Informed Search

Philipp Koehn

21 February 2019

Heuristic

From Wikipedia:

any approach to problem solving, learning, or discovery
that employs a practical method
not guaranteed to be optimal or perfect
but sufficient for the immediate goals

Outline

- Best-first search
- A* search
- Heuristic algorithms
  - hill-climbing
  - simulated annealing
  - genetic algorithms (briefly)
  - local search in continuous spaces (very briefly)

best-first search

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 return node
       fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

- Search space is in form of a tree
- 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

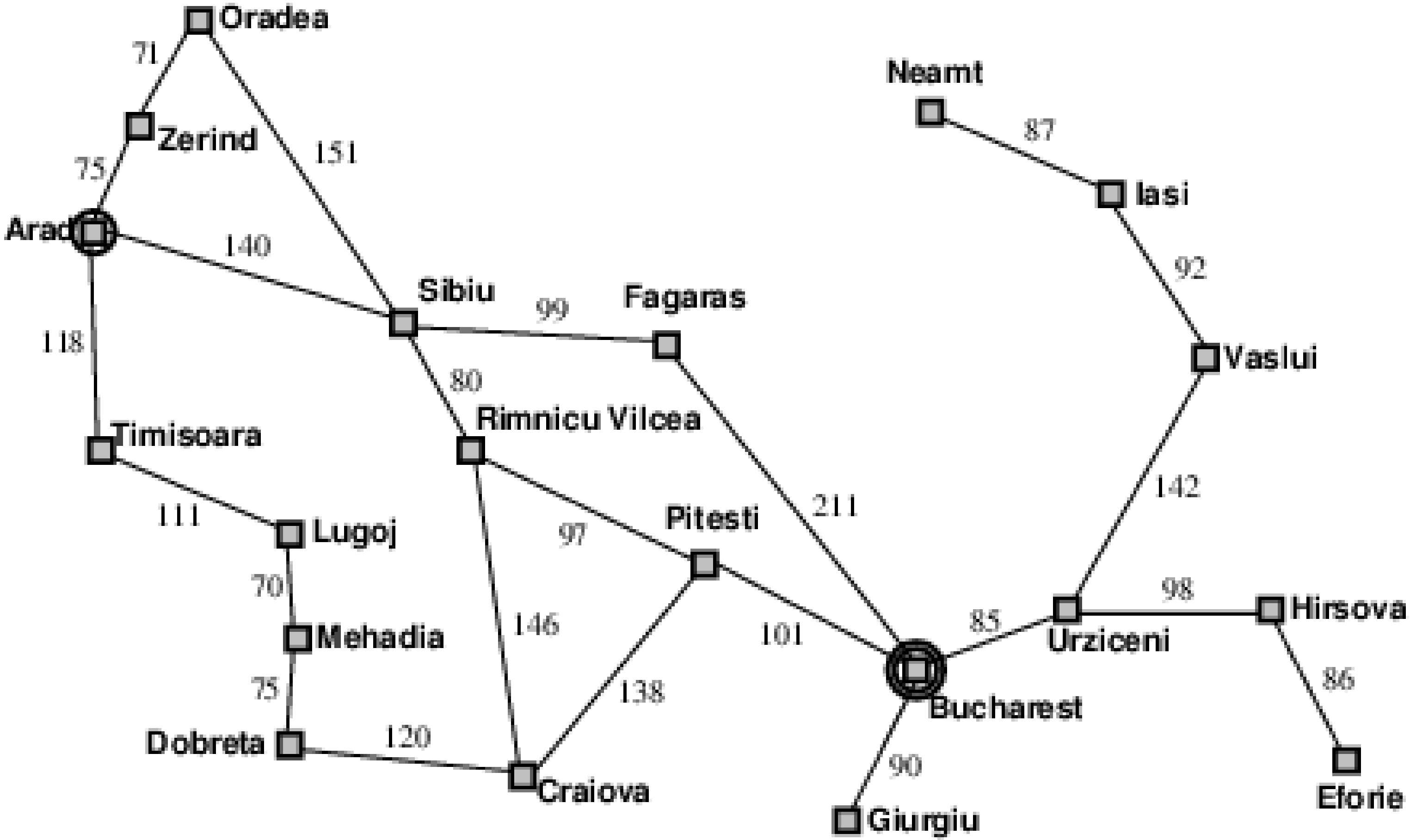
- Implementation:
  fringe is a queue sorted in decreasing order of desirability

- Special cases
  - greedy search
  - A^* search

Romania


Legend:
International border
Motorway
Motorway under construction
Main road
Secondary road
Railroad
Airport
Port
Cities and towns (nr. of inhabitants):
> 1.000.000
300.000 - 1.000.000
100.000 - 300.000
50.000 - 100.000
20.000 - 50.000
< 20.000
BUCHAREST National capital
Major urban concentration (over 500.000 inhabitants)
Scale of altitudes and depths:
3000 m
2500 m
2000 m
1500 m
1000 m
500 m
250 m
100 m
50 m
0 m
-500 m
-1000 m
-1500 m
-2000 m
Scale: 1:1.800.000
0 (km) 100
0 (mi) 60
STEREO 70 projection, S-42 (Romania) datum
UKRAINE
HUNGARY
SERBIA
BULGARIA
MOLDOVA
BLACK SEA
TRANSYLVANIAN PLATEAU
WALLACHIAN PLAIN
DOBRUJAN PLATEAU
DANUBE DELTA
BUCHAREST
CLUJ-NAPOCA
TIMIŞOARA
IAŞI
CONSTANŢA
CRAIOVA
BRAŞOV
GALAŢI
PLOIEŞTI
ORADEA
BRĂILA
ARAD
PITEŞTI
SIBIU
BACĂU
TÂRGU MUREŞ
BAIA MARE
BUZĂU
BOTOŞANI
SATU MARE
CHIŞINĂU
BELGRADE
DEBRECEN
SZEGED
RUSE
CHERNIVTSI
TIRASPOL
Danube

Romania


Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Lugoj
97
Pitesti
70
98
Mehadia
146
85
Hirsova
101
Urziceni
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

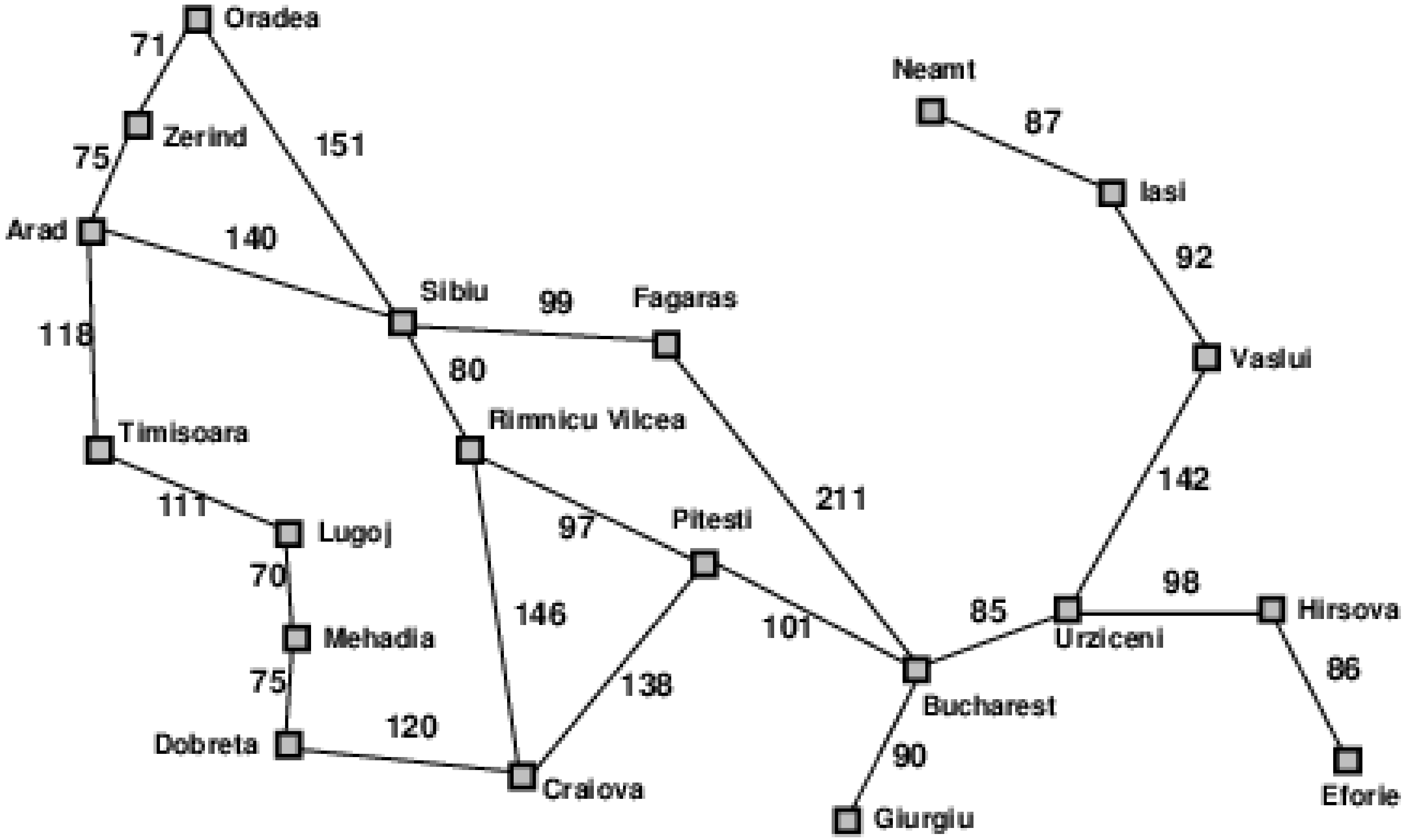
Greedy Search

- State evaluation function $h(n)$ (**heuristic**)
  = estimate of cost from n to the closest goal

