Informed search algorithms

Lecturer: Eibe Frank

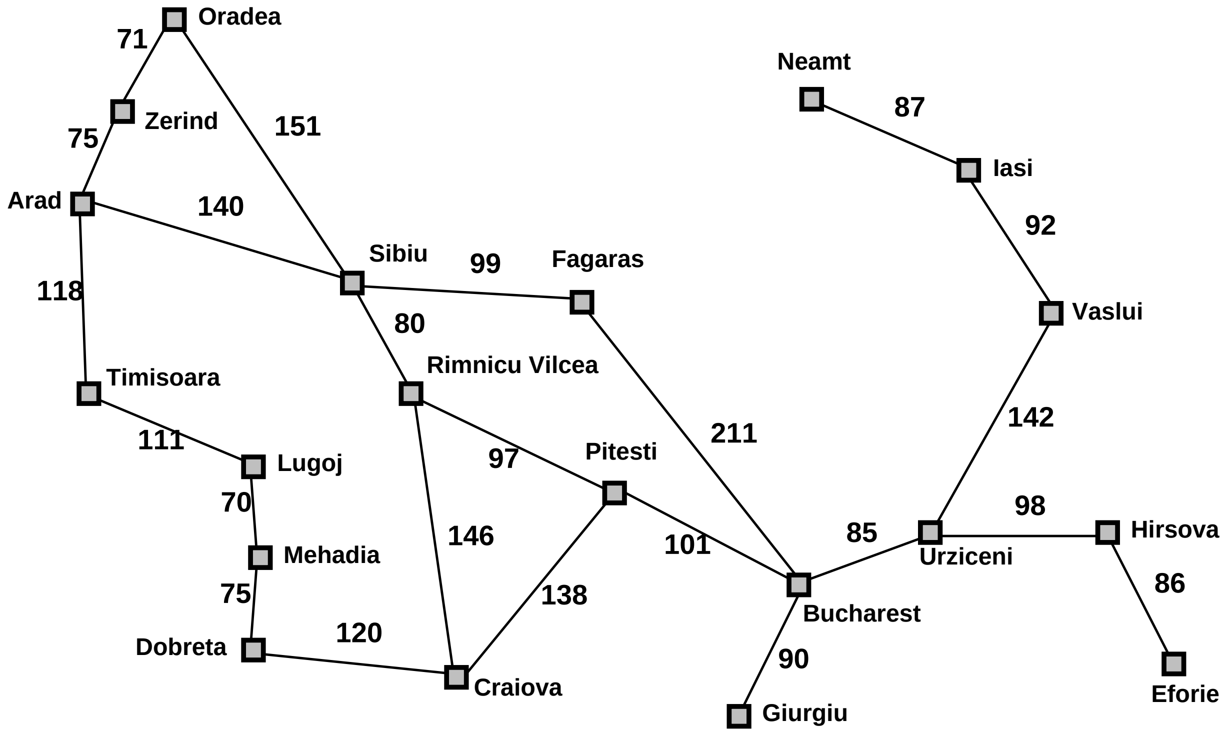
Based on "Artificial Intelligence"
by S. Russell and P. Norvig
Sections 4.1-4.3

- Best-first search
- Greedy search and A* search
- Heuristics
- Hill-climbing
- Simulated annealing
- Genetic algorithms

Best-first search

- An instance of *tree search* (or *graph search*)
- Idea: expand most desirable unexpanded node
- Need an estimate of "desirability" for each node provided by an **evaluation function** $f(n)$
  - Key component: **heuristic function** $h(n)$ that provides estimated cost of cheapest path from node n to goal node
  - Note: we assume that $h(n) = 0$ if n goal node
- *fringe* becomes queue sorted according to desirability
- Special cases:
  - Greedy search
  - A* search

Map of Romania



Straight−line distance to Bucharest

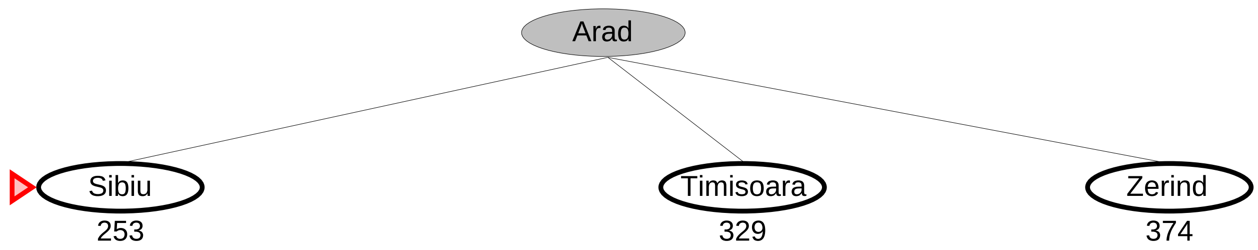
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

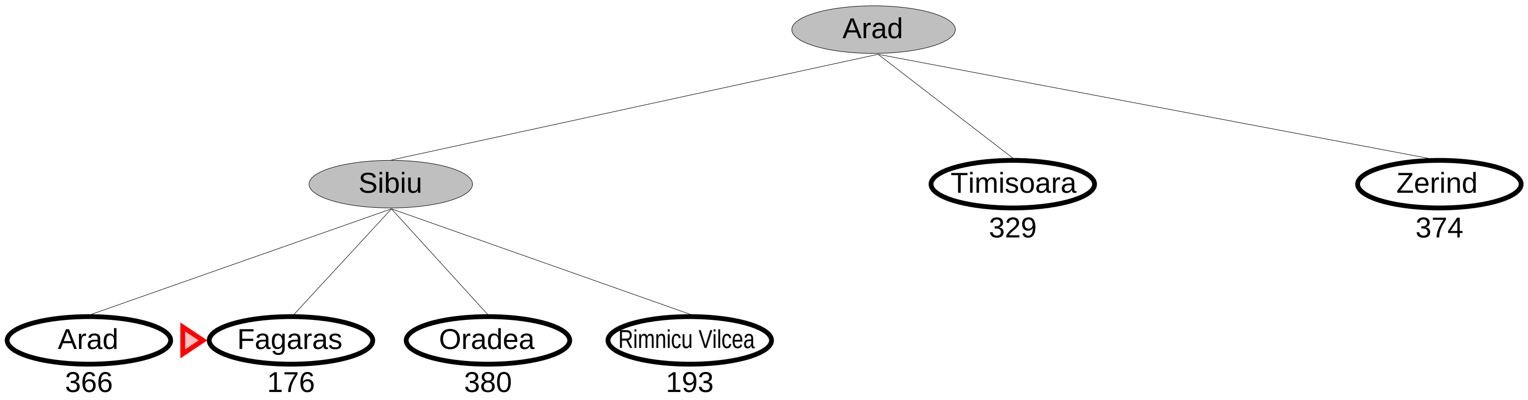
- Expands node that appears to be closest to goal, i.e. $f(n) = h(n)$
- Example: $h_{\mathrm{SLD}}(n)$ = straight-line distance from n to Bucharest
  - Assumes that straight-line distance is correlated with actual road distances
- Note: heuristic function cannot be computed from problem description itself

