Informed search algorithms

Outline

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

Best-first search

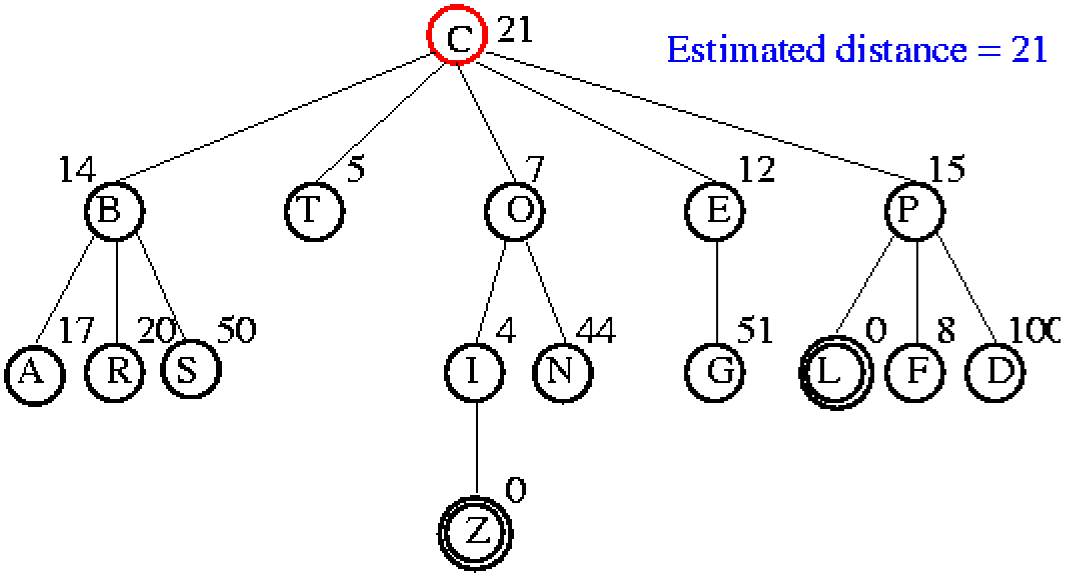
- Idea: use an evaluation function *f(n)* for each node
  - f(n) provides an estimate for the total cost.
  - → Expand the node n with smallest f(n).

- Implementation:

  Order the nodes in fringe increasing order of cost.

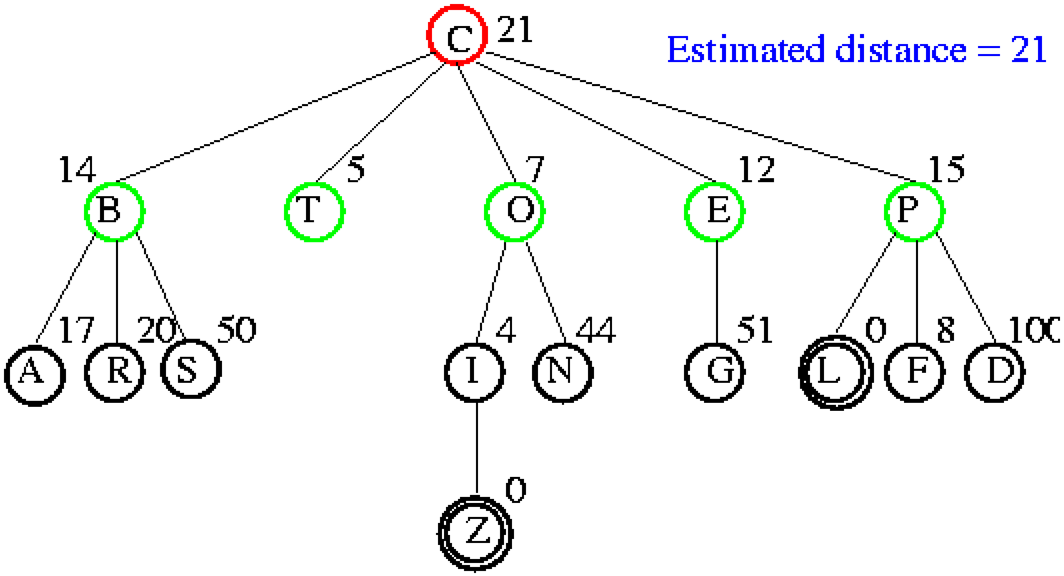
- Special cases:
  - greedy best-first search
  - A^* search

Example



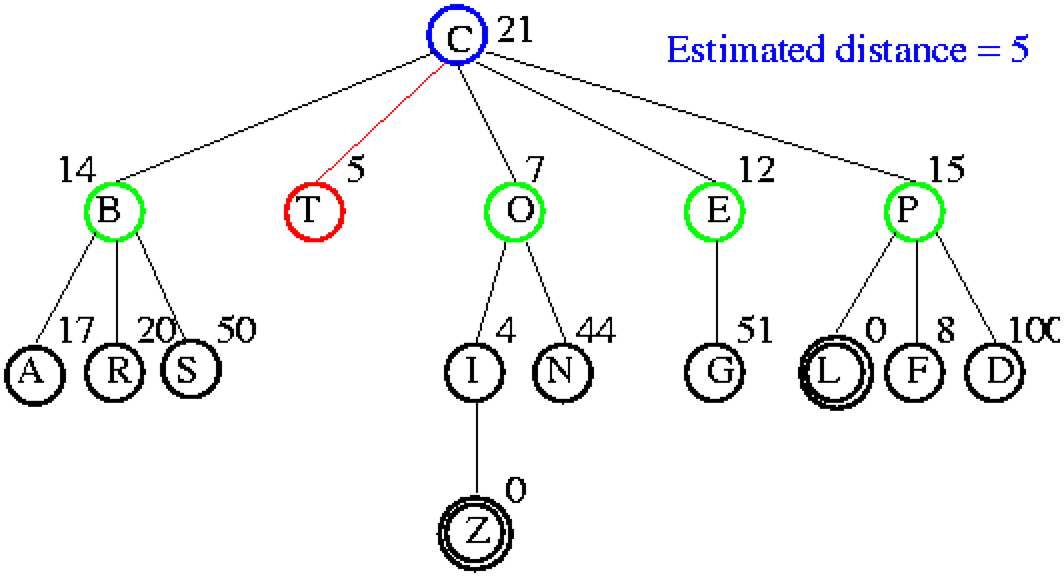
C 21
Estimated distance = 21
14
B
T
5
O
7
E
12
P
15
A
17
R
20
S
50
I
4
N
44
G
51
L
0
F
8
D
100
Z
0