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

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

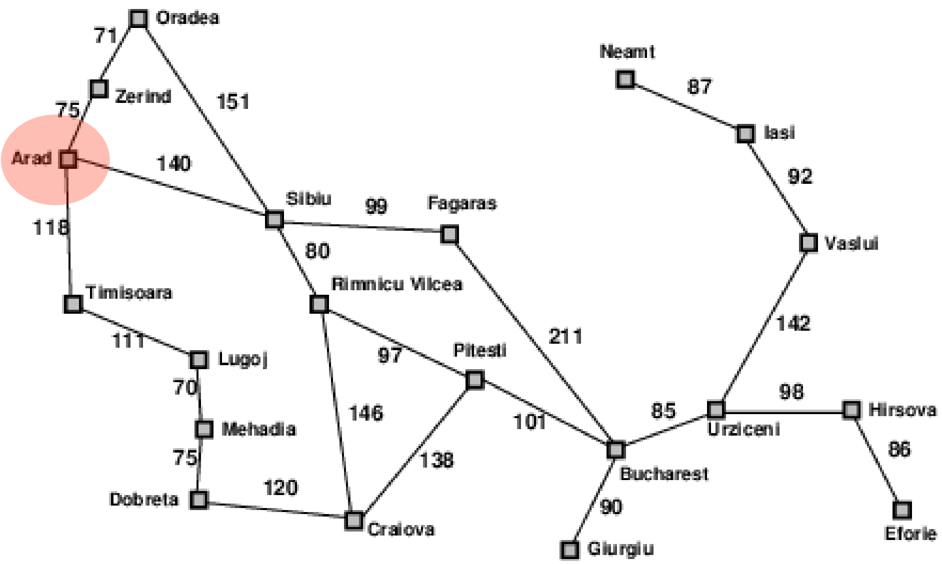
Romania with Step Costs in km



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

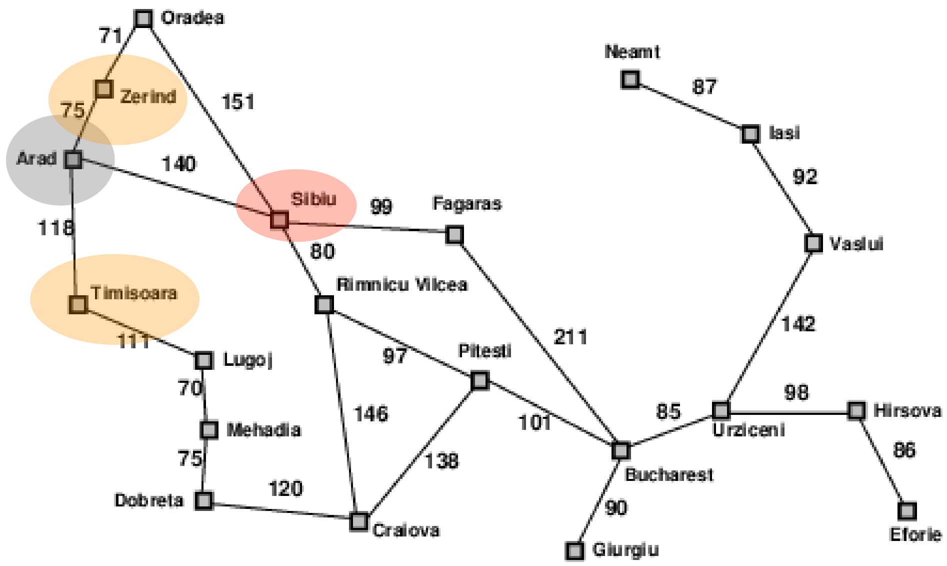
Greedy Search Example



Straight–line distance to Bucharest

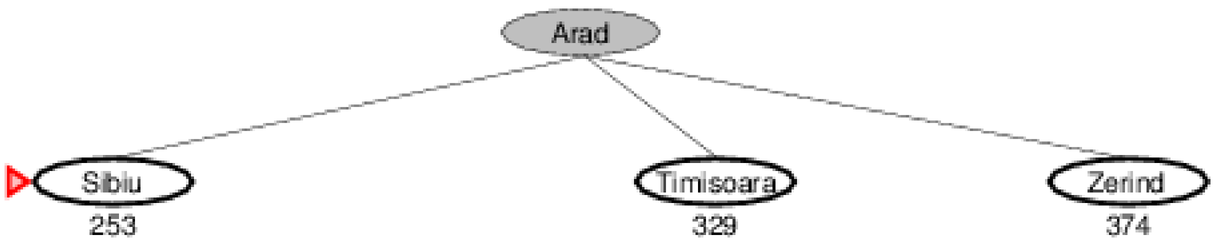
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search Example

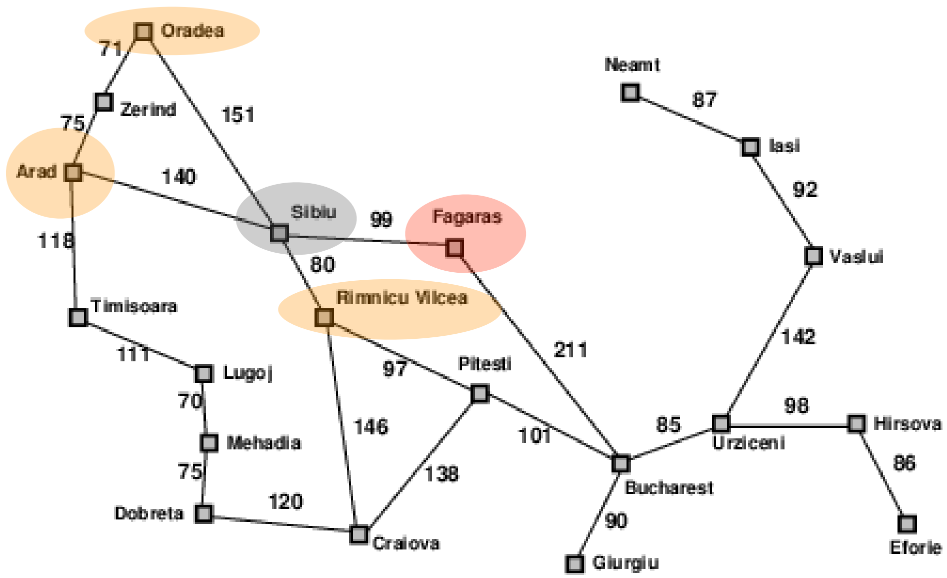


Straight−line distance to Bucharest

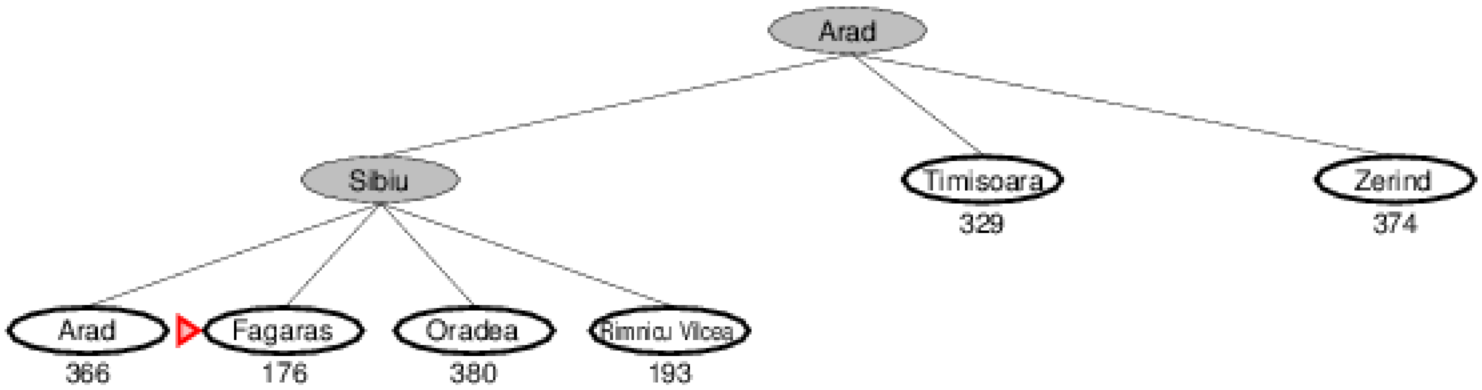
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



