


Part II
Methods of AI

Chapter 2
Problem-solving

Part II: Methods of AI

Chapter 2 - Problemsolving

2.1 Uninformed Search

2.2 Informed Search

2.3 Constraint Satisfaction Problems

UNIVERSITÄT DES SAARLANDES

DFKI


2.2 Informed Search

Heuristic Search Algorithms

Review: Tree search

```
function TREE-SEARCH (problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE [problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT (fringe)
        if GOAL-TEST [problem] applied to STATE (node) succeeds then
            return node
        fringe ← INSERTALL (EXPAND (node, problem), fringe)
```

A strategy is defined by picking the *order of node expansion*

Best-first search

Idea: use an *evaluation function* for each node

- estimate of “desirability”

Expand most desirable unexpanded node

Implementation:

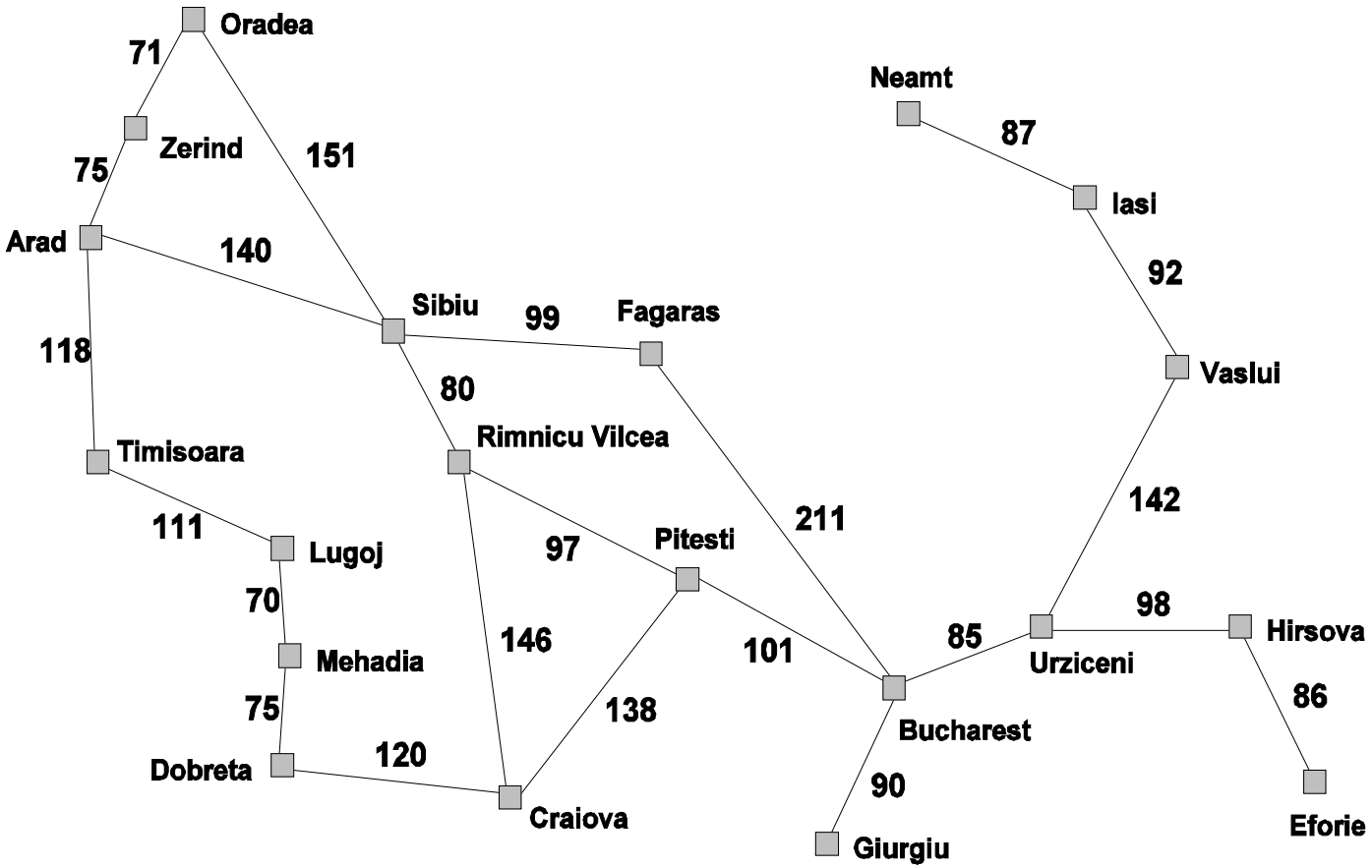
fringe is a queue sorted in decreasing order of desirability