Example



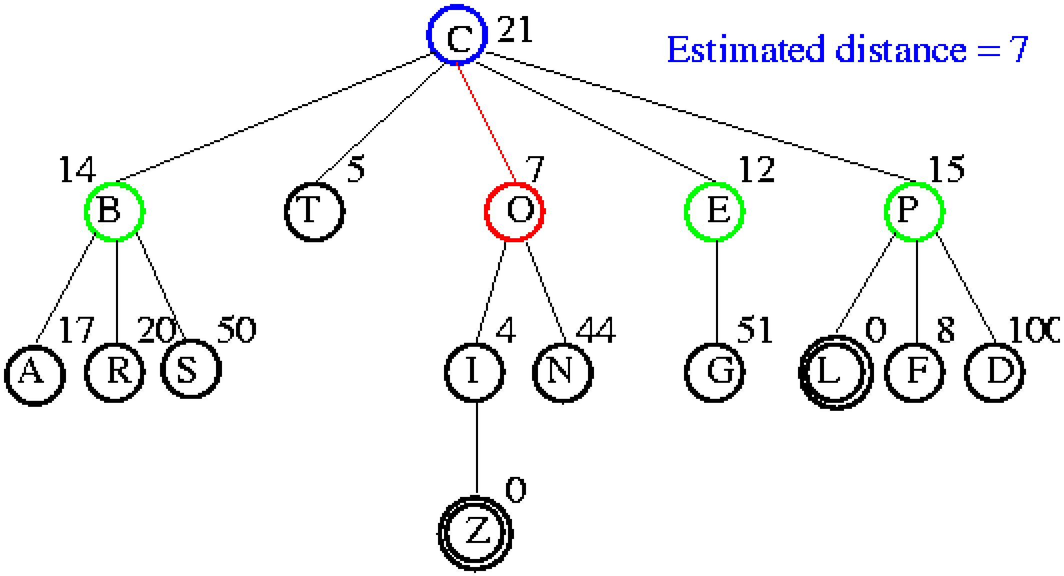
C 21
Estimated distance = 21
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

Example



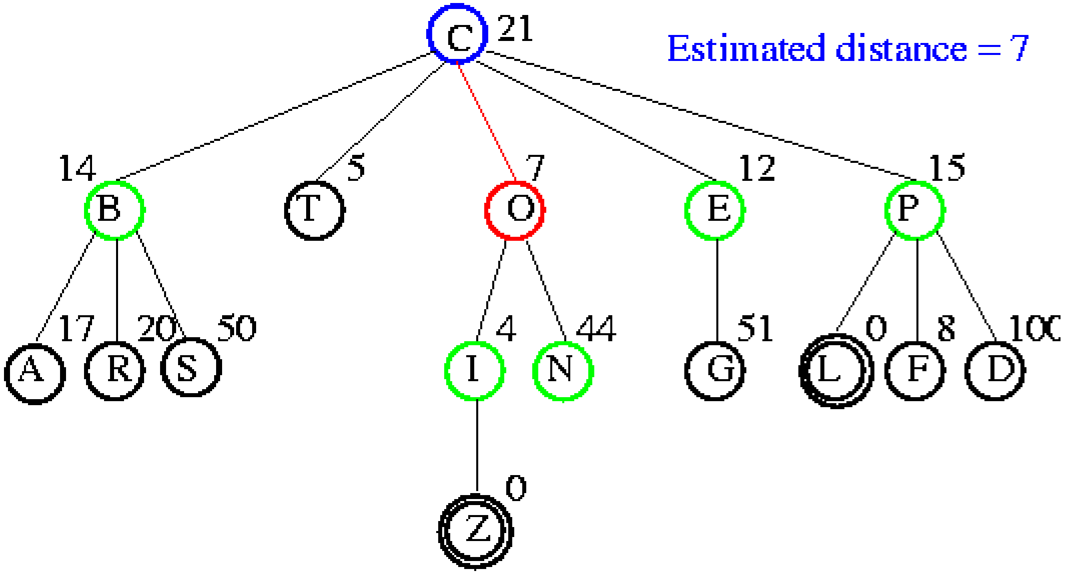
C 21
Estimated distance = 5
14
B
T
5
7
O
12
E
15
P
17
A
R
20
S
50
I
4
N
44
G
51
L
0
F
8
D
Z
0

Example



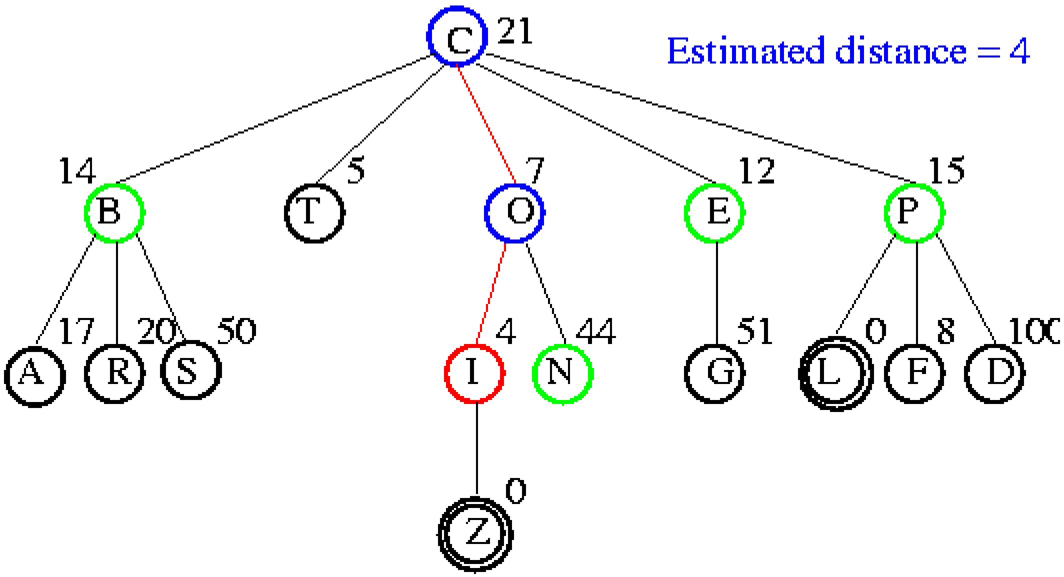
C 21
Estimated distance = 7
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
D
0 Z

