Informed Search

Informed search strategies

- Use the problem-specific knowledge beyond the definition of the problem itself to find more efficient solutions that uninformed strategy
  - Best-first search
    - Greedy best-first search/Búsqueda voraz primero el mejor
    - A^*
  - Heuristics
  - Local search algorithms
    - Hill-climbing search/Búsqueda de escalada
    - Simulated annealing search/Búsqueda de temple simulado
    - Local beam search/Búsqueda de haz local
    - Genetic algorithms/Algoritmos genéticos
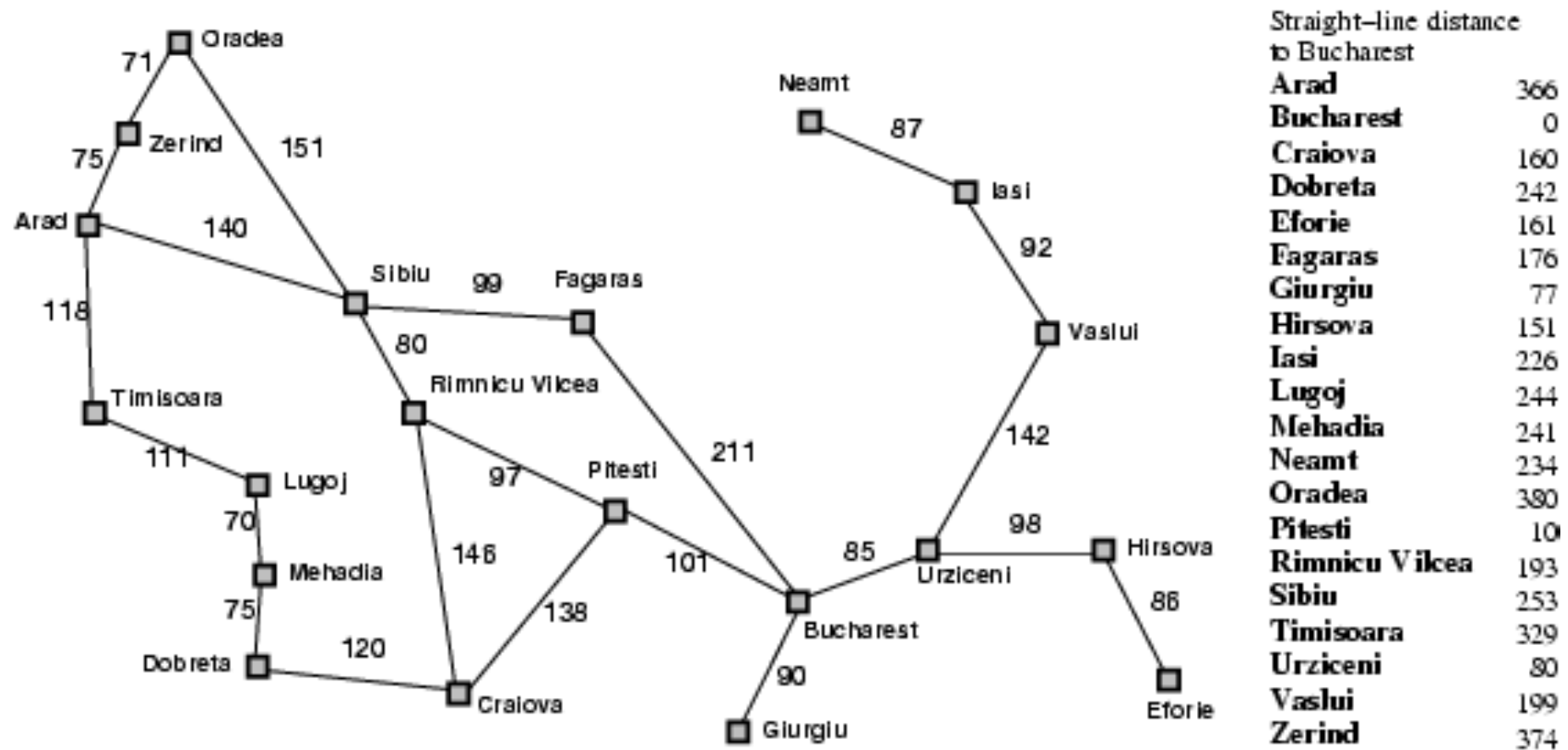
Best-first search

- ☐ Idea: use an evaluation function $f(n)$ for each node
  - ■ estimate of "desirability"
  - ■ Expand most desirable unexpanded node

- ☐ <u>Implementation</u>:

  Order the nodes in fringe in decreasing order of desirability

- ☐ Special cases:
  - ■ greedy best-first search
  - ■ A^* search

Romania with step costs in km

Informed search strategies

- ☐ Best-first search
  - ■ **Greedy best-first search**
  - ■ A^* search
- ☐ Heuristics
- ☐ Local search algorithms
  - ■ Hill-climbing search
  - ■ Simulated annealing search
  - ■ Local beam search
 - ■ Genetic algorithms

Greedy best-first search

- ☐ Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from n to *goal*

- ☐ Greedy best-first search expands the node that appears to be closest to the goal

- ☐ Implementation: as a priority queue to keep the fringe in ascending order of f-values

