



Solving problems by searching

Informed or Heuristic Search

Outline

- Search strategy
- Best-first search
 - Greedy best-first search
 - A^* search
- Local search algorithm or Hill-climbing search
 - Simulated annealing search
 - Genetic algorithms

Stochastic algorithms

Search Strategy

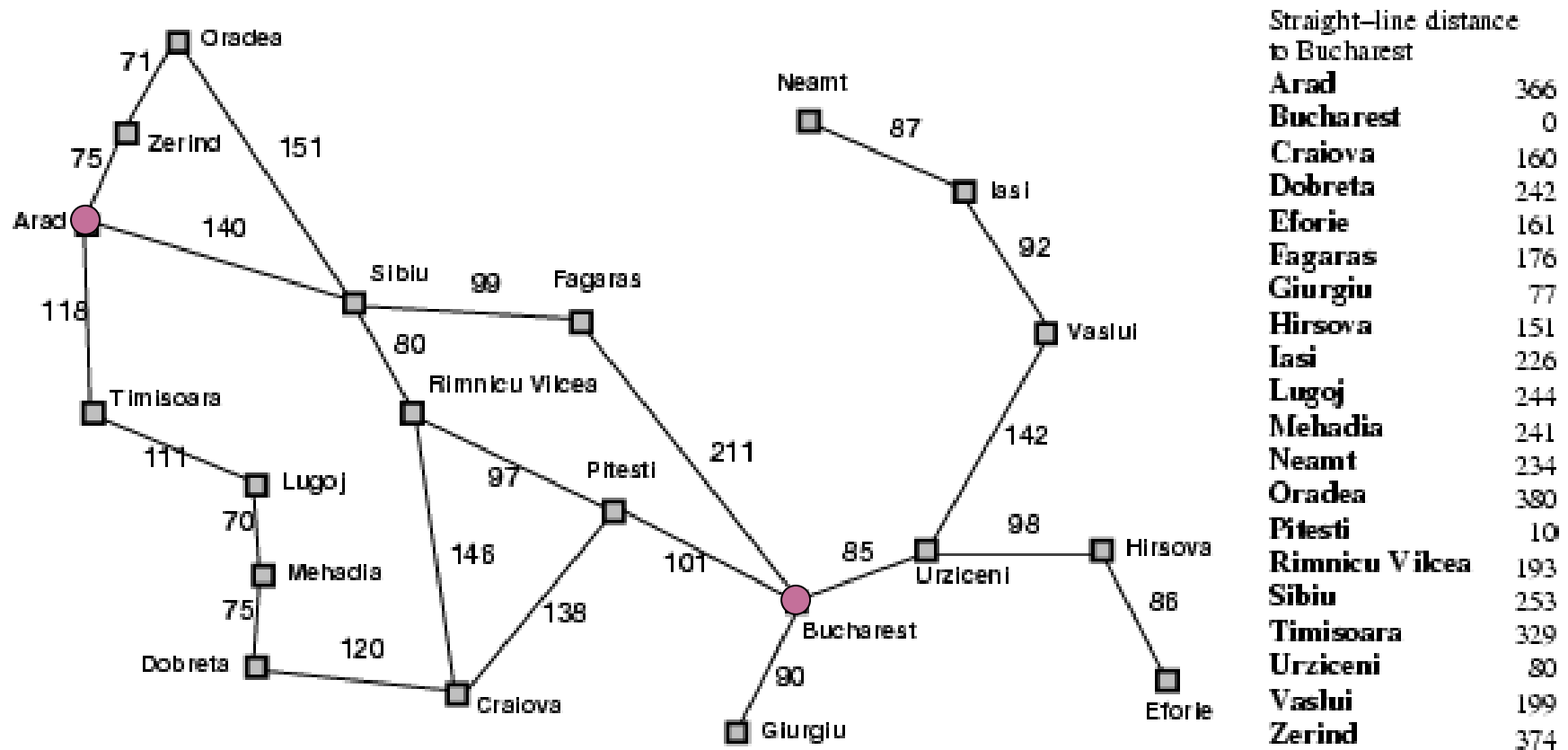
- ✓ **Uninformed search strategies** can find solutions to problems by systematically generating new states and testing them against the goal. *Unfortunately, these strategies are incredibly inefficient in most cases.*
- ✓ However, **informed search strategies** use problem specific knowledge can find more efficient solutions.
- ✓ A tree-search strategy is defined by picking the **order of node expansion.**
- ✓

Best-first search

- It is an instance of the general tree-search or graph-search algorithm in which a node is selected for expansion based on an evaluation function.
- Idea: use an evaluation function $f(n)$ for each node expansion
 - Estimate "desirability"
 -
 - Expand most desirable unexpanded node
 -
- Implementation:

Order the nodes in fringe in decreasing order of desirability based on $f(n)$. In this algorithm, there are different evaluation functions. Typically, it uses a heuristic function $h(n)$ that estimates the cost (the cheapest path cost from node n to a goal node) of a solution from n .
- Special cases:

Romania with step costs in km



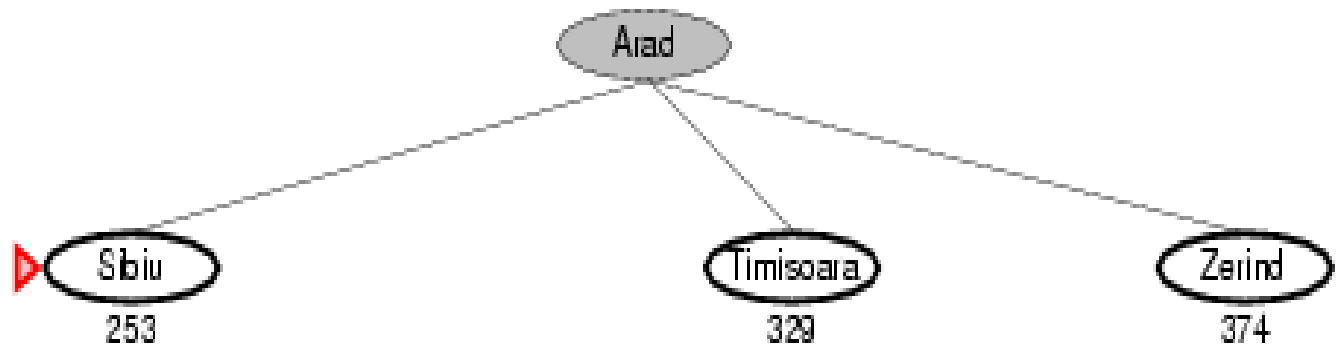
Greedy best-first search

- Greedy best-first search expands the node that appears to be closest to goal, on the grounds that this is likely to lead to a solution quickly.
- Thus, it evaluates nodes by using just the heuristic evaluation function $f(n) = h(n)$
= estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance (SLD) from n to goal (Bucharest)
- Notice that h_{SLD} can not be directly computed from the problem description, it takes certain amount of experience and is, therefore, a useful heuristic.
- It is not optimal, but is often efficient.
-