Example



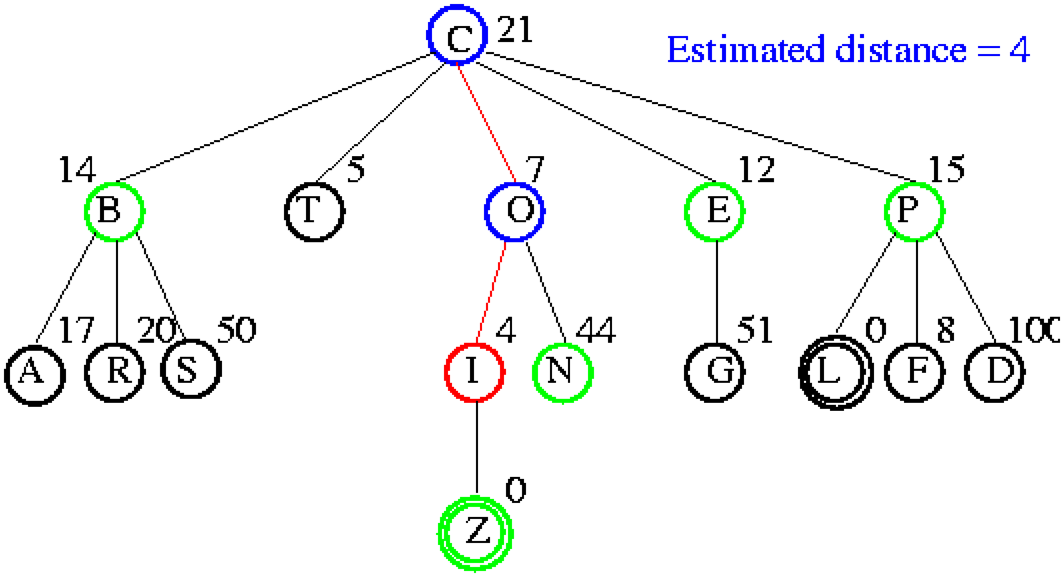
C 21
Estimated distance = 7
14 B
5 T
7 O
12 E
15 P
17 A
20 R
50 S
4 I
44 N
51 G
0 L
8 F
100 D
0 Z

Example



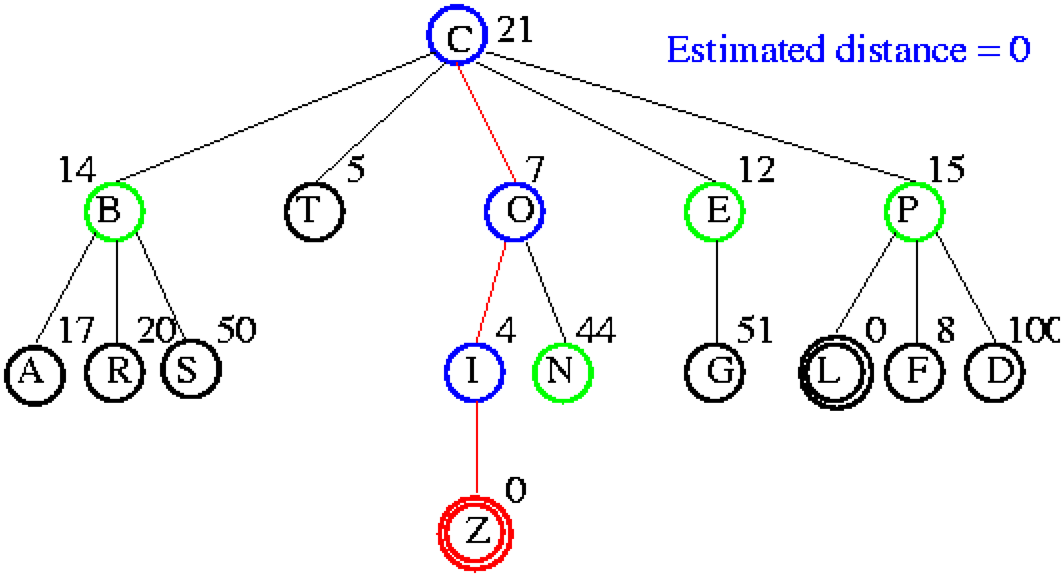
C 21
Estimated distance = 4
14
B
T 5
7
O
12
E
15
P
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 10C
Z 0

Example



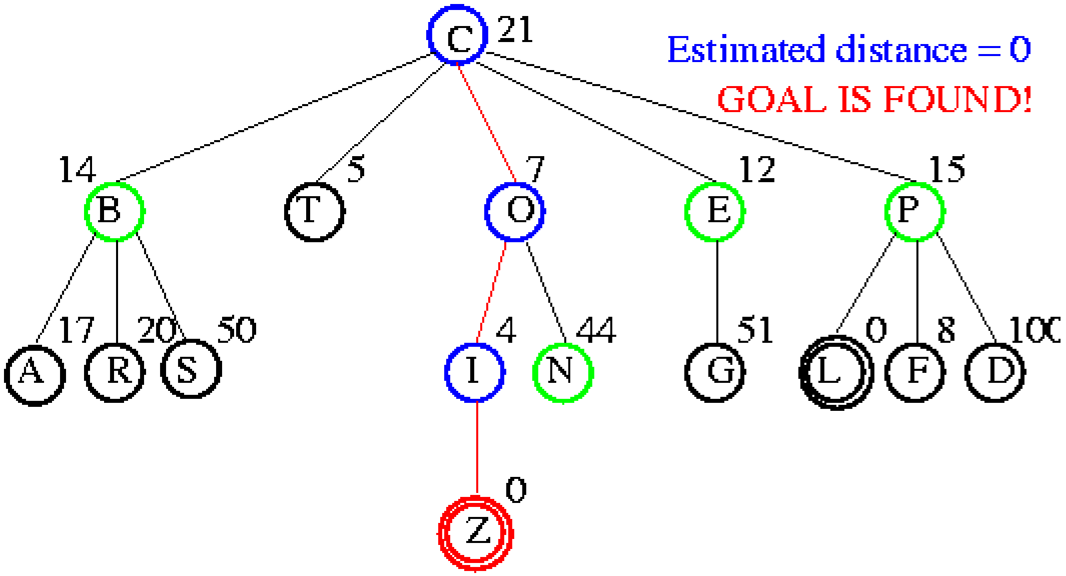
C
21
Estimated distance = 4
14
B
5
T
7
O
12
E
15
P
17
A
20
R
50
S
4
I
44
N
51
G
0
L
8
F
10C
D
0
Z