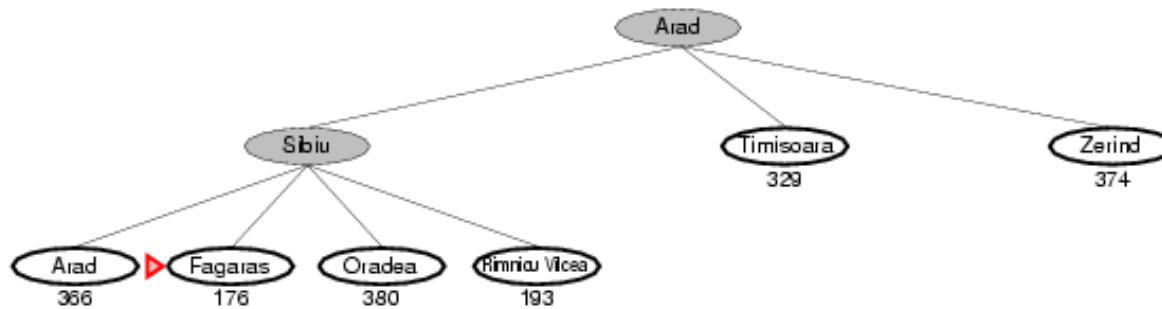- ☐ e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
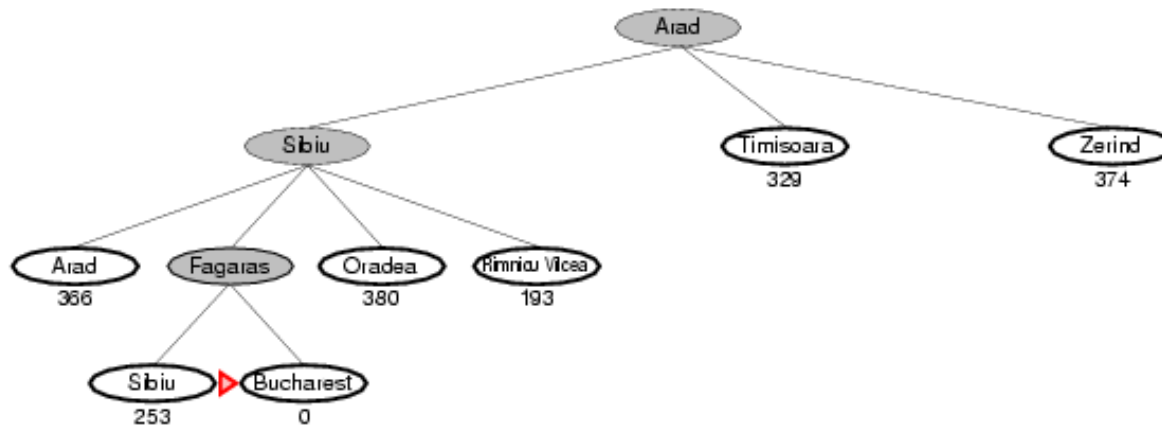
Greedy best-first search example

Arad
366

Greedy best-first search example
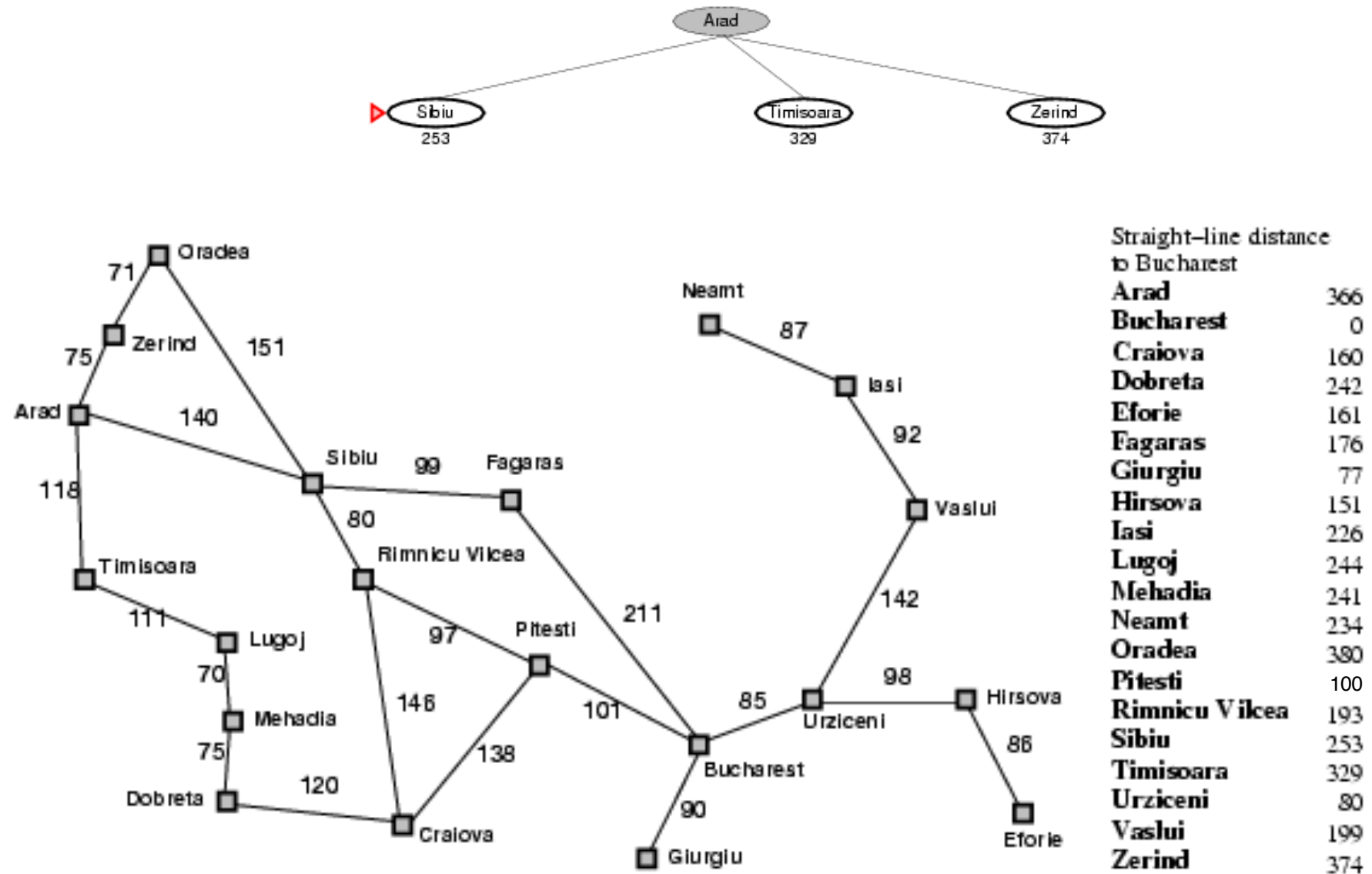
Greedy best-first search example