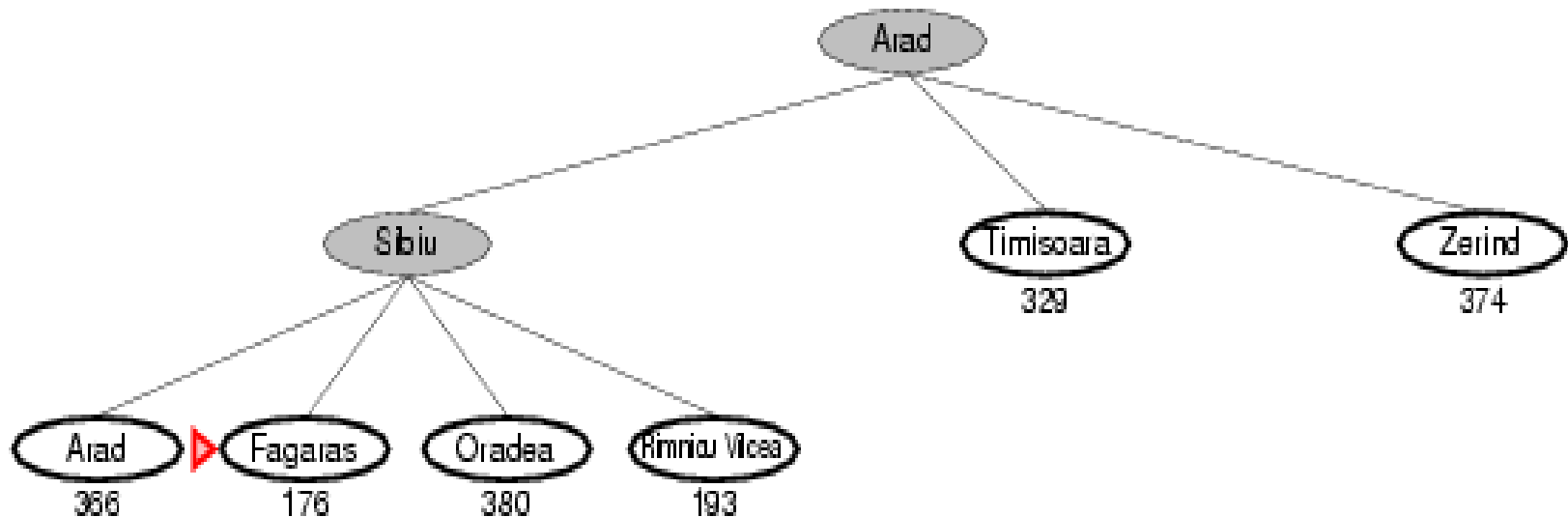
Greedy best-first search example



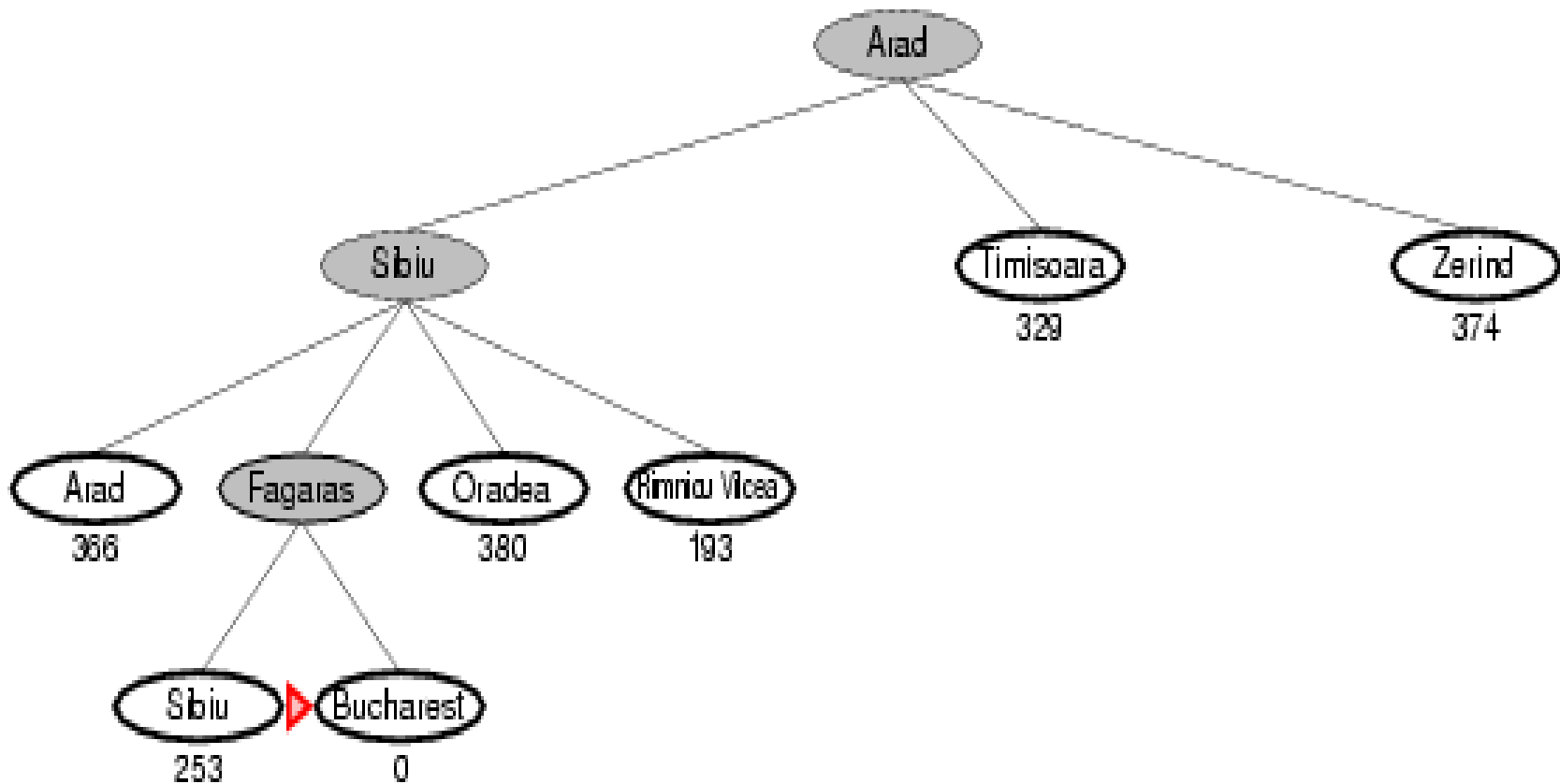
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
-
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
-
- Space? $O(b^m)$ -- keeps all nodes in memory
-
- Optimal? No

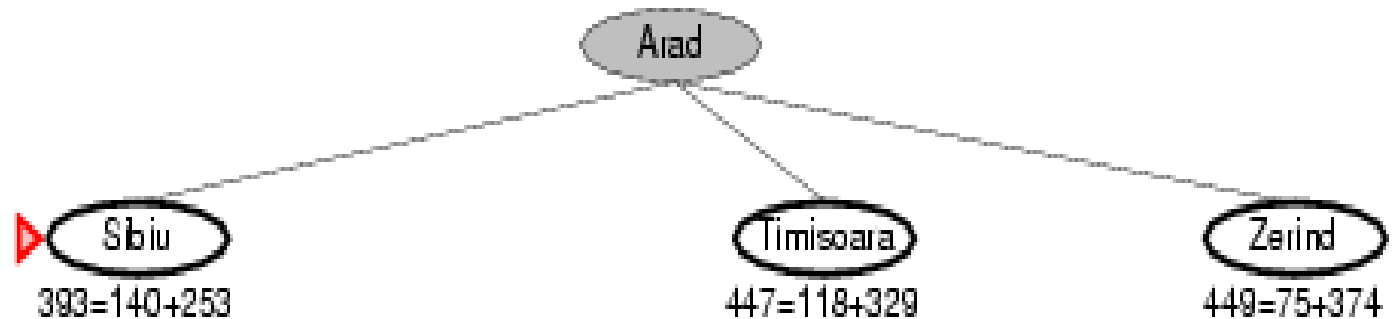
A* (A-star) search

- It is most widely-known form of best-first search.
-
- Evaluation function $f(n) = g(n) + h(n)$
- - $g(n)$ = cost so far to reach n
 - $h(n)$ = estimated cost from n to goal
 - $f(n)$ = estimated total cost of path through n to goal
- Expand nodes with minimal $f(n)$.
- The strategy is reasonable if A* is complete and optimal.
- A* is complete and optimal provided that we guarantee $h(n)$ is admissible (for tree-search) or consistent (for graph-search).