Special cases:

greedy search

A* search

Romania with step costs in km



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

UNIVERSITÄT DES SAARLANDES

DFKI

Greedy search

Evaluation function $h(n)$ (heuristic)

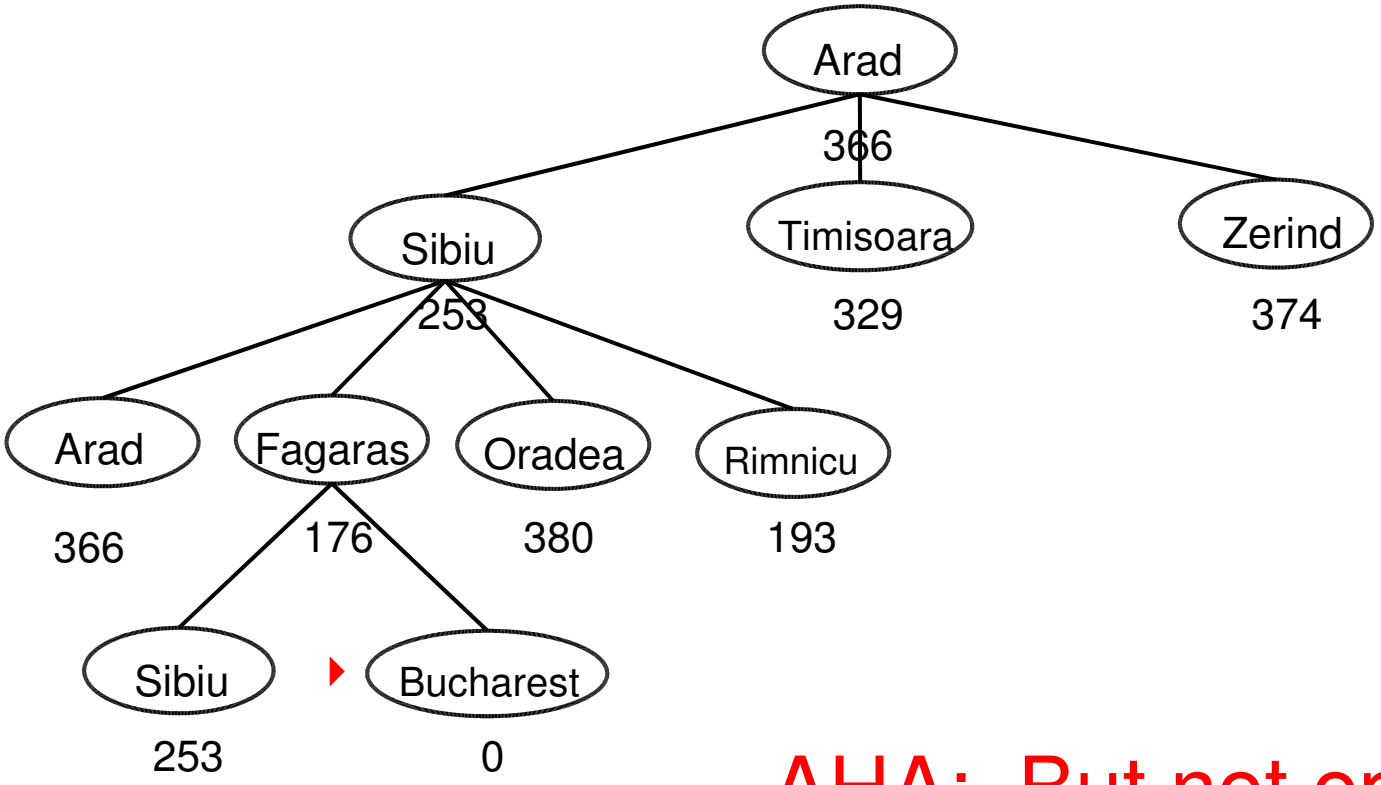
= estimate of cost from n to the closest goal

For Example:

$h_{SLD}(n)$ = *straight-line distance* from n to Bucharest

Greedy search expands the node that *appears* to be closest to the goal

Greedy search example



AHA: But not optimal!!!!