Greedy best-first search example

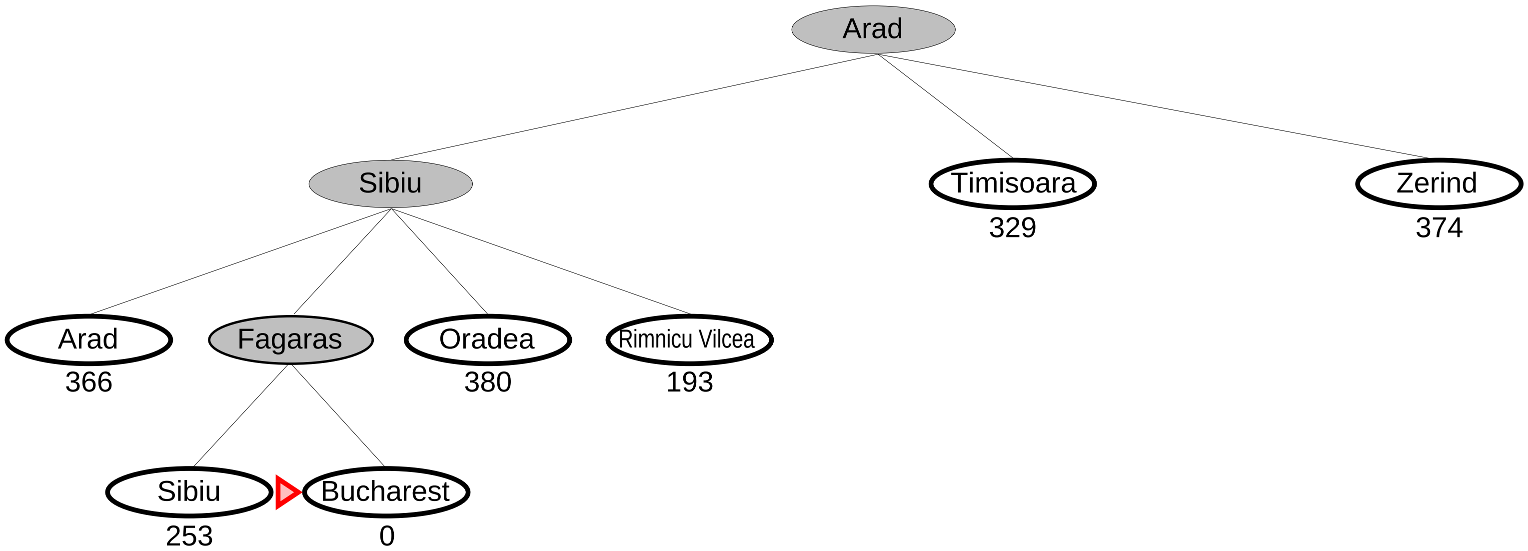
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



Properties of greedy best-first search

- Complete?
- Time?
- Space?
- Optimal?

Properties of greedy best-first search

- Complete?
  - No (e.g. going from Iasi to Oradea)
  - Complete in finite spaces with repeated-state checking
- Time?
- Space?
- Optimal?

Properties of greedy best-first search

- Complete?
  - No (e.g. going from Iasi to Oradea)
  - Complete in finite spaces with repeated-state checking
- Time?
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?
- Optimal?

Properties of greedy best-first search

- Complete?
  - No (e.g. going from Iasi to Oradea)
  - Complete in finite spaces 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?

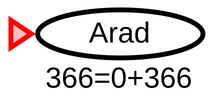
Properties of greedy best-first search

- Complete?
  - No (e.g. going from Iasi to Oradea)
  - Complete in finite spaces 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

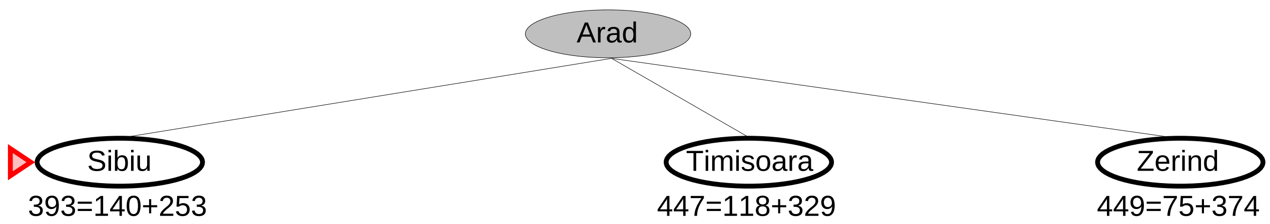
A^* search

- 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 through n to goal
- A^* search uses an **admissible** heuristic: $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.
  - Example: $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