Greedy best-first search example

Properties of greedy best-first search

- ☐ <u>Complete?</u> No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →

- ☐ <u>Time?</u> $O(b^m)$, but a good heuristic can give dramatic improvement

- ☐ <u>Space?</u> $O(b^m)$ -- keeps all nodes in memory

- ☐ <u>Optimal?</u> No

Similar to depth-first search
Each state has b successors (branching factor)
d is the depth of the shallowest solution

Informed search strategies

- ☐ Best-first search
    - ■ Greedy best-first search
    - ■ **A* search**
- ☐ Heuristics
- ☐ Local search algorithms
    - ■ Hill-climbing search
    - ■ Simulated annealing search
    - ■ Local beam search
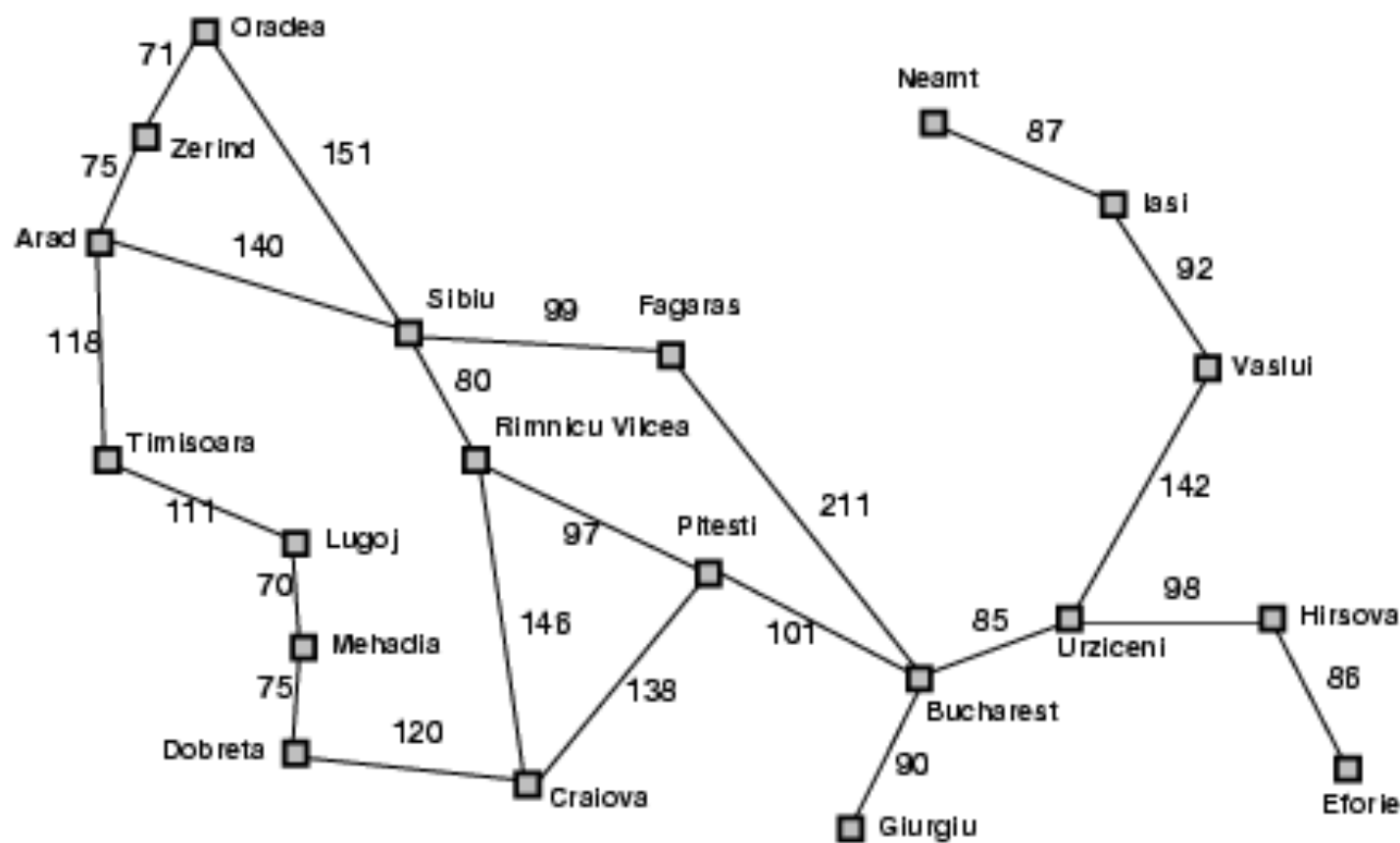   - ■ Genetic algorithms