Properties of greedy search

Complete?? No - can get stuck in loops, e.g.,

lasi → Neamt → lasi → Neamt →

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$ — keeps all nodes in memory

Optimal?? No

Conclusion: Only useful for local search. For heuristic non-local search use A*!

A*- Search

Idea: avoid expanding paths that are already expensive

Evaluation function: $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n

$h(n)$ = estimated cost to goal from n

$f(n)$ = estimated total cost of path from root through n to goal

A*- Search

A*-search uses an *admissible* heuristic:

always: $h(n) \leq h^*(n)$

where $h^*(n)$ is the *true* cost from n.

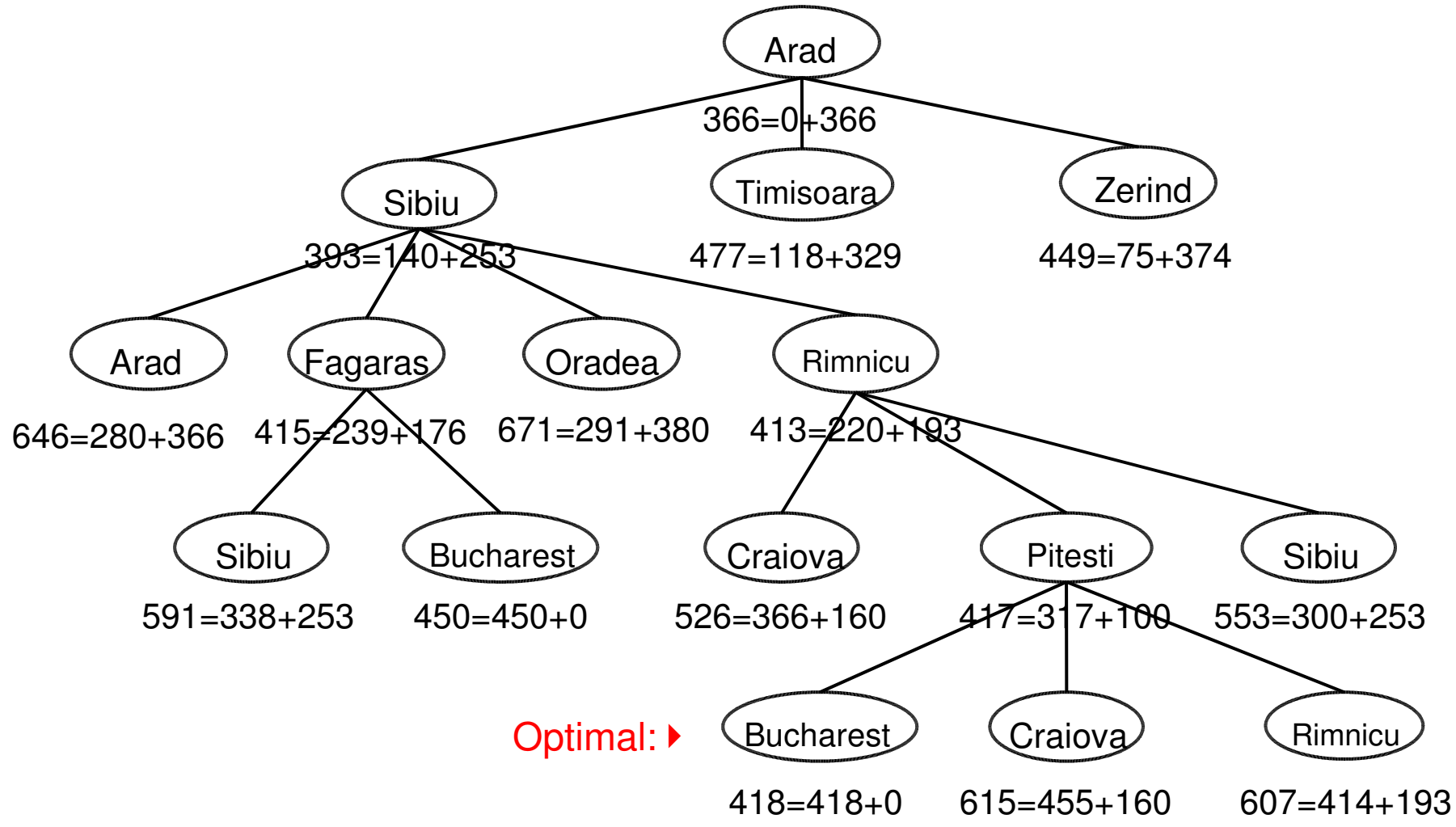
Also $h(n) \geq 0$, so $h(G) = 0$ for any goal G