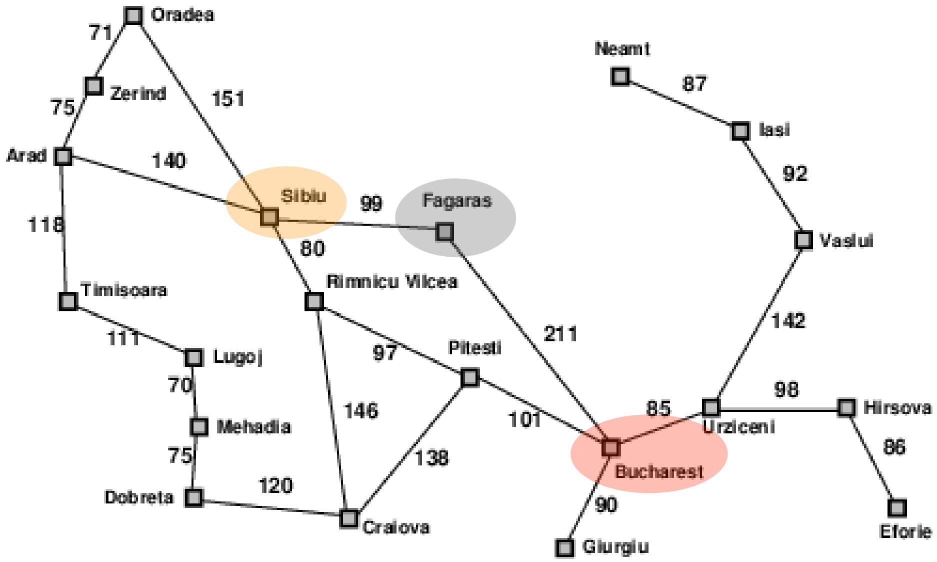
Greedy Search Example



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

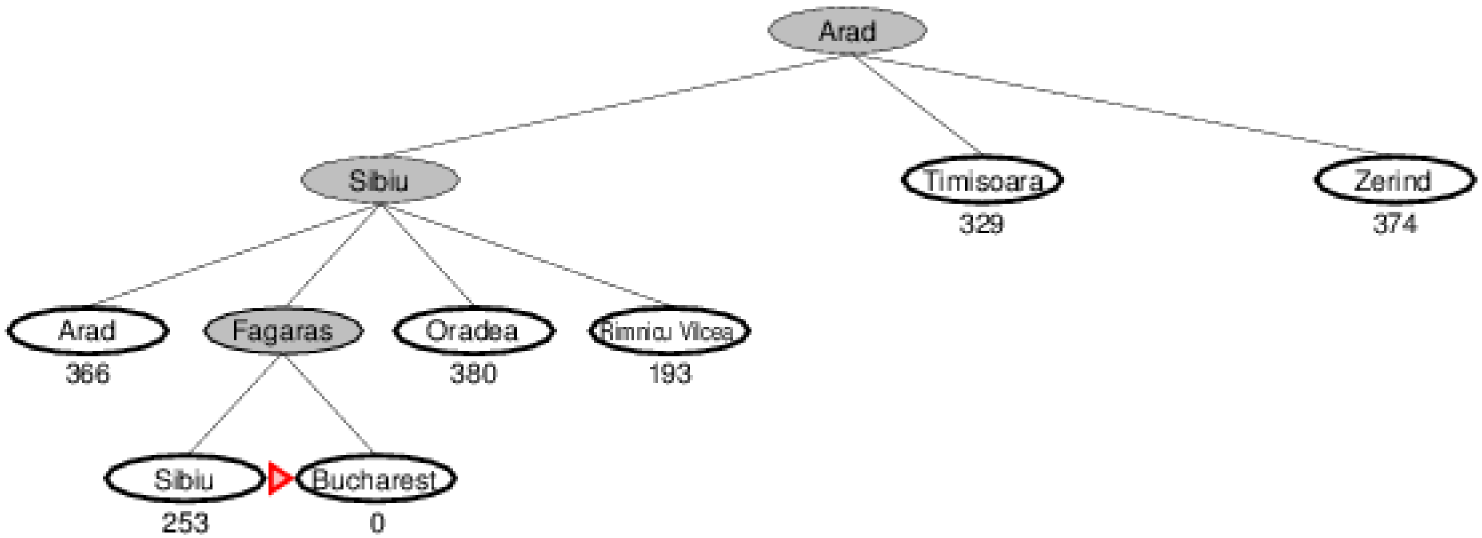


Greedy Search Example



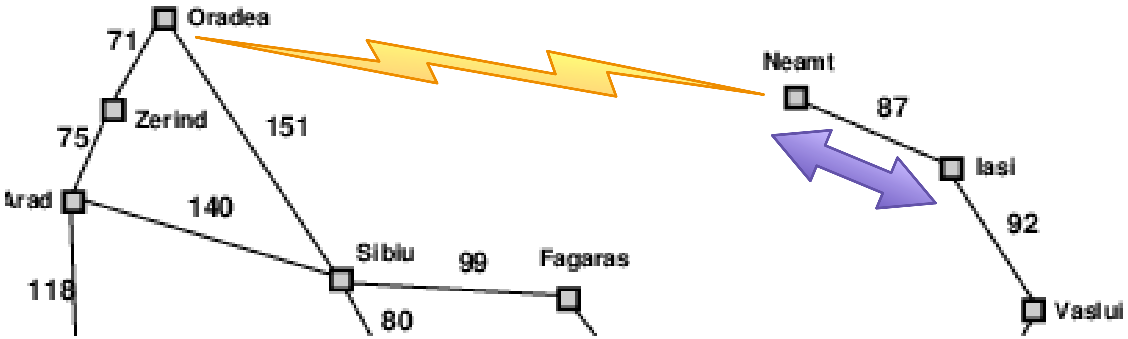
Straight–line distance to Bucharest

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



Properties of Greedy Search

- Complete? No, can get stuck in loops, e.g., with Oradea as goal,
  Iasi → Neamt → Iasi → 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

a* search

A* Search

- Idea: avoid expanding paths that are already expensive

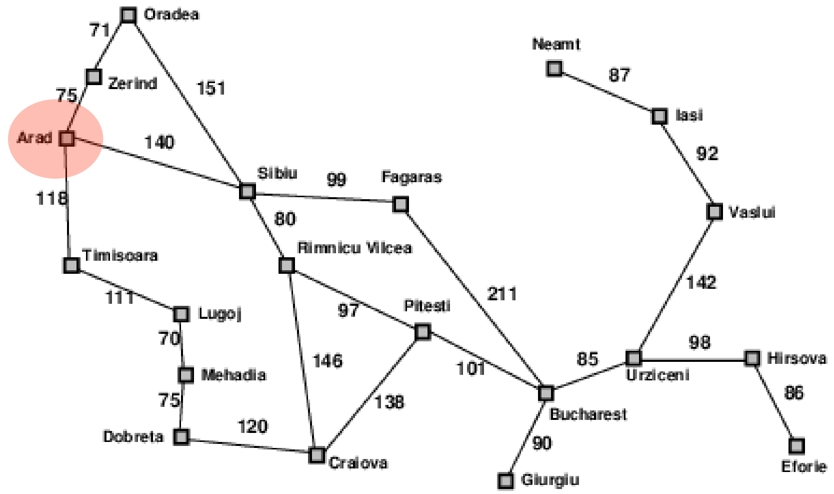
- State evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost to goal from n
  - $f(n)$ = estimated total cost of path through n to goal

- A* search uses an admissible heuristic
  - i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n
  - also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G

- E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

- Theorem: A* search is optimal

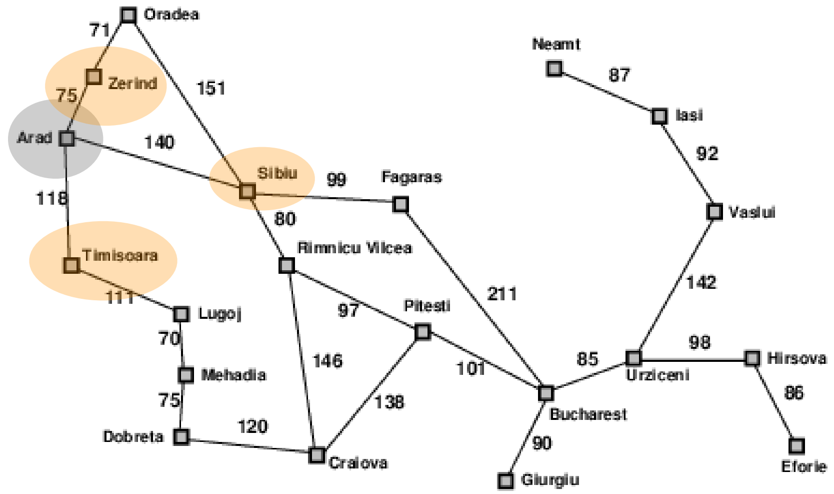
A* Search Example



Straight–line distance to Bucharest

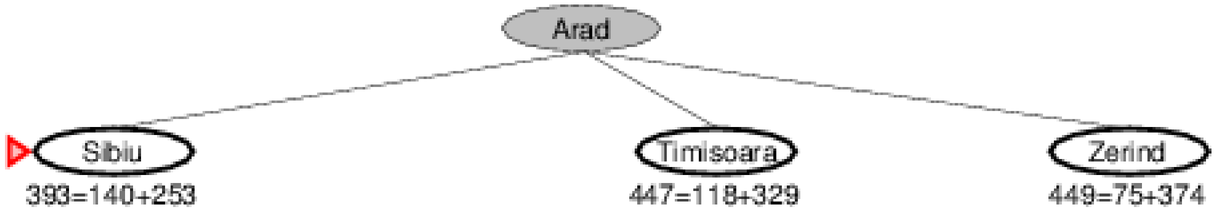
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A^* Search Example

