Algorithms

Exhaustive Search & Local Search

Hee-Kap Ahn
Graduate School of Artificial Intelligence
Dept. Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)

Textbooks: Algorithms by Jeff Erickson /Algorithms by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani.

Coping with NP-hardness

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

Theory says you're unlikely to find a polynomial-time algorithm unless you sacrifice one or more of the desired features.

- Optimality of the solution.
- Polynomial running time of the algorithm.
- Arbitrary instances of the problem.

Coping with NP-hardness

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

Theory says you're unlikely to find a polynomial-time algorithm unless you sacrifice one or more of the desired features.

- Optimality of the solution.
- Polynomial running time of the algorithm.
- Arbitrary instances of the problem.

You may have to rely on **intelligent exponential search** such as *backtracking* and *branch and bound*. These algorithms take exponential time in worst-case, but could be very efficient on typical instances if they are designed with deliberation.

A **heuristic algorithm** relies on ingenuity, intuition, a good understanding of the application, meticulous experiments, and insights from other fields of research. It may work well, but with no guarantees on either the running time or the quality of the solution.

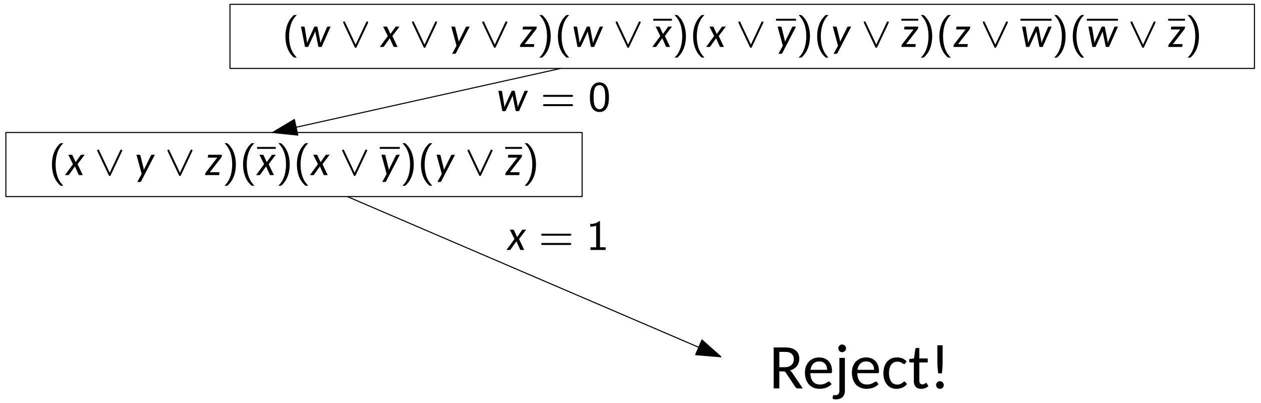
Backtracking

- Compute candidates to the solution incrementally, one small piece at a time.
- For multiple alternatives to the solution, evaluate every alternative recursively and abandon it (backtracks) immediately if it cannot be a component to the solution.

Consider an instance of SAT consisting of 6 clauses from 4 variables w, x, y, z.

$$\phi(w,x,y,z) = (w \vee x \vee y \vee z)(w \vee \overline{x})(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{w})(\overline{w} \vee \overline{z}).$$

All assignments with $w = 0$ and $x = 1$ can be instantly eliminated.


$(w \vee x \vee y \vee z)(w \vee \overline{x})(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{w})(\overline{w} \vee \overline{z})$
$w = 0$
$(x \vee y \vee z)(\overline{x})(x \vee \overline{y})(y \vee \overline{z})$
$x = 1$
Reject!

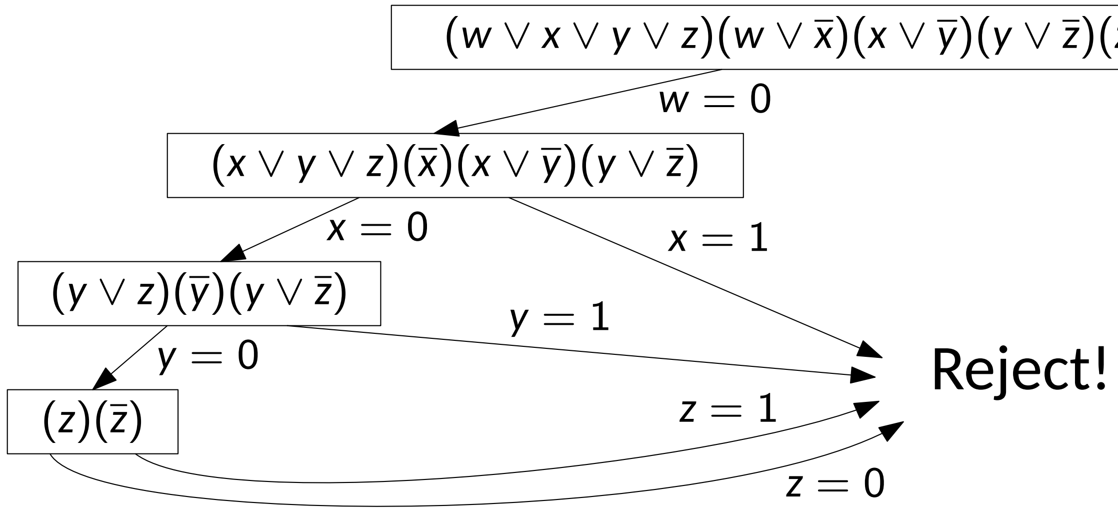
Backtracking

- Compute candidates to the solution incrementally, one small piece at a time.
- For multiple alternatives to the solution, evaluate every alternative recursively and abandon it (backtracks) immediately if it cannot be a component to the solution.

Consider an instance of SAT consisting of 6 clauses from 4 variables w, x, y, z.

$$\phi(w,x,y,z) = (w \vee x \vee y \vee z)(w \vee \bar{x})(x \vee \bar{y})(y \vee \bar{z})(z \vee \bar{w})(\bar{w} \vee \bar{z}).$$

All assignments with $w = 0$ and $x = 1$ can be instantly eliminated.