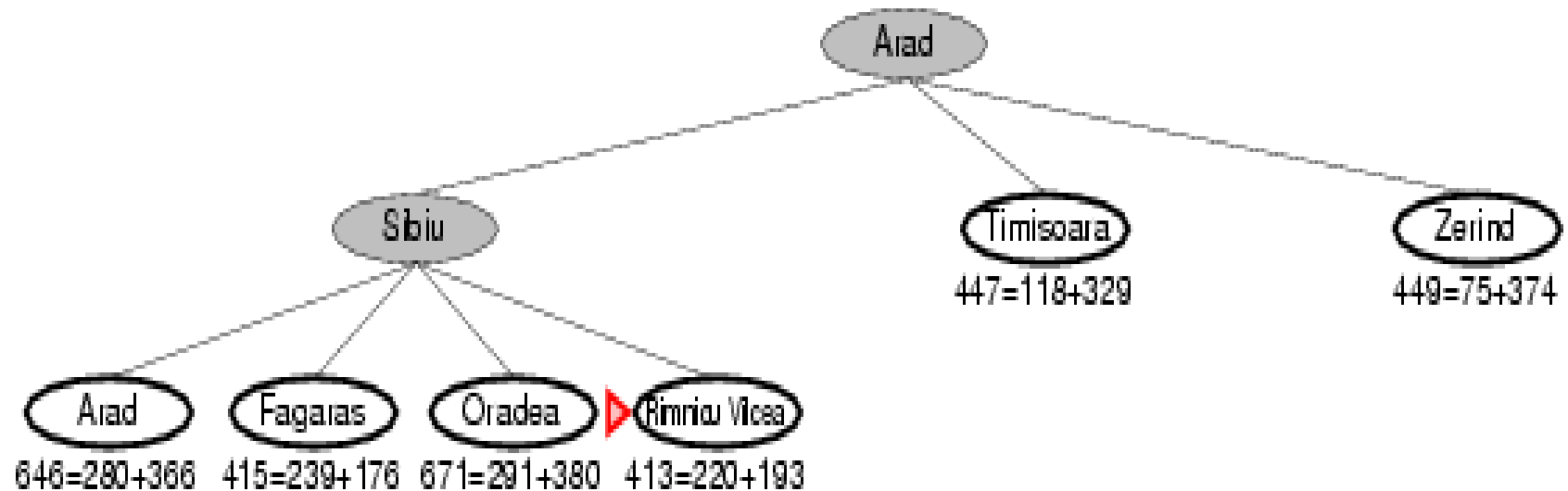
A* search example



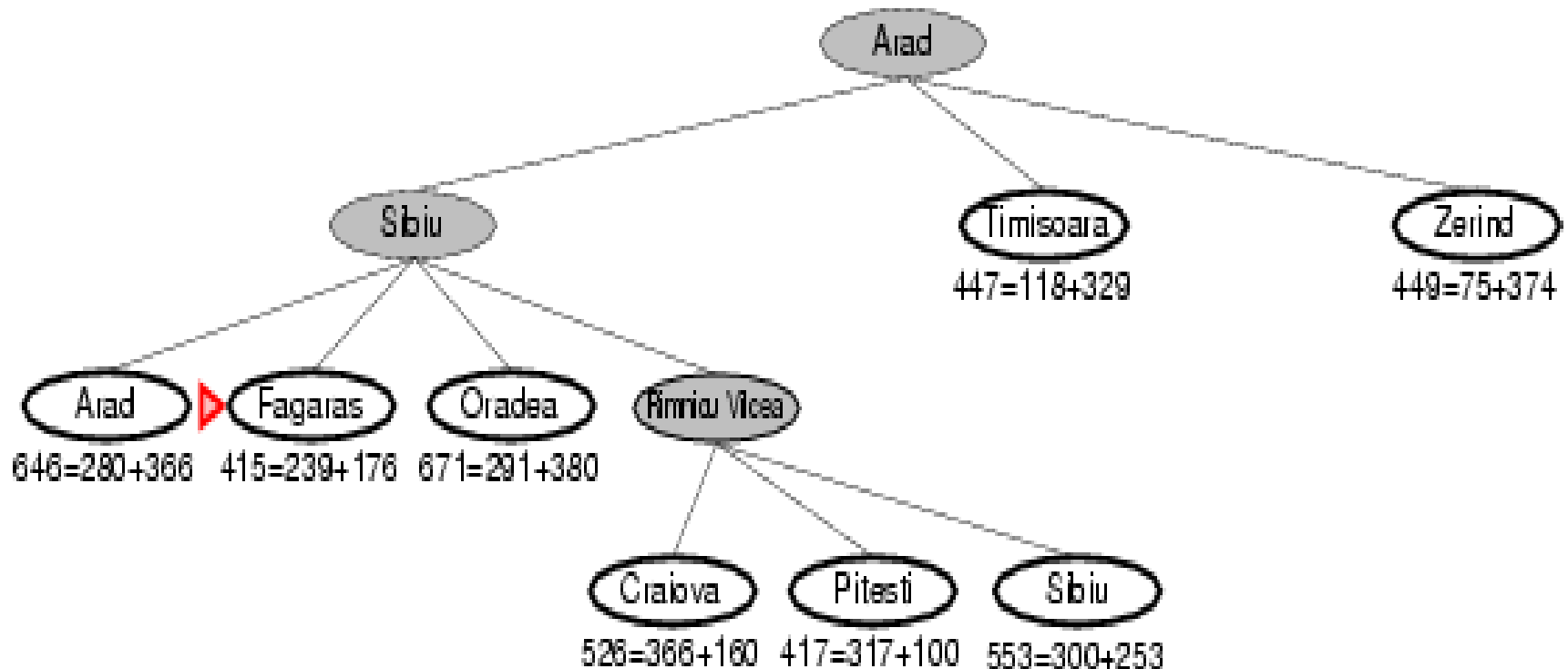
A* search example



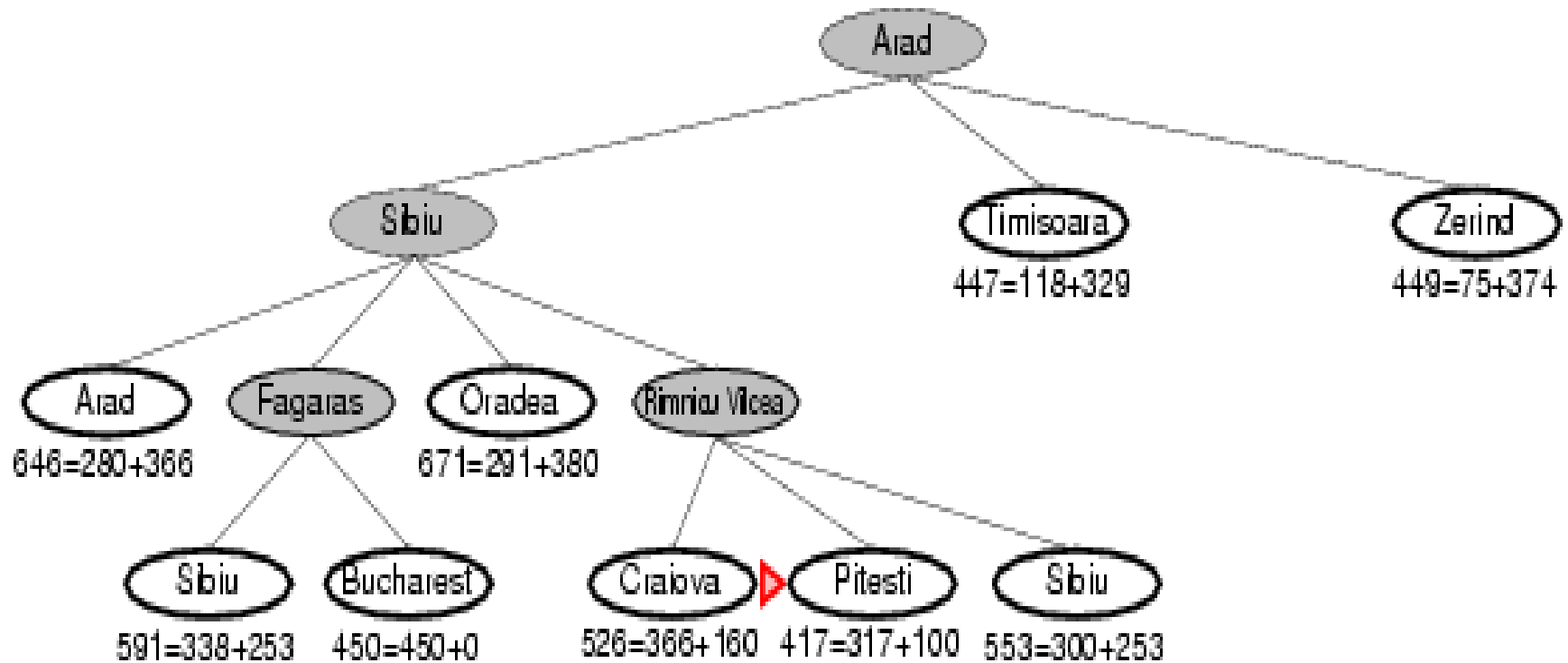
A* search example



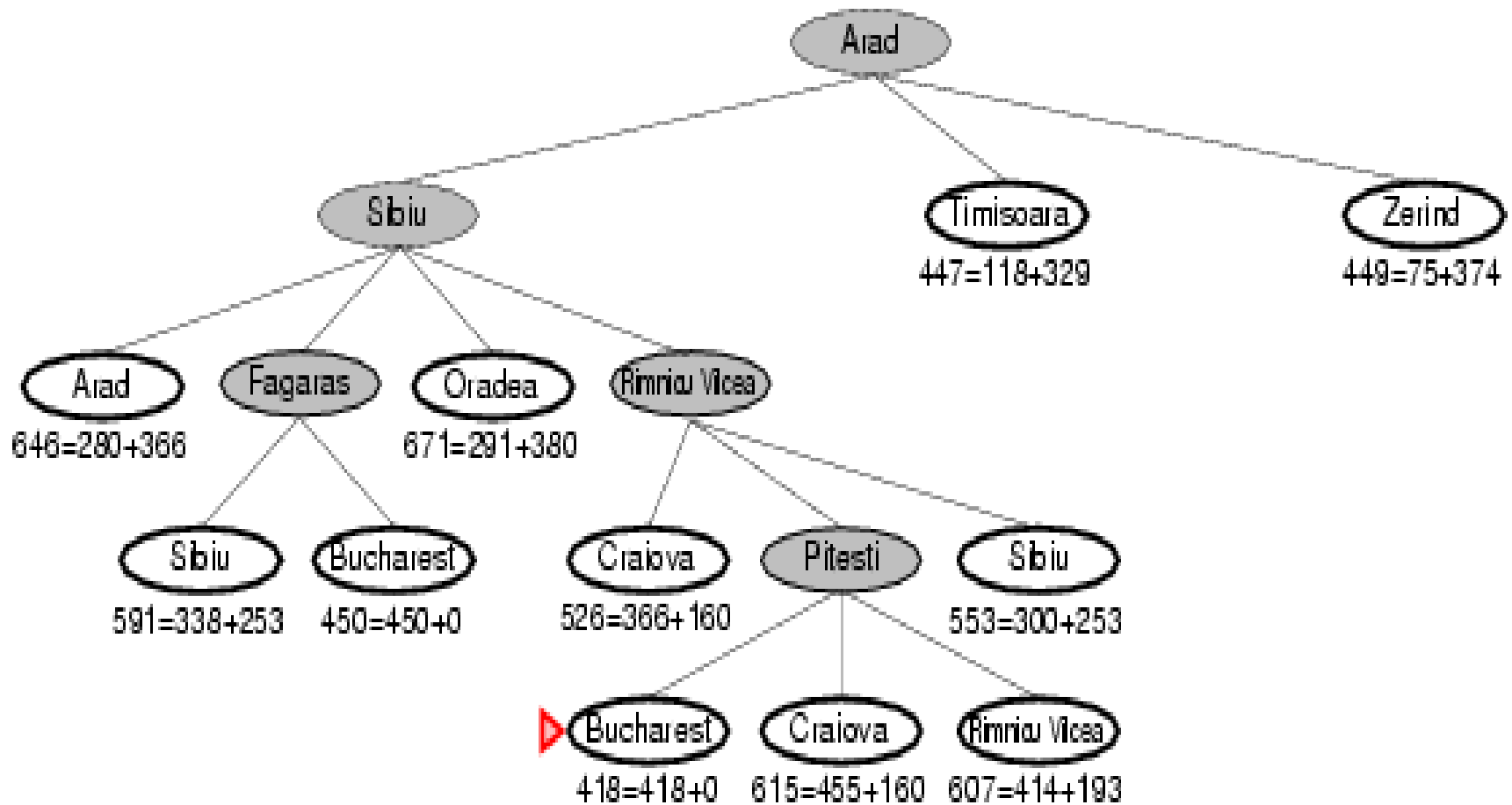
A* search example



A* search example



A* search example



Admissible heuristics

- Theorem: A^* is optimal if $h(n)$ is admissible heuristic, which means $h(n)$ never overestimates the cost to reach the goal.
- More clearly, an admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- A heuristic $h(n)$ is admissible if for every node n , $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n .
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance).
-

Properties of A*

- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
-
- Time? Exponential
-
- Space? Keeps all nodes in memory
-
- Optimal? Yes
-

Admissible heuristics

For the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance (city block distance), which indicates the sum of distances of the tiles from their goal positions.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