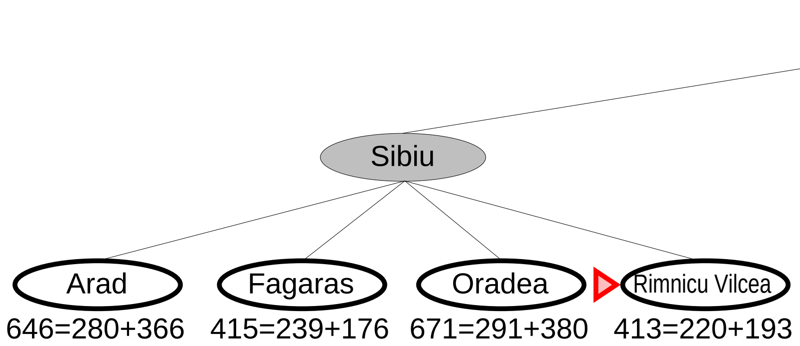
A* example



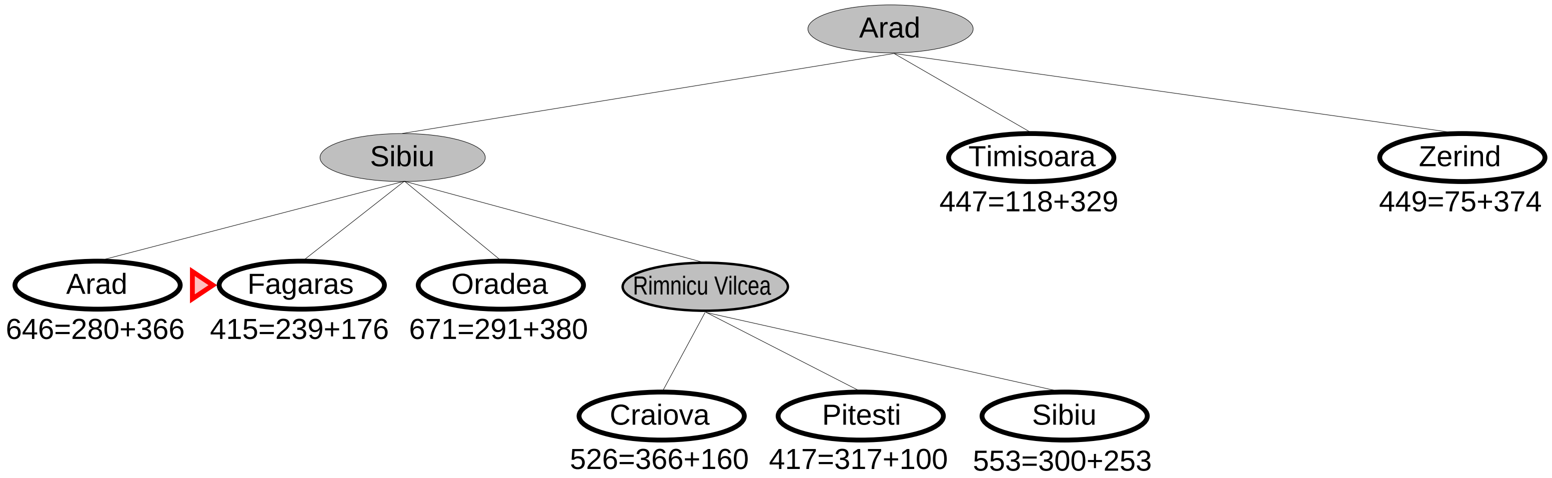
A* example


Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

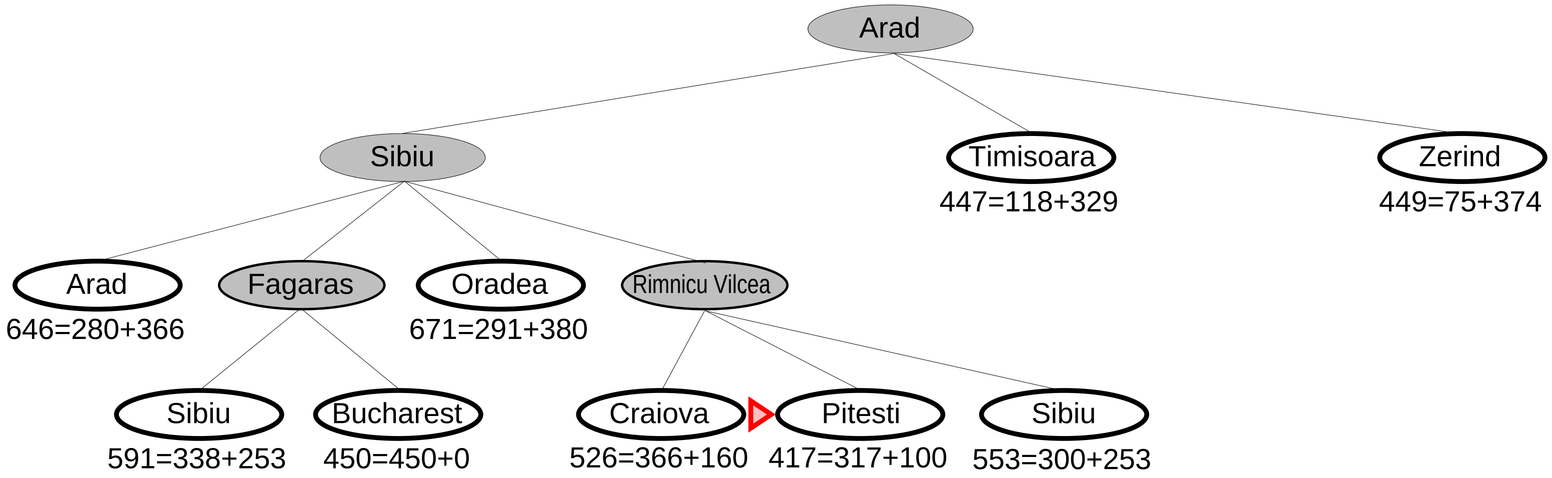
A* example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

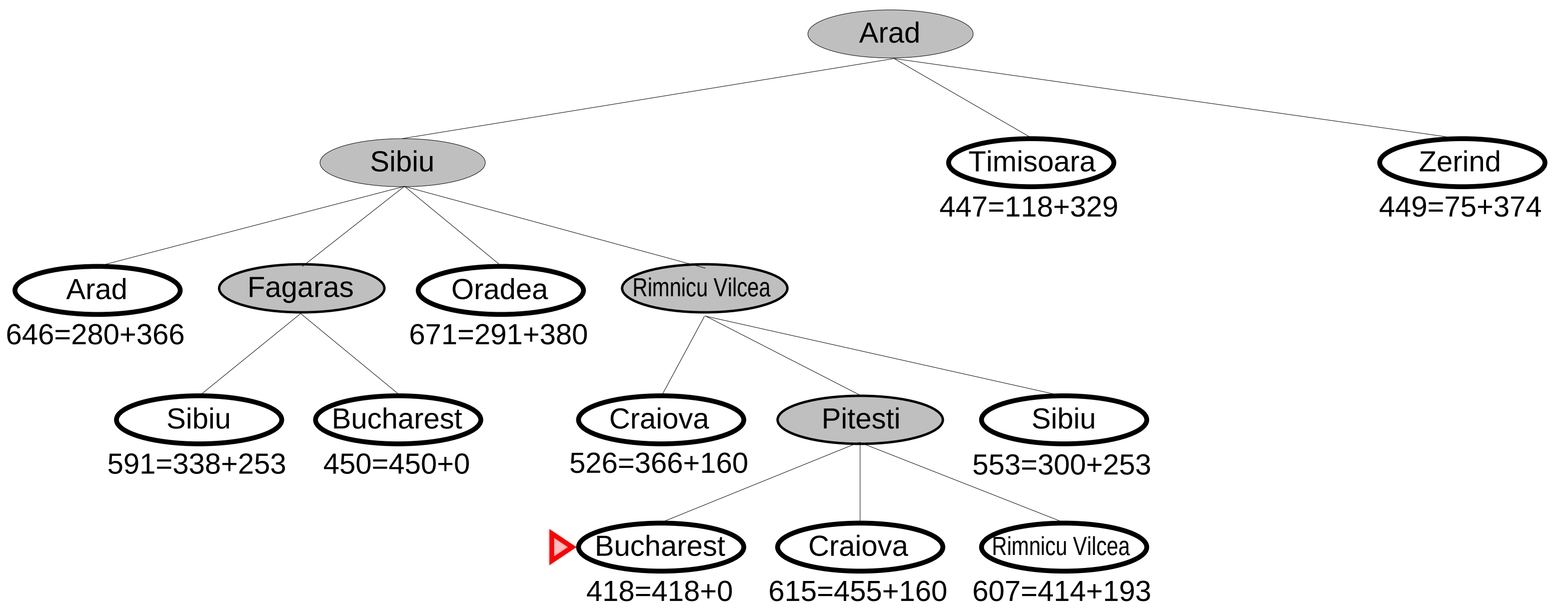
A* example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* example

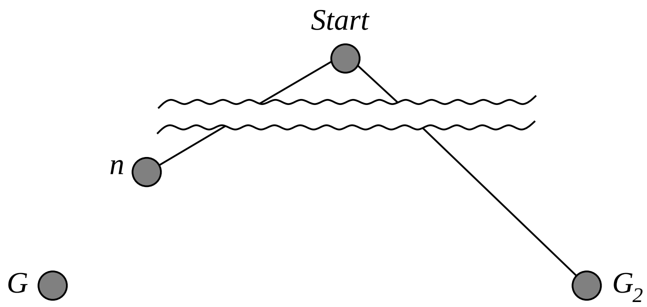

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Optimality of A* (standard proof)