$(w \vee x \vee y \vee z)(w \vee \bar{x})(x \vee \bar{y})(y \vee \bar{z})(z \vee \bar{w})(\bar{w} \vee \bar{z})$
$w = 0$
$(x \vee y \vee z)(\bar{x})(x \vee \bar{y})(y \vee \bar{z})$
$x = 0$
$x = 1$
$(y \vee z)(\bar{y})(y \vee \bar{z})$
$y = 1$
$y = 0$
$(z)(\bar{z})$
$z = 1$
$z = 0$
Reject!

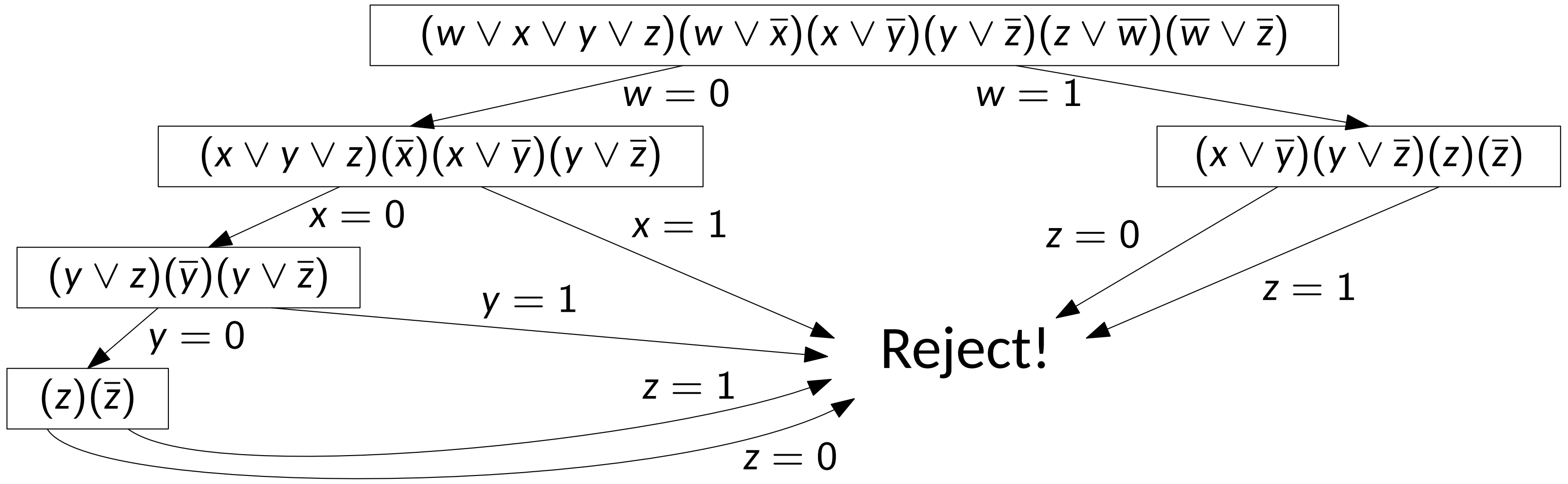
Backtracking

- Compute candidates to the solution incrementally, one small piece at a time.
- For multiple alternatives to the solution, evaluate every alternative recursively and abandon it (backtracks) immediately if it cannot be a component to the solution.

Consider an instance of SAT consisting of 6 clauses from 4 variables w, x, y, z.

$$\phi(w,x,y,z) = (w \vee x \vee y \vee z)(w \vee \overline{x})(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{w})(\overline{w} \vee \overline{z}).$$

All assignments with $w = 0$ and $x = 1$ can be instantly eliminated.


$(w \vee x \vee y \vee z)(w \vee \overline{x})(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{w})(\overline{w} \vee \overline{z})$
$w = 0$
$w = 1$
$(x \vee y \vee z)(\overline{x})(x \vee \overline{y})(y \vee \overline{z})$
$(x \vee \overline{y})(y \vee \overline{z})(z)(\overline{z})$
$x = 0$
$x = 1$
$z = 0$
$z = 1$
$(y \vee z)(\overline{y})(y \vee \overline{z})$
$y = 1$
$y = 0$
Reject!
$(z)(\overline{z})$
$z = 1$
$z = 0$

Backtracking

Backtracking explores the space of assignments, growing the tree only at nodes where there is uncertainty about the output.

But, which subproblem to expand next? Which branching variable to use?

> Choose the subproblem that contains the smallest clause and then branch on a variable in that clause.

The backtracking procedure has the following format.

```
Start with some problem $P_0$
Let $\mathcal{S} = \{P_0\}$, the set of active subproblems.
while $\mathcal{S}$ is nonempty:
        choose a subproblem $P \in \mathcal{S}$ and remove it from $\mathcal{S}$
        expand it into smaller subproblems $P_1, P_2, \dots, P_k$
        for each $P_i$:
            if $\boxed{test(P_i)}$ succeeds:  halt and announce this solution
            if $\boxed{test(P_i)}$ fails:  discard $P_i$
            Otherwise:  add $P_i$ to $\mathcal{S}$
Announce that there is no solution
```

recursive calls to subproblems

Uncertain

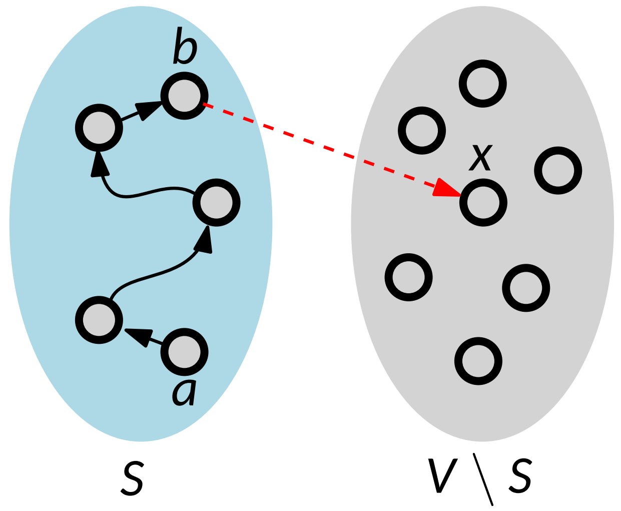
Branch-and-bound

A similar technique for optimization problems by using lower/upper bounds.