Example



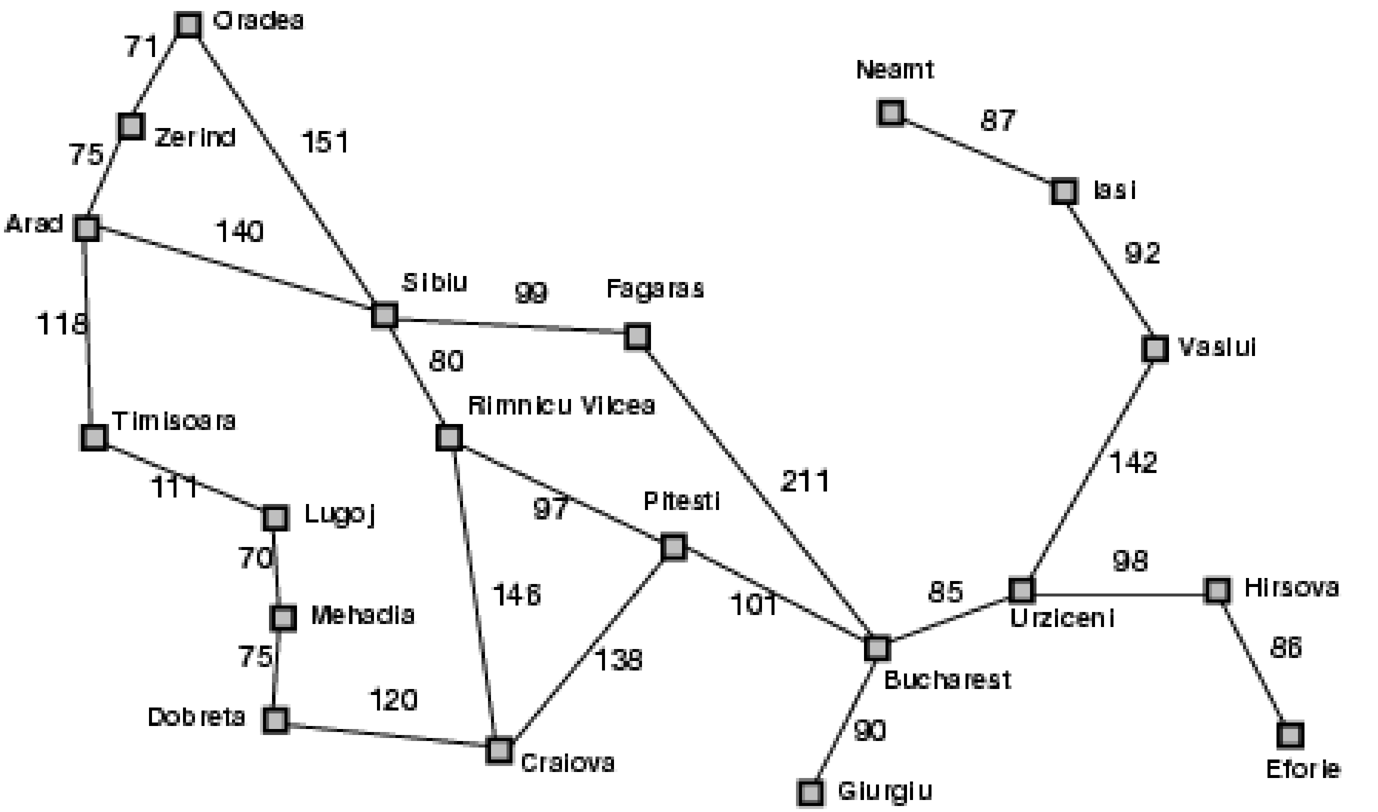
C 21
Estimated distance = 0
14
B
T 5
7
O
12
E
15
P
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

Example



Estimated distance = 0
GOAL IS FOUND!
C 21
14
B
T 5
O 7
E 12
P 15
A 17
R 20
S 50
I 4
N 44
G 51
L 0
F 8
D 100
Z 0

Romania with straight-line dist.

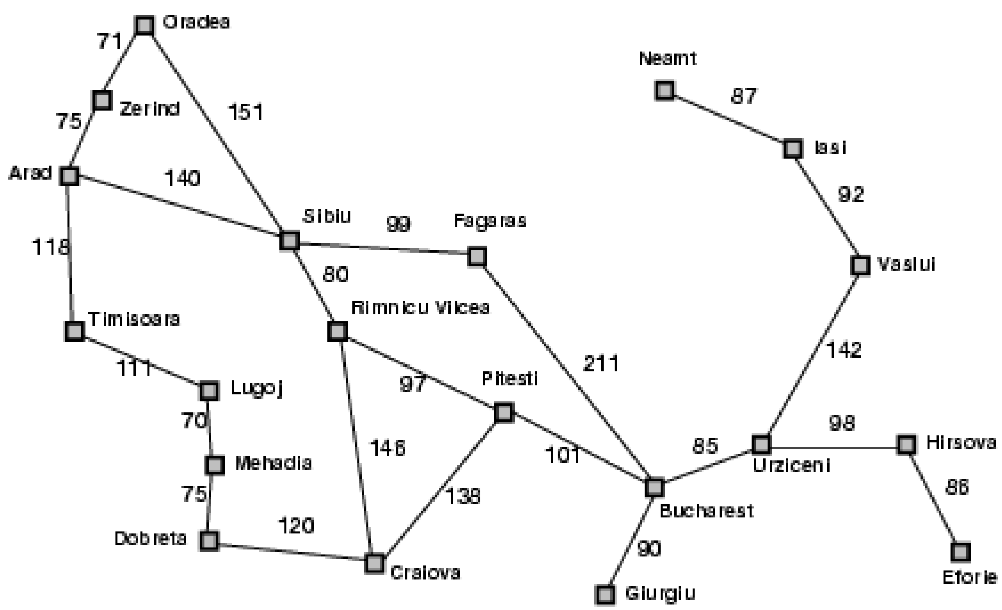


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

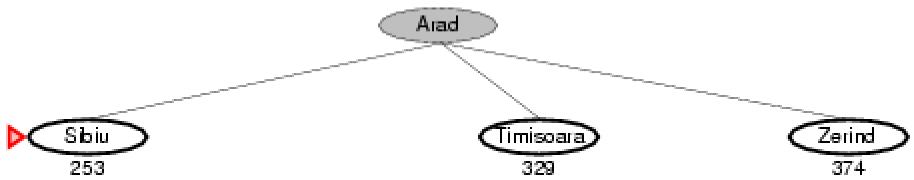
- f(n) = estimate of cost from *n* to *goal*
- e.g., $f_{SLD}(n)$ = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal.

