Search algorithms

- Uninformed search strategies
  - blind search: none extra information or knowledge beyond the definition of the problem itself
  - Challenging problem for Uniformed search
- Informed search strategies
  - heuristic search: some problem-specific knowledge used

- Questions and Answers

Summary of algorithms*

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Challenging from Complexity in space and in time

Lecture 5: Informed search algorithms

Chapter 4

Material

- Chapter 4 Section 1 - 3
- Exclude memory-bounded heuristic search

Outline

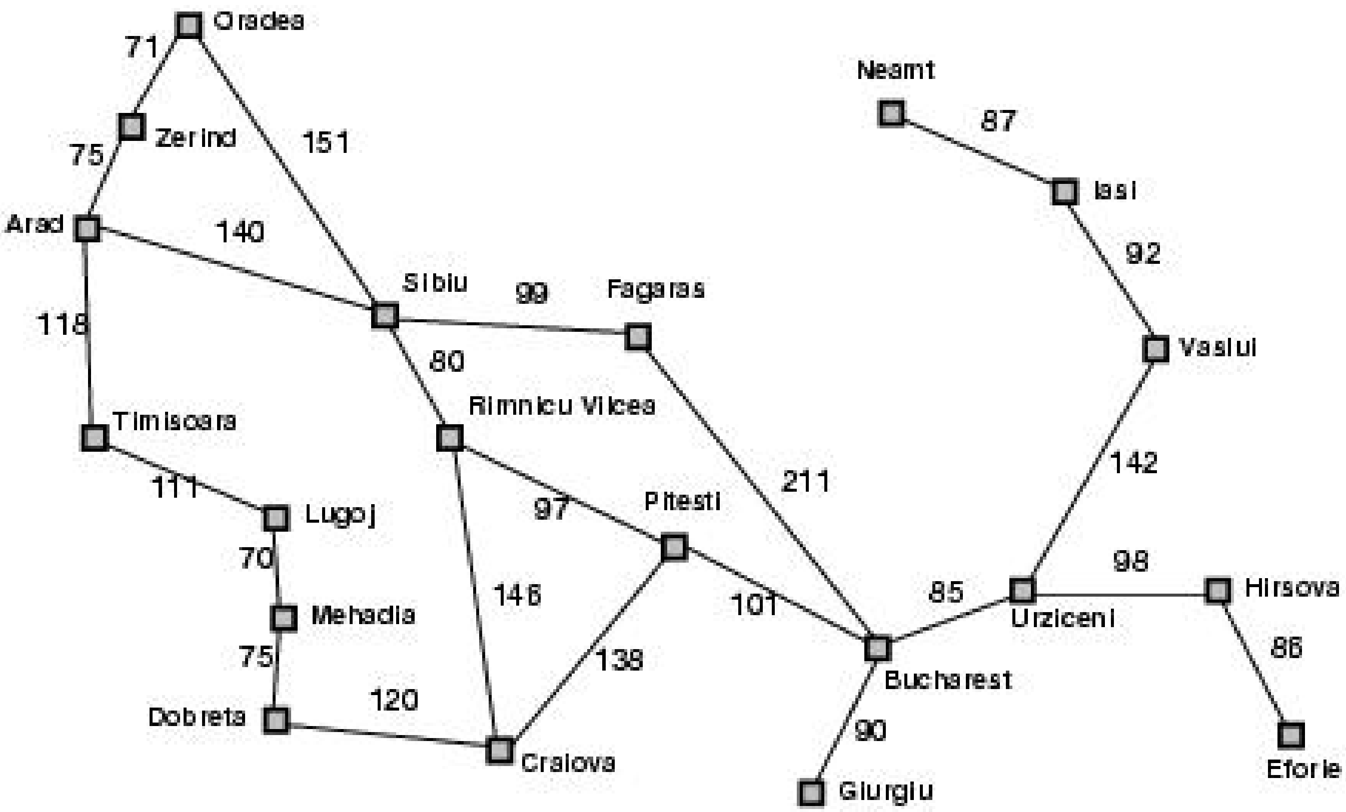
- Best-first search
- Greedy best-first search
- A^* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms
- Summary

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
    Expand most desirable unexpanded node
    e.g. the node with the lowest evaluation expanded first
- Implementation:
  Order the nodes in fringe in *decreasing* order of desirability

- Special cases:
  - greedy best-first 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	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

- Evaluation function $f(n)$ $=?$

$$f(n) = h(n) \text{ (heuristic)}$$

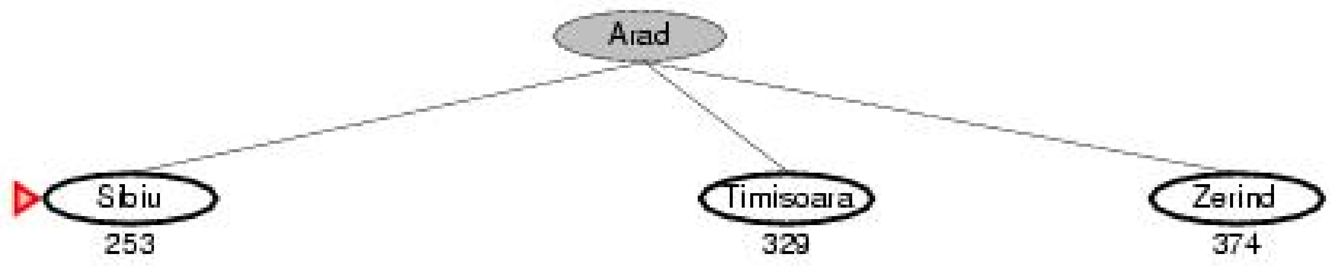
  – to estimate of cost from n to *goal*

e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

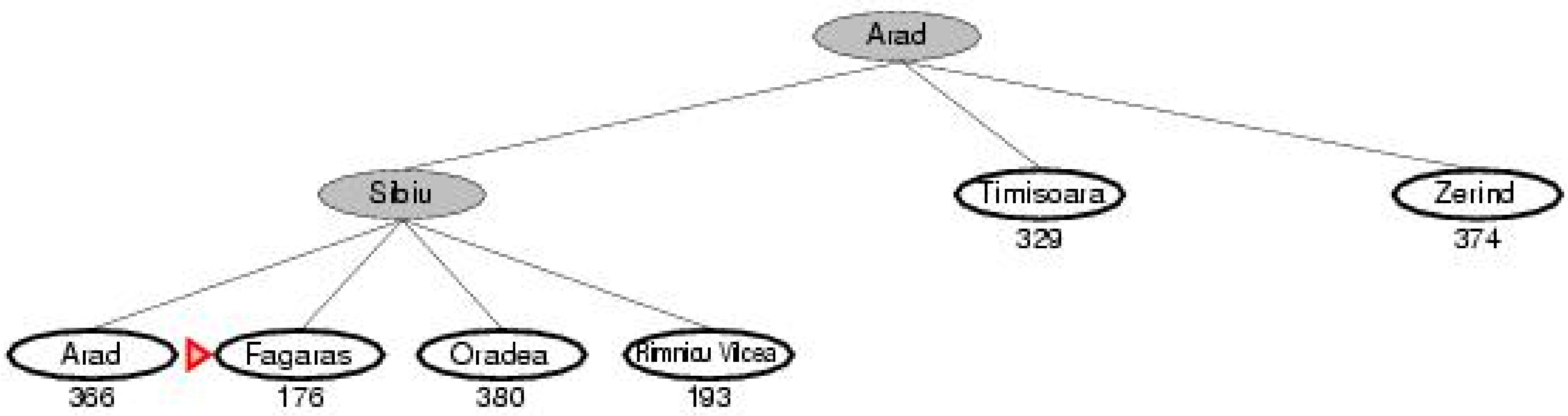
- Greedy best-first search expands the node that appears to be closest to goal