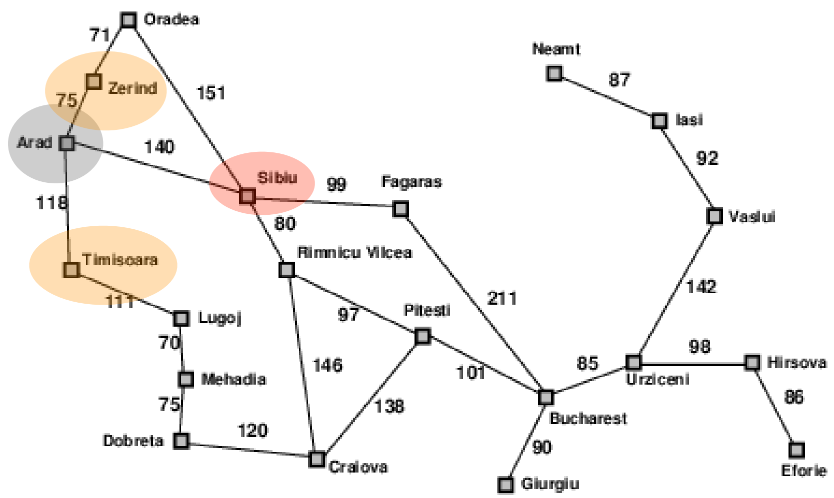


Straight–line distance to Bucharest

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

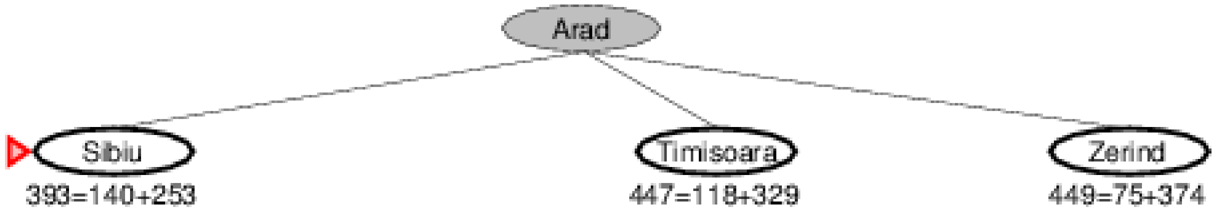


A* Search Example

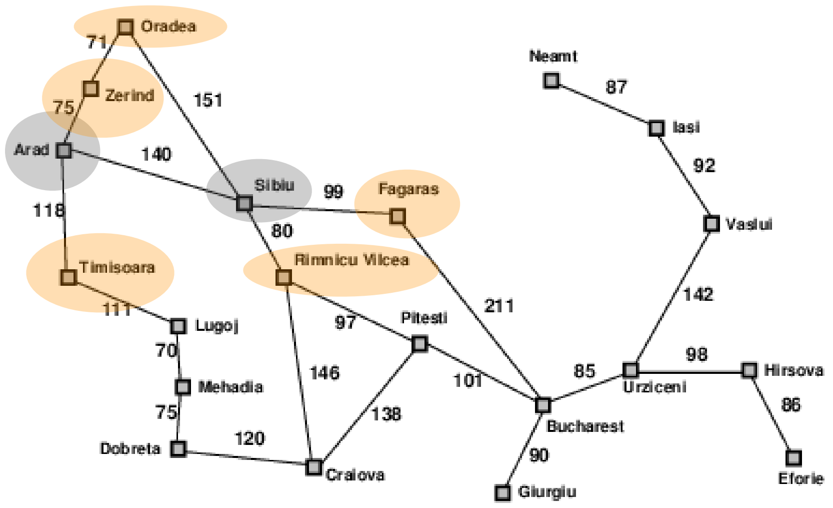


Straight–line distance to Bucharest

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

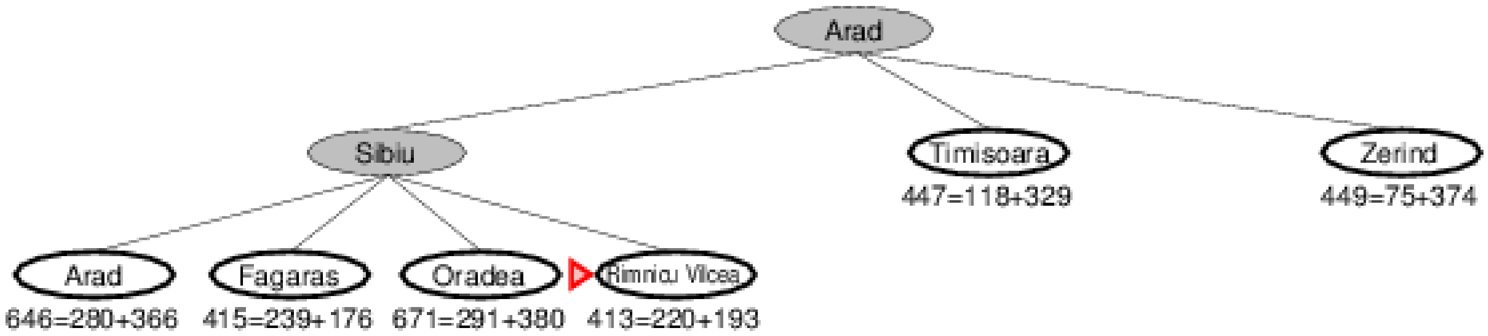


A* Search Example

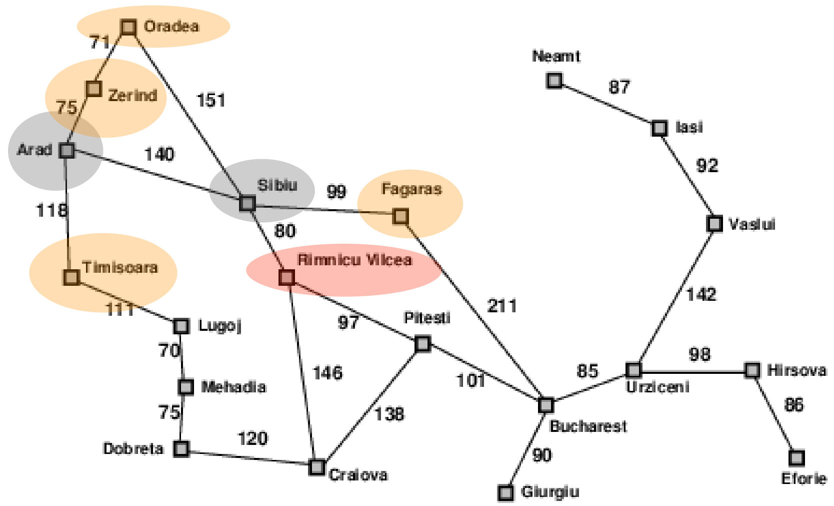


Straight–line distance to Bucharest

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

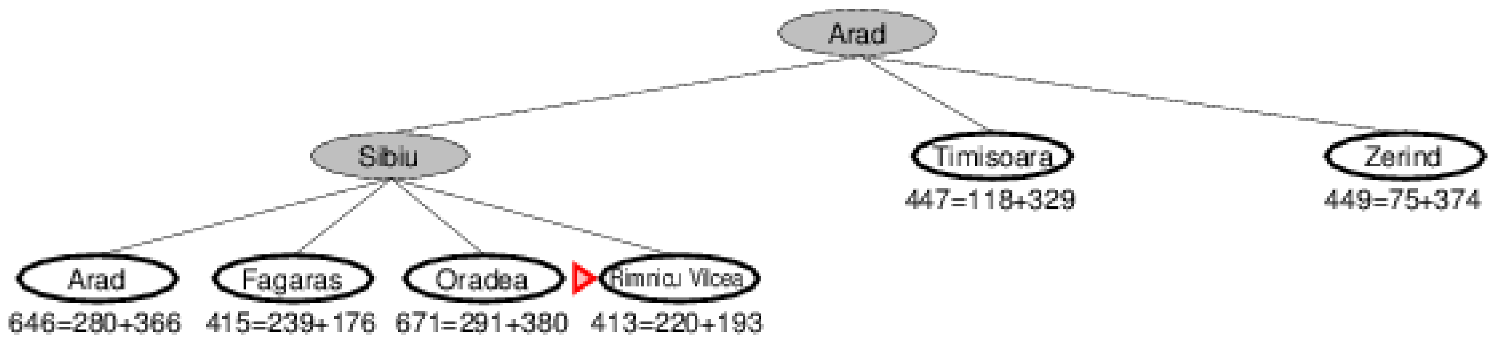


A* Search Example

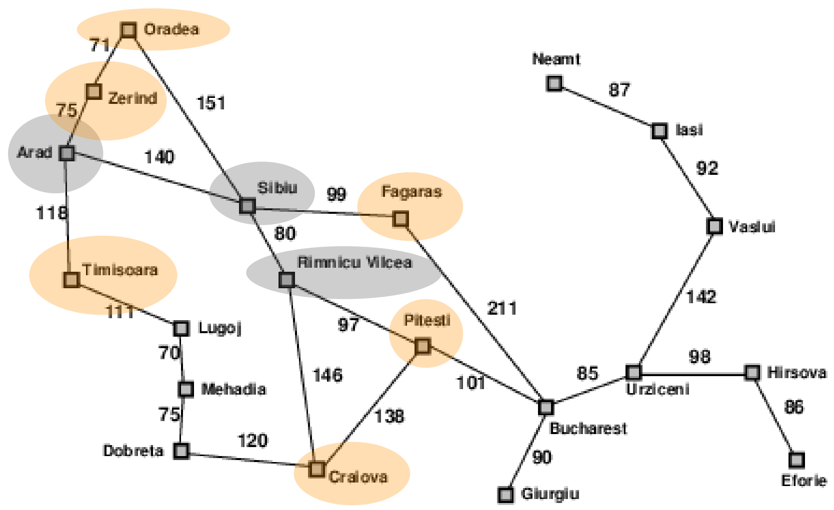


Straight–line distance to Bucharest

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

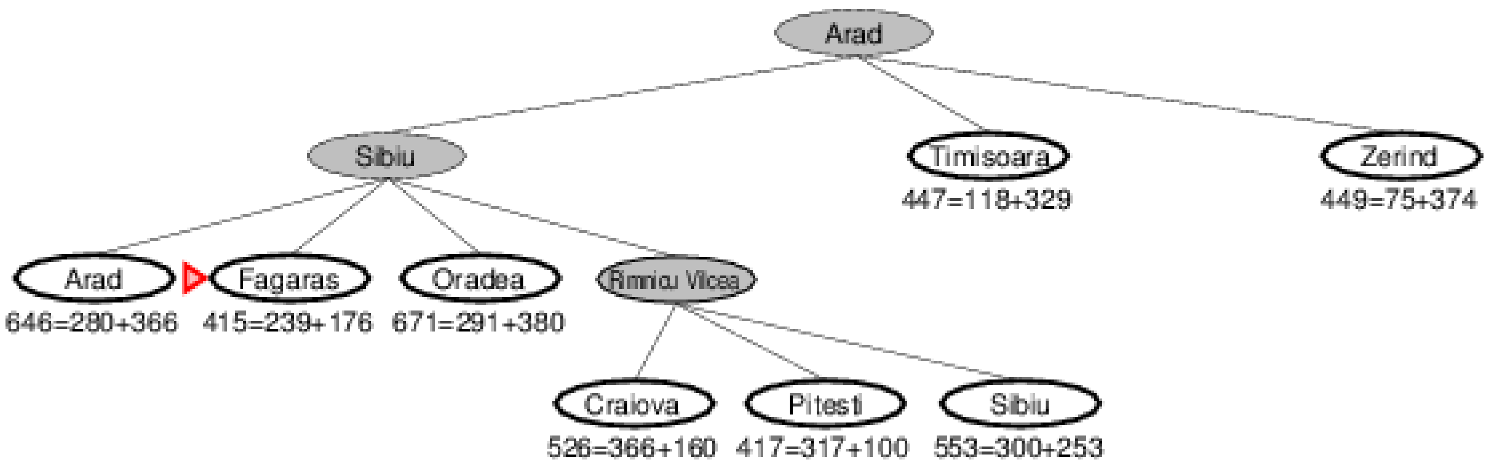


A* Search Example