For Example:

$h_{SLD}(n)$ never overestimates the actual road distance

Theorem: A* - Search is optimal

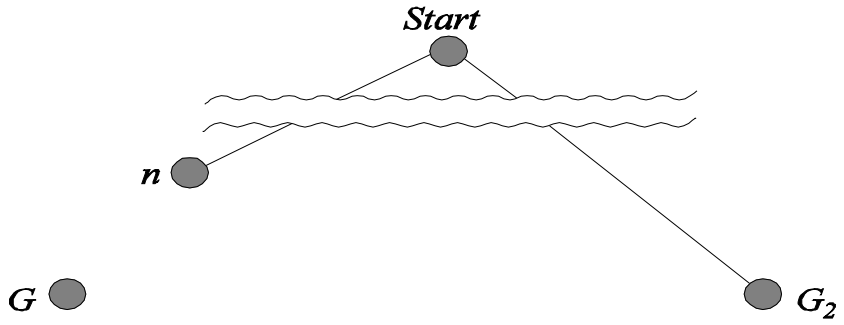
A*-Search: An Example



Optimality of A* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue.

Let n be an unexpanded node on a shortest path to an optimal goal G_1.



$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$> g(G_1)$ since G_2 is suboptimal

$\geq f(n)$ since h is admissible

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

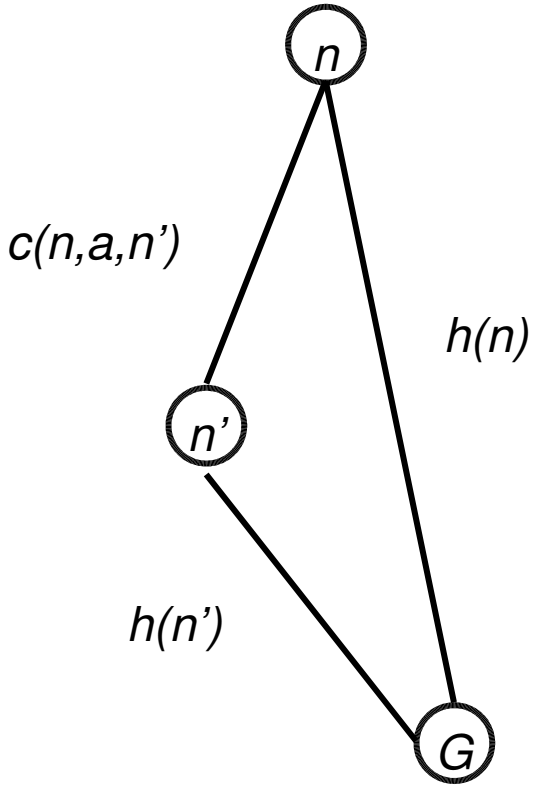
Consistency

A heuristic is consistent if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$