Greedy best-first search example

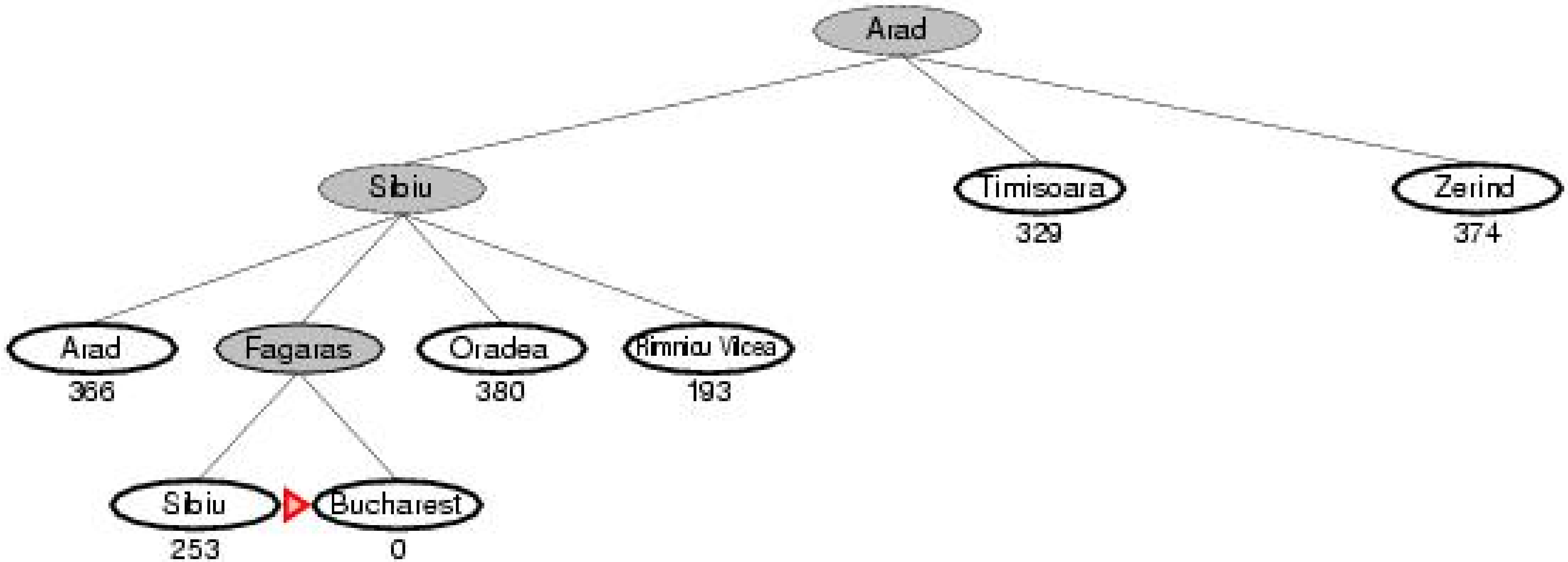
Greedy best-first search example



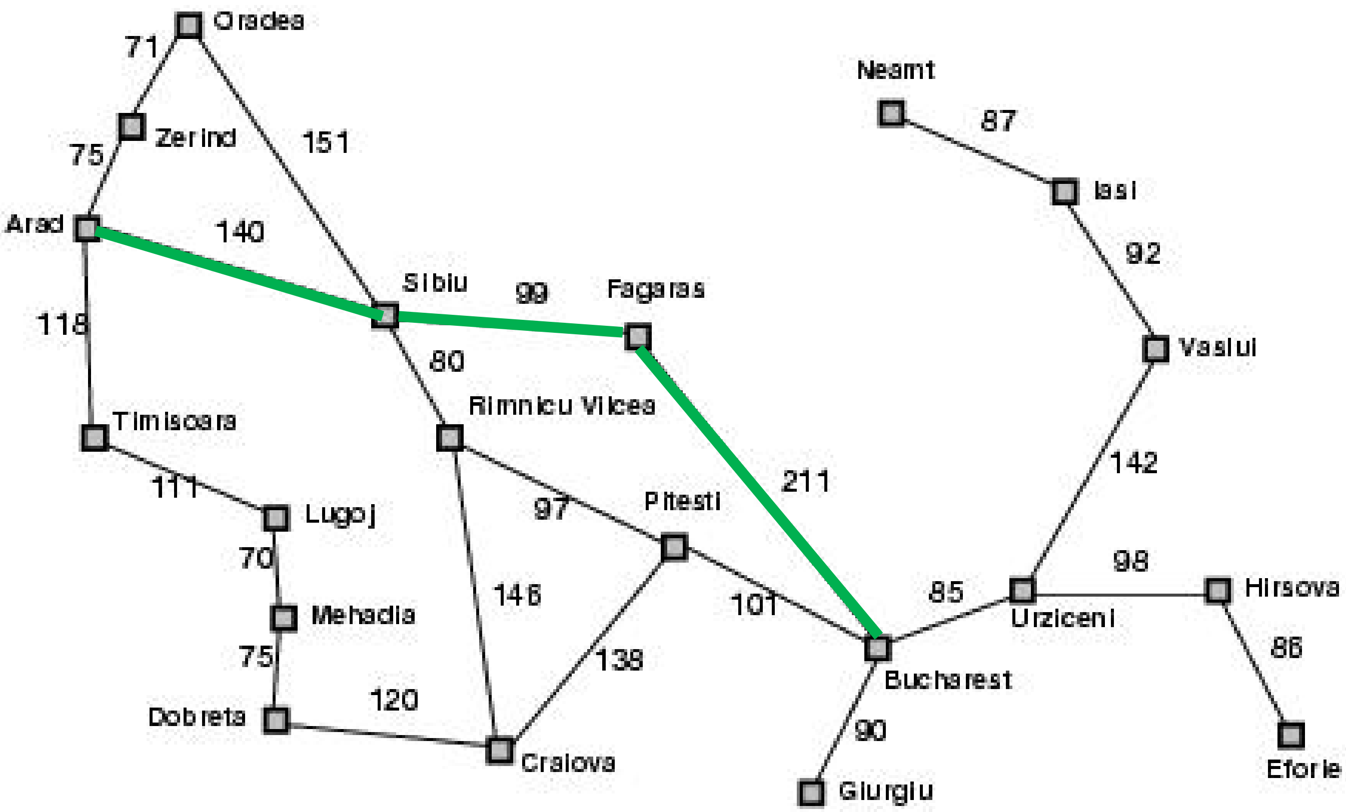
Greedy best-first search example



Greedy best-first search example



Result based on Greedy BFS

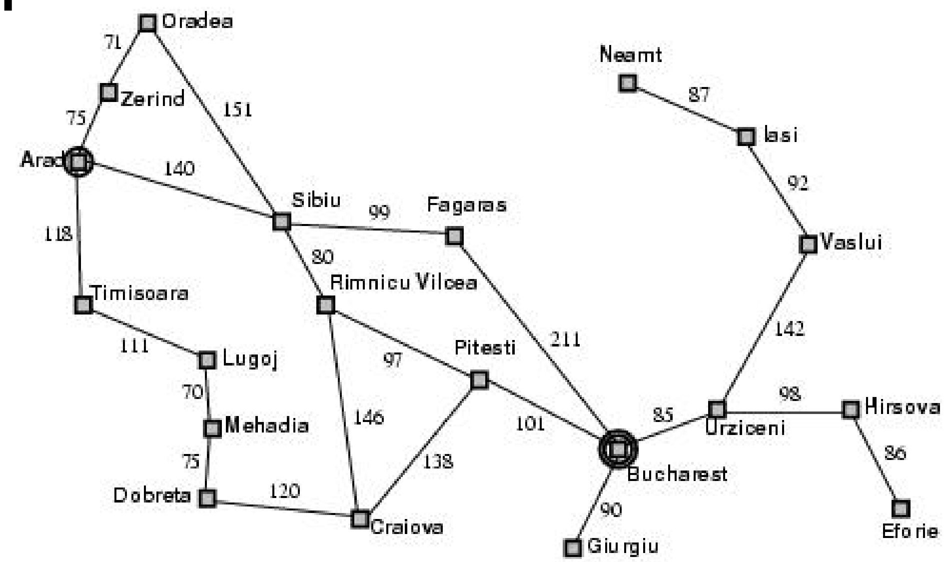


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy best-first search

- Complete? No – can get stuck in loops e.g., *Iasi → Bucharest?*
  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: to avoid expanding paths that are already expensive, how to improve f(n)?
- Evaluation function

$$\boxed{f(n) = g(n) + h(n)}$$

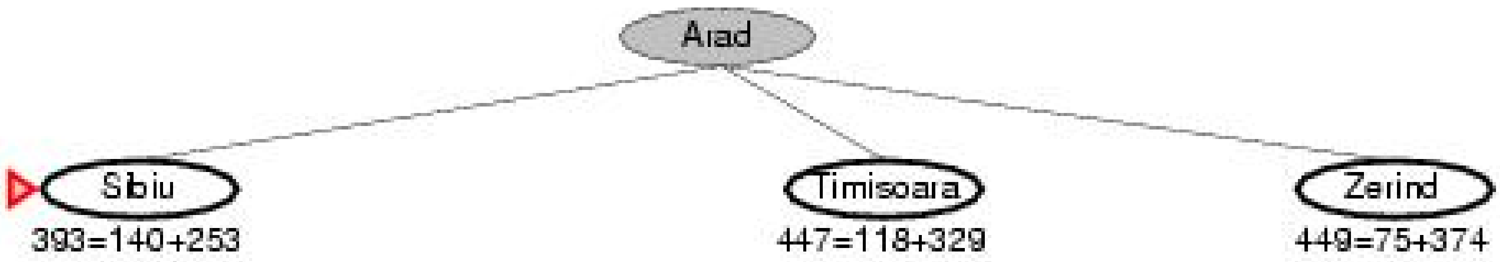
$g(n)$ = cost so far to reach n

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

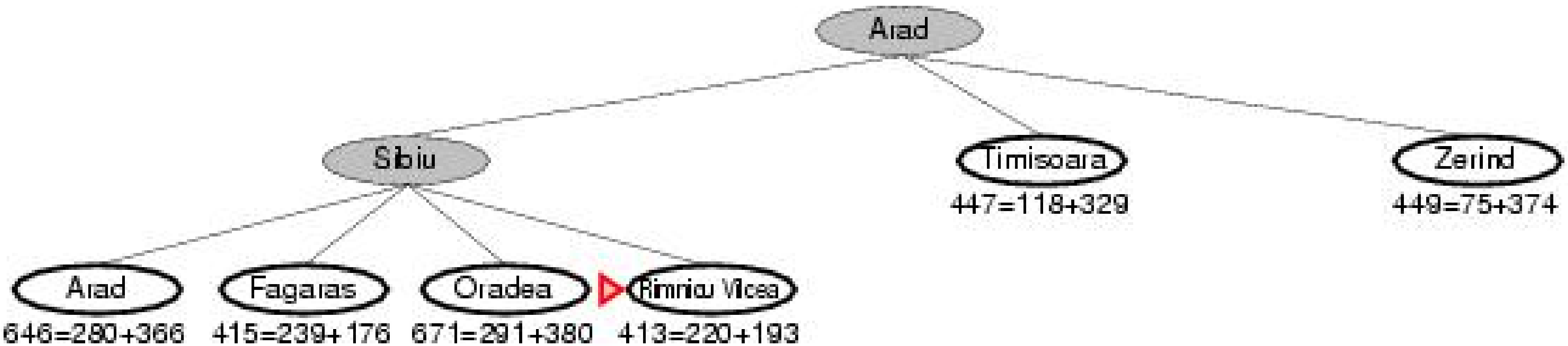
$f(n)$ = estimated total cost of path through n to goal