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

- ☐ A * is optimal if h(n) is an admissible heuristic such that h(n) never overestimates the cost to reach the goal
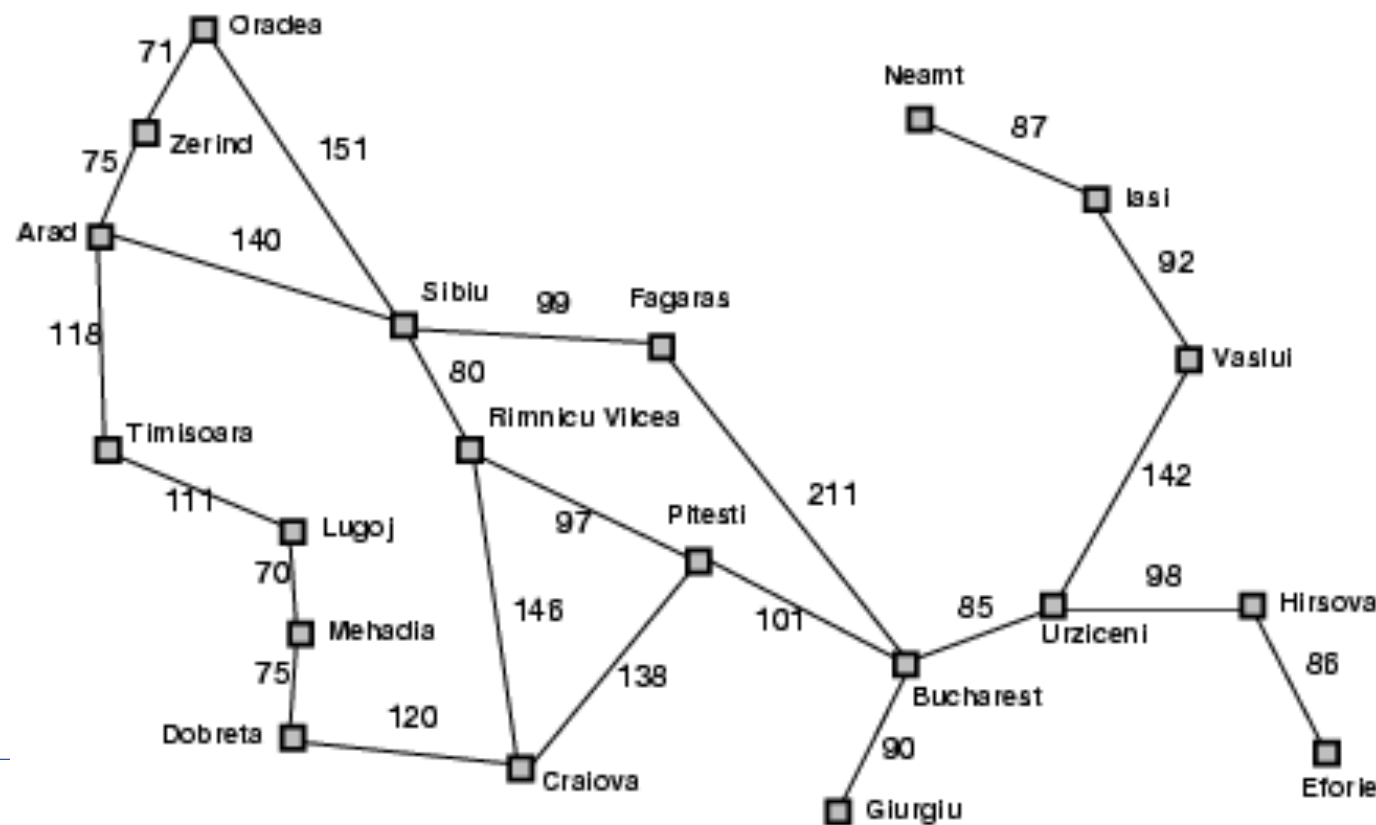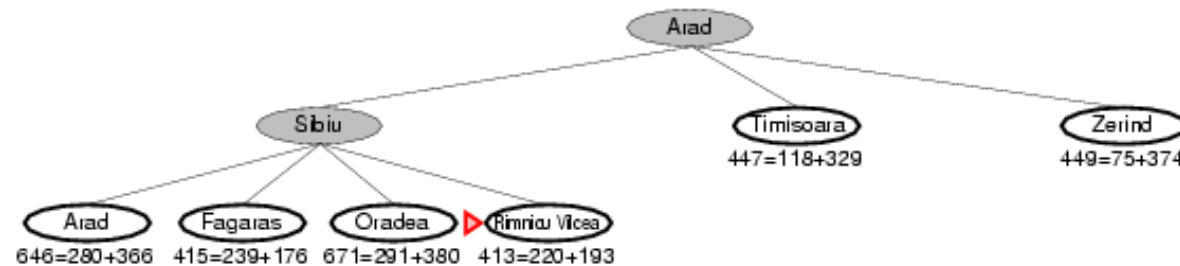
A* search example

Arad
366=0+366

A* search example

A* search example

A* search example

A* search example

A* search example

Informed search strategies

- ☐ Best-first search
- ☐ Greedy best-first search
- ☐ A* search
- ☐ **Heuristics**
- ☐ Local search algorithms
  - ■ Hill-climbing search
  - ■ Simulated annealing search
  - ■ Local beam search
  - ■ Genetic algorithms

Admissible heuristics

- ☐ Objective:
    - ■ produce a solution that is good enough for solving the problem at hand
    - ■ This solution may not be the best but approximate the exact solution

- ☐ $h(n)$ is admissible if for node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n

- ☐ An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic

- ☐ Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

- ☐ Theorem: If $h(n)$ is admissible, A* using `TREE-SEARCH` is optimal

Properties of A*

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

Admissible heuristics for 8-puzzle

☐ Heuristic: produce a solution in a reasonable time frame, good enough to solve the problem

☐ The average cost for the 8-puzzle are approx. 22 steps. Here are 26 steps.

☐ Branching factor is approx. 3

■ Empty in the middle, 4 mov

■ Empty in the corner, 2 mov

■ Rest cases, 3 mov

Start State   Goal State

☐ Depth-first search will look 3^{22} states

☐ If we keep track of repeated states, we could reduce to 170.000

☐ In the15-puzle = 10^{13}

Admissible heuristics for 8-puzzle

E.g., for the 8-puzzle:

☐ $h_1(n)$ = number of misplaced tiles
☐ $h_2(n)$ = total Manhattan distance
(i.e., + horizontal and vertical distance from desired location of each tile)

Start State

Goal State

☐ $h_1(S)$ = ?
☐ $h_2(S)$ = ?