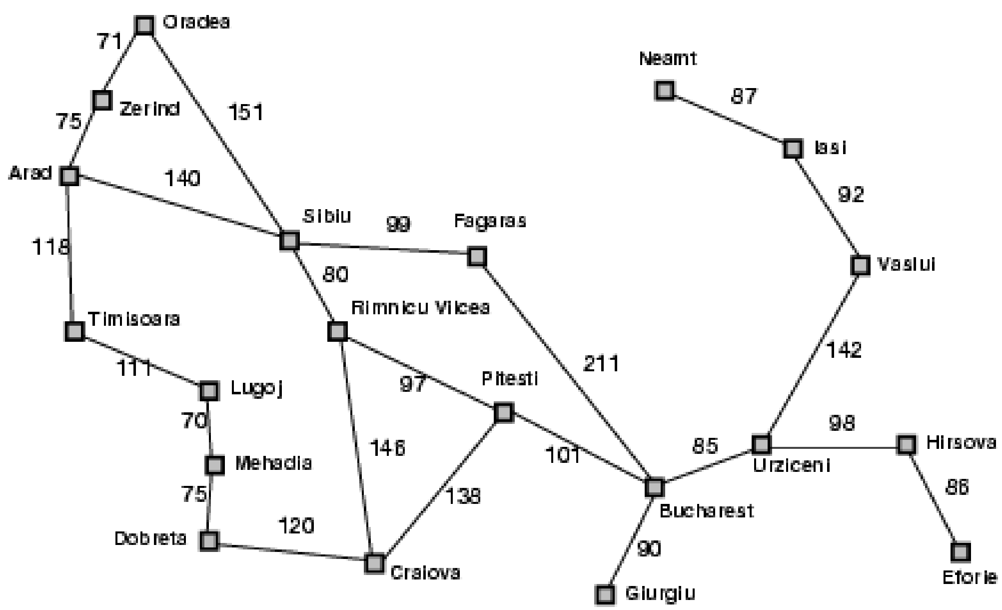
Greedy best-first search example



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 example

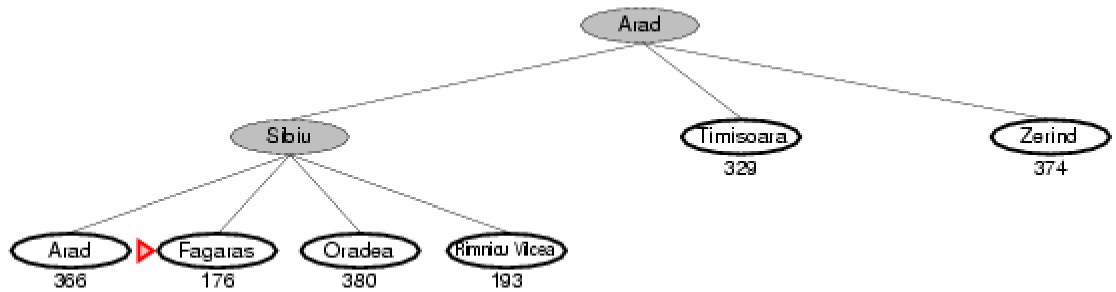


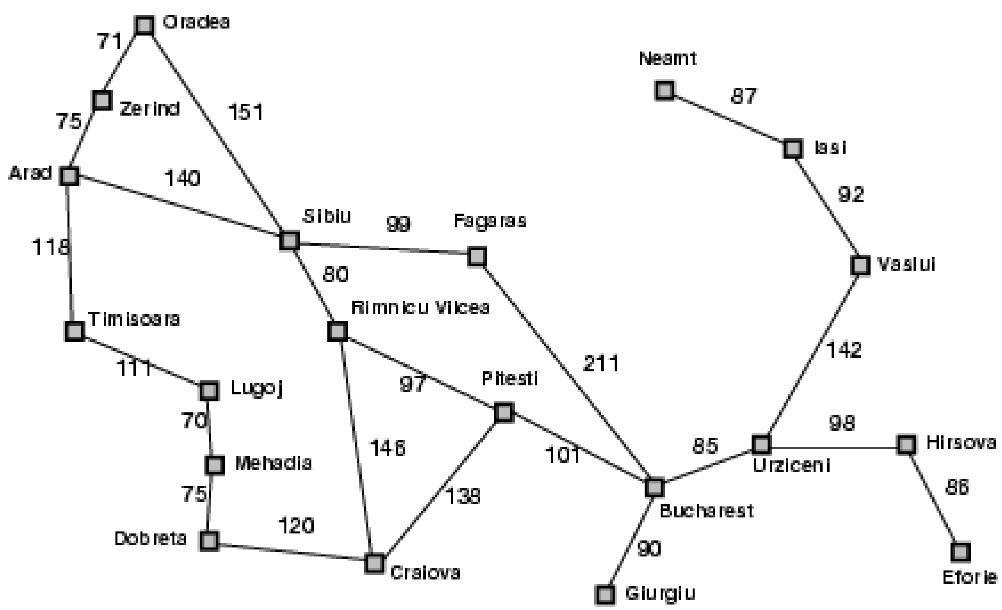


Straight–line distance to Bucharest

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

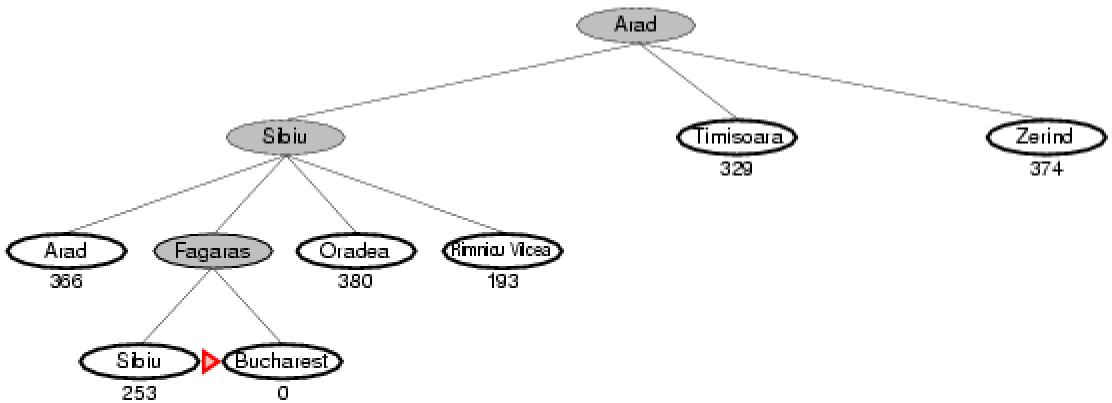
Greedy best-first search example

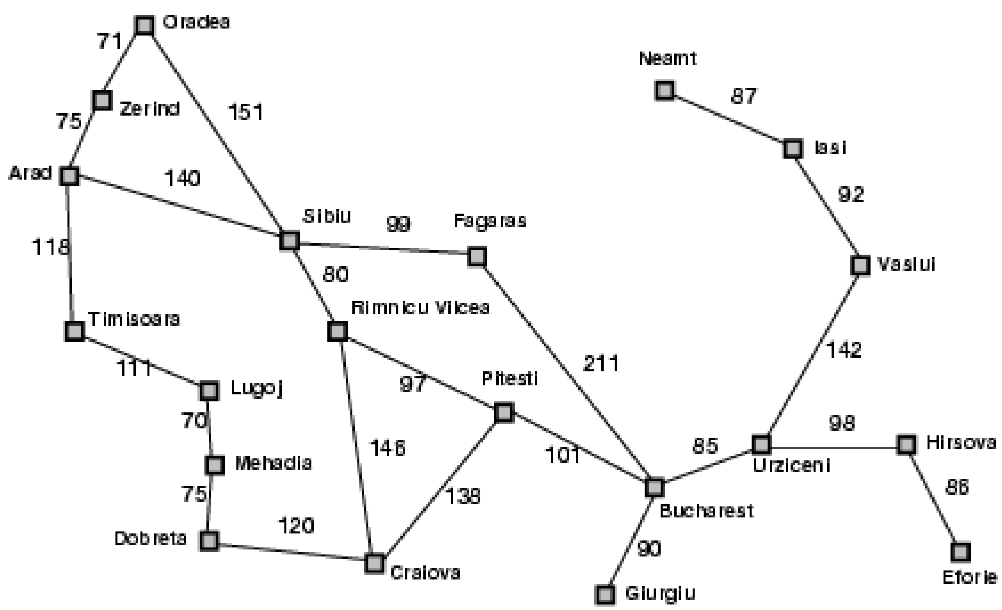




Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example





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

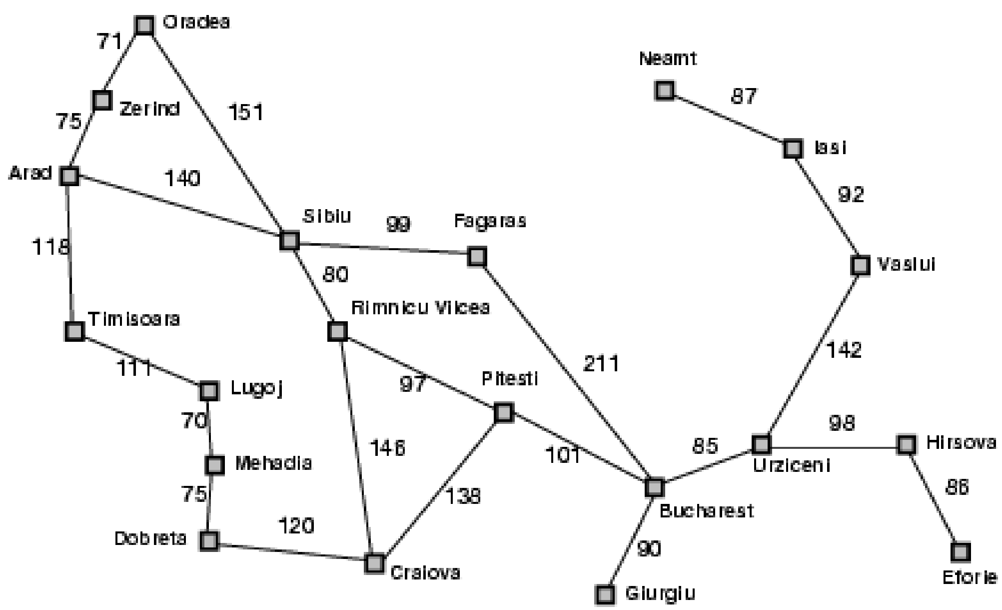
- Complete? No – can get stuck in loops.
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ - keeps all nodes in memory
- Optimal? No

e.g. Arad→Sibiu→Rimnicu Virea→Pitesti→Bucharest is shorter!

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
- Best First search has $f(n)=h(n)$