A* search example

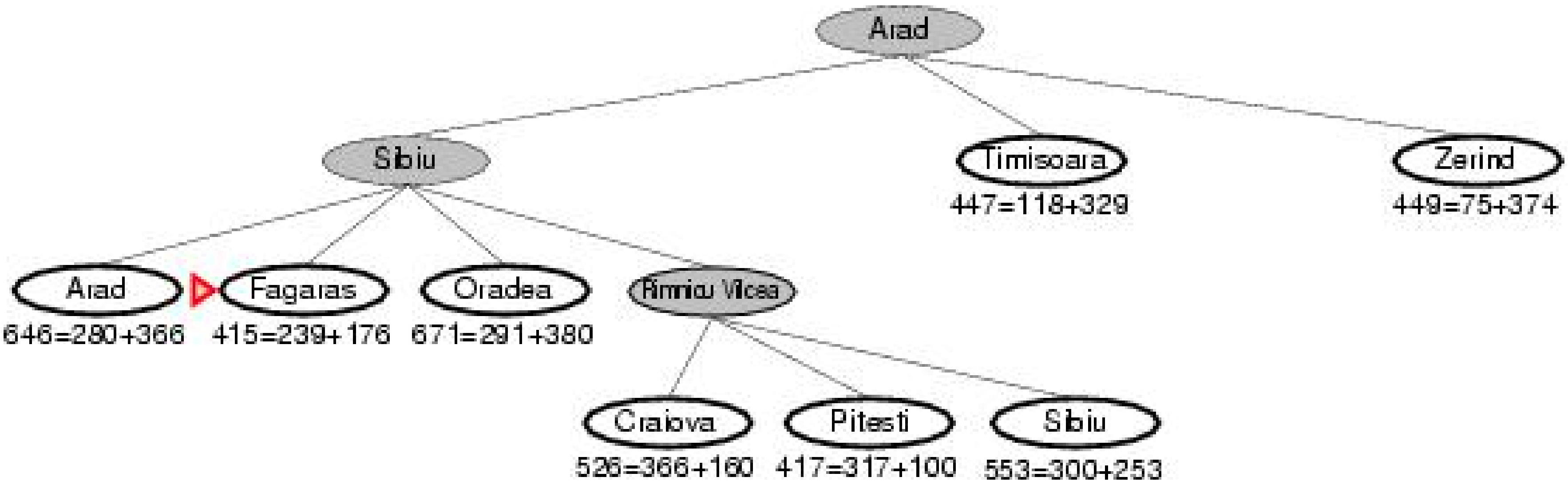
A* search example



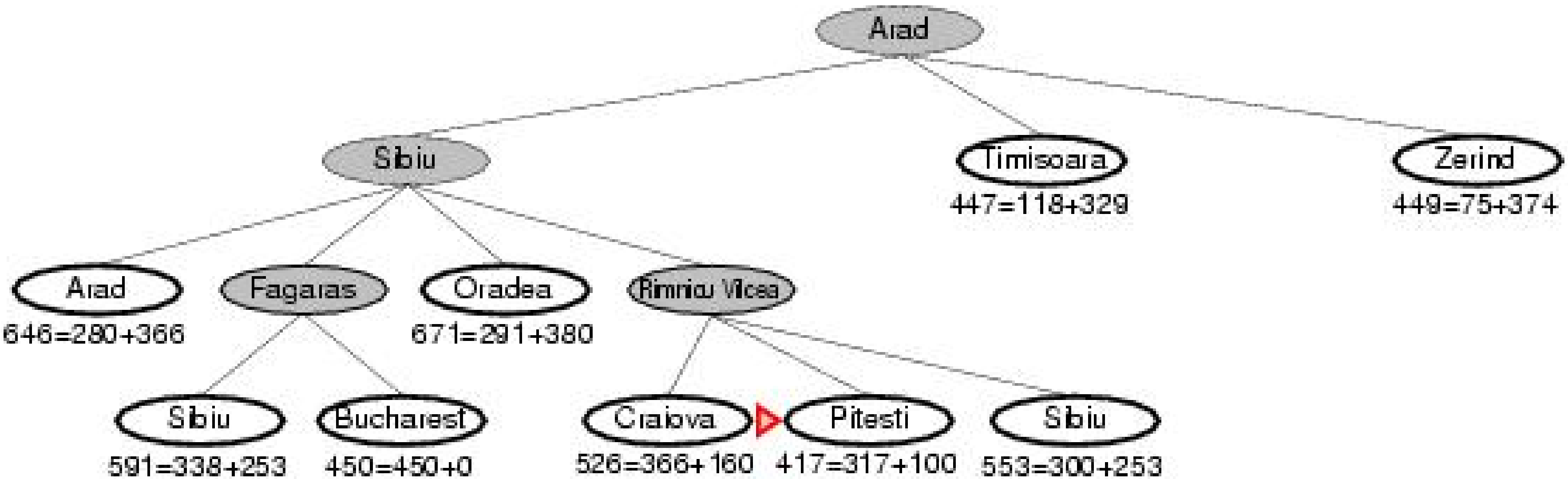
A* search example



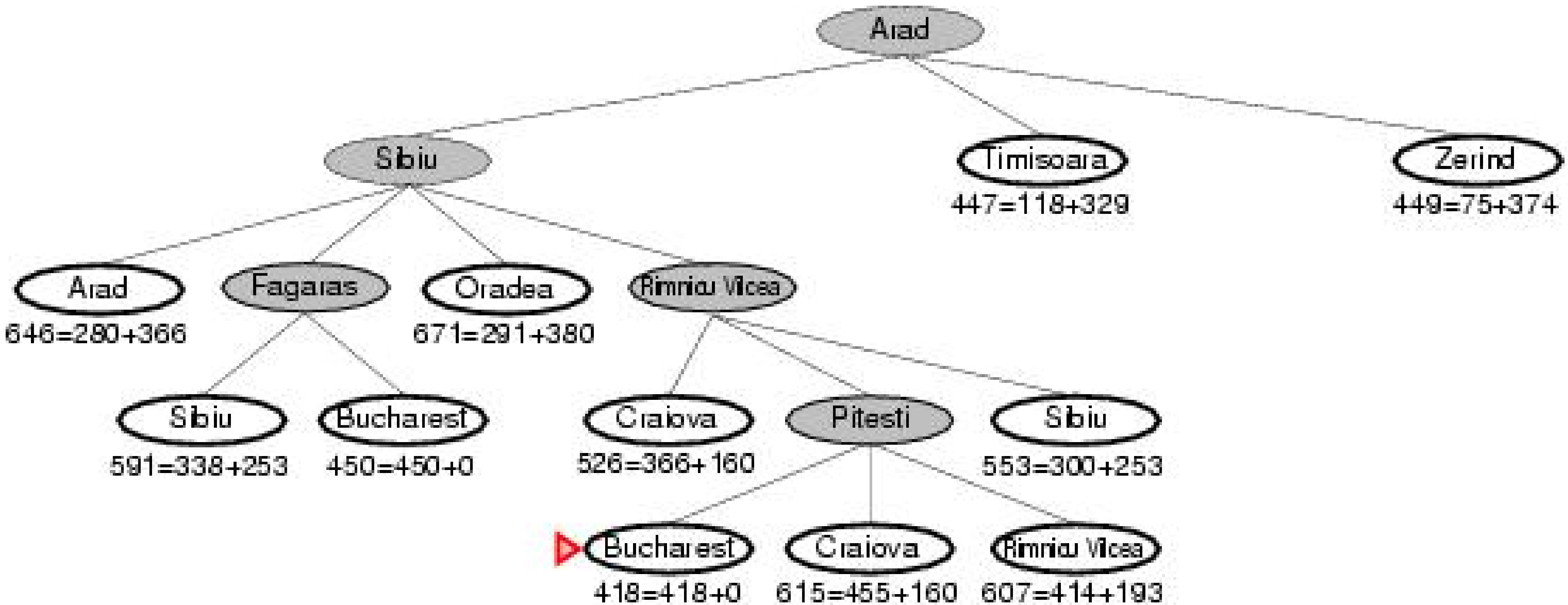
A* search example



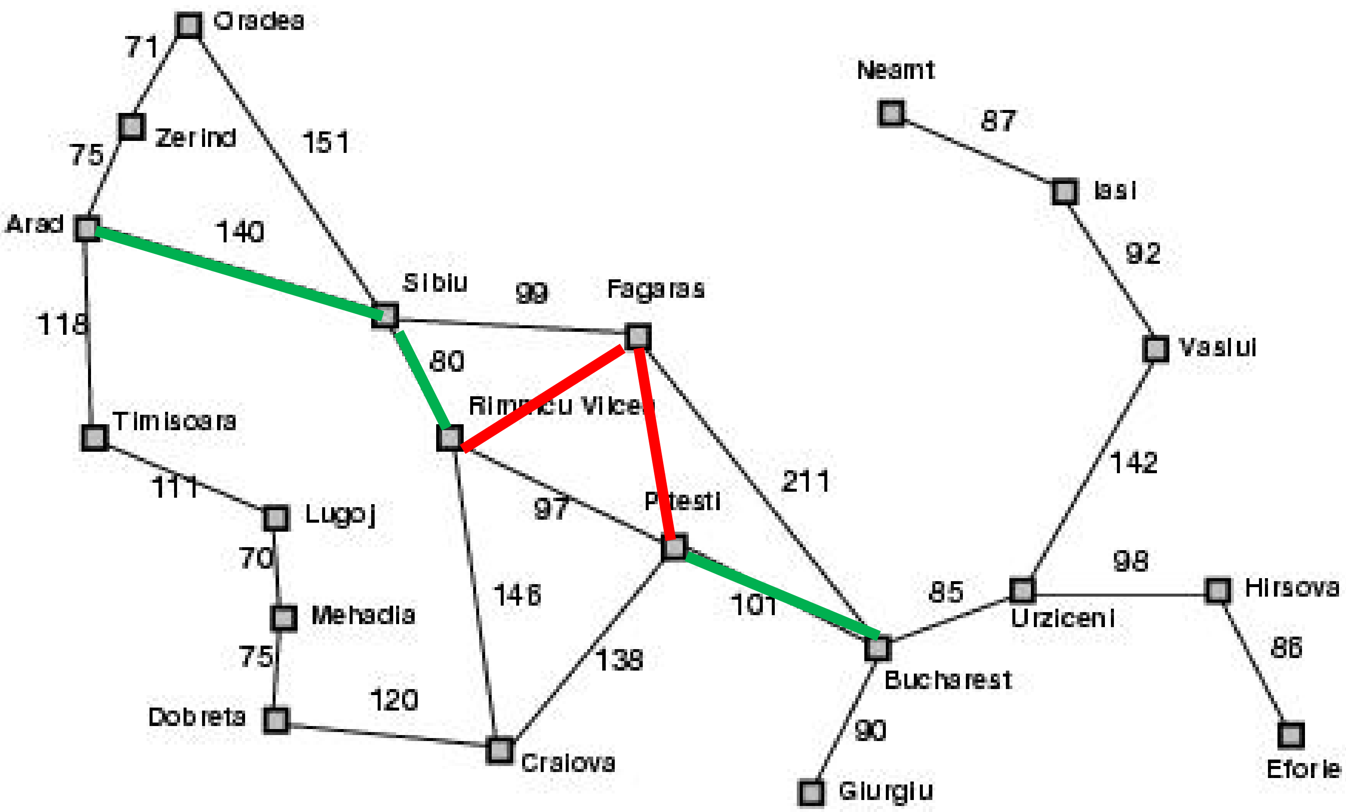
A* search example



A^* search example



Result based on A*



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

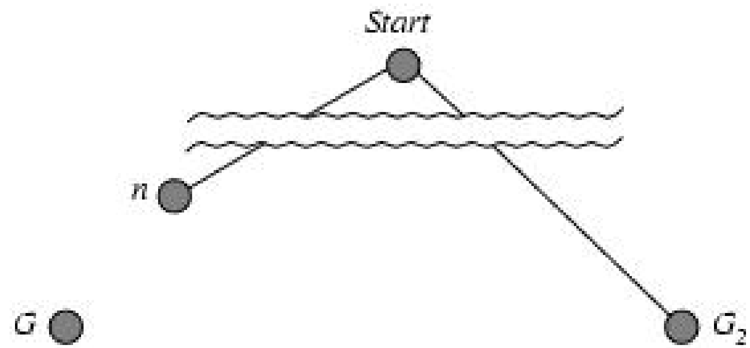
Admissible heuristics

- 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*.
- 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**

Optimality of A* (proof)

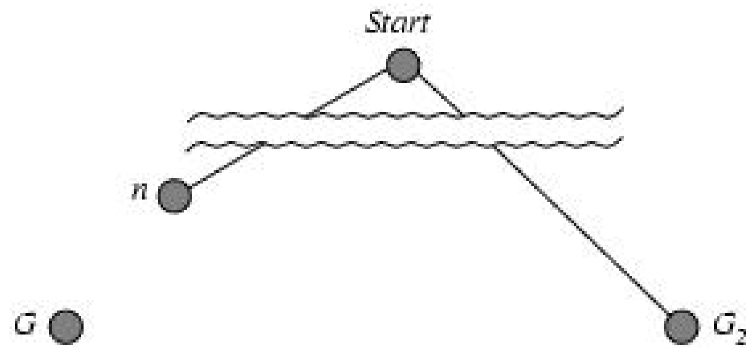
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A^* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



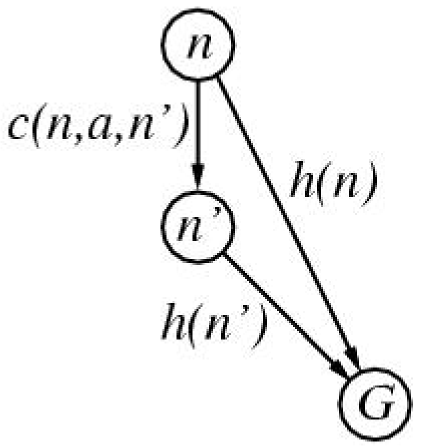
• $f(G_2)$	$> f(G)$	from above
• $h(n)$	$\leq h^*(n)$	since h is admissible
• $g(n) + h(n)$	$\leq g(n) + h^*(n)$	
• $f(n)$	$\leq f(G)$	

Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion

Consistent heuristics

- Assume: A heuristic is consistent if for every node n, every successor n' of n generated by any action a,

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



- Proof: If h is consistent, we have

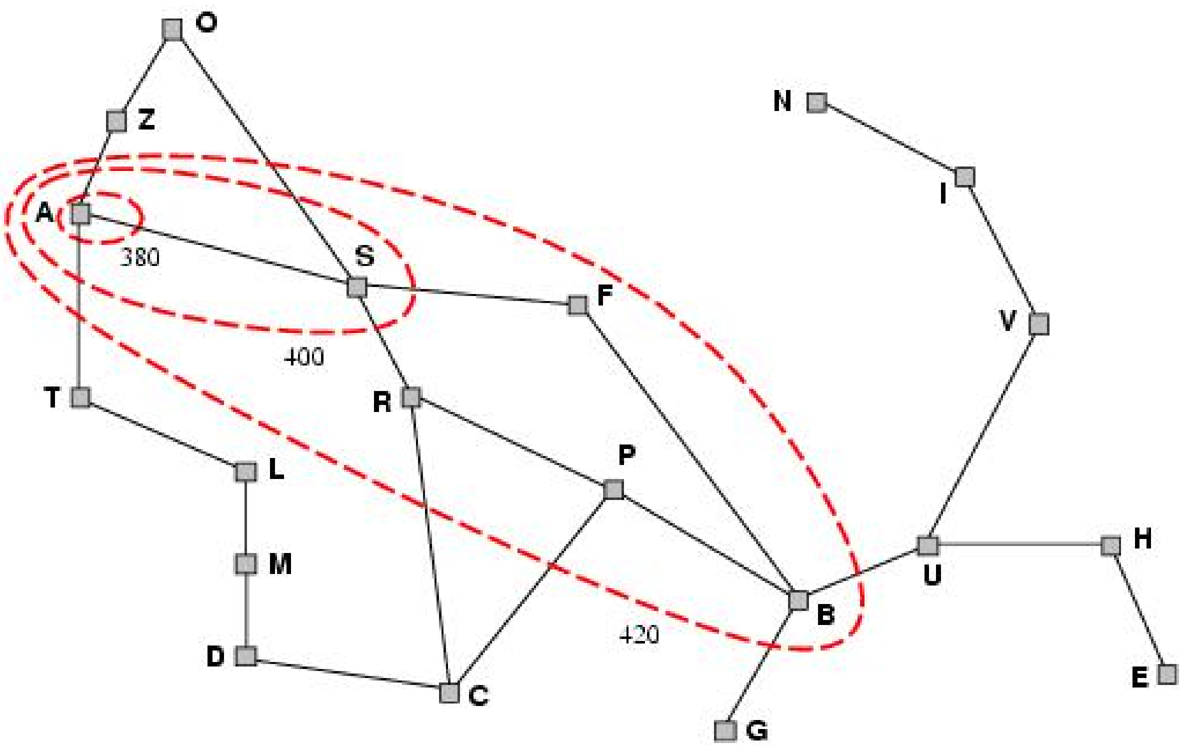
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

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

- **Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal**

Optimality of A*

- A* expands nodes in order of increasing *f* value
  Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Properties of A*

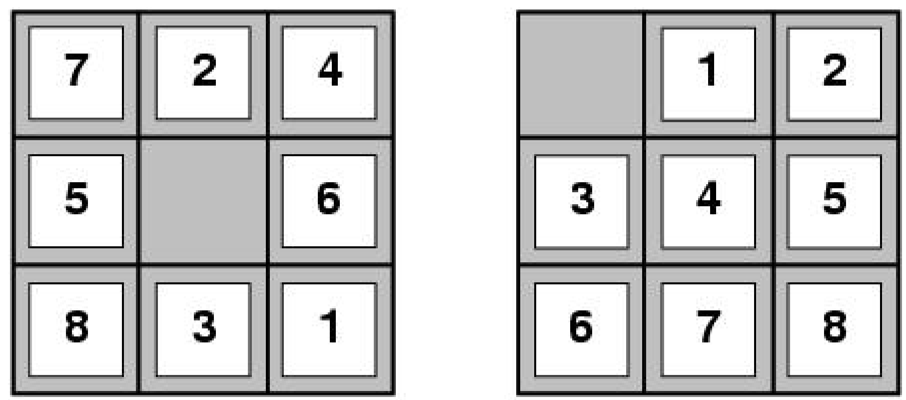
- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

Admissible heuristics

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

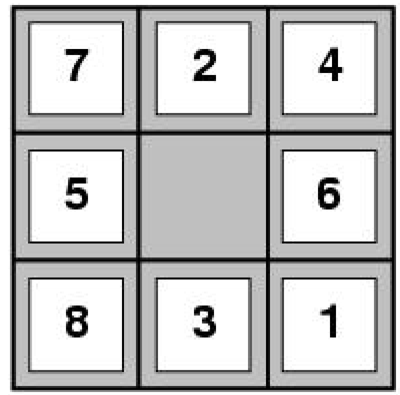
- $h_1(S)$ = ?
- $h_2(S)$ = ?