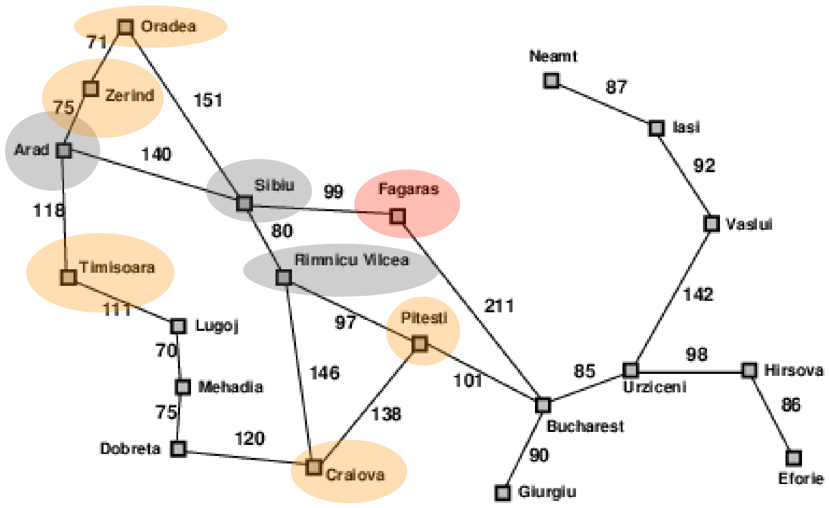


Straight–line distance to Bucharest

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

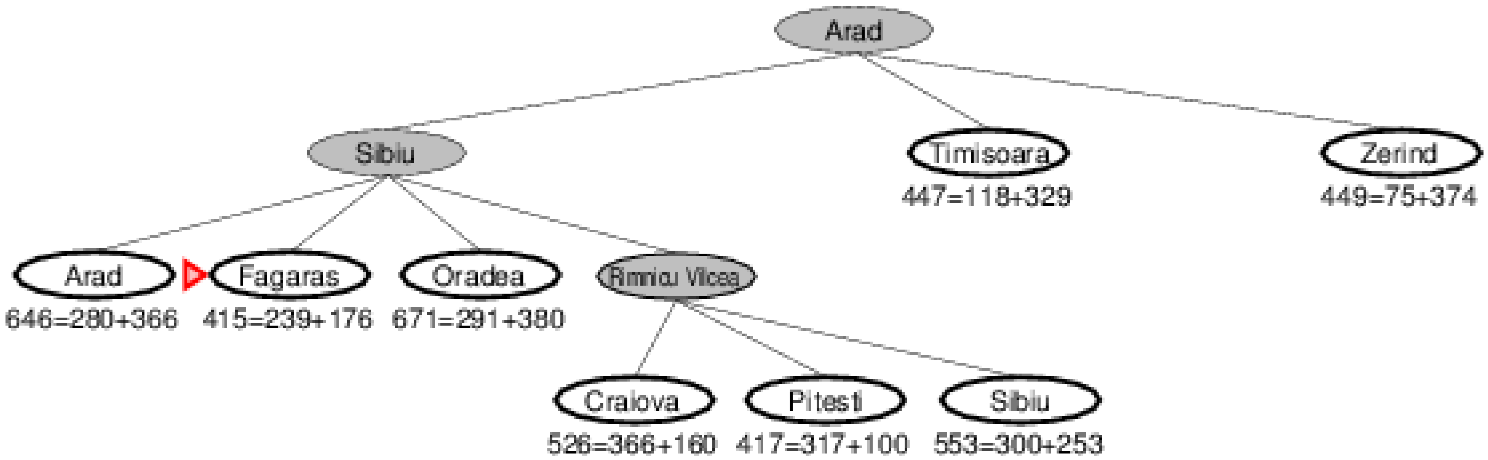


A^* Search Example

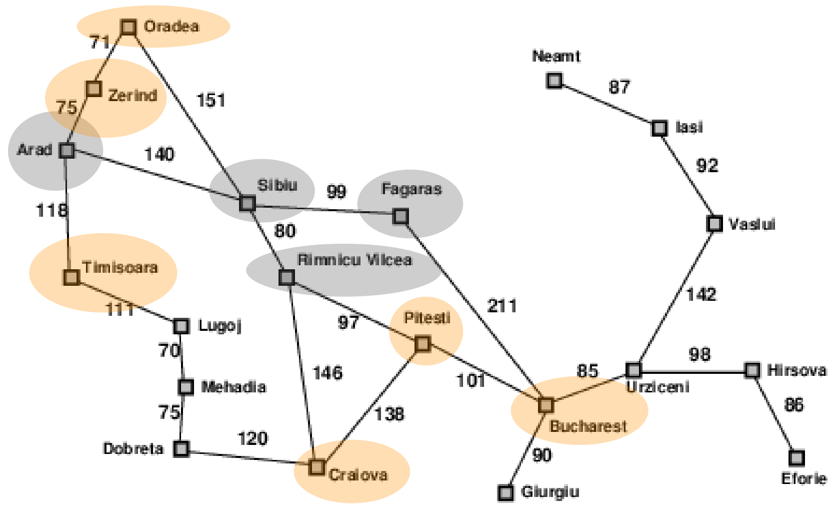


Straight–line distance to Bucharest

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

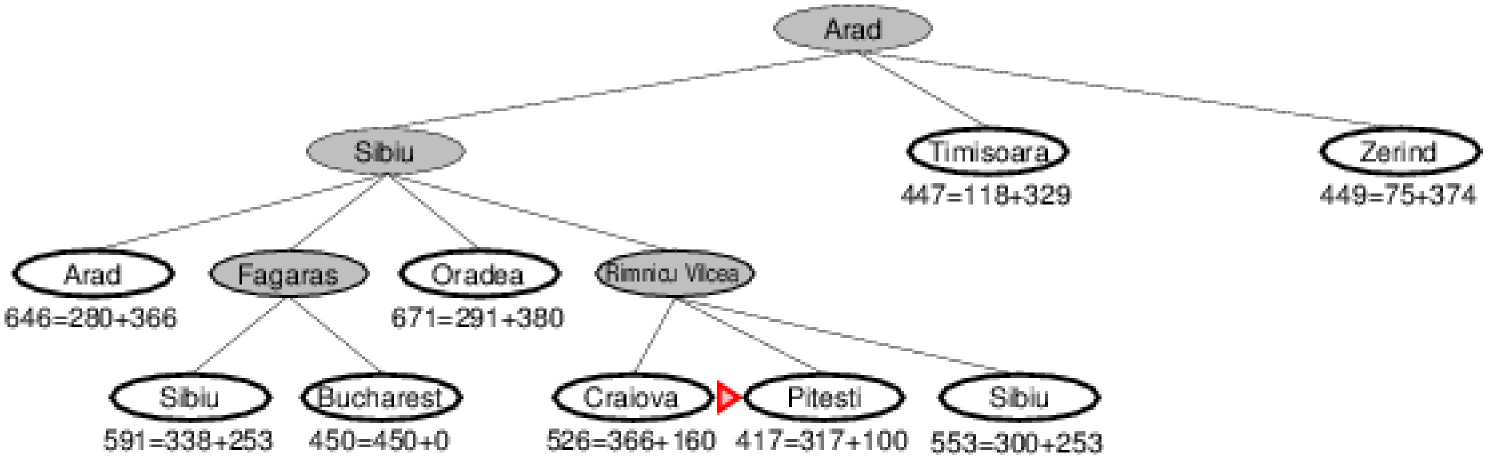


A* Search Example

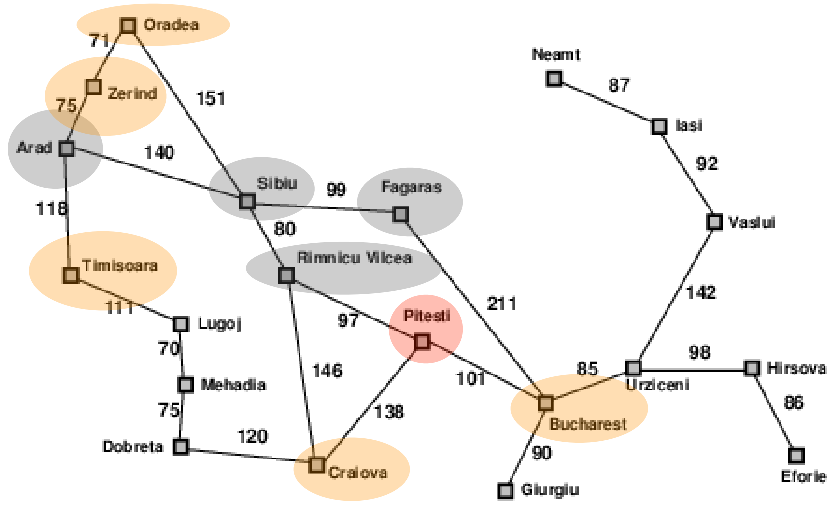


Straight–line distance to Bucharest

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

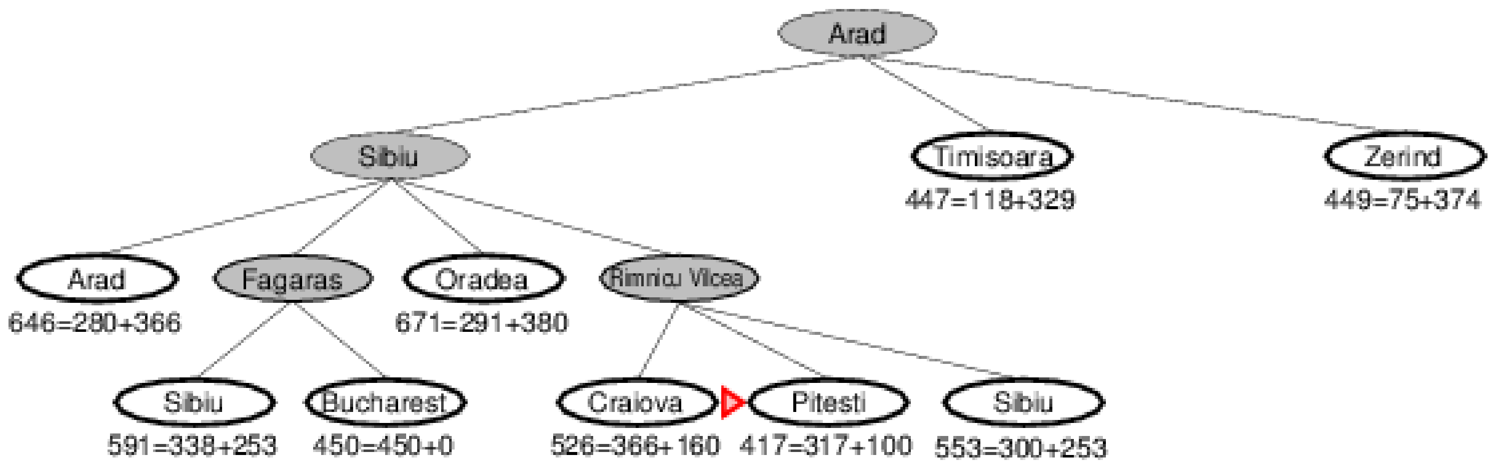


A* Search Example

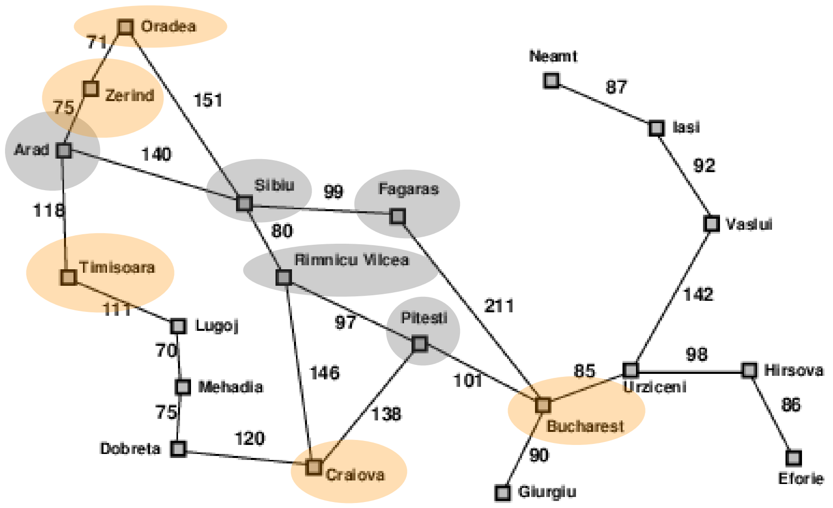


Straight-line distance to Bucharest

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

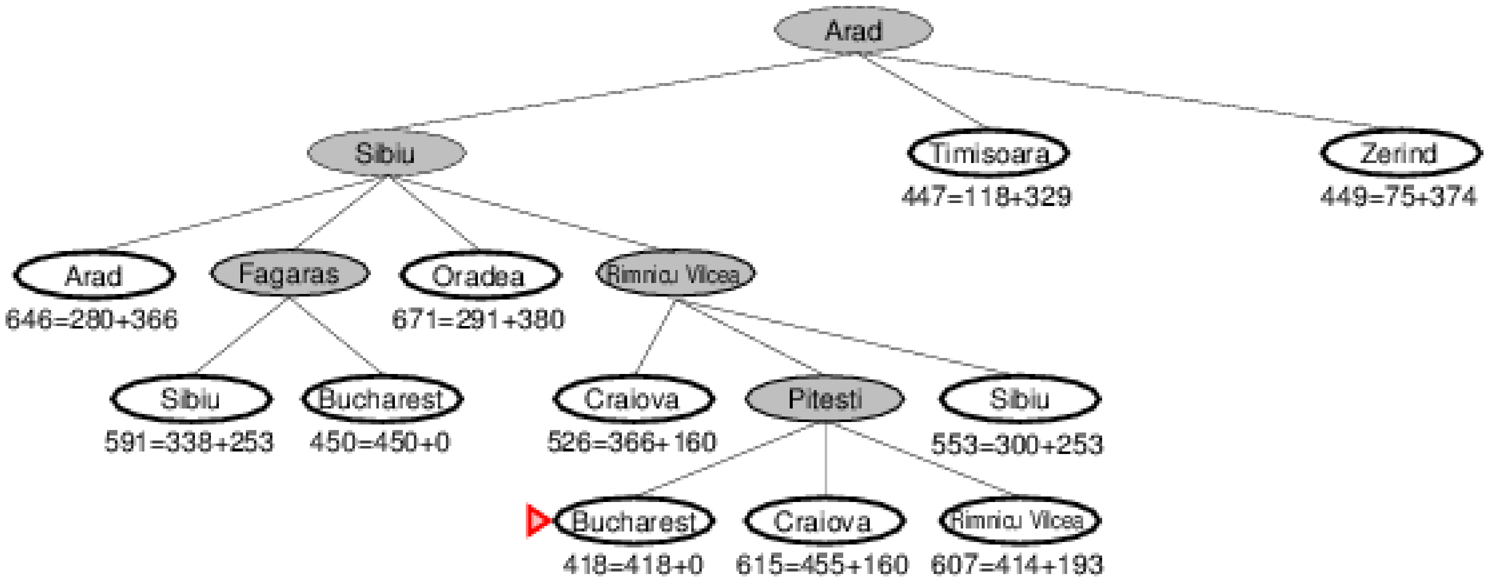


A* Search Example