- Suppose some suboptimal goal node G_2 is in the queue and let n be an unexpanded node on a shortest path to an optimal goal G

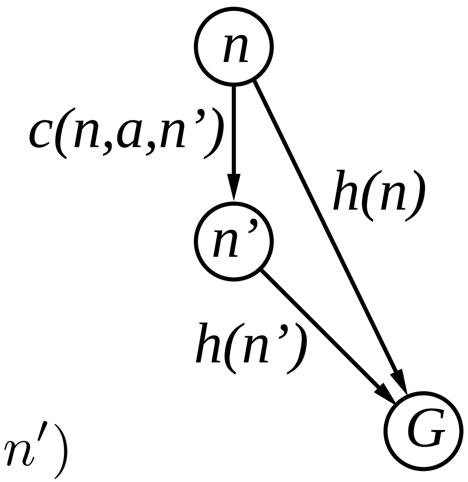


$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

- Since $f(G_2) > f(n)$, A* will never select G_2 for expansion
- Note: doesn't work for *graph search* because it can discard optimum path to a repeated state

Consistent heuristics

- A heuristic is **consistent** if
  $h(n) \leq c(n, a, n') + h(n')$



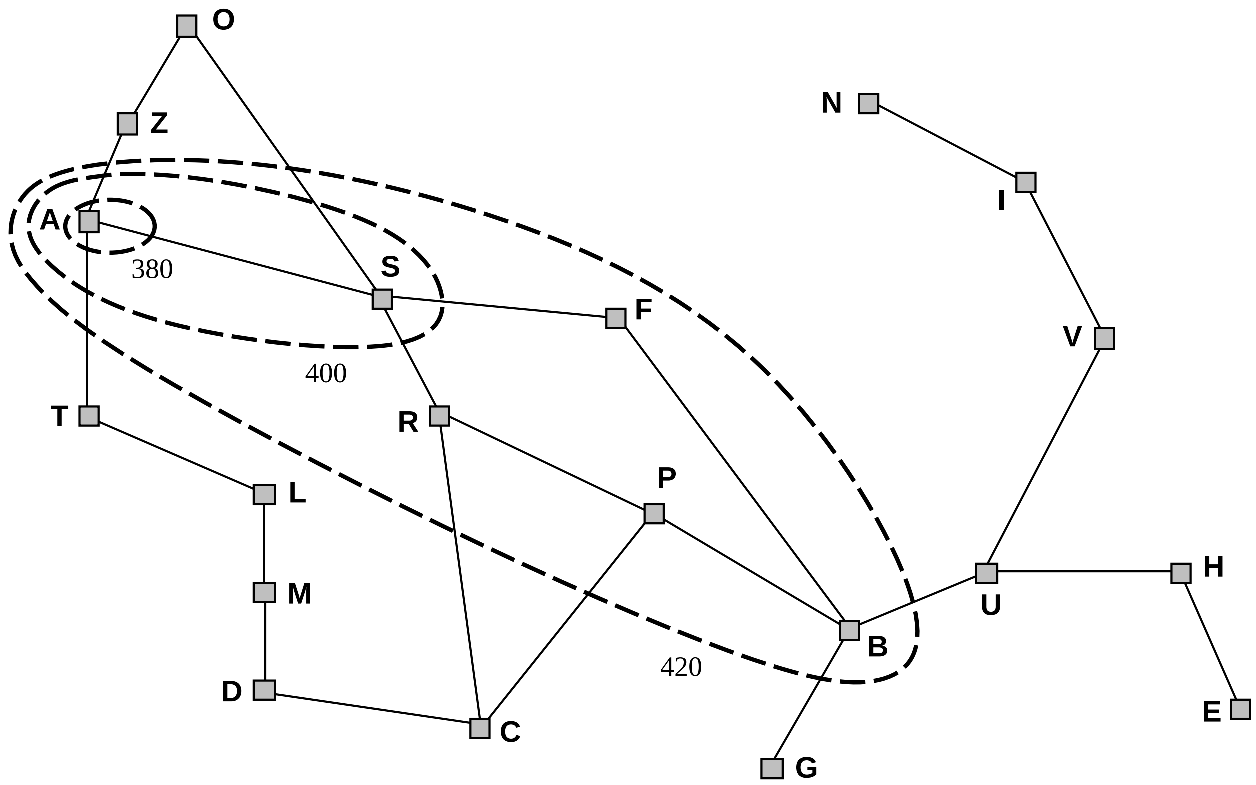
- If h is consistent, then

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

- This means $f(n)$ is nondecreasing along any path
- Hard to find: inconsistent admissible heuristics

Contours for A*

- If heuristic consistent then A* adds "f-contours" of nodes (similar to how breadth-first adds layers)
  - Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Properties of A* search

- Complete?
- Time?
- Space?
- Optimal?

Properties of A* search

- Complete?
  - Yes, unless there are infinitely many nodes with $f \leq C^*$
- Time?
- Space?
- Optimal?

Properties of A* search

- Complete?
  - Yes, unless there are infinitely many nodes with $f \leq C^*$
- Time?
  - Exponential unless $|h(n) - h^*(n)| \leq O(\log h^*(n))$, where $h^*(n)$ is the true cost of getting from n to the goal
  - I.e. unless error doesn't grow faster than log of path cost
  - This is not the case for most heuristics in practical use
- Space?
- Optimal?

Properties of A* search

- Complete?
  - Yes, unless there are infinitely many nodes with $f \leq C^*$
- Time?
  - Exponential unless $|h(n) - h^*(n)| \leq O(\log h^*(n))$, where $h^*(n)$ is the true cost of getting from n to the goal
  - I.e. unless error doesn't grow faster than log of path cost
  - This is not the case for most heuristics in practical use
- Space?
  - Has to keep all nodes in memory
  - Expands all nodes with $f(n) < C^*$, some nodes with $f(n) = C^*$, and no nodes with $f(n) > C^*$
- Optimal?