Admissible heuristics

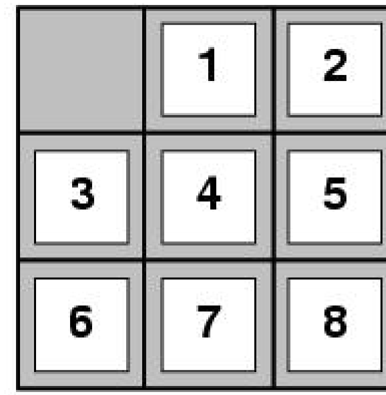
E.g., for the 8-puzzle:

- $h_1(n)$ = 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

Summary

- Greedy Best-first Search
- A^* search
- Properties of search
  - Complete
  - Optimal
    - Local or global
- Heuristic *h(n)*
  - Admissible
  - Consistent

Assignment

– Chap 3: exercise 3.14, 3.21, 3.23

*Handed in next Tuesday

To be continued

Informed search algorithms

- Greedy Best-first Search
- A^* search
  - Heuristic *h(n)*
- Questions and Answers
  - Algorithms

Lecture 6: Informed search algorithms(2)

Chapter 4

Outline

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

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
- local search algorithms
  - keep a single "current" state, try to improve it
  - *Gradient Descent algorithm*

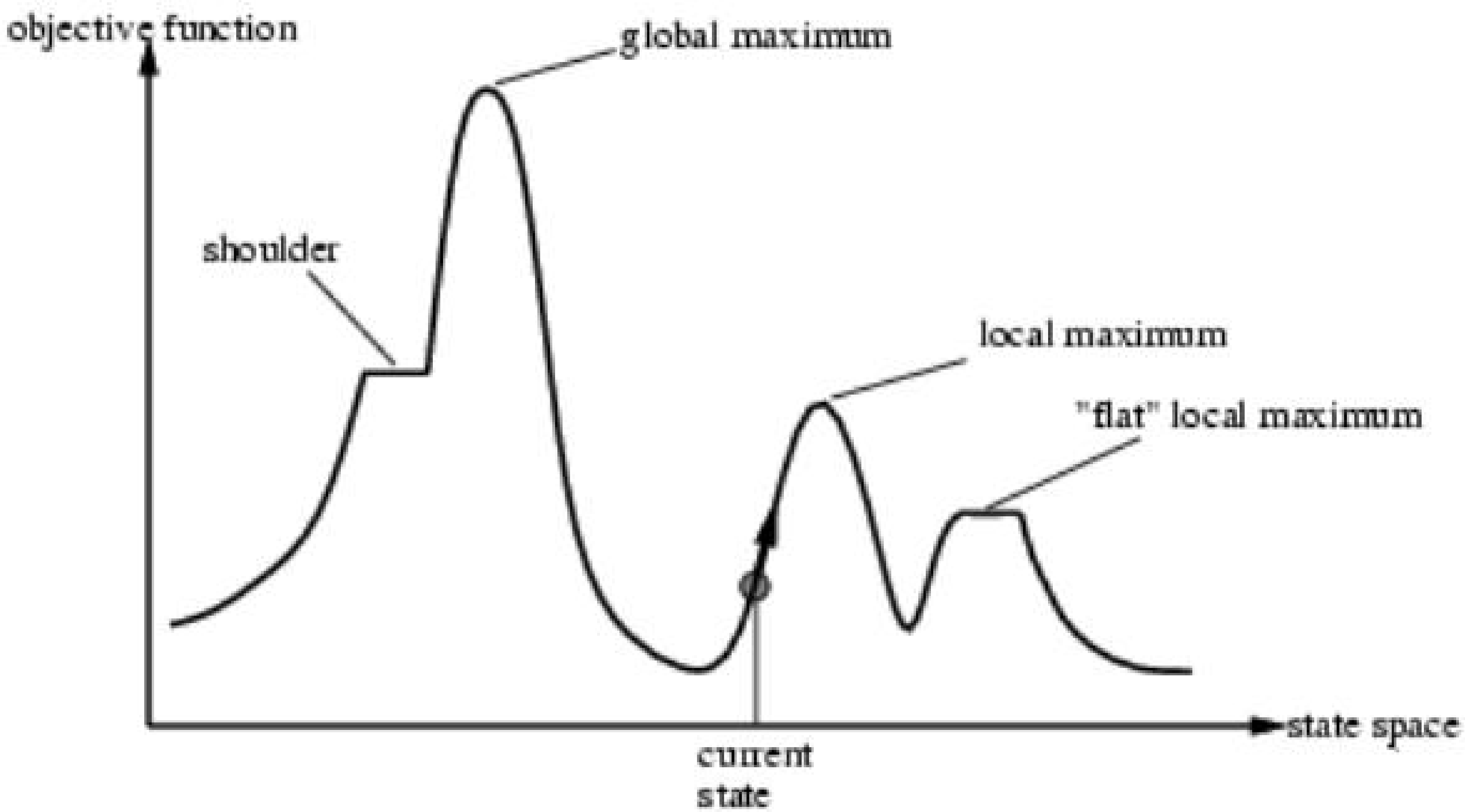


Fig. A one-dimensional state-space landscape

Different strategies for hill-climbing search:

• Gradient Descent algorithm / • Simulated Annealing algorithm / • GA

Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Hill-climbing search

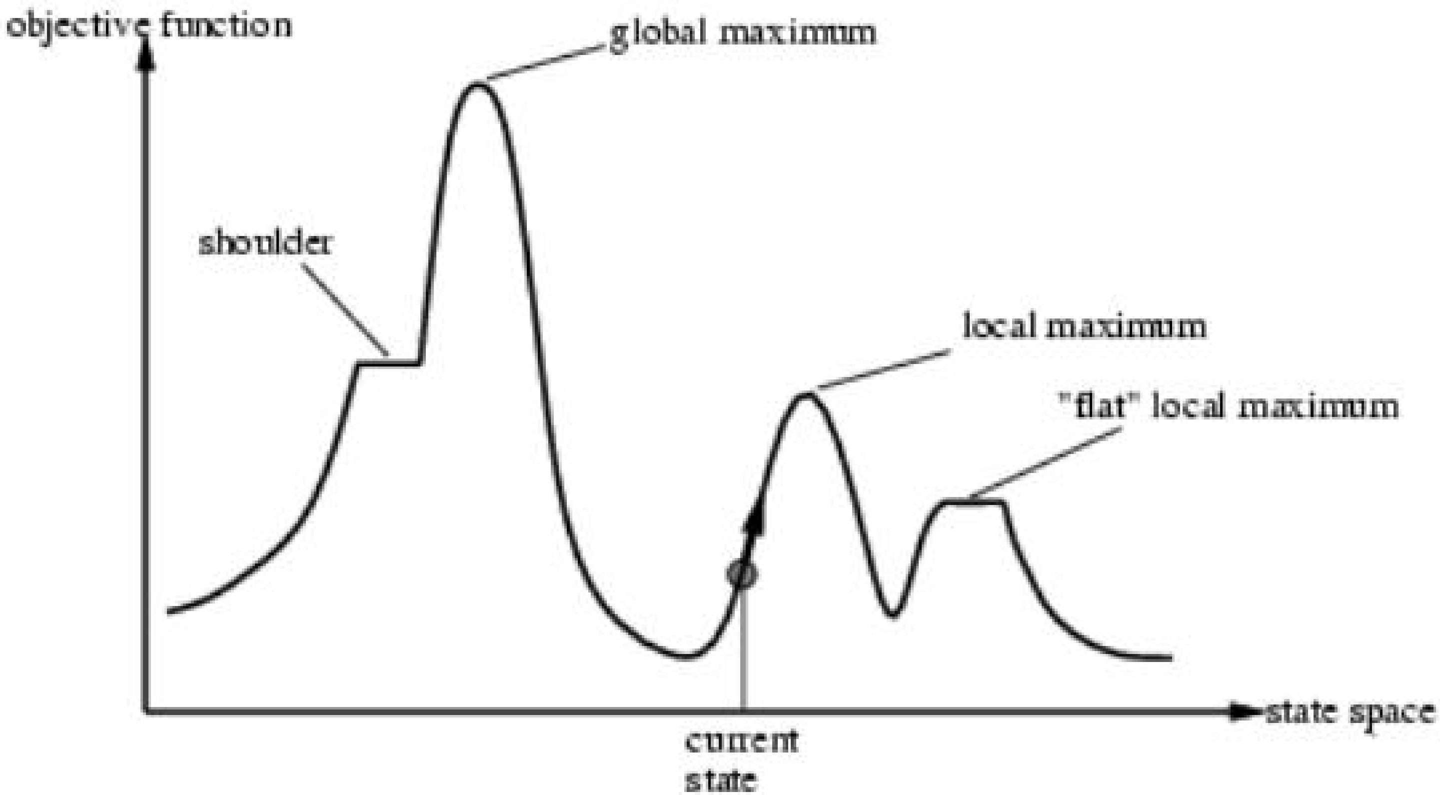
- "Like climbing Everest in thick fog with amnesia"

```
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
```