■ $h_1(S) = ?$

■ $h_2(S) = ?$

Admissible heuristics

- $h_1(n)$ is an admissible heuristic, because it is clear that any tile that is out of place must be moved at least once.
- $h_2(n)$ is also admissible, because any move can do one step closer to the goal.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ $3+1+2+2+2+3+3+2 = 18$
- Neither of these $h_1(n)$ and $h_2(n)$ overestimates the true solution cost, which is 26.

Relaxed problems

- A problem with fewer restrictions on the actions is called a **relaxed problem**
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original 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
-

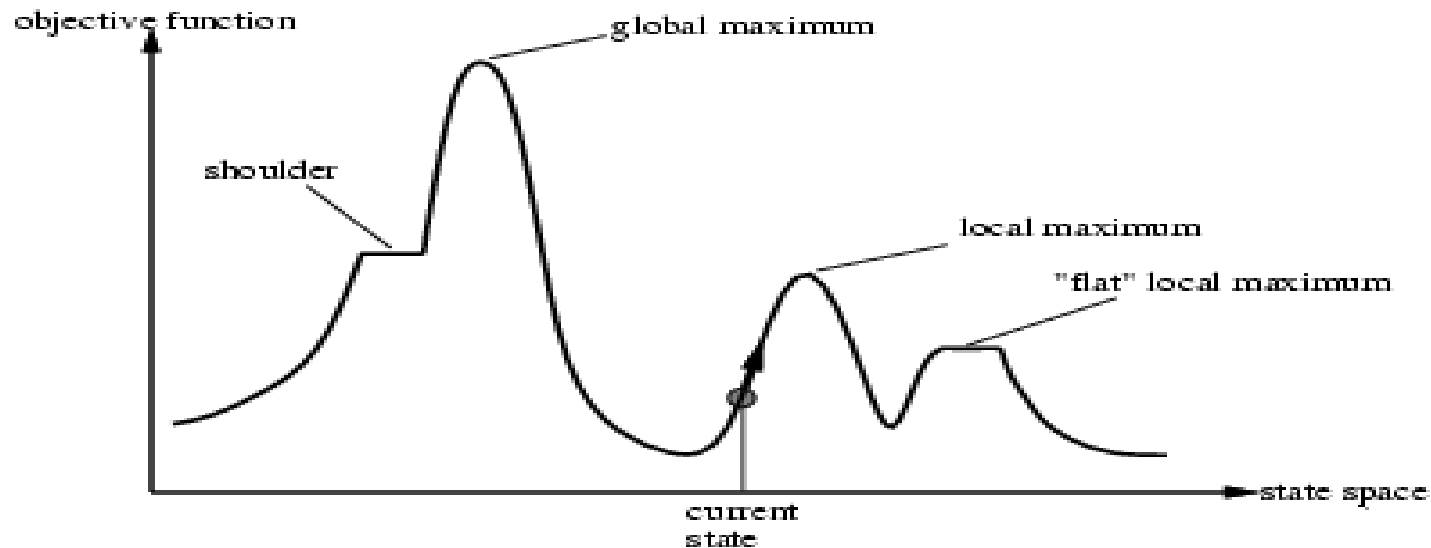
Local search algorithms and optimization problems

- In many optimization problems, the **path** to the goal is irrelevant; the goal state itself is the solution. For example, in the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.
-
- This class of problems includes many important applications such as IC design, factory-floor layout, telecommunications network optimization, vehicle routing, and portfolio management.
- **Local search algorithms** operate using a single current state (rather than multiple paths) and generally move only to neighbors of that state. Typically, the paths followed by the search are not retained.
- Though **local search algorithms** are not systematic, however, it has two major advantages:
 - Use very little memory – usually a constant amount.
 - They often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are not suitable.

Local search algorithms and optimization problems

- In addition to finding goals, local search algorithms are useful for solving optimization problems, in which the aim is to find the best state according to an objective function.
- For example, nature provides an objective function – reproductive fitness. In this case, Darwinian evolution (genetic algorithm) could be seen as attempting to optimize, but there is no “goal test” and no “path cost” for this problem.

Local search algorithm



Consider state space landscape. A landscape has both “location” and “elevation”.