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

Start State

Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1

- h_2 is better for search

- Typical search costs (average number of nodes expanded):

- $d=12$ IDS = 3,644,035 nodes
  $A^*(h_1) = 227$ nodes
  $A^*(h_2) = 73$ nodes
- $d=24$ IDS = too many nodes
  $A^*(h_1) = 39,135$ nodes
  $A^*(h_2) = 1,641$ nodes

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

Informed search strategies

- ☐ Best-first search
- ☐ Greedy best-first search
- ☐ A* search
- ☐ Heuristics
- ☐ **Local search algorithms**
  - ■ Hill-climbing search
  - ■ Simulated annealing search
  - ■ Local beam search
  - ■ Genetic algorithms

Local search algorithms

- ☐ In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- ☐ State space = set of "complete" configurations. Find configuration satisfying constraints, e.g., n-queens
- ☐ In such cases, we can use local search algorithms
- ☐ Keep a single "current" state, try to improve it
- ☐ Work with one current state and generally moves to the neighboring state
- ☐ The paths followed by the search are not retained
  - ■ They use little memory
  - ■ You can find reasonable solutions in large state spaces or infinite

Example: *n*-queens

☐ Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal

Informed search strategies

- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
  - **Hill-climbing search**
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

Hill-climbing search

- It's just a loop that moves in the direction of increasing value
  - Ends when it reaches a peak where no neighbor has a higher value
  - The search tree is not kept, just a data structure of the current node to check the goal condition and its objective function value
- "Like climbing Everest in thick fog with amnesia"

Hill-climbing search
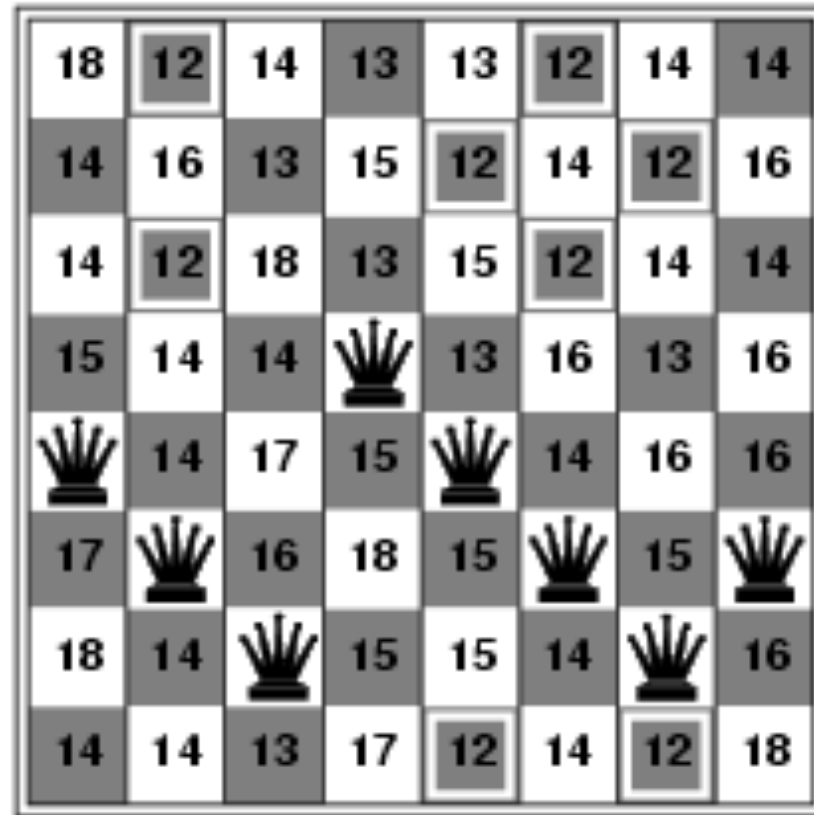
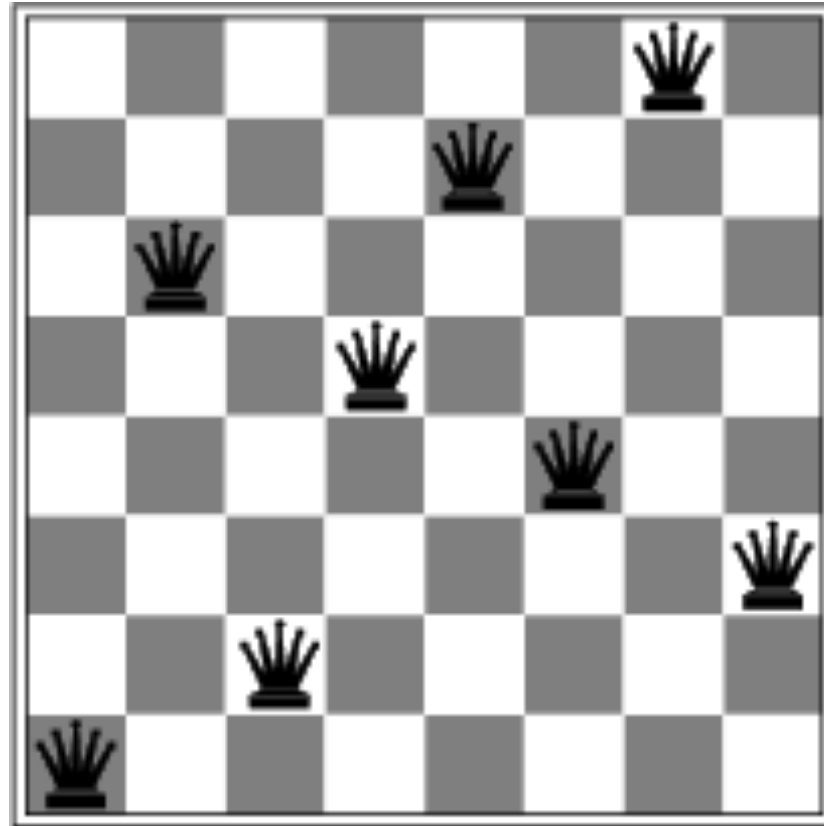☐ Problem: depending on initial state, can get stuck in local maxima
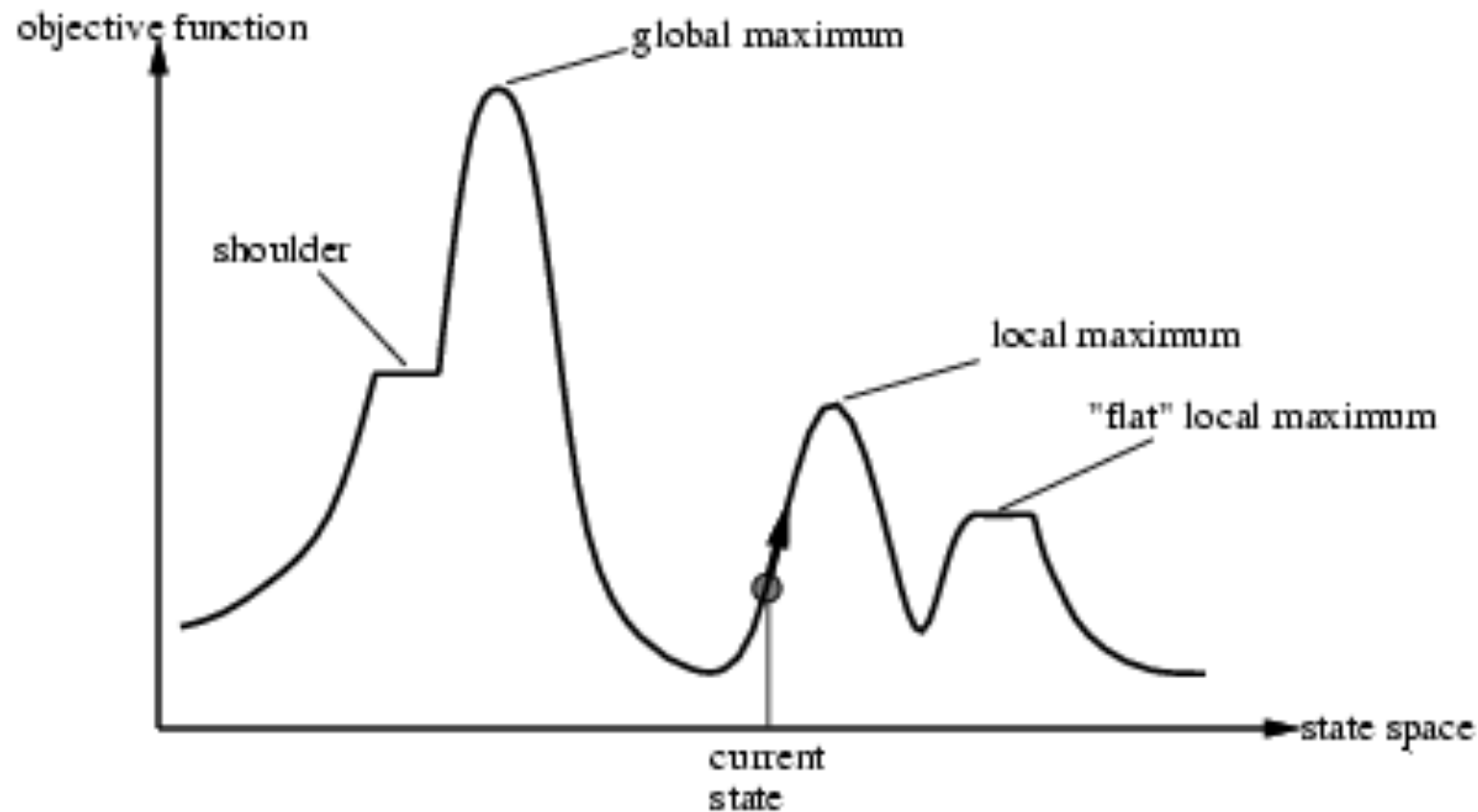
Hill-climbing search: 8-queens problem