Hill-climbing search

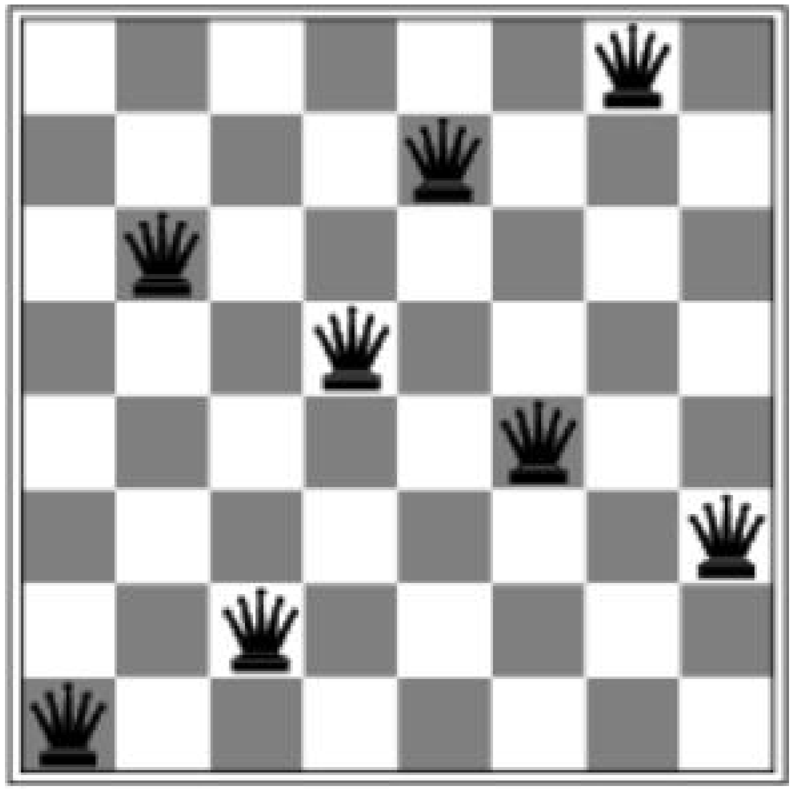
- Problem: depending on initial state, can get stuck in local maxima



Hill-climbing search: 8-queens problem

- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = *?* for the above state **17**