To eliminate a partial solution quickly for a minimization problem, we use a lower bound on the cost - without knowing the exact cost.

> If the lower bound of a partial solution exceeds the best solution known so far, we eliminate the partial solution immediately.

Consider a TSP instance on a complete weighted graph $G = (V, E)$. Let $[a, S, b]$ denote a partial solution which is a simple path from a to b passing through the vertices of S, where $a, b \in S \subseteq V$.



At each step, extend a particular solution $[a, S, b]$ by a single edge (b, x), where $x \in V \setminus S$.

$\Rightarrow |V \setminus S|$ subproblems of the from $[a, S \cup \{x\}, x]$.

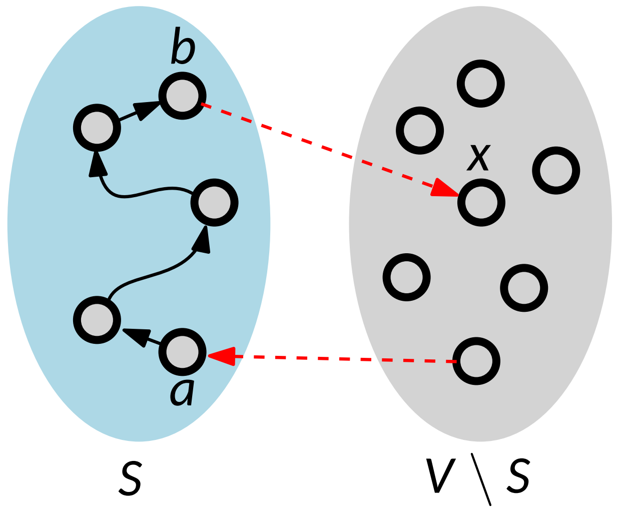
Branch-and-bound

A similar technique for optimization problems by using lower/upper bounds.

To eliminate a partial solution quickly for a minimization problem, we use a lower bound on the cost - without knowing the exact cost.

> If the lower bound of a partial solution exceeds the best solution known so far, we eliminate the partial solution immediately.

Consider a TSP instance on a complete weighted graph $G = (V, E)$. Let $[a, S, b]$ denote a partial solution which is a simple path from a to b passing through the vertices of S, where $a, b \in S \subseteq V$.



At each step, extend a particular solution $[a, S, b]$ by a single edge (b, x), where $x \in V \setminus S$.

$\Rightarrow |V \setminus S|$ subproblems of the from $[a, S \cup \{x\}, x]$.

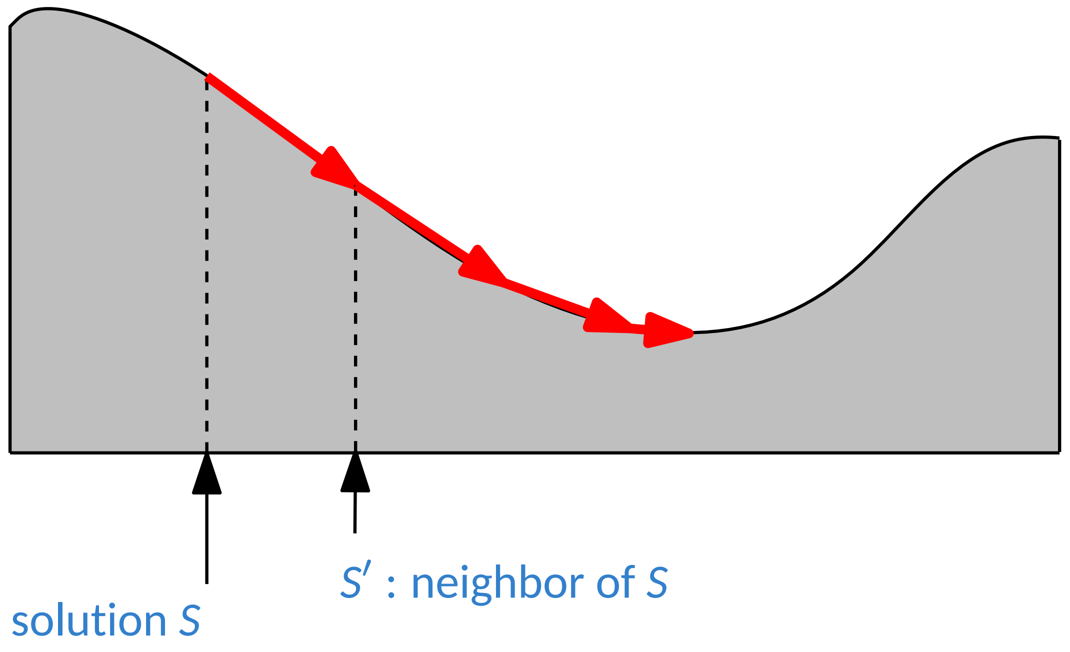
The cost of the remainder of the tour $\geqslant$ (1) the lightest edge from $V \setminus S$ to a
+ (2) the lightest edge from b to $V \setminus S$
+ (3) an MST of $V \setminus S$.

Branch-and-bound

```
Start with some problem $P_0$
Let $\mathcal{S} = \{P_0\}$, the set of active subproblems.
bestsofar $= \infty$
**while** $\mathcal{S}$ is nonempty:
      choose a subproblem $P \in \mathcal{S}$ and remove it from $\mathcal{S}$
      expand it into smaller subproblems $P_1, P_2, \ldots, P_k$
      For each $P_i$:
          **if** $P_i$ is a complete solution:  update bestsofar
          **else** if lowerbound($P_i$) $<$ bestsofar:  add $P_i$ to $\mathcal{S}$
**return** bestsofar
```

Local Search

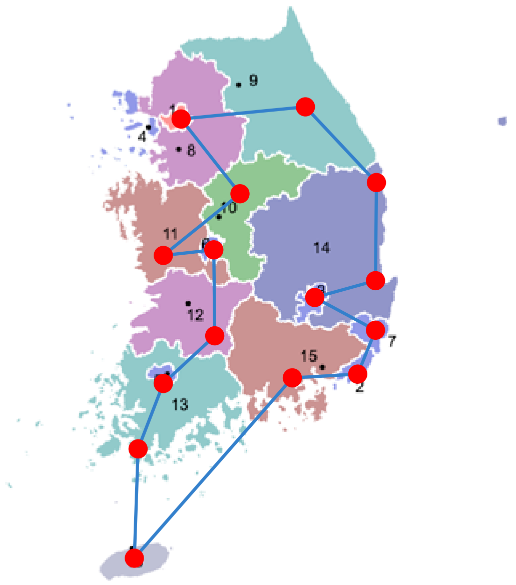
Local search. An algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a “nearby" one.