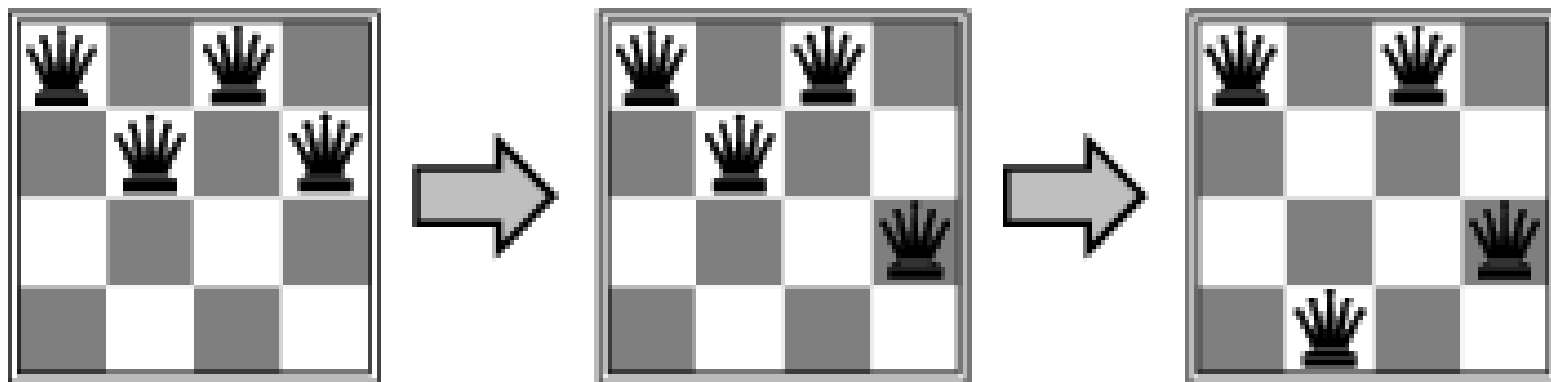
If elevation corresponds to cost function, then the aim is to find the lowest valley – a **global minimum**. If elevation corresponds to an objective function, the aim is to find the highest peak – a **global maximum**. Local search algorithms explore this landscape.

A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.

Example: n -queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal (that means no queen attacks any other; a queen attacks any piece in the same row, column or diagonal)

■



One queen per column.

Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

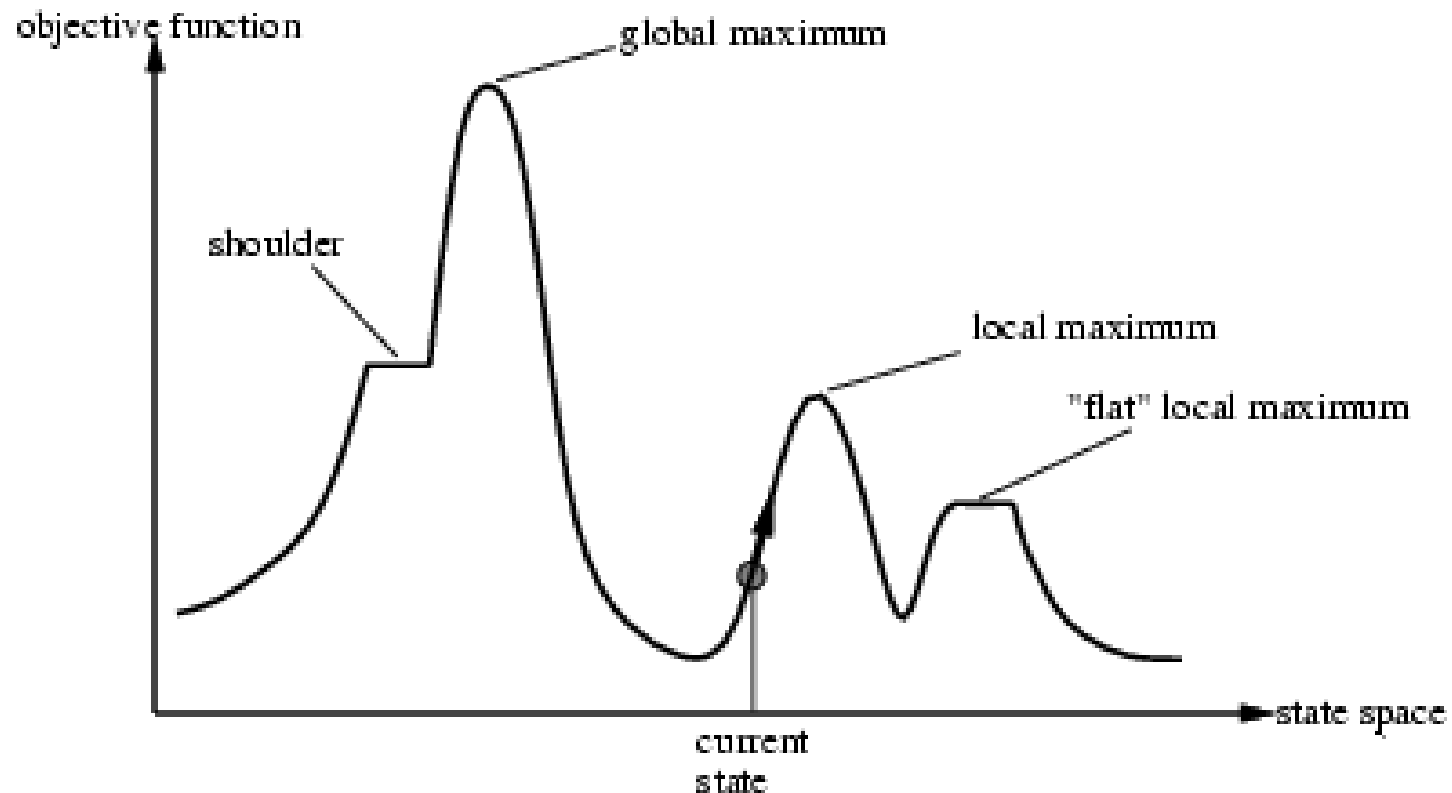
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node