Properties of A* search

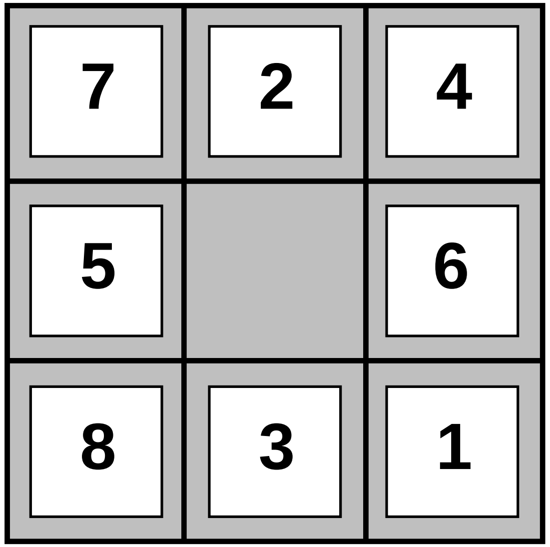
- Complete?
  - Yes, unless there are infinitely many nodes with $f \leq C^*$
- Time?
  - Exponential unless $|h(n) - h^*(n)| \leq O(\log h^*(n))$, where $h^*(n)$ is the true cost of getting from n to the goal
  - I.e. unless error doesn't grow faster than log of path cost
  - This is not the case for most heuristics in practical use
- Space?
  - Has to keep all nodes in memory
  - Expands all nodes with $f(n) < C^*$, some nodes with $f(n) = C^*$, and no nodes with $f(n) > C^*$
- Optimal? Yes

Memory-bounded heuristic search

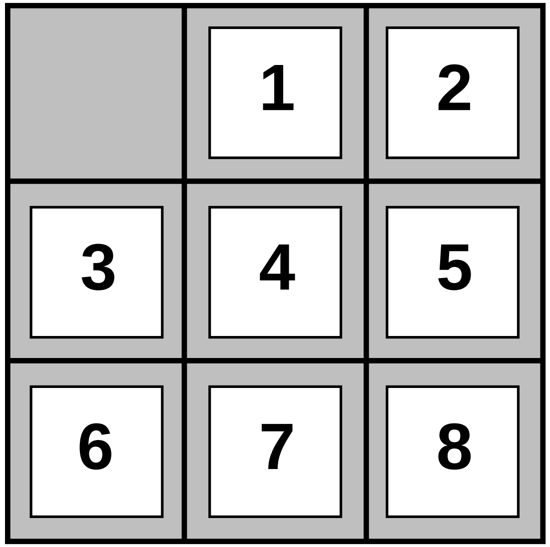
- **SMA*** (simplified memory-bounded A*)
  - Proceeds just like A* until memory is full
  - If memory is full, it drops the *worst* node (the one with the highest f-value) and backs up its value to its parent
  - I.e. when all descendants of a node are forgotten, we still have an idea how worthwhile it is to expand the node
  - A subtree is regenerated only when *all other paths* have been shown to be worse than the forgotten path
- Complete if shallowest goal node is reachable with available memory
- Optimal if shallowest optimal goal node is reachable
- Other algorithms: **IDA*** and **RBFS**

Admissible heuristics

- E.g, for the 8-puzzle
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = total Manhattan distance



Start State

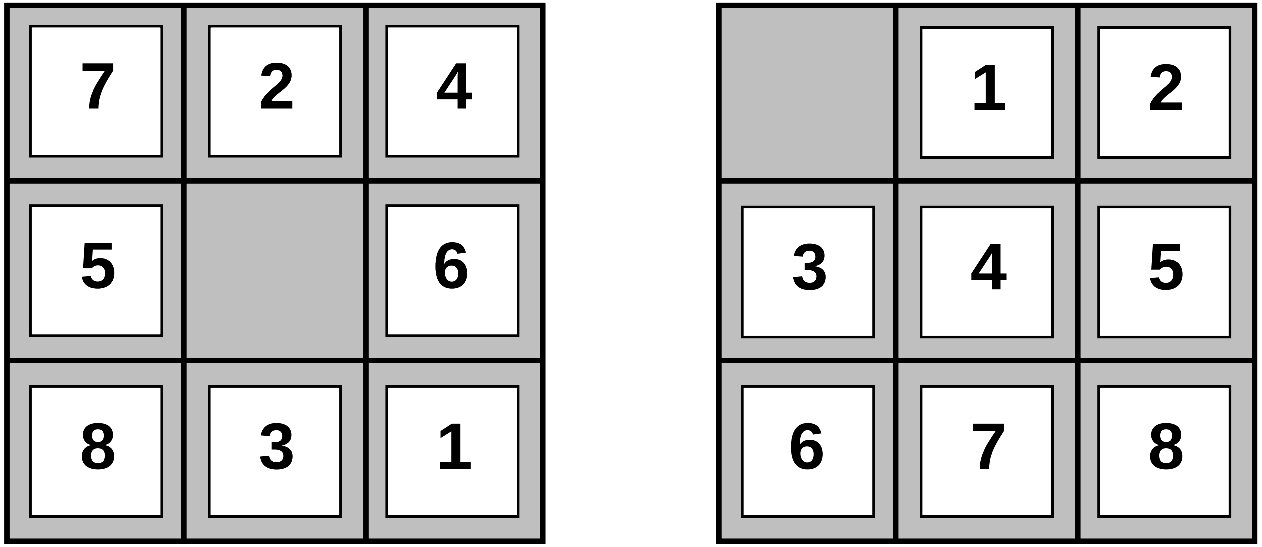


Goal State

  - $h_1(n) = ?$
  - $h_2(n) = ?$

Admissible heuristics

- E.g, for the 8-puzzle
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = total Manhattan distance



Start State **Goal State**

  - $h_1(n) = 8$
  - $h_2(n) = 3{+}1{+}2{+}2{+}2{+}3{+}3{+}2 = 18$

Dominance

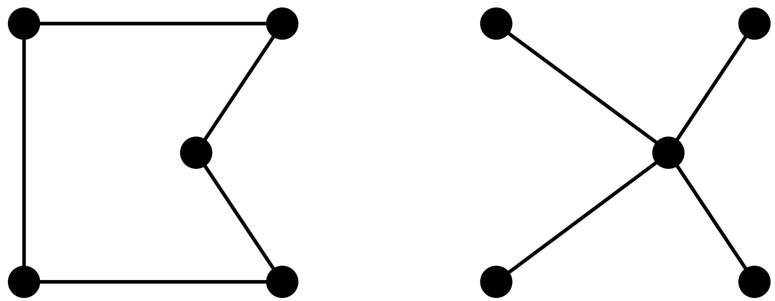
- If $h_2(n) \geq h_1(n)$ for all n (and both admissible!) then h_2 **dominates** h_1 and is better for search
- Typical search costs: $d = 14$
  - IDS = 3,473,941 nodes
  - $A^*(h_1) = 539$ nodes
  - $A^*(h_2) = 113$ nodes
- Typical search costs: $d = 24$
  - IDS $\approx$ 54,000,000,000 nodes
  - $A^*(h_1) = 39{,}135$ nodes
  - $A^*(h_2) = 1{,}641$ nodes

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, the $h_1(n)$ 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 cost of the real problem