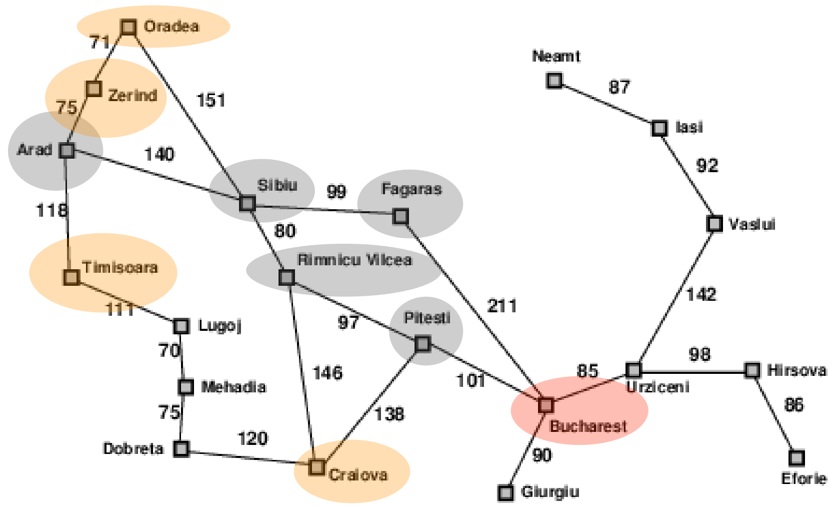


Straight−line distance
to Bucharest

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

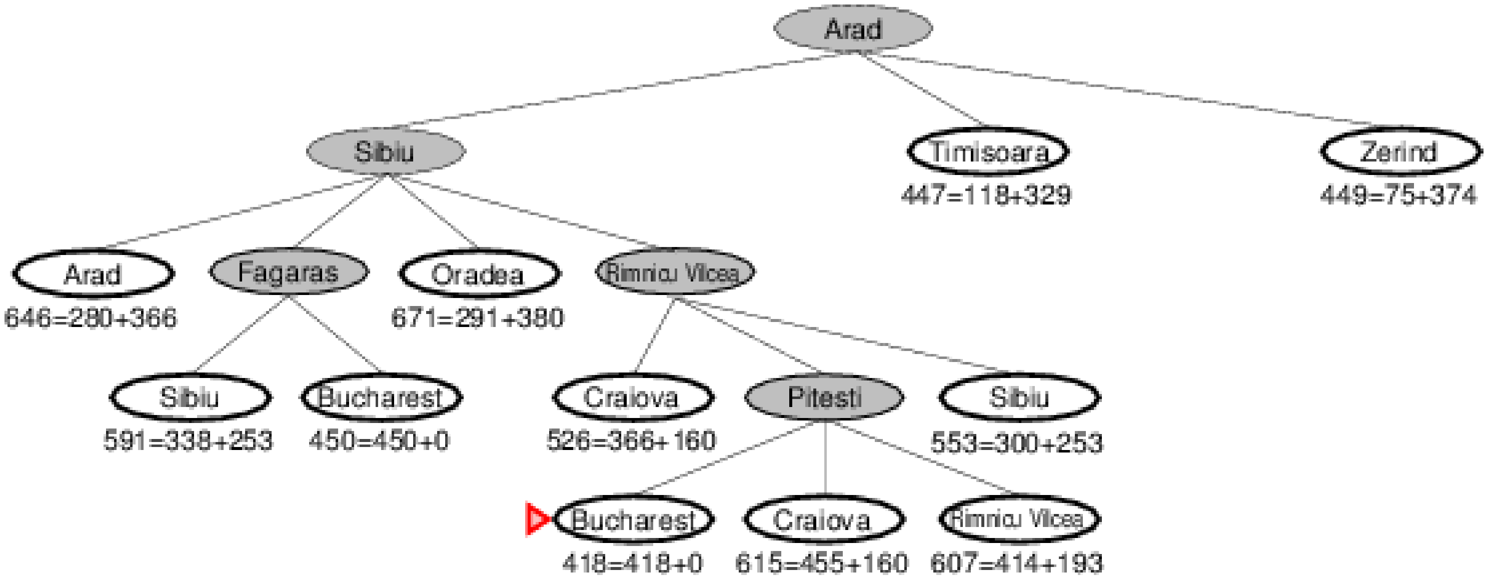


A* Search Example



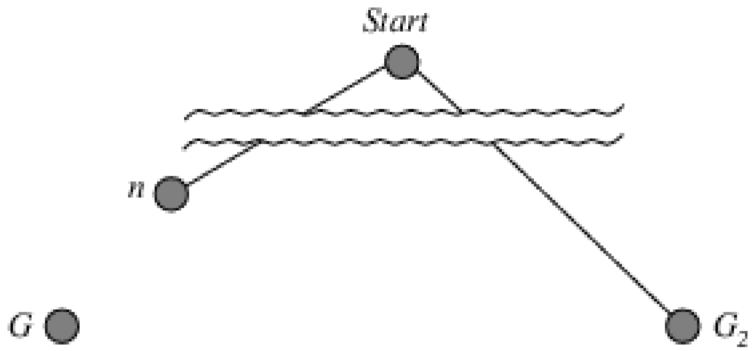
Straight–line distance to Bucharest

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



Optimality of A*

- 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



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

- Since $f(G_2) > f(n)$, A* will never terminate at G_2

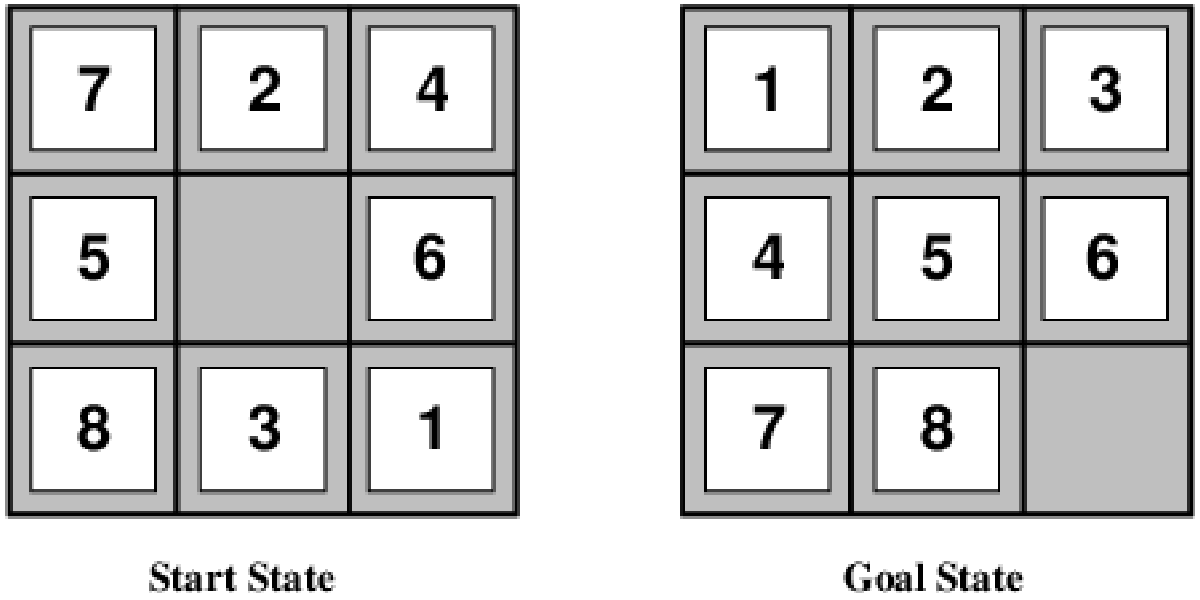
Properties of A*

- Complete? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time? Exponential in [relative error in $h \times$ length of solution]
- 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 $f(n) > C^*$