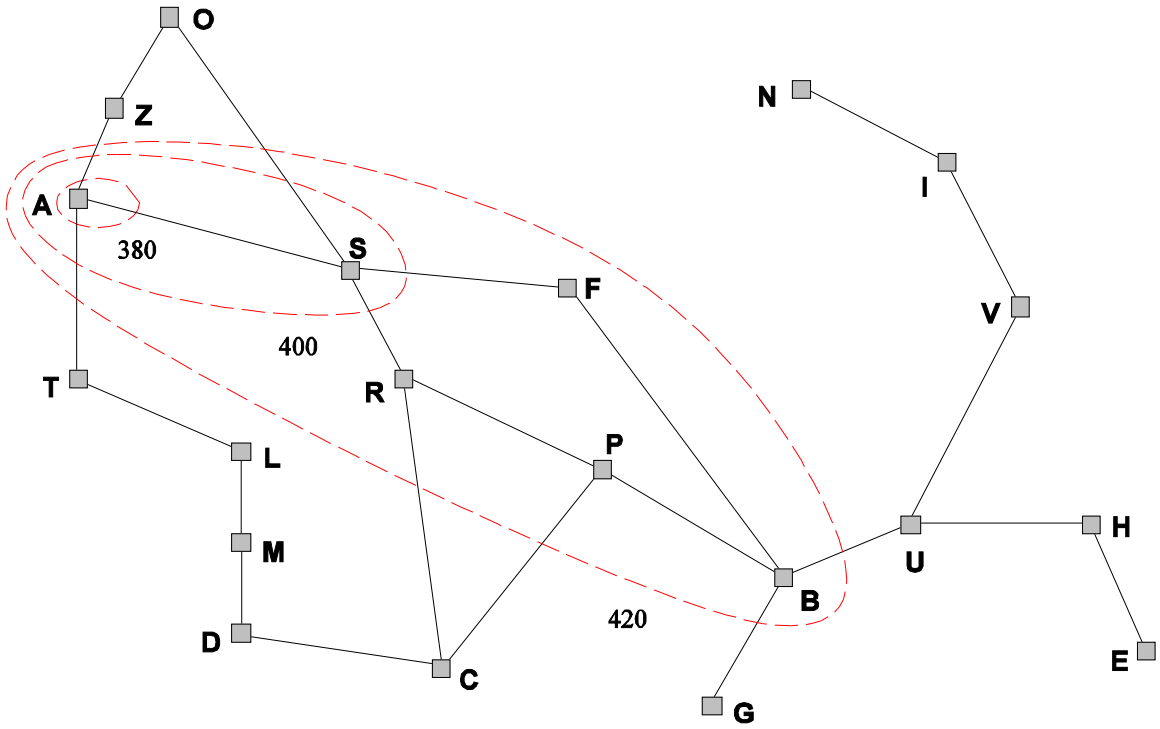
i.e., $f(n)$ is non-decreasing along any path.

Optimality of A* (more useful)

Lemma: **A* expands nodes in order of increasing *f*-value**

Gradually adds "*f*-contours" of nodes (cf. breadth-first adds layers)

Contour *i* has all nodes with $f = f_i$, where $f_i < f_{i+1}$



UNIVERSITÄT DES SAARLANDES

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h x length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes — cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

Properties of A*

Requirements:

Tree search: *h* must be admissible

Graph search: *h* must be even constant

h is consistent $\Leftrightarrow$ *f* non-deceasing

$\Rightarrow$ *h* is admissible

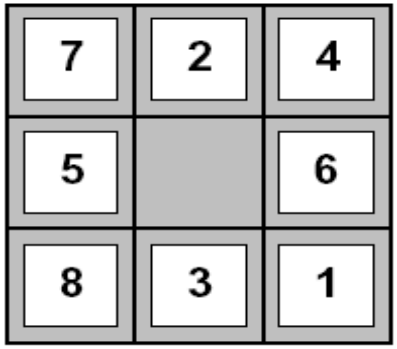
Most admissible heuristics are consistent

Admissible Heuristics

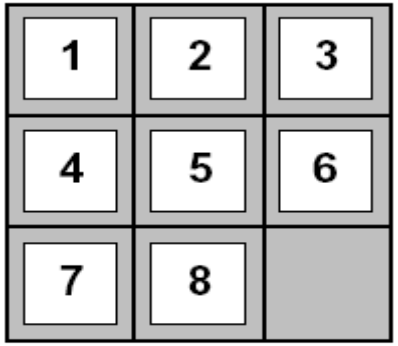
For Example: the 8-Puzzle

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance = number of squares from desired location