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

```
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 search

- Annealing
  - is actually a term used in the metallurgical industry
  - probably by heating the crystal to a very high temperature and then slowly cooling down, reducing defects in the crystal (reaching the most stable state with the lowest energy)
- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Questions:
  - What will be if ***T*** remain a big value?
  - What will be if ***T*** decrease extremely quickly?

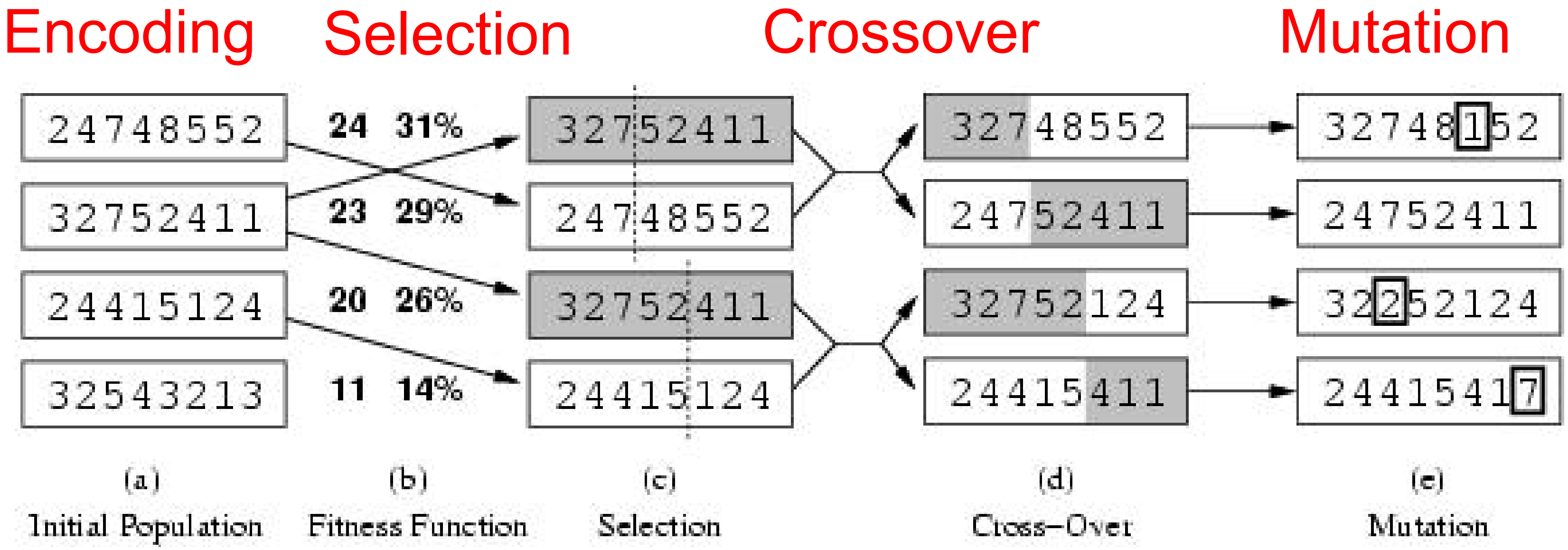
Local beam search

- 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

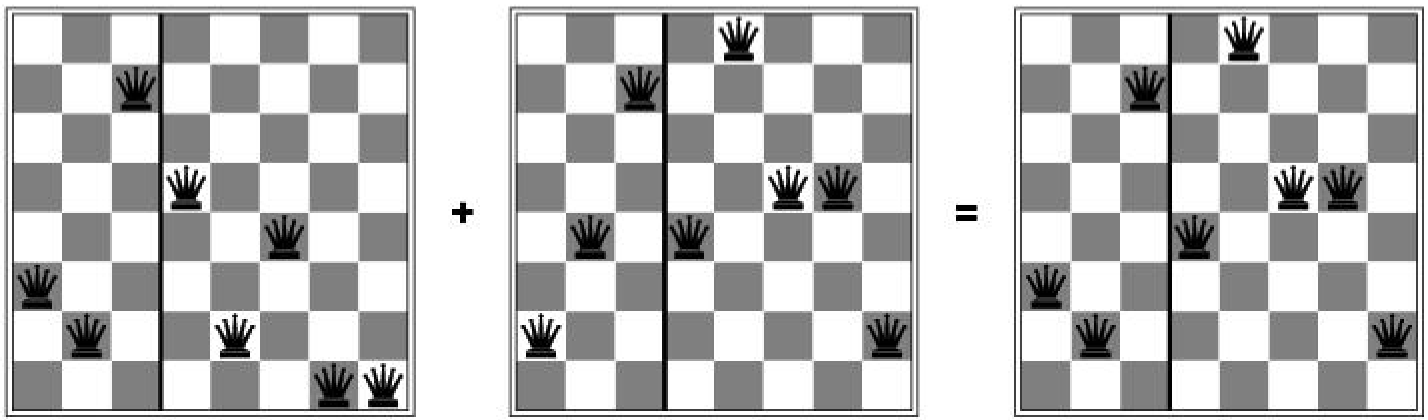
- Ideas
  - A successor state is generated by combining two parent states
    Start with *k* randomly generated states (population)
    A state is represented as a string over a finite alphabet (often a string of 0s and 1s) *--encoding*
    Evaluation function (fitness function). Higher values for better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

Genetic algorithms

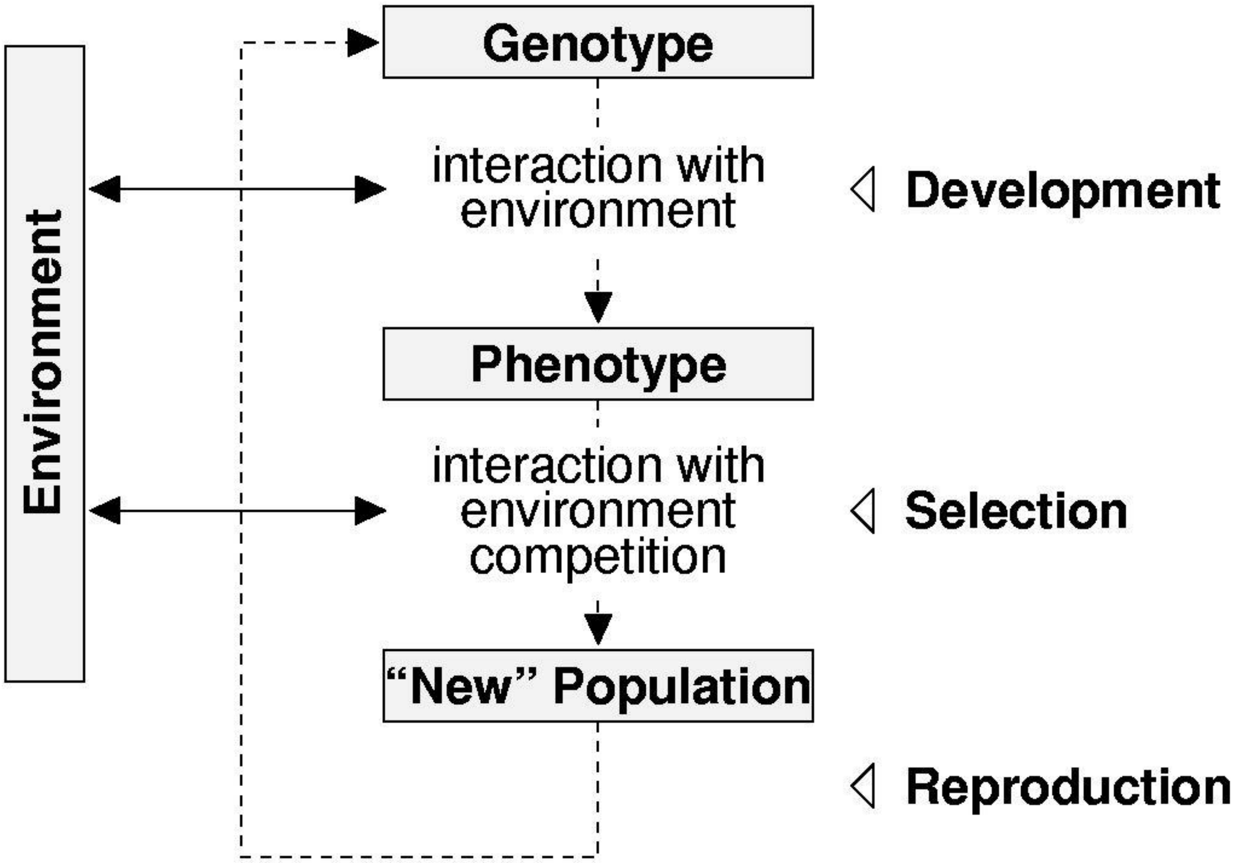


- Encoding: 8-bit string
- Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times 7/2 = 28$)
- $24/(24+23+20+11) = 31\%$
- $23/(24+23+20+11) = 29\%$ etc

Genetic algorithms



“Grand” evolutionary scheme

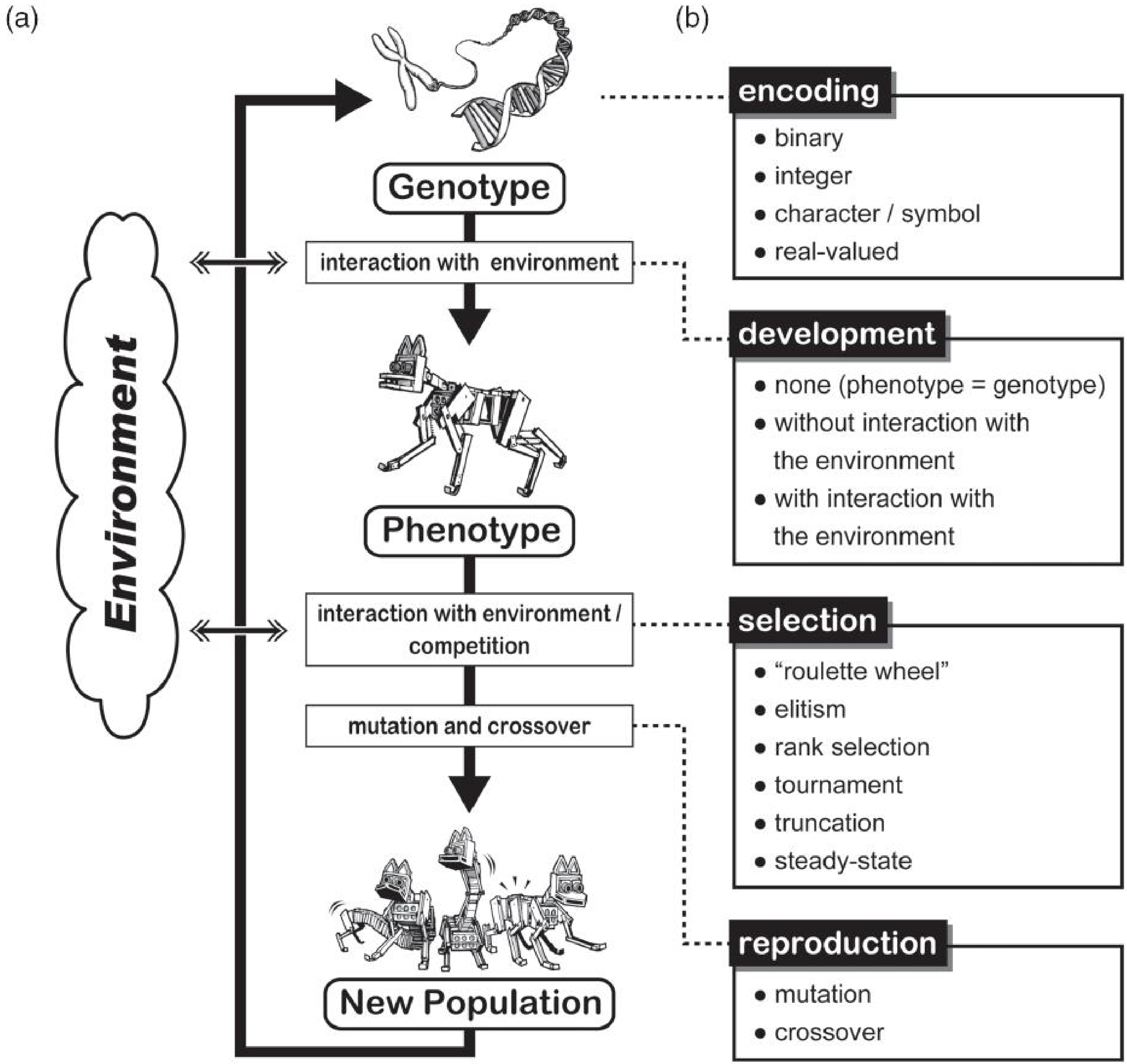


- ***Q & A :***

Exploration
vs
Exploitation

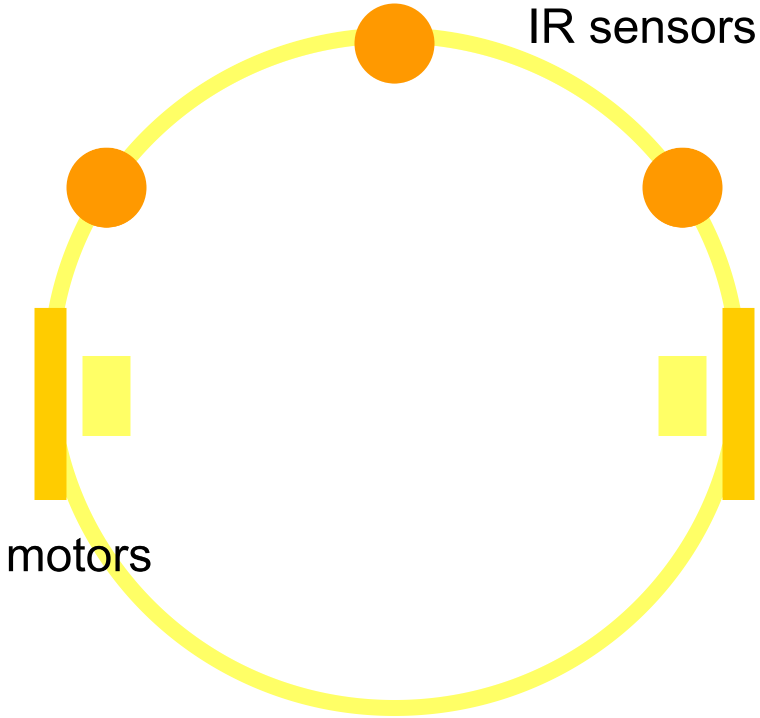
encoding	development	selection	reproduction
• binary • many-character • real-valued	• no development (phenotype = genotype) • development with and without interaction with the environment	• “roulette wheel” • elitism • rank selection • tournament • truncation • steady-state	• mutation • crossover

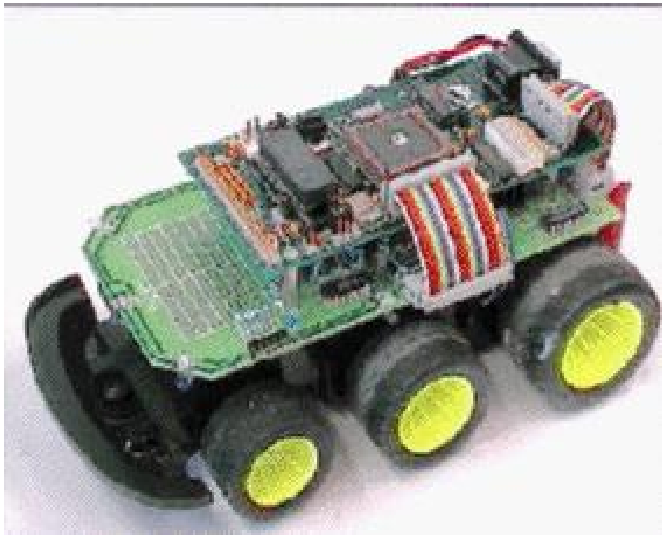
Cycle for artificial evolution



from:
"How the body…"