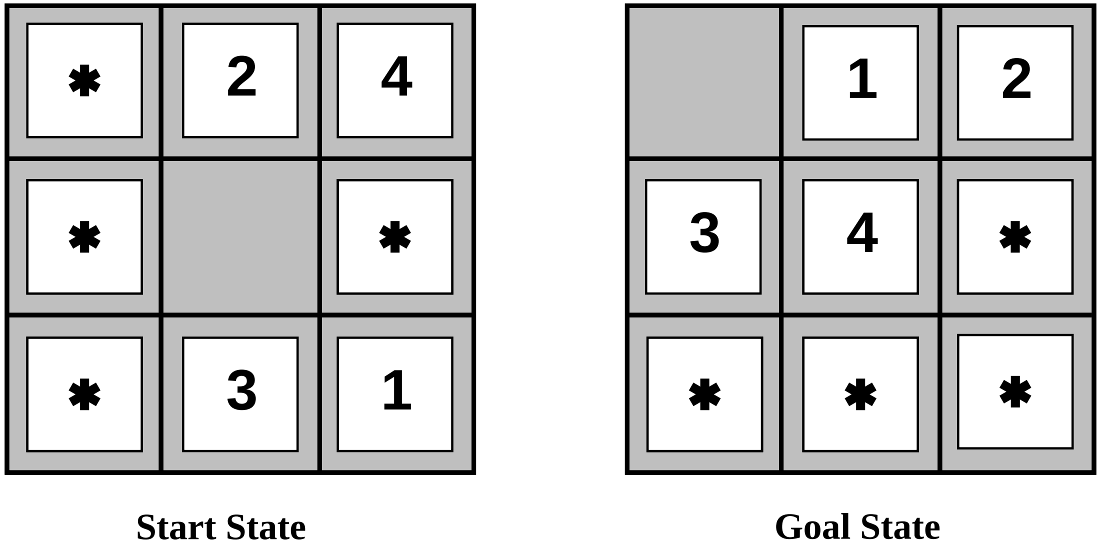
More on relaxed problems

- Admissible heuristic for traveling salesman problem: sum of costs for minimum spanning tree
- Lower bound on the shortest TS tour
- Minimum spanning tree can be computed in $O(n^2)$



Pattern databases

- Idea: store exact solution costs for subproblem instances
  - Optimum solution cost of subproblem is lower bound on optimum solution cost of complete problem



- Works really well with **disjoint patterns** where problem can be divided up so that each move only affects one subproblem
  - Then we can just add the costs for the subproblems!

Learning heuristics from experience

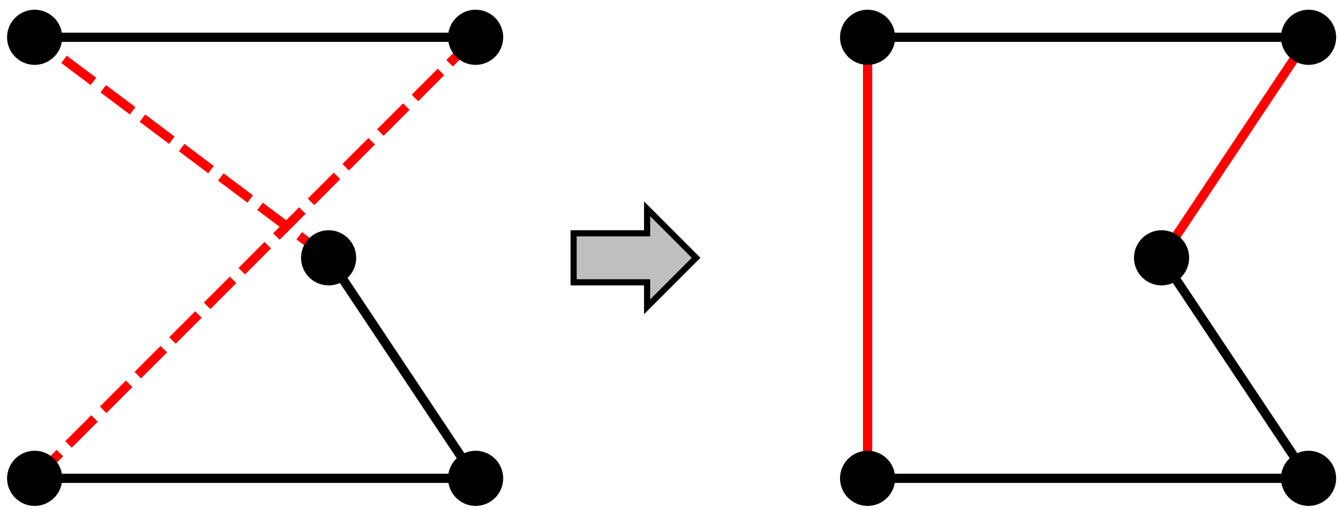
- **Inductive learning algorithms** can be used to learn a heuristic function given some **training examples**
  - Each example consists of a state from the solution path and the actual cost of the solution from that point
  - Learning algorithms: neural nets, decision tree learners, etc.
- Each example needs to be described by **features** of the state that are relevant to its evaluation
  - E.g.: "number of misplaced tiles" or "number of pairs of adjacent tiles that are also adjacent in goal state"
- Example: heuristic function could be linear combination of features values, i.e. $h(n) = c_1 * x_1(n) + c_2 * x_2(n)$

Local search algorithms

- In many search and **optimization problems**, the path is irrelevant, and we are only interested in the goal state
- **Local search** algorithms operate by maintaining a single **current state**
  - Requires only constant space
  - Usually based on a **complete state formulation** where every state is a *potential* solution

