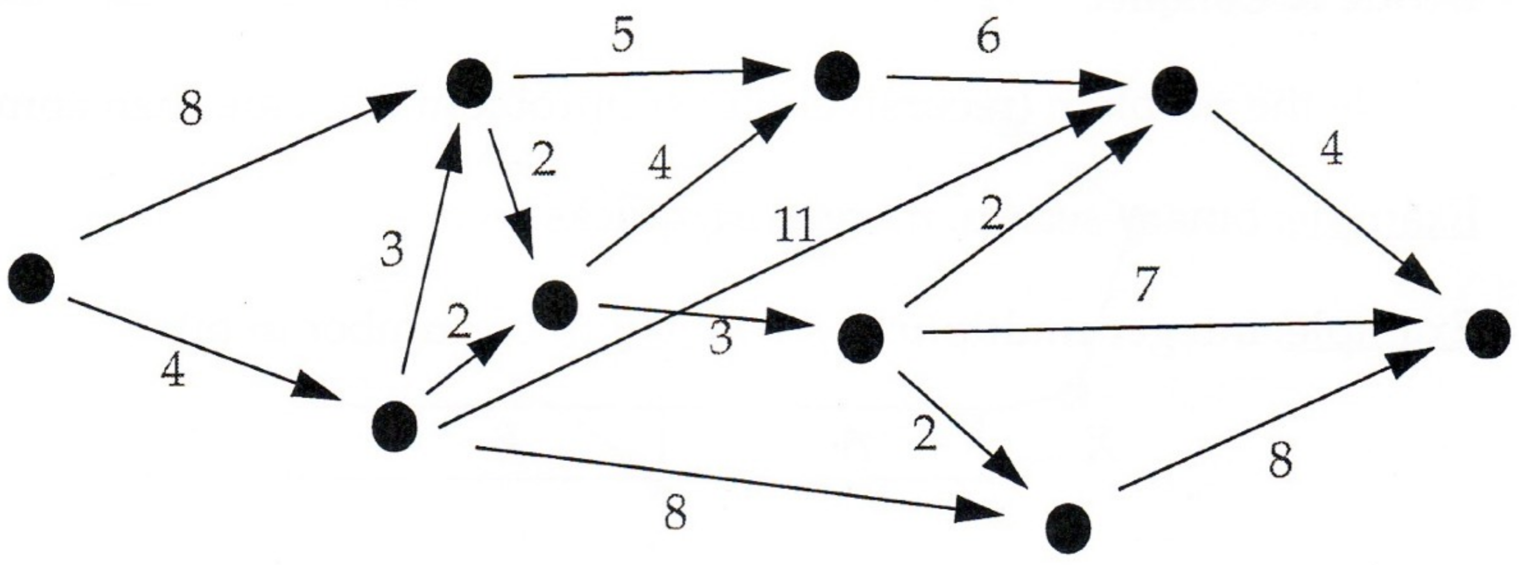
- For the non-standard parts, use standard algorithm design techniques

- Invent a completely new kind of algorithm...

Main discrete algorithm design techniques

- **Divide and conquer**. Split the problem, solve the subproblems, merge.
- **Dynamic programming**. Solve a sequence of subproblems until you get to your problem.
- **Combinatorial search**. Test all possibilites or a subset of them. Use branch and bound to speed up.
- **Greedy algorithm/heuristic**. Do what seems best in every step. Is often heuristic.
- **Local search**. Take a solution and repeatedly improve it by making small changes. Is often heuristic.
- …

Solving the directed shortest path problem with dynamic programming



Traverse nodes from left to right and mark with distance from origin.

Circumvents the combinatorial explosion!