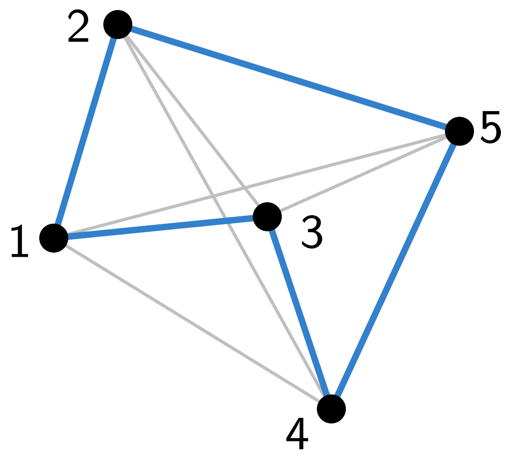
In general, it is not difficult to design an approach, but it is often very difficult to say anything about the quality of the solution.

Traveling Salesperson Problem

TSP. Given a list of cities and their pairwise distances, find the shortest tour visiting all cities exactly once.



TSP is often modelled as an undirected weighted complete graph.



$(1, 2, 5, 4, 3)$

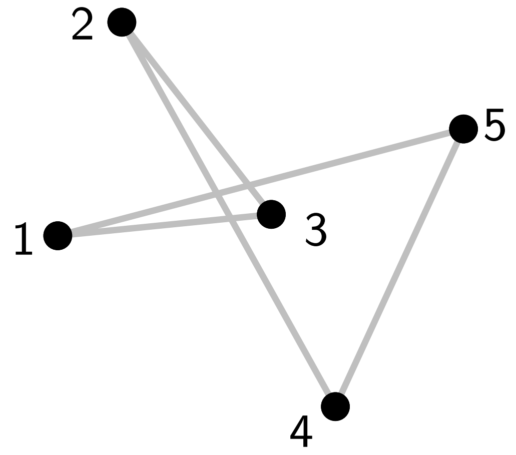
Brute-force search

$(1, 2, 3, 4, 5)$
$(1, 3, 5, 2, 4)$
$(1, 3, 4, 2, 5)$
$\vdots$

There are $(n-1)!$ cases.

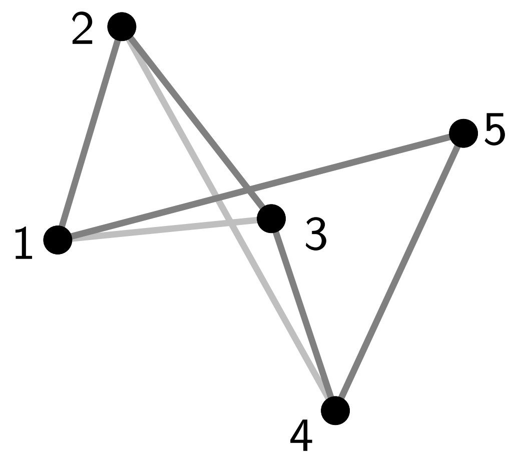
if $n = 50$, then there are
$49! = 608281864034267560872252163321295376887552831379210240000000000$ cases.

Local Search



$(1, 3, 2, 4, 5)$

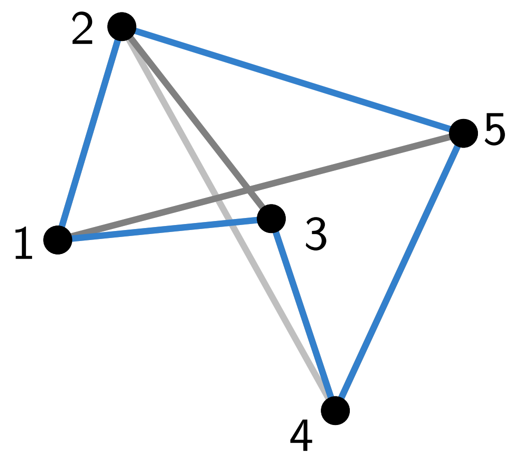
Local Search



$$\begin{tikzcd}[column sep=tiny] (1, & \boxed{3,} \arrow[r, dashed, bend left] & \boxed{2,} \arrow[l, dashed, bend right] & 4, 5) \end{tikzcd}$$