  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
```

Amnesia: loss of memory









Hill-climbing search

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

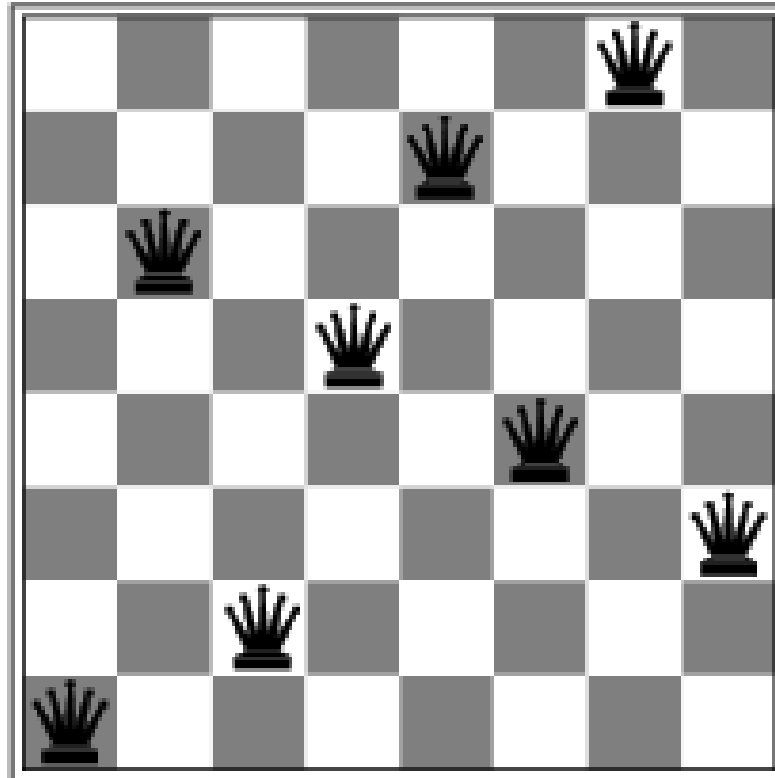


Hill-climbing search: 8-queens problem

- Heuristic cost function, h = number of pairs of queens that are attacking each other, either directly or indirectly.
- $h = 17$ for the above state
- The global minimum of this function is zero, which occurs only at perfect solutions.
- The values shows all of its successors, with best successors having $h = 12$

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$



Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but **gradually decrease** their frequency.
- Simulated annealing returns optimal solutions when given an appropriate cooling schedule.
-

```
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
     $\Delta E \leftarrow \text{VALUE}[\textit{next}] - \text{VALUE}[\textit{current}]$ 
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 
```


Properties of simulated annealing search

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.
- Widely used in VLSI layout, airline scheduling, etc.
-

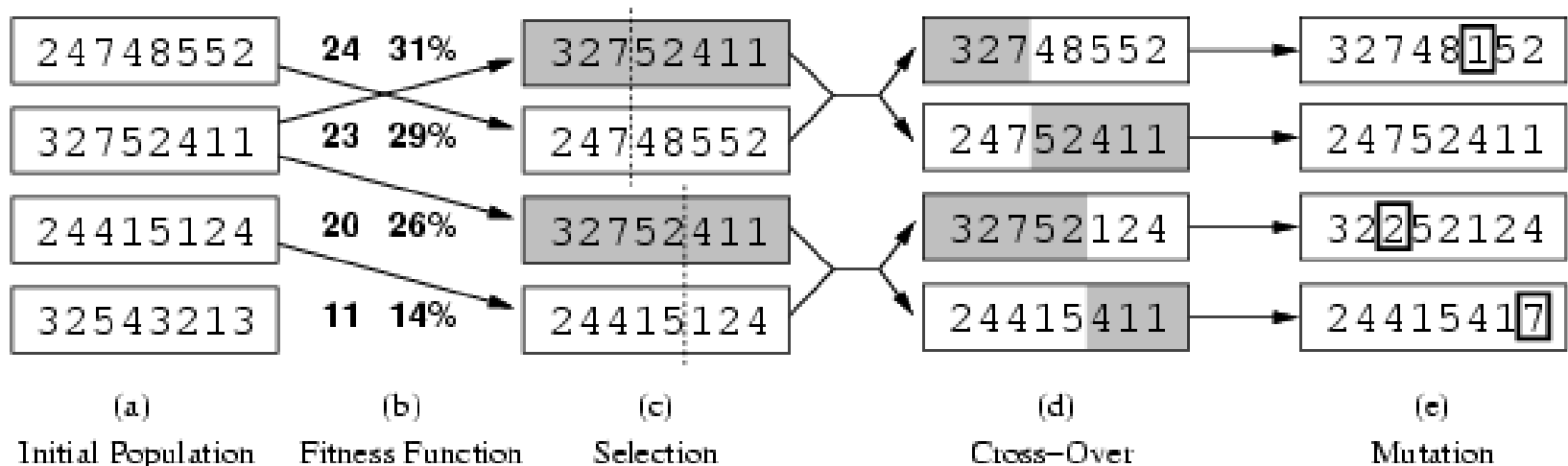
Local beam search

- Keep track of k states rather than just one
-
- Start with k randomly generated states
-
- At each iteration, all the successors of all k states are generated
-
- If any one is a goal state, stop; else select the k best successors from the complete list and repeat.

Genetic algorithms

- A successor state is generated by combining two parent states
-
- Start with k randomly generated states (**population**)
-
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
-
- Evaluation function (**fitness function**). Higher values for better states.
-

Genetic algorithms



- Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times 7/2 = 28$)

■

- $24/(24+23+20+11) = 31\%$

■

- $23/(24+23+20+11) = 29\%$ etc

Genetic algorithms