Evolving a neural controller for an agent

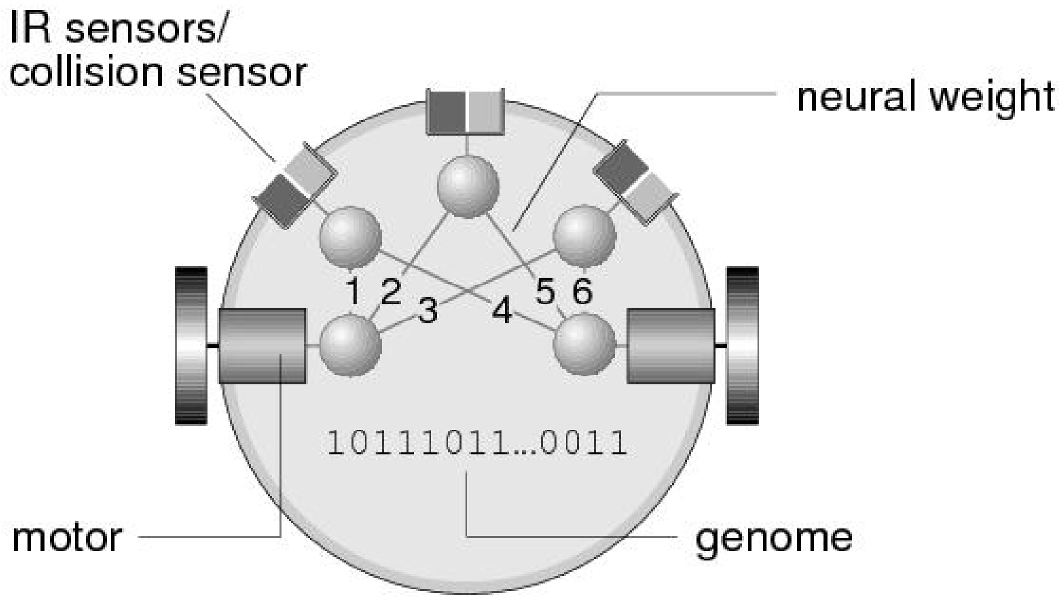




how to proceed?

Encoding in genome

a.



b.	initial genome	1101	0110	0001	0011	1010	1100
	encodes weights (numbers)	1	2	3	4	5	6
c.	initial weights after "development"	.37	−.1	−.43	−.3	.16	.3

Encoding in genome "development"

1. take the one single individual with the highest fitness
2. choose another individual from the population at random, irrespective of fitness, for sexual reproduction
3. add the fittest individual to the new population

fittest individual (highest rank)

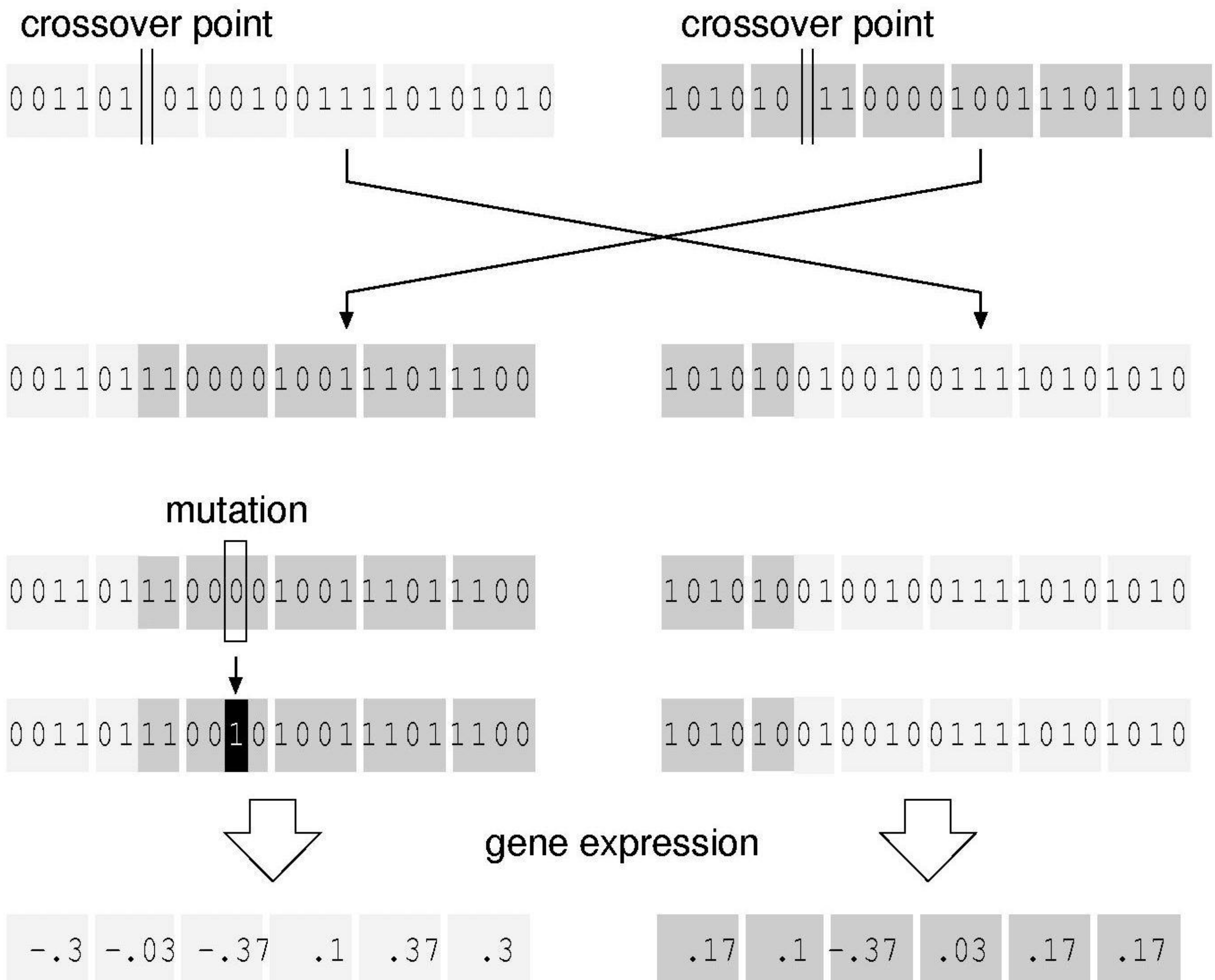
	1	2	3	4	5	6
initial genome	0011	0101	0010	0111	1010	1010
encoded weights	-.3	-.17	-.37	.03	.17	.17

other individual

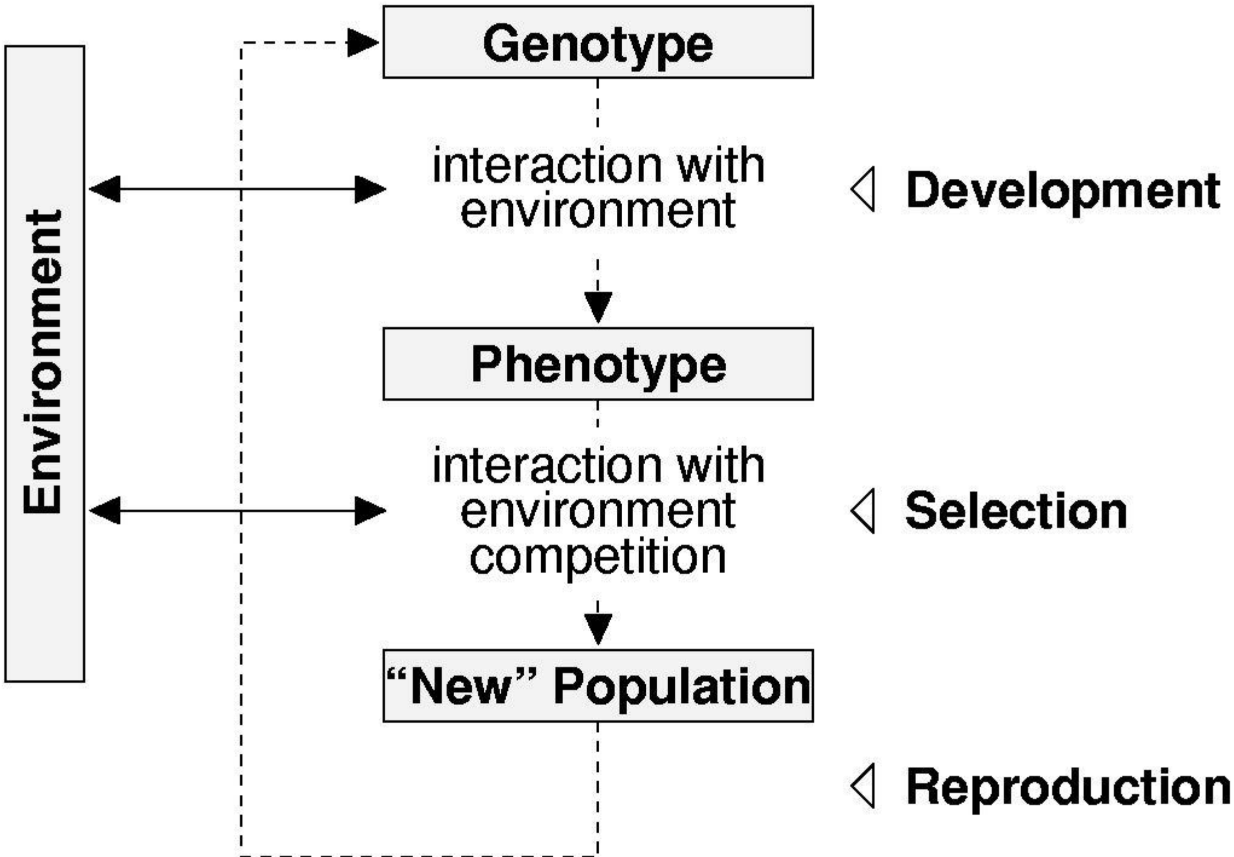
	1	2	3	4	5	6
initial genome	1010	1011	0000	1001	1101	1100
encoded weights	.17	.23	-.5	.1	.37	.3

encoding	**development**	**selection**	**reproduction**
• binary • many-character • real-valued	• no development (phenotype = genotype) • development with and without interaction with the environment	• "roulette wheel" • elitism • rank selection • tournament • truncation • steady-state	• mutation • crossover

Reproduction: Crossover and mutation “development”



“Grand” evolutionary scheme



encoding	development	selection	reproduction
• binary • many-character • real-valued	• no development (phenotype = genotype) • development with and without interaction with the environment	• “roulette wheel” • elitism • rank selection • tournament • truncation • steady-state	• mutation • crossover

Summary

- Informed Search Algorithms
- Beyond classical search algorithms
  - Local search algorithms
  - Simulated annealing search
  - Genetic algorithms
- Still challenging
  - Neural Networks (NN, ANN)
  - EA (GA, EP, ES, GP)
  - Deep Learning
  - ……

Assignment

– Readings: Chap 4
– Chap 4: exercise 4.1

*Handed in next Tuesday