$(1, 2, 3, 4, 5)$

Local Search

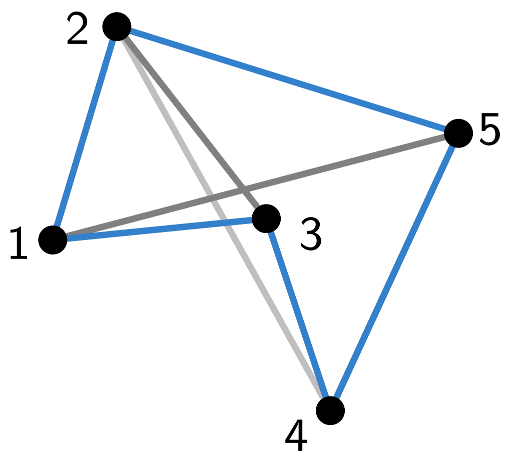


$(1, \boxed{3,}\boxed{2,} 4, 5)$

$(1, 2, \boxed{3,} 4, \boxed{5})$

$(1, 2, 5, 4, 3)$

Local Search

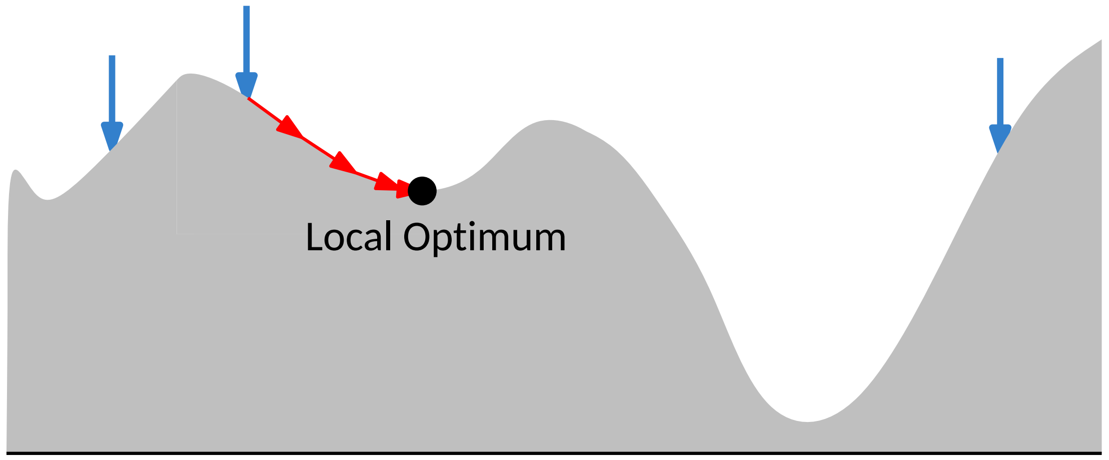


$(1, \boxed{3,} \boxed{2,} 4, 5)$

$(1, 2, \boxed{3,} 4, \boxed{5})$

$(1, 2, 5, 4, 3)$

Whenever local search methods can get stuck in a local minimum, starting from a different initial configuration.



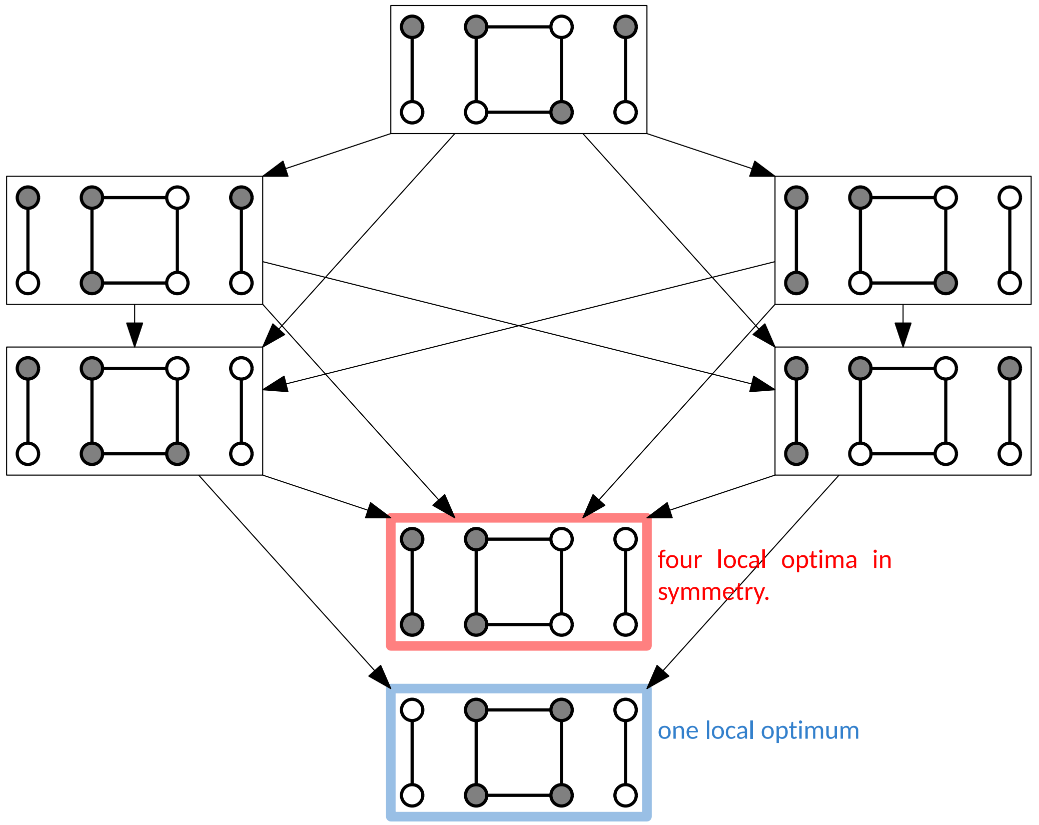
Randomization and Restarts

As the problem size grows exponentially, repeating the local search gets ineffective.

Randomization. (1) picks a random initial solution and (2) chooses a local move when several local optima are available.

Repeating local search with different random seed on each invocation. If the probability of reaching a good local optimum on any given run is p, then within $O(1/p)$ runs such a solution is likely to be found.

Example - Graph partitioning of 8 nodes into two equal-sized subsets. If two end nodes of an edge are in different subsets, it costs 1. Find a partition with minimum cost.



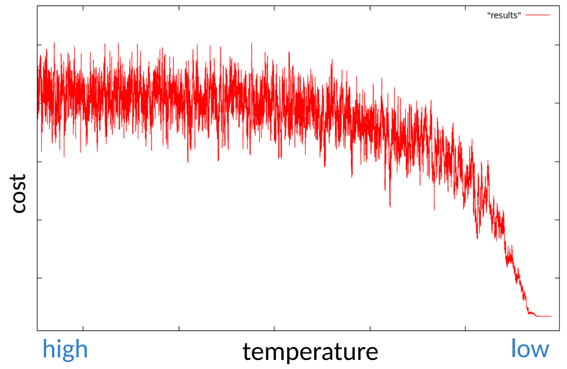
If local search starts at a random solution, and at each step a random neighbor of lower cost is selected, the search is at most four times as likely to wind up in a bad solution than a good one.