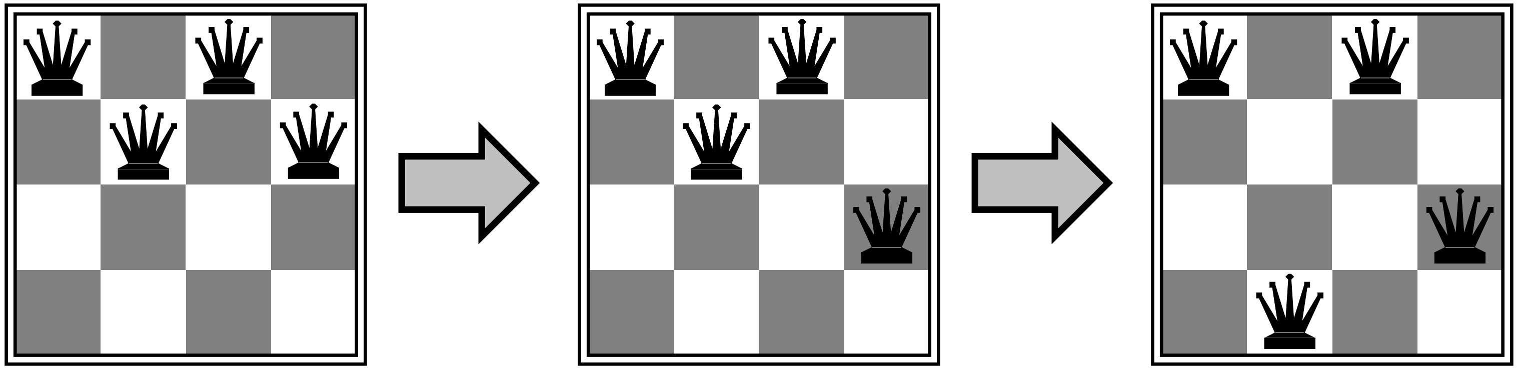
Example: traveling salesman problem

- Start with any configuration, perform pairwise changes

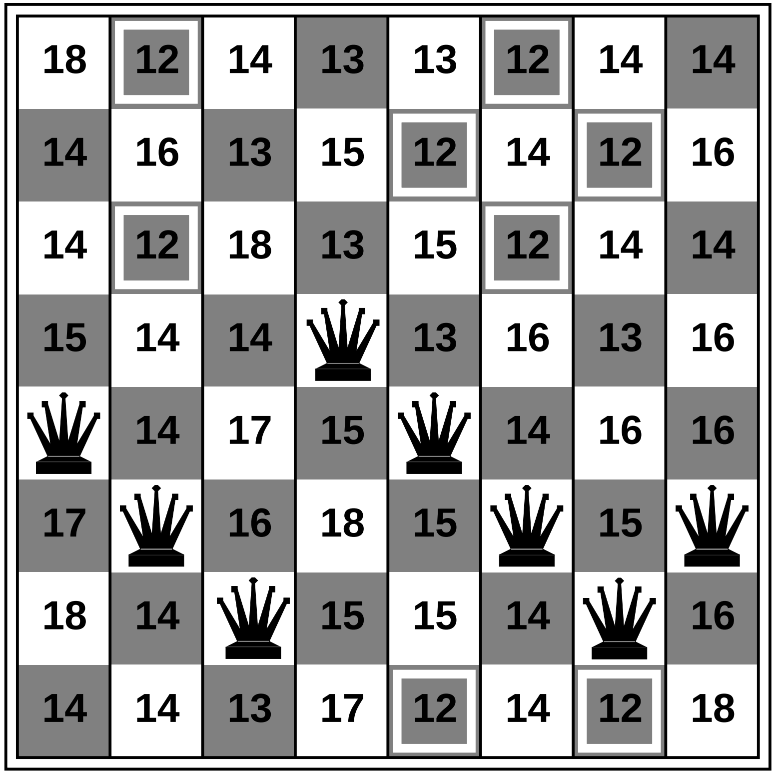


Example: n-queens

- Put n queens on $n \times n$ board with no two queens on the same row, column, or diagonal
- Move a queen to reduce the number of conflicts

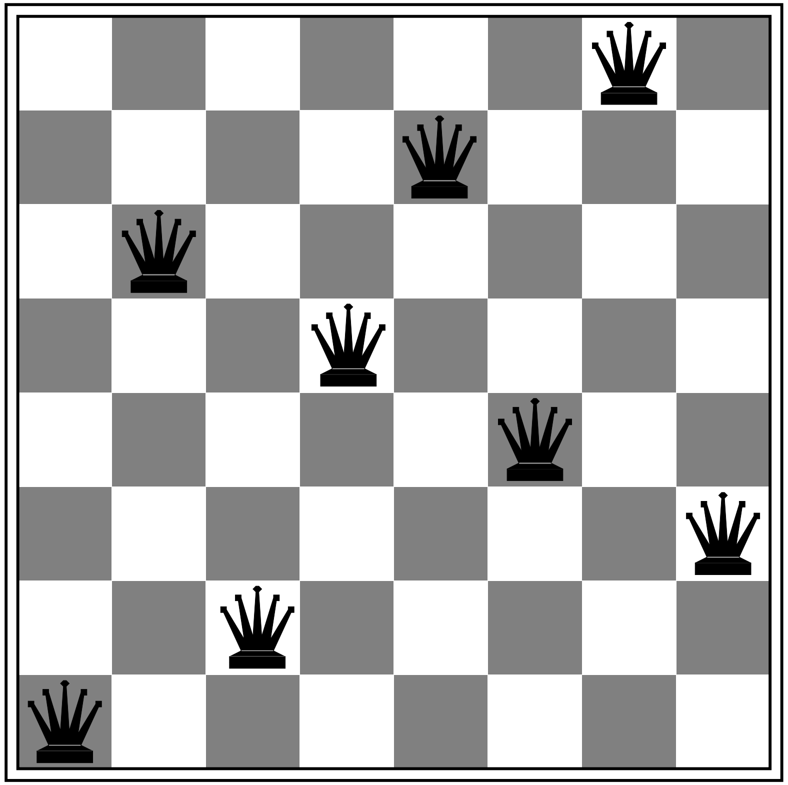


Example: n-queens successors



- A state with heuristic cost estimate 17 and the costs for its successors
- Cost = number of pairs of queens that are attacking each other

Example: n-queens local minimum



- A state with cost 1 and no escape route

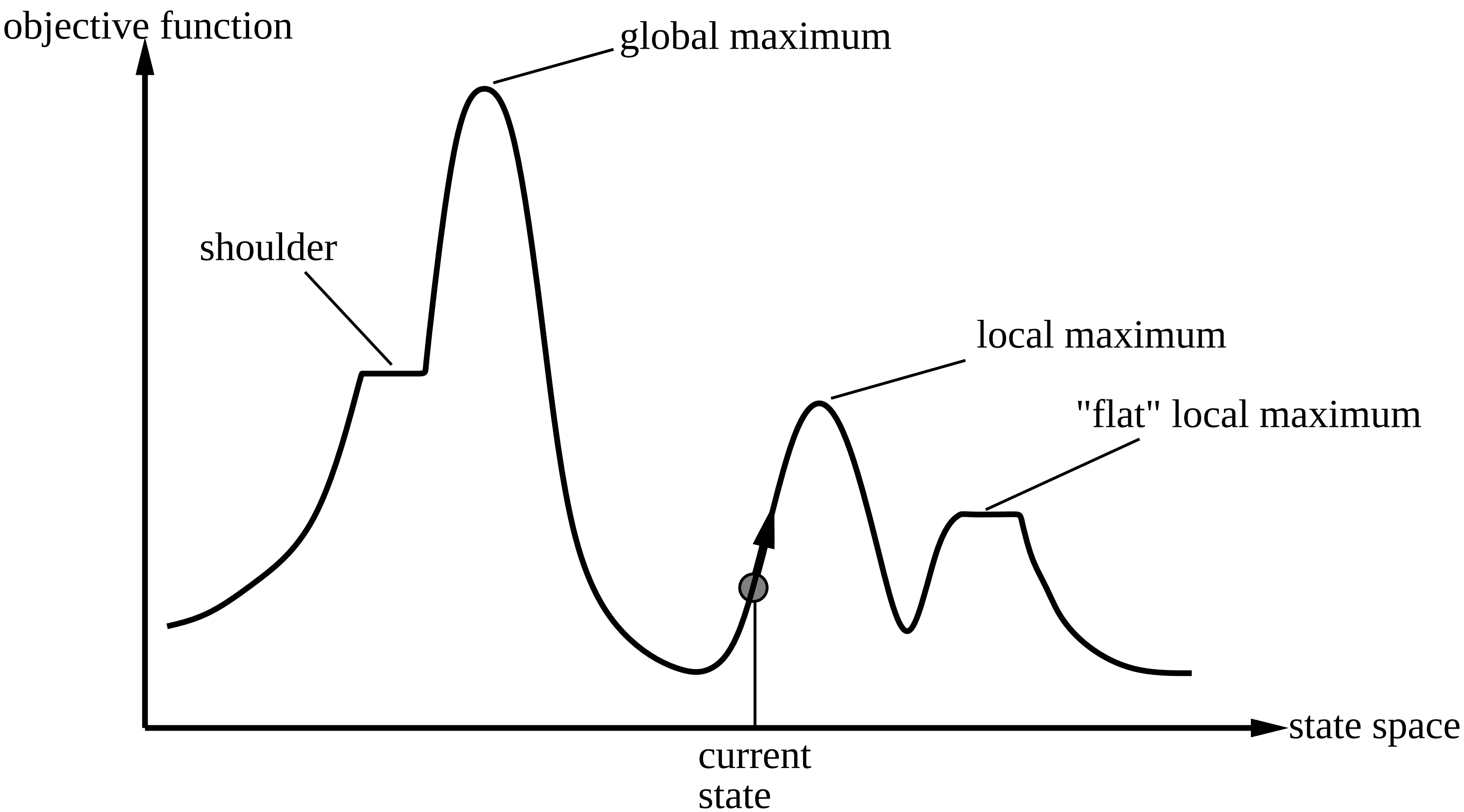
Hill-climbing

- "Like climbing Everest in thick fog with amnesia"