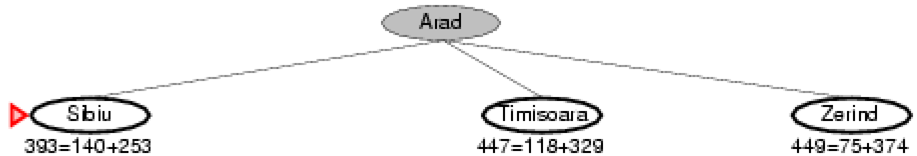
A* search example

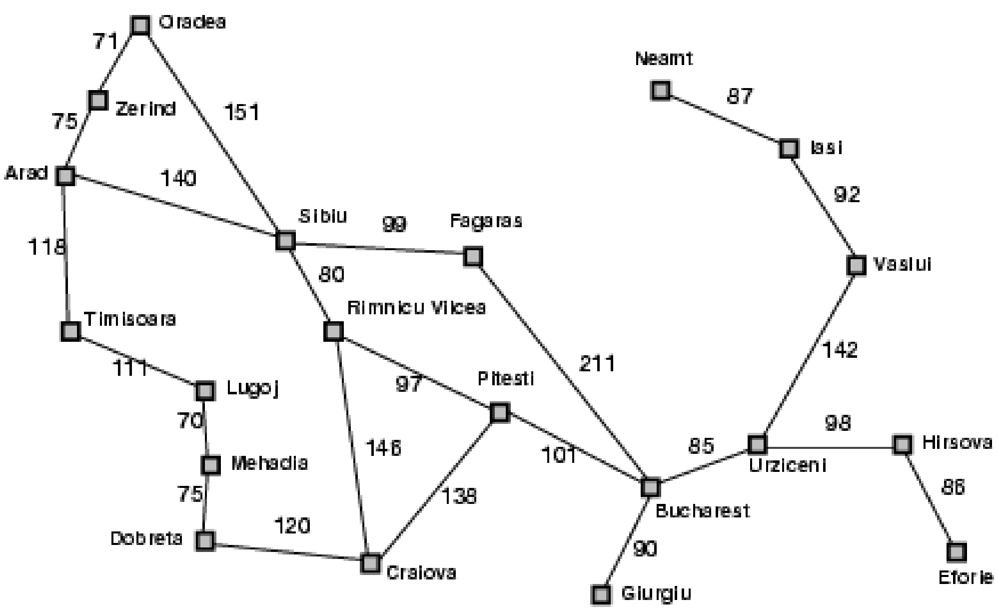


Straight–line distance to Bucharest

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

A* search example

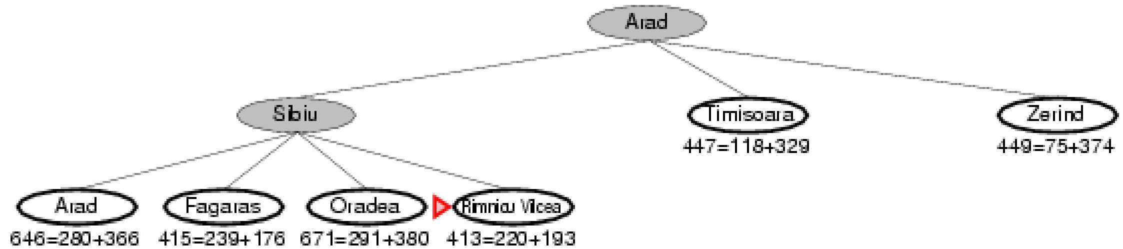


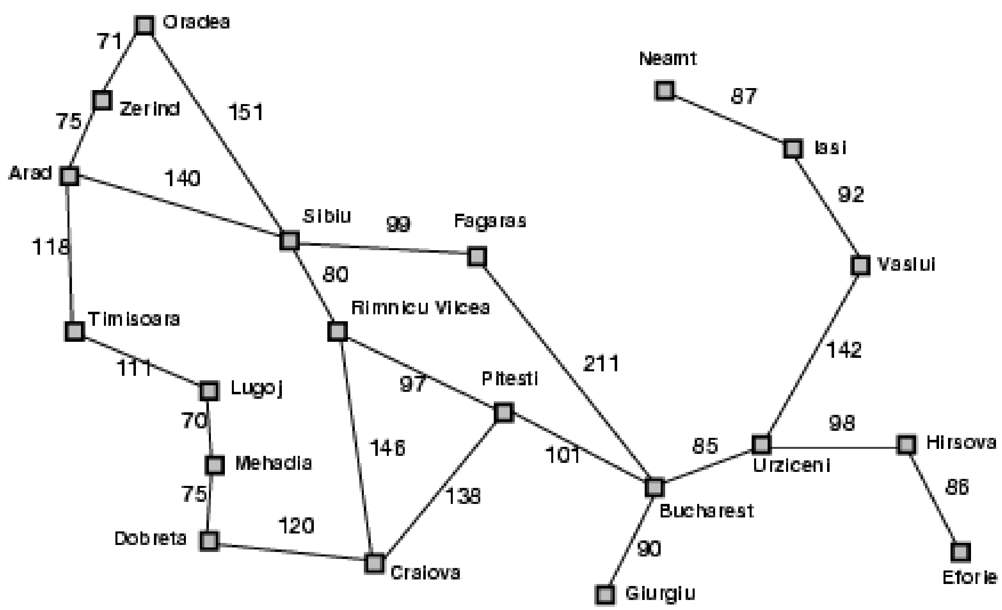


Straight–line distance
to Bucharest

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

A* search example

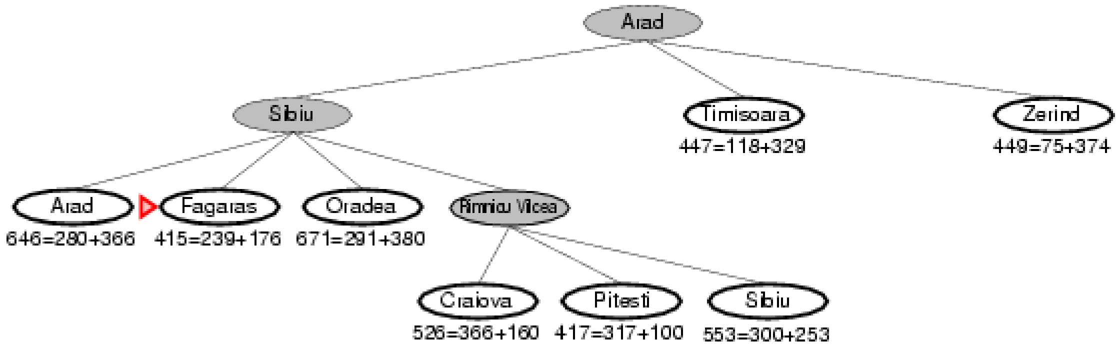


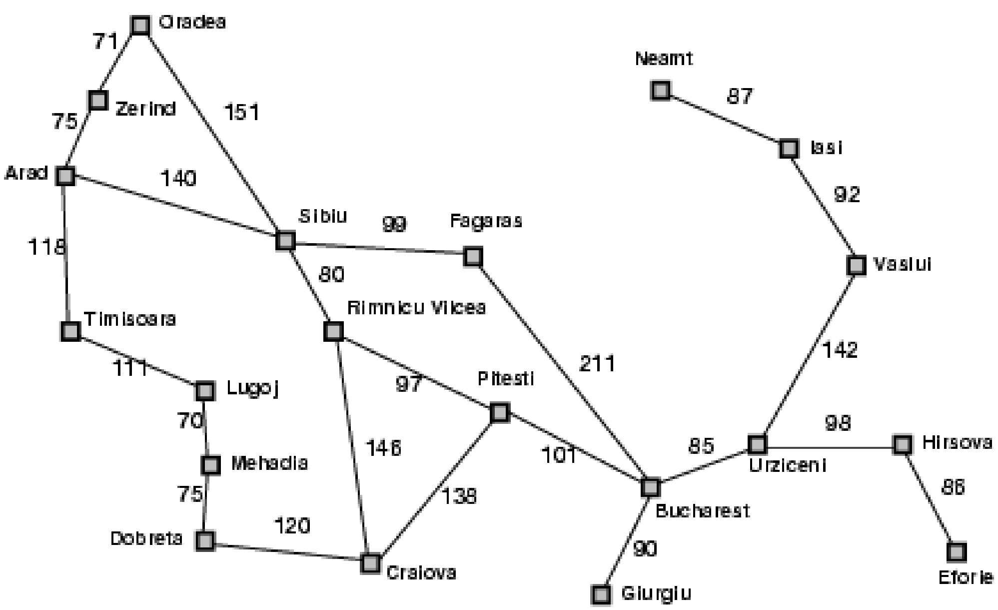


Straight–line distance to Bucharest