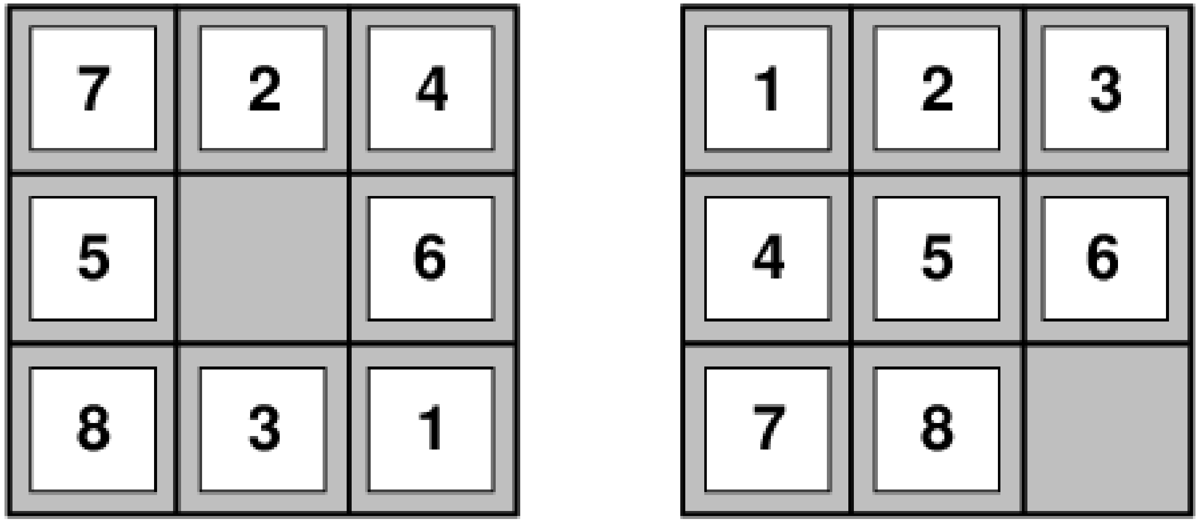
Admissible Heuristics

- E.g., for the 8-puzzle



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)

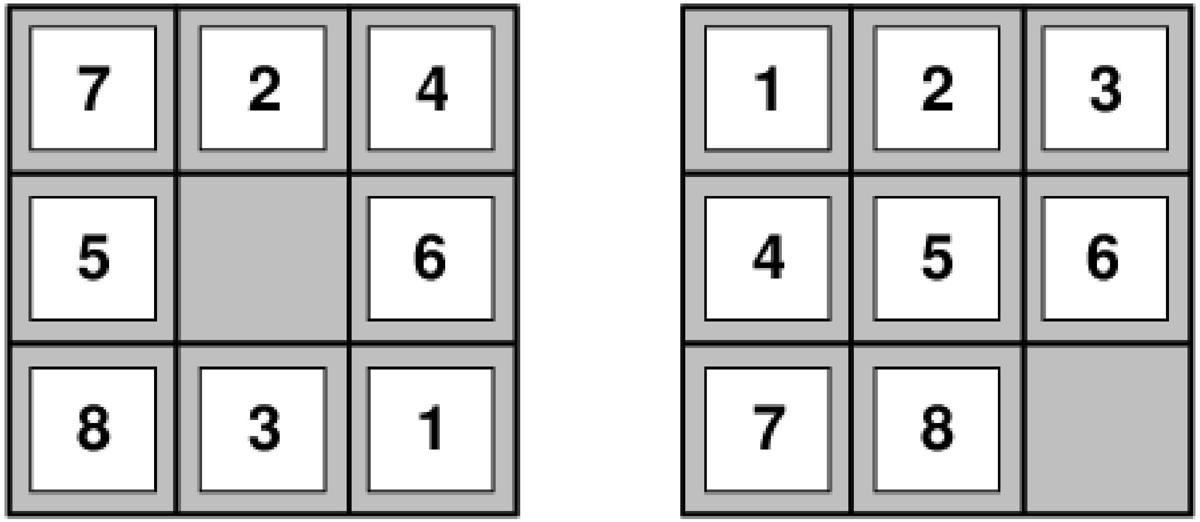


Start State   Goal State

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

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

- $h_1(S) =?$ 6
- $h_2(S) =?$ 4+0+3+3+1+0+2+1 = 14

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
  $\rightarrow h_2$ dominates h_1 and is better for search

- Typical search costs (d = depth of solution for 8-puzzle)

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

- Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

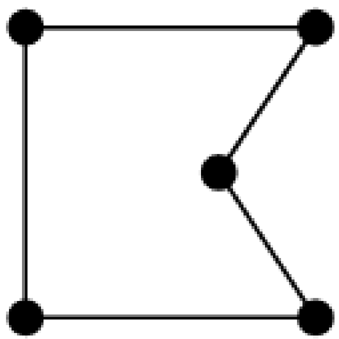
  is also admissible and dominates h_a, h_b

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** $\Rightarrow h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to **any adjacent square** $\Rightarrow h_2(n)$ 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

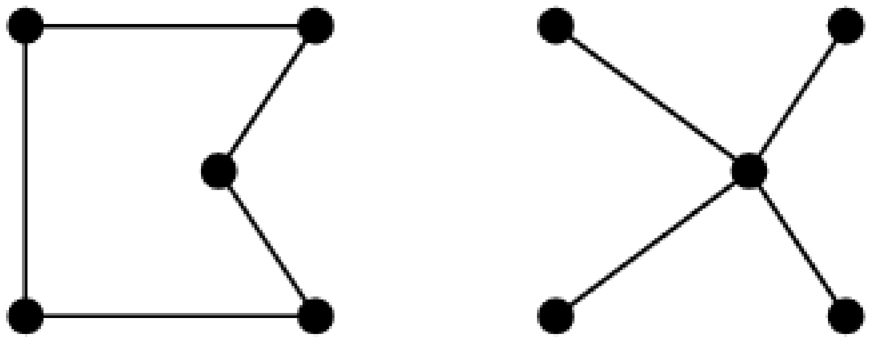
Relaxed Problems

- Well-known example: travelling salesperson problem (TSP)
- Find the shortest tour visiting all cities exactly once



Relaxed Problems

- Well-known example: travelling salesperson problem (TSP)
- Find the shortest tour visiting all cities exactly once



- Minimum spanning tree
  - can be computed in $O(n^2)$
  - is a lower bound on the shortest (open) tour

Summary: A*

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
  - incomplete and not always optimal
- A* search expands lowest $g + h$
  - complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search)
- Admissible heuristics can be derived from exact solution of relaxed problems

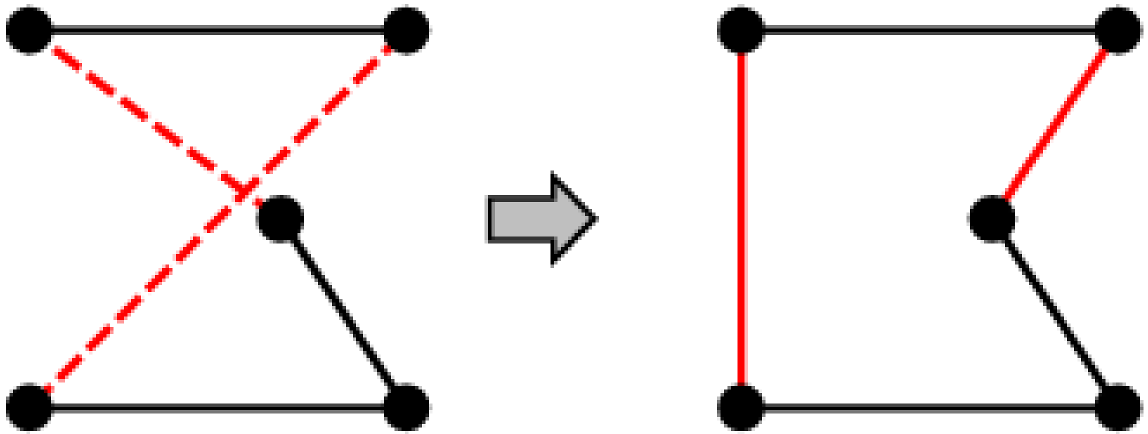
iterative improvement algorithms

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
  - 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

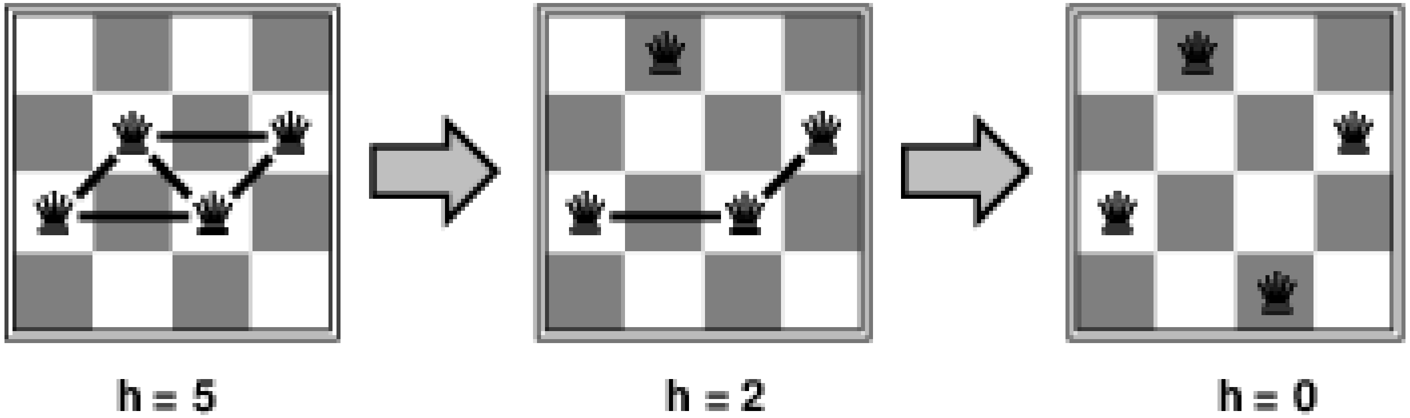


- Variants of this approach get within 1% of optimal quickly with 1000s of cities