- ☐ *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- ☐ *h = 17* for the above state
- ☐ The figure also shows the values of all successors, top successors have h = 12

Hill-climbing search: 8-queens problem

- A local minimum with h = 1 (obtained in 5 steps)

Hill-climbing search

- The algorithm gets stuck for several reasons::
  - Local Maximum: it is a peak that is higher than each of its neighbours, but lower than the maximum overall
  - Ridges: cause a sequence of local maxima that make navigation difficult
  - Plateau (flat): can lead to a local maximum where there is no ascendant exit or a terrace to advance
- In the 8-queens, it gets stack in 86% and solve 14% cases
- If we allow lateral movements with the hope that we find a terrace (limiting them if reach a local maximum, e.g.100):→ 94% success
- Variants:
  - Stochastically (randomly chooses upward movements)
  - Random restart (the initial states are generated randomly)

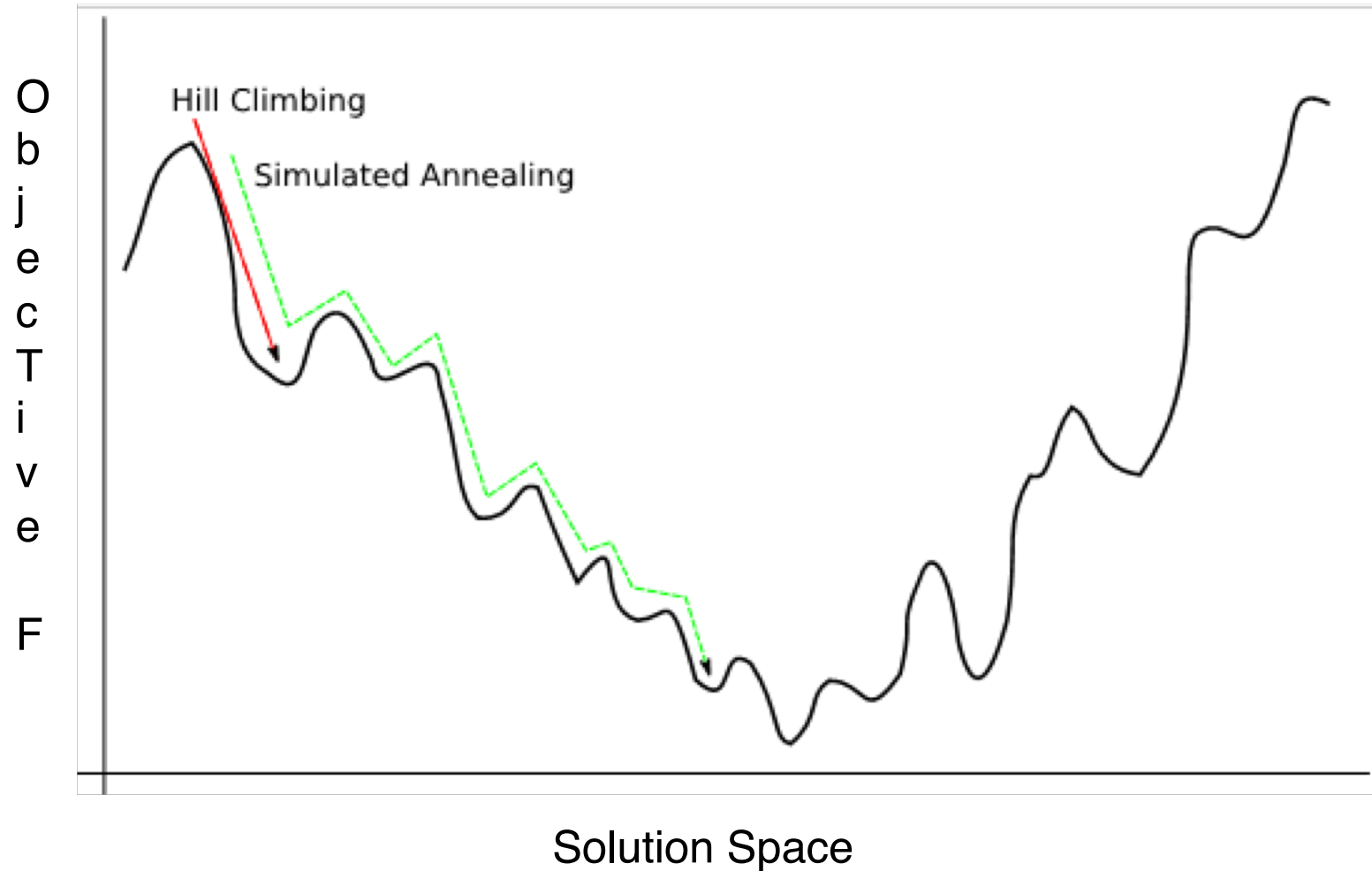# Informed search strategies

- ☐ Best-first search
- ☐ Greedy best-first search
- ☐ A* search
- ☐ Heuristics
- ☐ Local search algorithms
  - ◼ Hill-climbing search
  - ◼ **Simulated annealing search**
  - ◼ Local beam search
  - ◼ Genetic algorithms

Simulated annealing search

- Process of tempering or hardening metals by heating and then cooling them gradually
- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency
- It combines hill-climbing with random generation successor
- Good for problems with a large search space, optimum is surrounded by many local optima
- Problem: determine the values of the parameters, and requires an important work of experimentation that depends on each problem
- Widely used in VLSI layout, airline scheduling, etc

Simulated annealing search

Informed search strategies

- ☐ Best-first search
- ☐ Greedy best-first search
- ☐ A* search
- ☐ Heuristics
- ☐ Local search algorithms
  - ■ Hill-climbing search
  - ■ Simulated annealing search
  - ■ **Local beam search**
  - ■ Genetic algorithms

Local beam search

- Idea: 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
  - Alternatively stochastic LBS randomly choose k successors, with the probability of choosing a successor as an increasing function of its value