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

A* search example

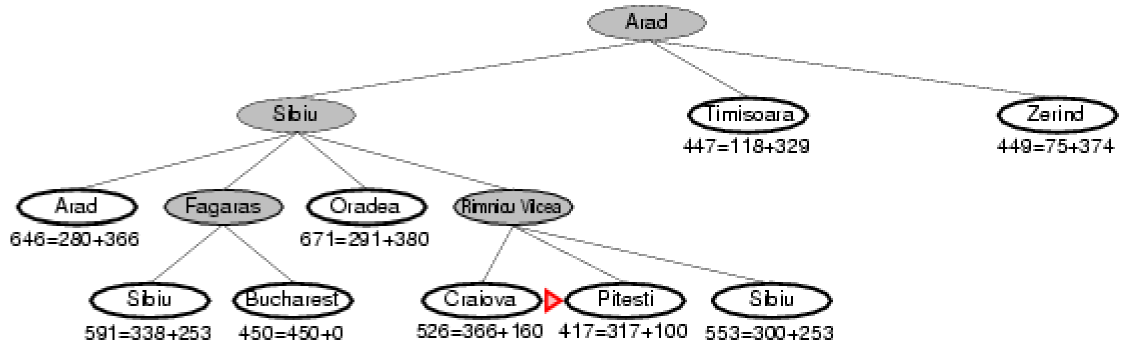


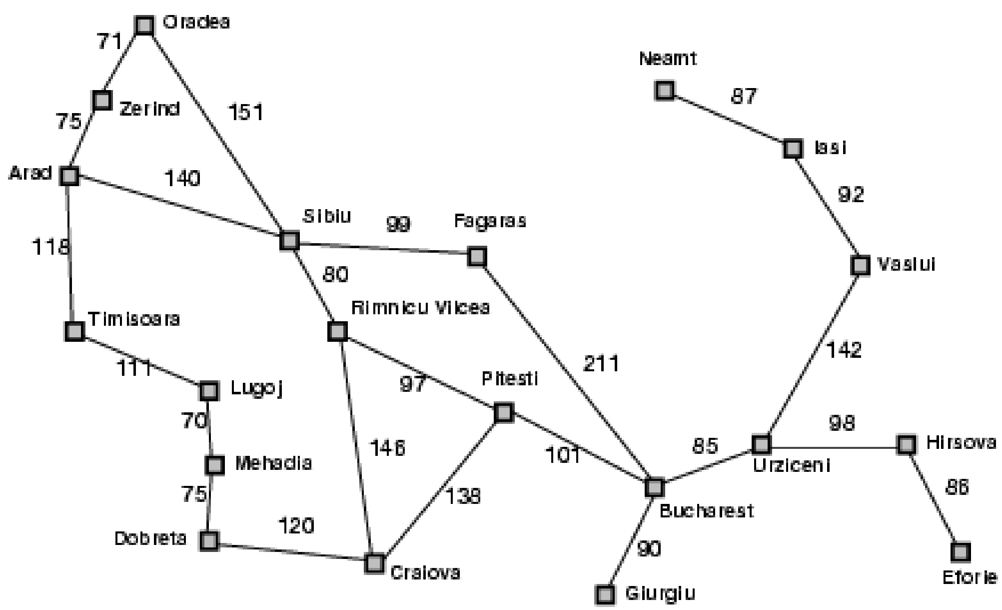


Straight–line distance to Bucharest

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	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

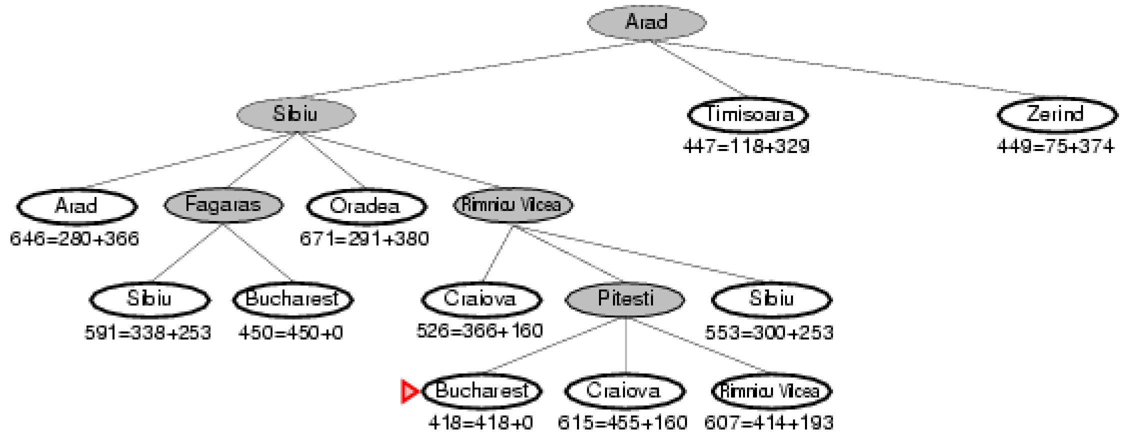


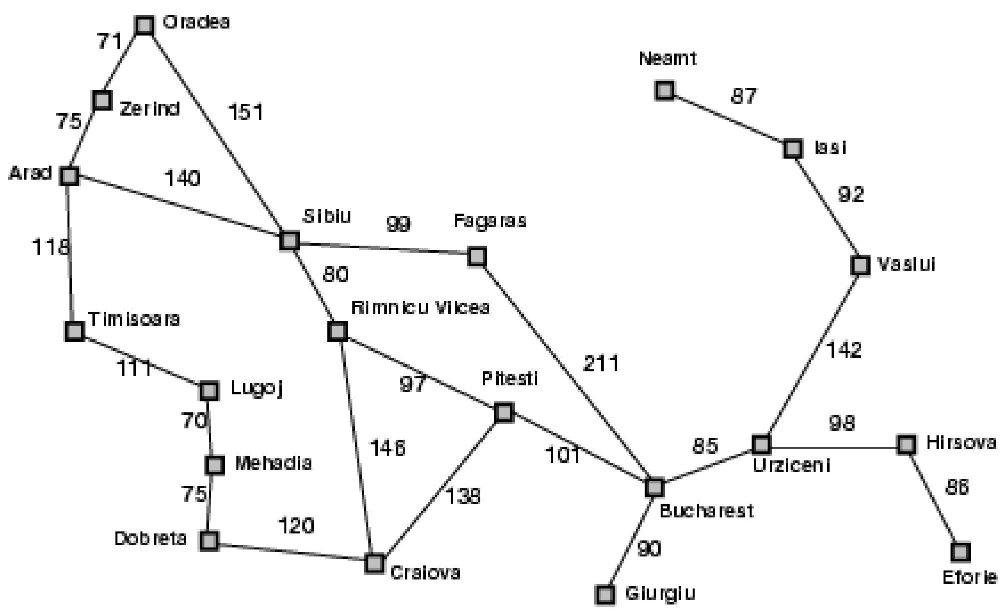


Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	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

A* search example





Straight–line distance
to Bucharest