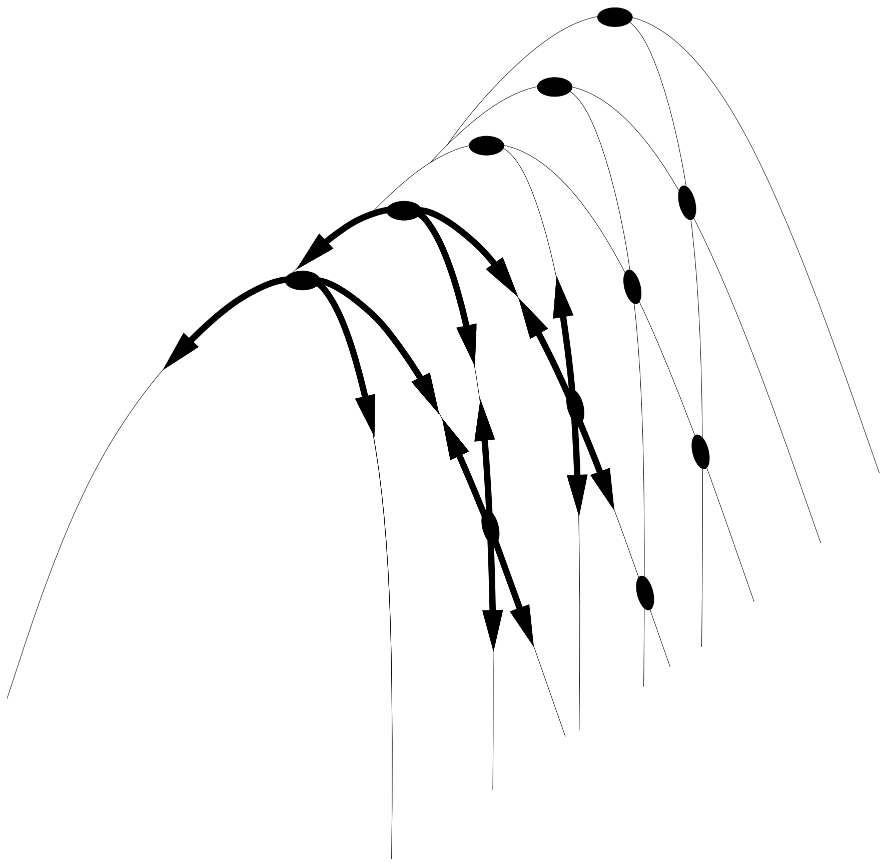
```
function Hill-Climbing( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   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
```

Objective function



Ridge



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^(Δ E/T)
```

Properties of simulated annealing

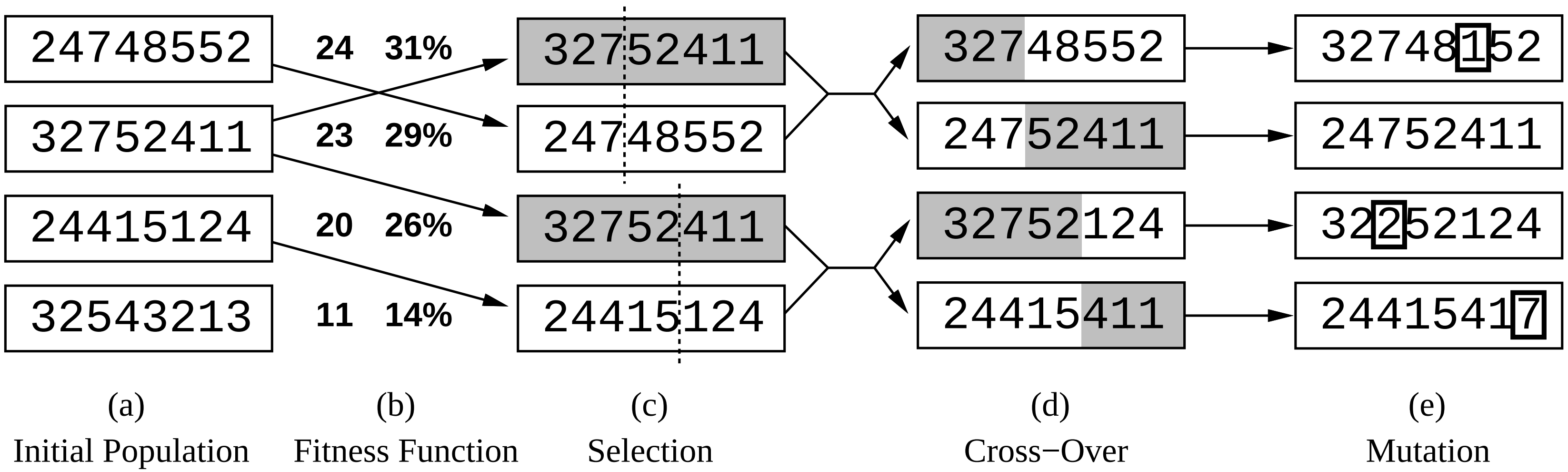
- In metallurgy **annealing** is the process used to temper or harden metals and glass
  - Material is heated to a high temperature and then gradually cooled down
- It can be shown that simulated annealing reaches the best state if the "temperature" T decreases slowly enough
  - Is this an interesting guarantee?
- Devised by Metropolis *et al.*, 1953, for physical process modeling
- Has been used for VLSI layout, airline scheduling, and other large optimization tasks

Local beam search

- Idea: keep track of k states instead of only one (as in hill-climbing search)
  - Begins with k randomly generated states
  - At each step, all successors of all k states are generated
  - If one of them is a goal, the algorithm halts
  - Otherwise, the k best are selected, the rest discarded, and the algorithm repeats
- **Stochastic beam search** chooses k successors at random, with selection probability being increasing function of node's value
  - Can help preventing premature convergence
  - Similar to process of natural selection

Genetic algorithms

- Variant of stochastic beam search where states are generated by combining two parents (instead of modifying a single state)
  - Starts with k random states (**population**)
  - Each state (or **individual**) is represented as a string over a finite alphabet
  - **Mutation** operator is applied after offspring has been generated from selected parents using **crossover**



Example crossover

- First two parents and first offspring from previous slide