Simulated Annealing

Gibbs-Boltzmann function. The probability of finding a physical system in a state with energy E is proportional to $e^{-E/(kT)}$, where $T > 0$ is temperature and k is a constant.

- For any temperature $T > 0$, function is monotone decreasing function of energy E.
- System more likely to be in a lower energy state than higher one.
  - T large: high and low energy states have roughly same probability
  - T small: low energy states are much more probable

```
Let $s$ be any starting solution and $T$ be a fixed temperature
repeat
    randomly choose a solution $s'$ in the neighborhood of $s$
    if $\Delta = \text{cost}(s') - \text{cost}(s) < 0$:
        replace $s$ by $s'$
    else:   replace $s$ by $s'$ with probability $e^{-\Delta/T}$
```

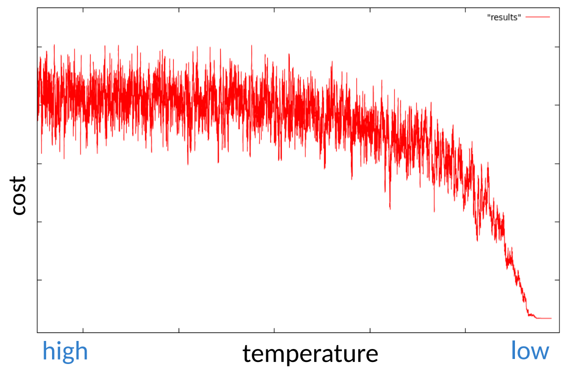


Simulated Annealing

Gibbs-Boltzmann function. The probability of finding a physical system in a state with energy E is proportional to $e^{-E/(kT)}$, where $T > 0$ is temperature and k is a constant.

- For any temperature $T > 0$, function is monotone decreasing function of energy E.
- System more likely to be in a lower energy state than higher one.
  - T large: high and low energy states have roughly same probability
  - T small: low energy states are much more probable

```
Let s be any starting solution and T be a fixed temperature
repeat
    randomly choose a solution s′ in the neighborhood of s
    if $\Delta = \text{cost}(s') - \text{cost}(s) < 0$:
        replace s by s′
    else: replace s by s′ with probability $e^{-\Delta/T}$
```



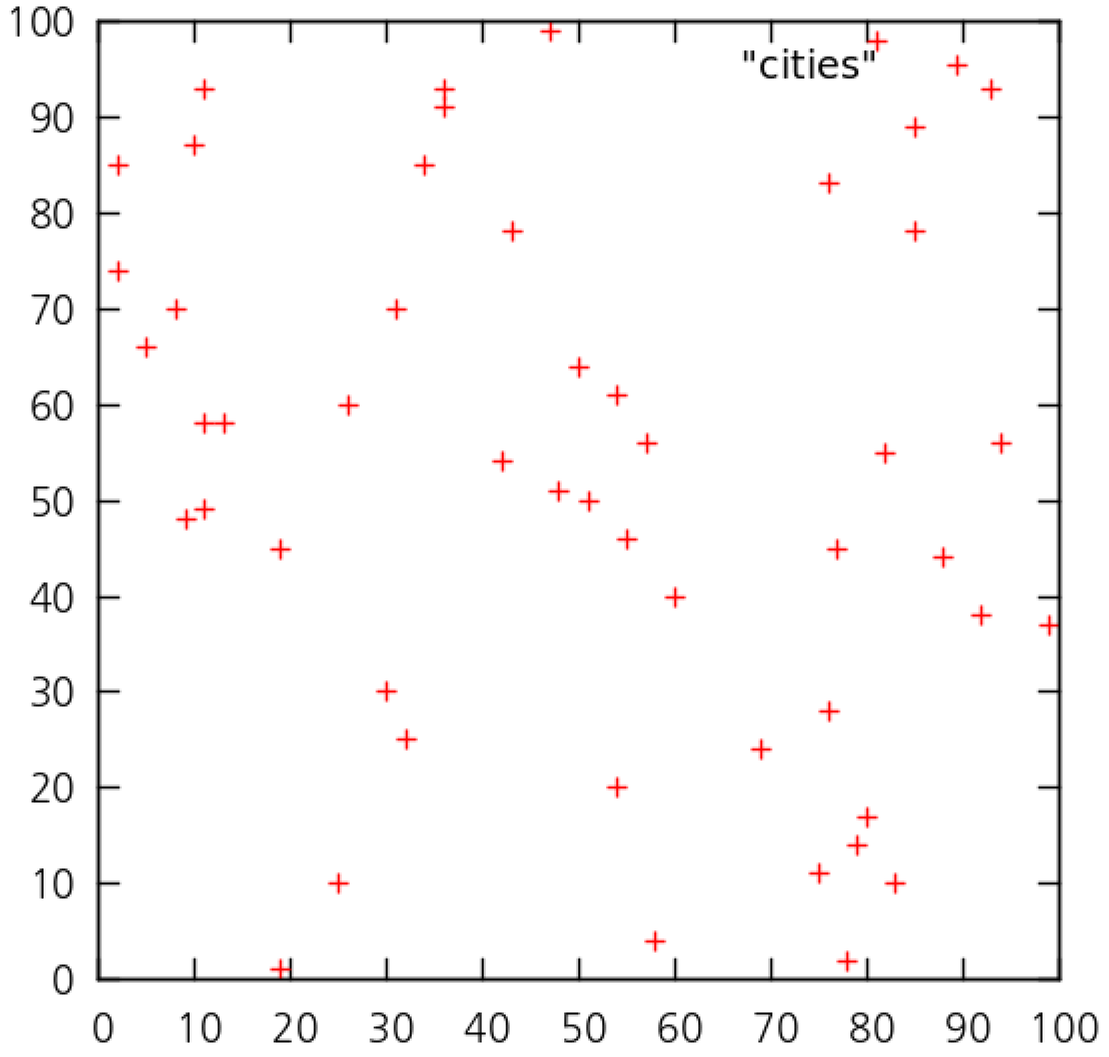
Simulated annealing. Makes occational excursions, at a significant cost, out of low-cost region to escape local optima.

- T large : probability of accepting an uphill move is large
- T small : uphill moves are almost never accepted.
- Idea : to start with T large and then gradually reduce it to zero
- Cooling schedule: $T = T(i)$ at iteration i.