Start State



Goal State

$h1(S)$ =?? 7

$h2(S)$ = ?? 4+0+3+3+1+0+2+1 = 14

UNIVERSITÄT DES SAARLANDES

Dominance

If $h2(n) \geq h1(n)$ for all n (both admissible)
then $h2$ *dominates* $h1$ and is better for search

Typical search costs:

$d = 14$	IDS = 3,473,941
	$A^*(h_1)$ = 539 nodes
	$A^*(h_2)$ = 113 nodes
$d = 24$	IDS ≈ 54,000,000,000 nodes
	$A^*(h_1)$ = 39,135
	$A^*(h_2)$ = 1,641 nodes

UNIVERSITÄT DES SAARLANDES

DFKI

Relaxed problems

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution

If the rules are relaxed so that a tile can move to *any adjacent square*, then $h_2(n)$ gives the shortest solution

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Optimization problems

Local search problems

\+ Objective Function

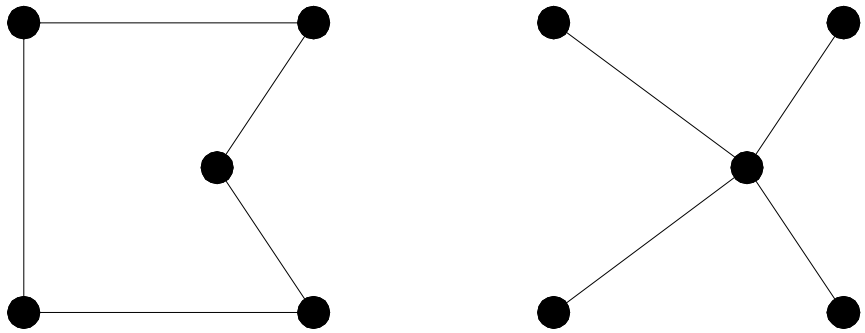
= Optimalisation problem

Find the optimal solution!

Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)

Find the shortest tour visiting all cities exactly once



Minimum spanning tree can be computed in $O(n^2)$

and is a lower bound on the shortest (open) tour

Iterative improvement algorithms

In many optimization problems, *path* is irrelevant;
the goal state itself is the solution

Then state space = set of "complete" configurations;

find *optimal* configuration, e.g. TSP

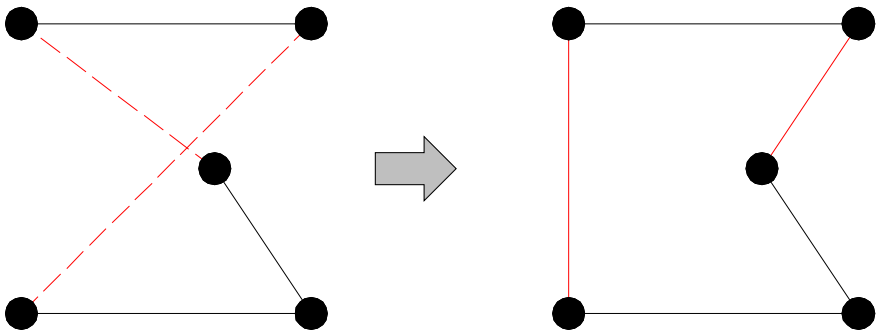
or, find configuration satisfying constraints, e.g. timetable

In such cases, can use *iterative improvement* algorithms;
keep a single "current" state, try to improve it

Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges



Hill-climbing (or gradient ascent/descent)