Example: n-Queens

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

- Move a queen to reduce number of conflicts



- Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1$ million

Hill-Climbing

- For instance Gradient Ascent (or Descent)

- "Like climbing Everest in thick fog with amnesia"

1. Start state = a solution (maybe randomly generated)
2. Consider neighboring states, e.g.,
   - move a queen
   - pairwise exchange in traveling salesman problem
3. No better neighbors? Done.
4. Adopt best neighbor state
5. Go to step 2

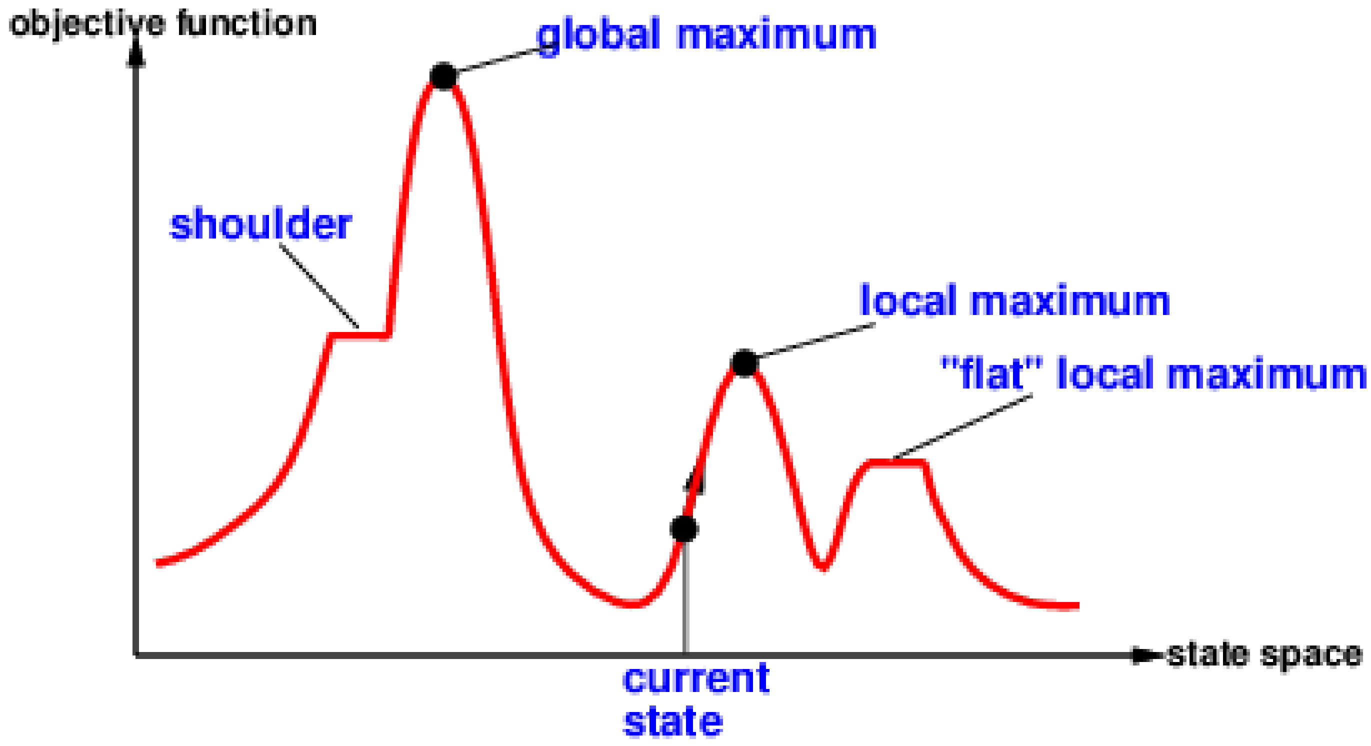
Hill-Climbing


```
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
```

Hill-Climbing

- Useful to consider state space landscape



- Random-restart hill climbing overcomes local maxima—trivially complete
- Random sideways moves ☺ escape from shoulders ☹ loop on flat maxima

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*
    ∆*E* ← VALUE[*next*] – VALUE[*current*]
    if ∆*E* > 0 **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Properties of Simulated Annealing

- At fixed "temperature" T, state occupation probability reaches Boltzman distribution

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

- T decreased slowly enough $\Longrightarrow$ always reach best state x^* because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

- Is this necessarily an interesting guarantee?

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

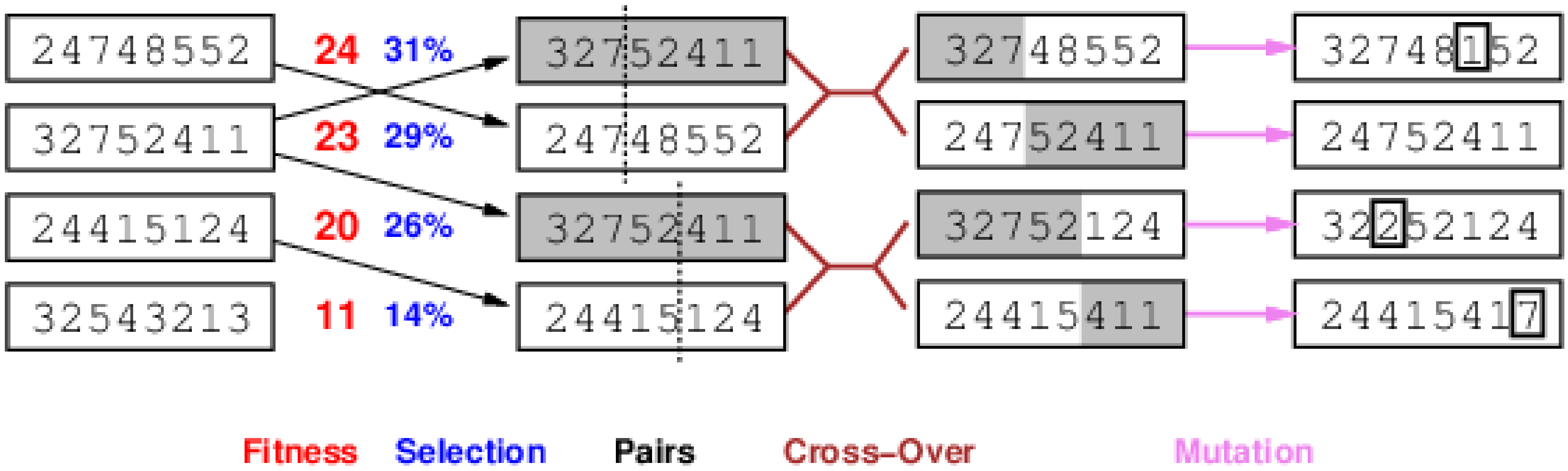
- Widely used in VLSI layout, airline scheduling, etc.

Local Beam Search

- Idea: keep k states instead of 1; choose top k of all their successors
- Not the same as k searches run in parallel!
- Problem: quite often, all k states end up on same local hill
- Idea: choose k successors randomly, biased towards good ones
- Observe the close analogy to natural selection!

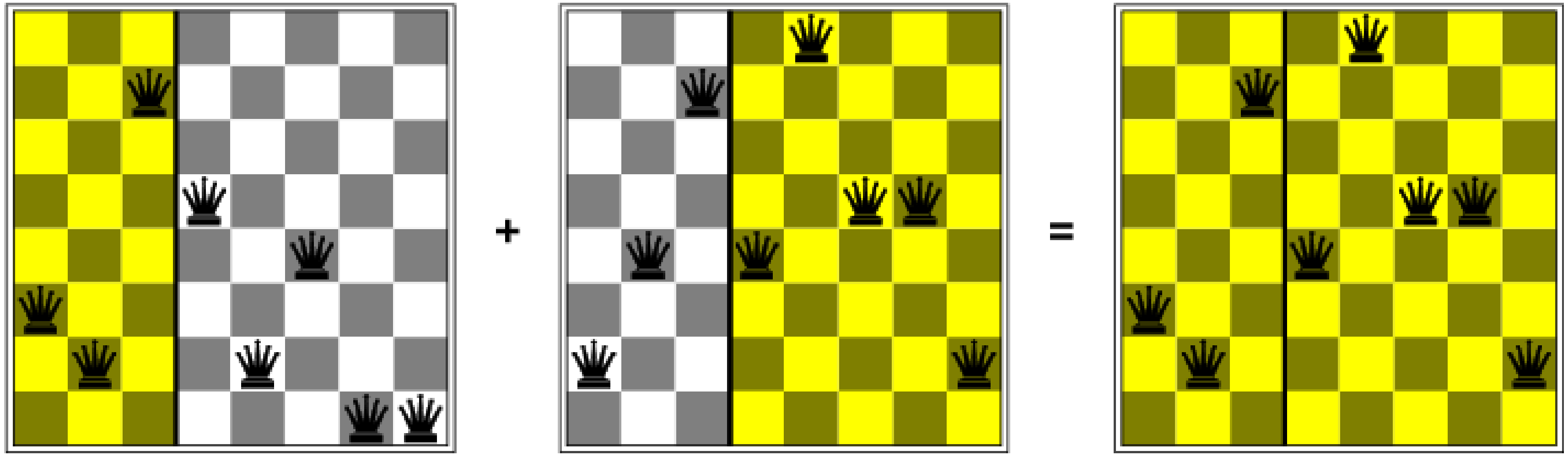
Genetic Algorithms

- Stochastic local beam search + generate successors from **pairs** of states



Genetic Algorithms

- GAs require states encoded as strings (GPs use programs)
- Crossover helps **iff substrings are meaningful components**



Summary

- Exact search
  - exhaustive exploration of the search space
  - search with heuristics: a*

- Approximate search
  - hill-climbing
  - simulated annealing
  - local beam search (briefly)
  - genetic algorithms (briefly)