Example - TSP

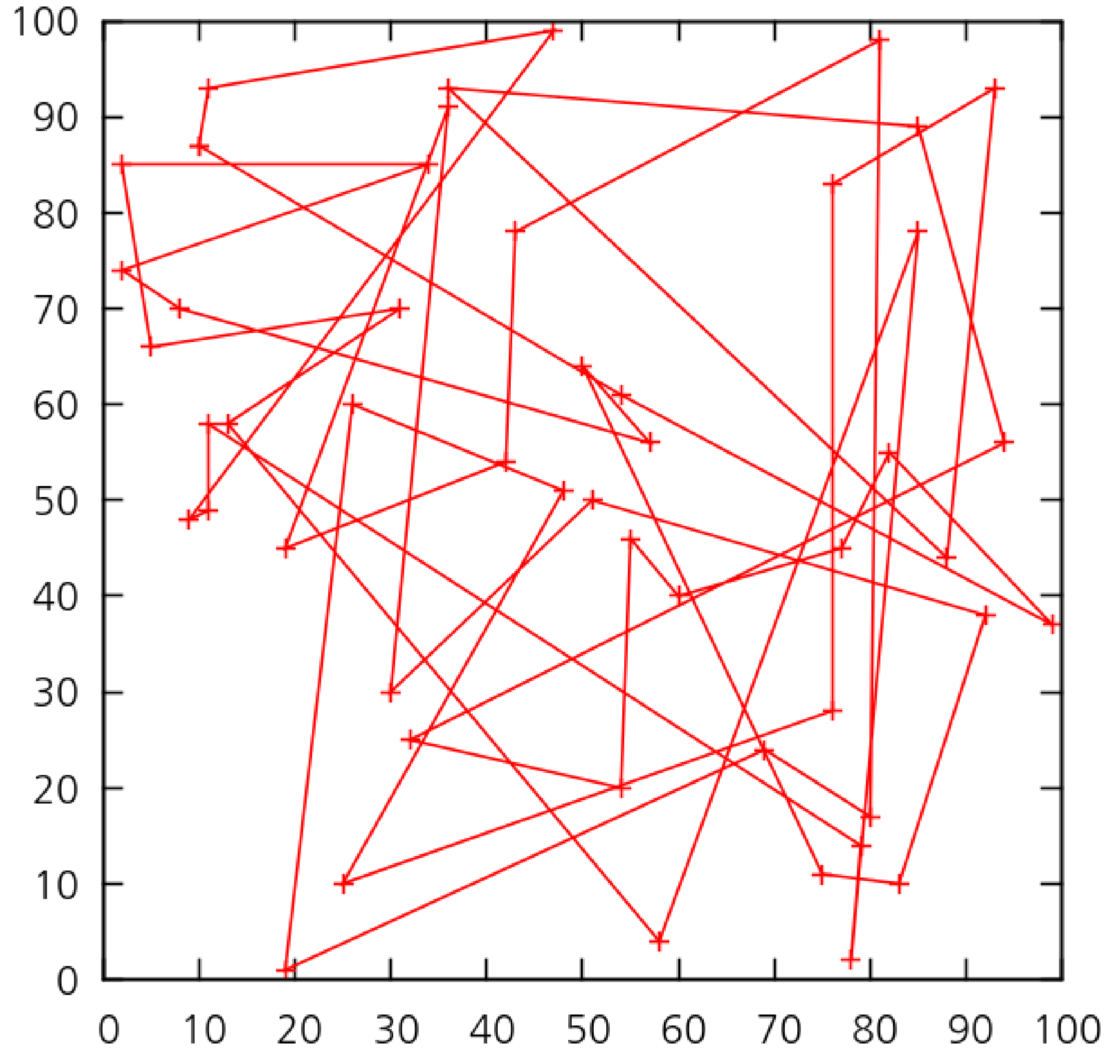
50 cities in the plane.

$49! = 608281864034267560872252163321295376887552831379210240000000000$



Example - TSP

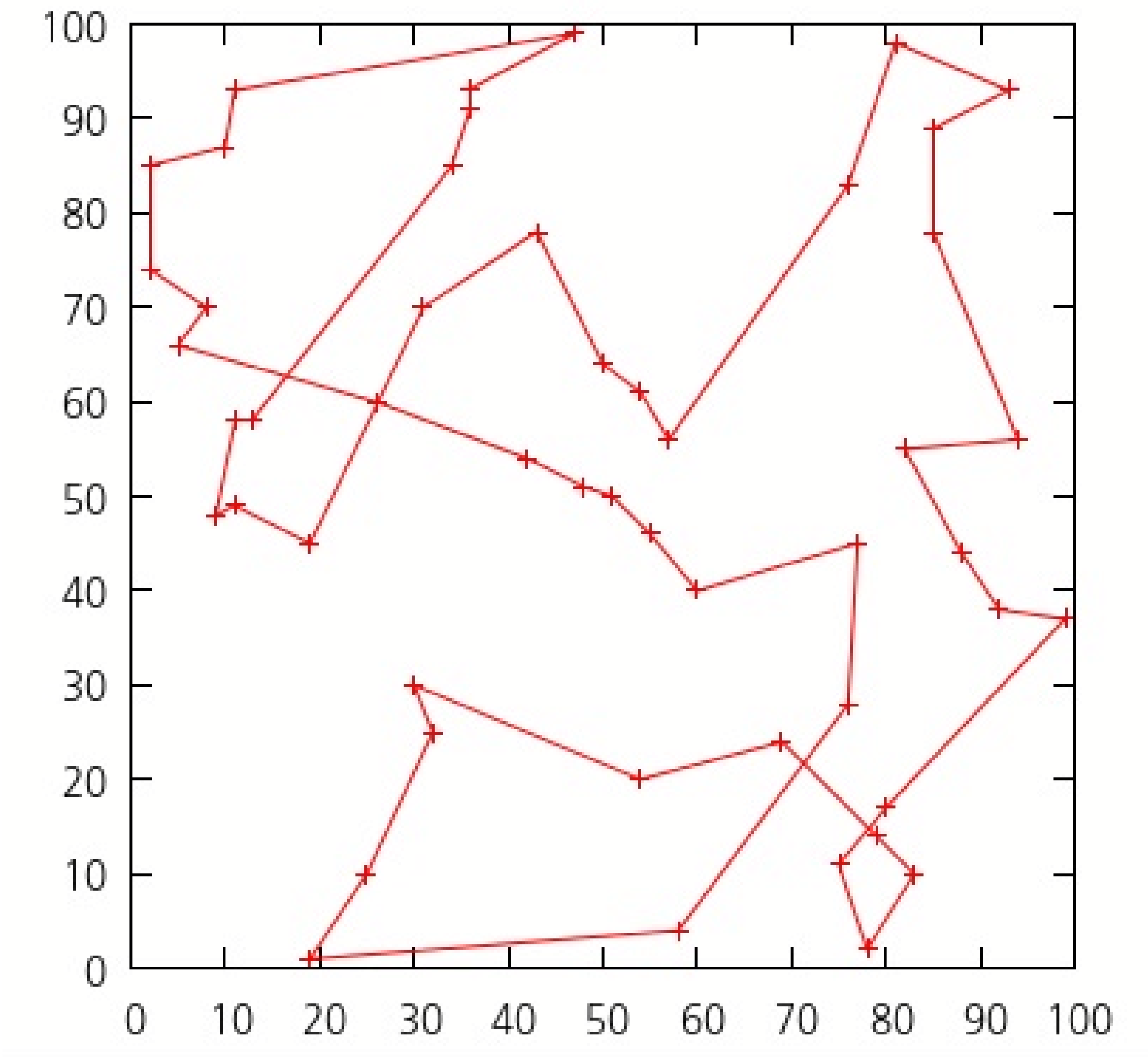
50 cities in the plane.
$49! = 608281864034267560872252163321295376887552831379210240000000000$