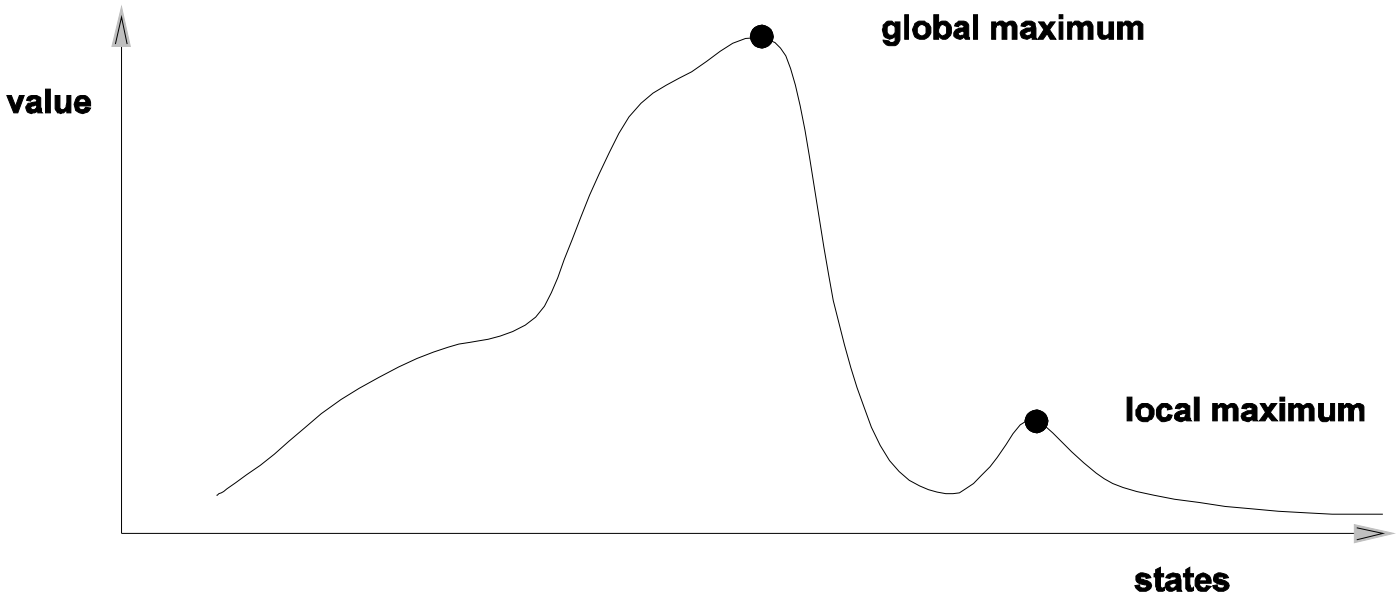
„Like climbing Everest in thick fog with amnesia“

```
function Hill-Climbing(problem) returns a state that is
   a local maximum
   inputs: problem, a problem
   local variables:        current, a node
                           neighbor, a node
   current ← Make-Node(Initial-State[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if Value[neighbor] < Value[current] then return
                           State[current]
      current ← neighbor
   end
```

UNIVERSITÄT DES SAARLANDES

Hill-climbing contd.

Problem: depending on initial state, can get stuck on local maxima



In continuous spaces, problems with choice of step size, slow convergence

Simulated Annealing

At fixed “temperature” T, state occupation probably reaches Boltzman distribution

$$p(x) = a\, e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Rightarrow$ always reach best state

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Simulated annealing

Idea: escape local maxima by allowing some „bad“ moves
but gradually decrease their size and frequency

```
function Simulated-Annealing(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to “temperature”
   local variables: current, a node
                    next, a node
                    T, a “temperature” controlling prob. of downward steps
   current ← Make-Node[Initial-State[problem]]
   for      t ← 1 to ∞ do
            T ← schedule[t]
            if T = 0 then return current
            next ← a randomly selected successor of current
            ΔE ← Value[next] - Value[current]
            if ΔE > 0 then current ← next
            else current ← next only with probability e^(ΔE/T)
```

Local Beam Search

- Local search (no path)
- K states in parallel instead of a single one
- States with poor performance are removed and others generate more than one successor states or completely new nodes are generated
  - **stochastically**
  - **genetically**

Online Search

Agent knows nothing but
- Whether current state is a goal state
- Possible actions in current state
- After performing an action: cost of this action

$$\text{Competitive Ratio} := \frac{\text{Expected costs of online search}}{\text{Minimal cost of informed agent}}$$

Examples: Safely Explorable Maze (no cliffs, no beasts)

see Learning Realtime A*

UNIVERSITÄT DES SAARLANDES

DFKI

Part II: Methods of AI

Chapter 2 - Problemsolving

2.1 Uninformed Search

2.2 Informed Search

2.3 Constraint Satisfaction Problems