Combinatorial search: branch & bound

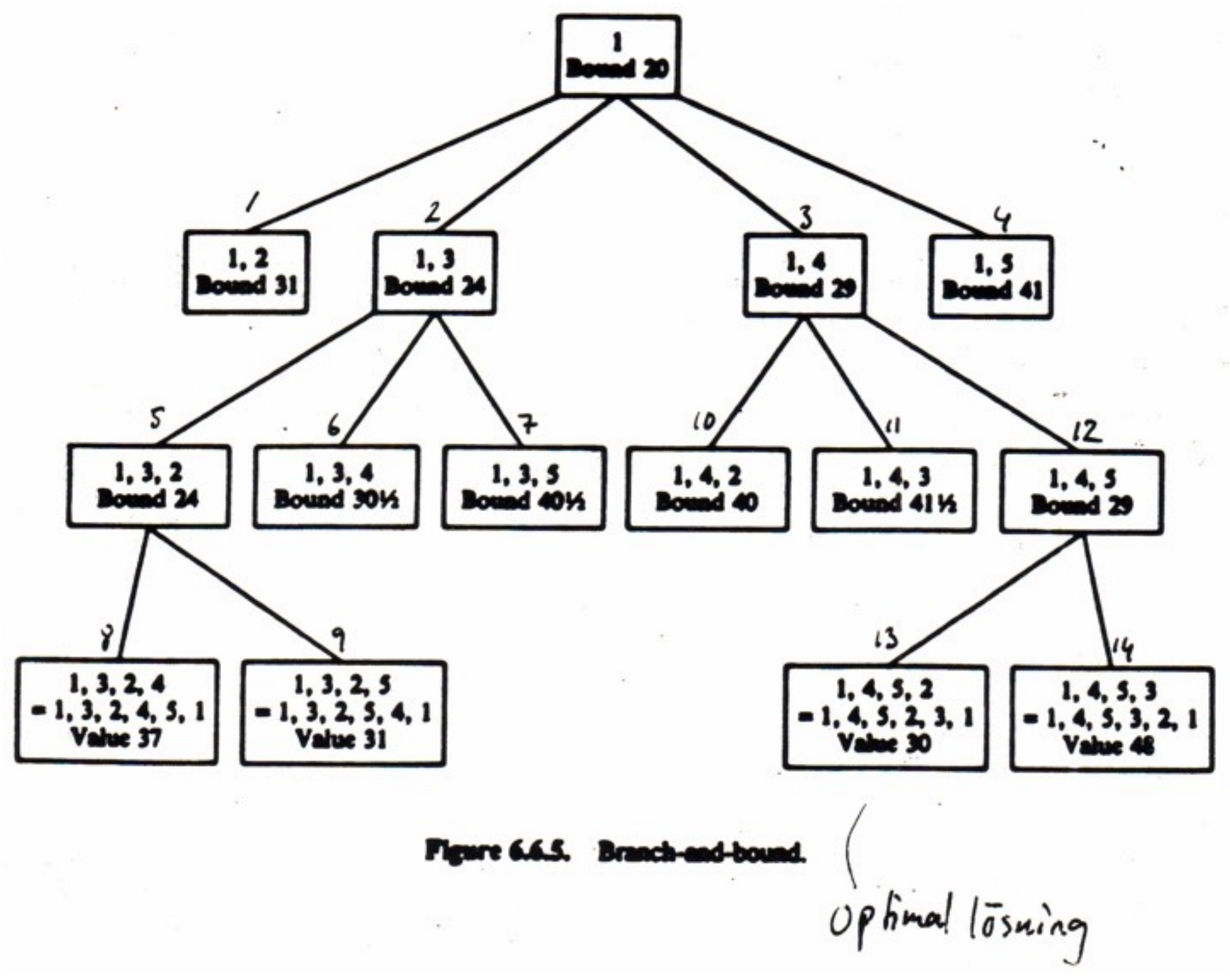
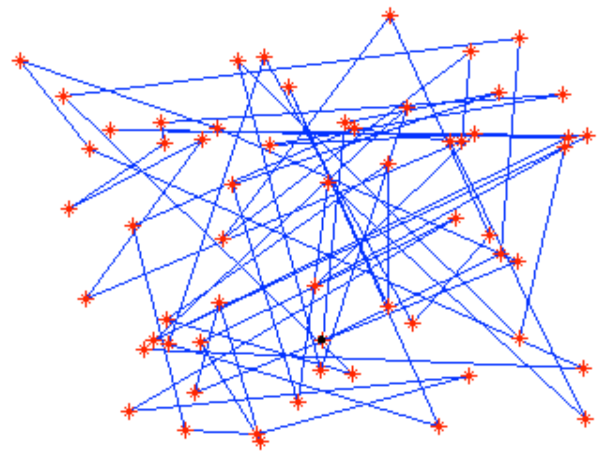
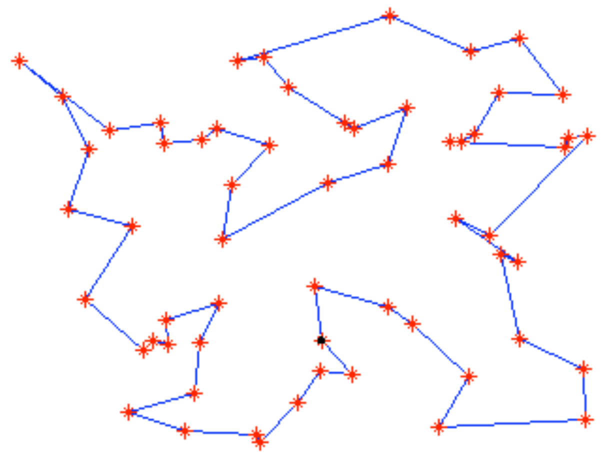