Brute-force search 100, 000 tries : cost 1901.069

Example - TSP

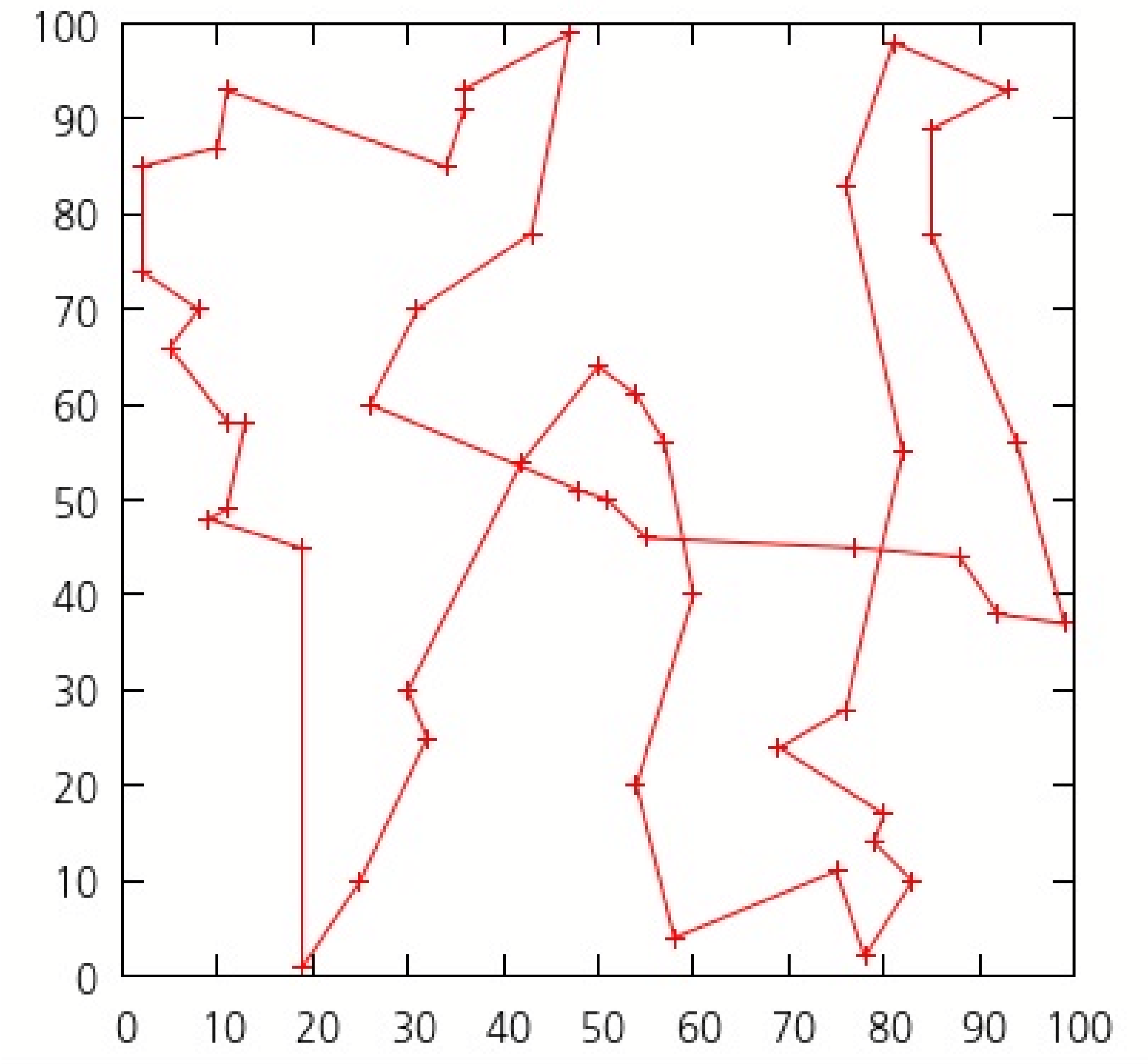
50 cities in the plane.
$49! = 608281864034267560872252163321295376887552831379210240000000000$



Repeated local search : cost 680.031

Example - TSP

50 cities in the plane.
$49! = 608281864034267560872252163321295376887552831379210240000000000$



Simulated annealing : cost 639.763