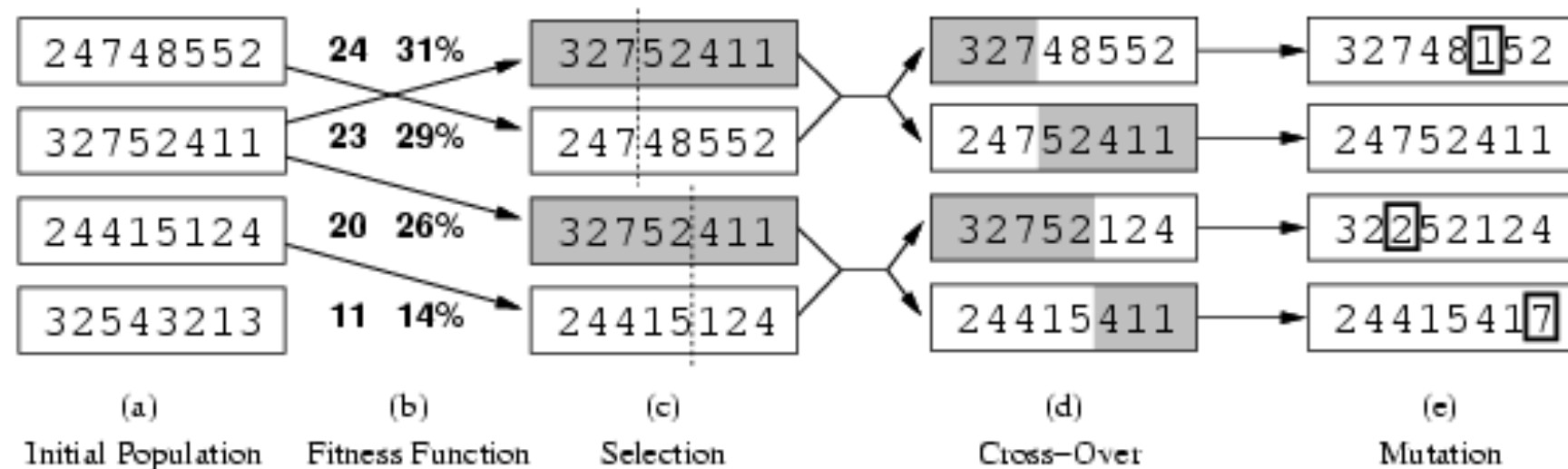# Informed search strategies

- ☐ Best-first search
- ☐ Greedy best-first search
- ☐ A* search
- ☐ Heuristics
- ☐ Local search algorithms
  - ■ Hill-climbing search
  - ■ Simulated annealing search
  - ■ Local beam search
  - ■ **Genetic algorithms**
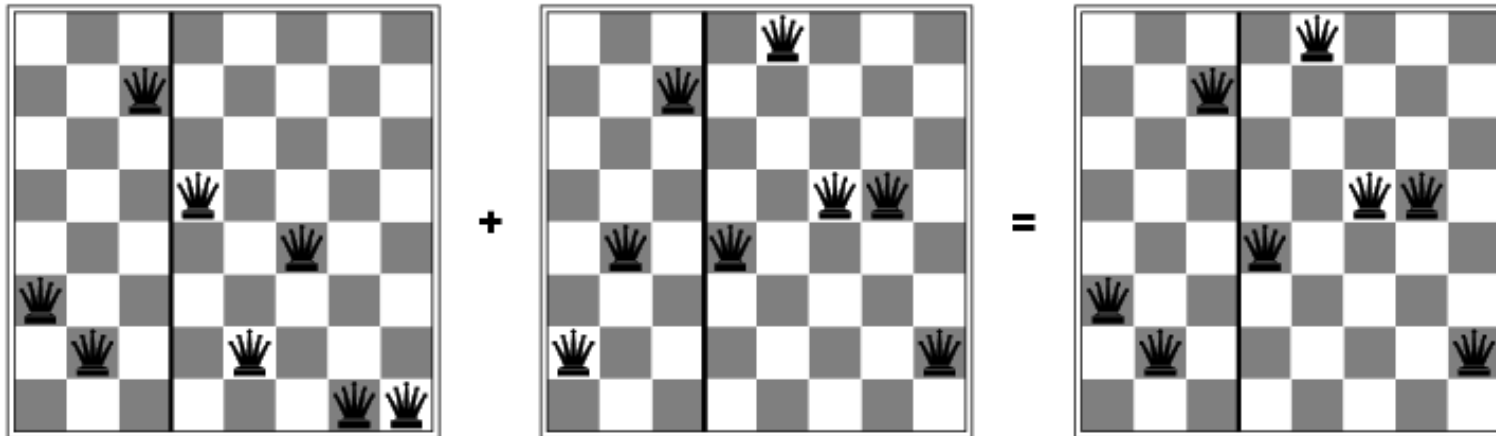
Genetic algorithms

- 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

- Produce the next generation of states by selection, crossover, and mutation

Genetic algorithms

| 24748552 | 24 31% | 32752411 | 32748552 | 3274852 |
32752411	23 29%	24748552	24752411	24752411
24415124	20 26%	32752411	32752124	3252124
32543213	11 14%	24415124	24415411	2441541
(a)	(b)	(c)	(d)	(e)
Initial Population	Fitness Function	Selection	Cross-Over	Mutation

- ☐ The initial population is represented by chains of 8 digits representing states of 8 queens

Genetic algorithms

□ It corresponds to the 2 parents of figure (c)
327 52411 (5 attacked queens)
247 48552 (4 attacked queens)

Genetic algorithms
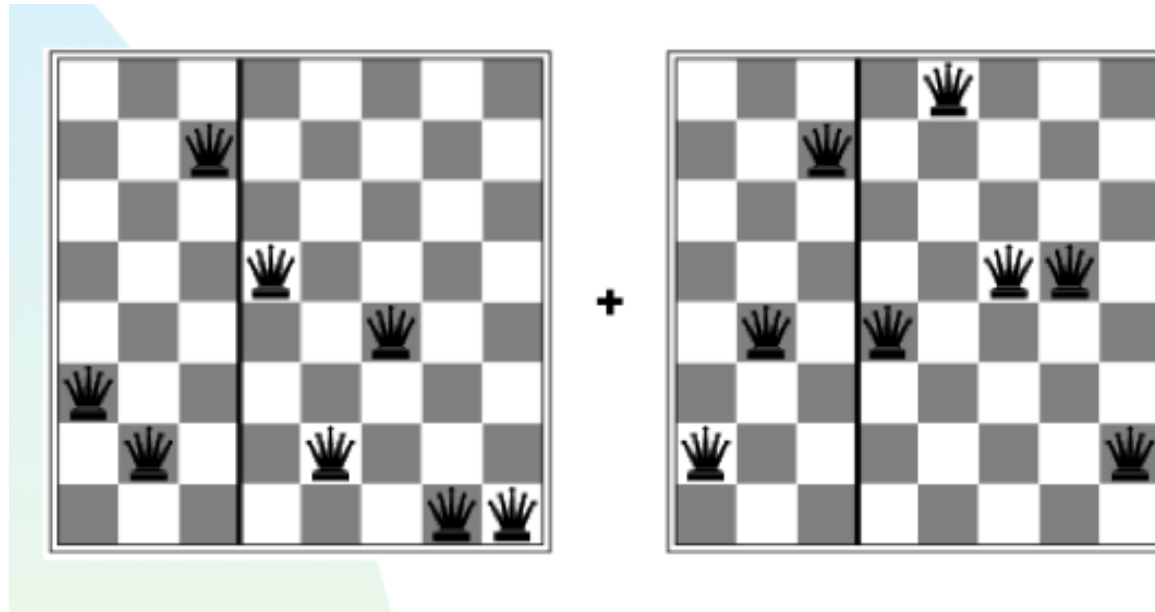
☐ Fitness function: number of NON-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)

■ Right: 247 48552 (4 attacked queens) = 28 – 4 = 24

   24/(24+23+20+11) = 31%

■ Left: 327 52411 (5 attacked queens) = 28 – 5 = 23

   23/(24+23+20+11) = 29%

Genetic algorithms

(a)	(b)	(c)	(d)	(e)
Initial Population	Fitness Function	Selection	Cross–Over	Mutation

probabilities

- ☐ Crossing points are chosen and d) the offspring are created crossing the chain parental crosspoint
- ☐ In e) each position is subject to random mutation with a small probability

Informed Search