Figure 6.6.5. Branch-and-bound.

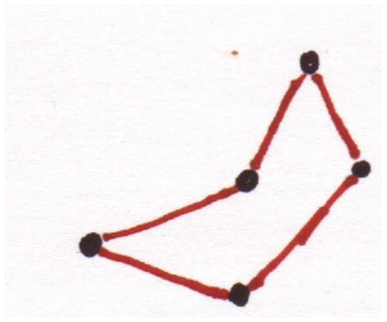
Local search for the TSP

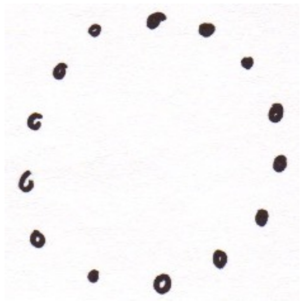


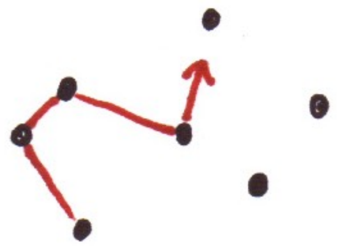


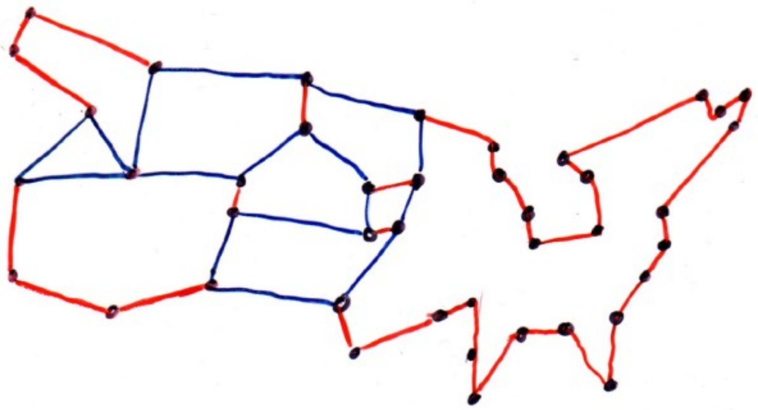
from Wikipedia

How solve in practice: TSP examples...









A lot of mathematical engineering!

Which algorithm design technique?

What algorithm design technique you should use depends on:

- **Do you think the problem might be polynomial**, then go for no less than a perfect solution (main alt:: D&C,DP, greedy)

- **If you think the problem is exponential**, you will probably have to accept some compromise:
  - Use an exponential algorithm and avoid solving large problems. If you tune it, maybe you will be able to solve your problem.
  - Use a heuristic that is faster but may give lower quality

- **If you have no idea** try all methods (in several ways) and then analyze your algorithms!