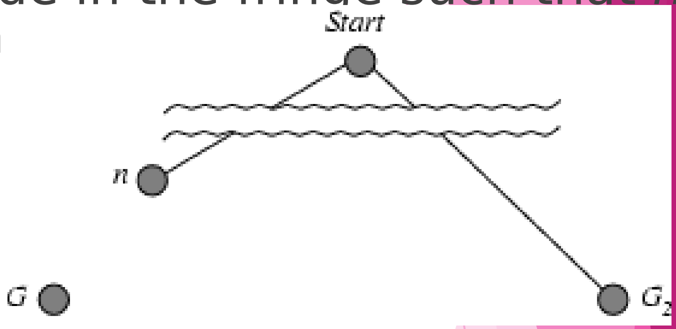
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

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 goa



- $f(G_2)$ > f(G) copied from last slide
- h(n) ≤ h*(n) since h is admissible (*under*-estimate)
- g(n) + h(n) ≤ g(n) + h*(n) from above
- f(n) ≤ f(G) since g(n)+h(n)=f(n) & g(n)+h*(n)=f(G)
- f(n) < f(G2) from top line.

Hence: n is preferred over G2

Consistent heuristics

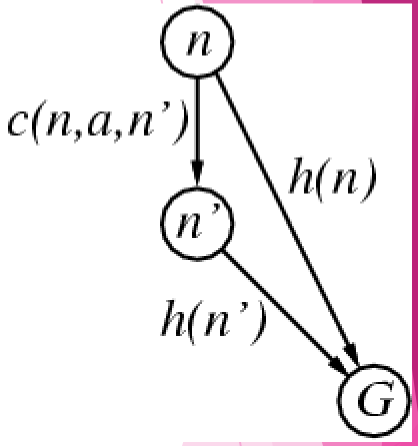
- A heuristic is consistent if for every node *n*, every successor *n*' of *n* generated by any action *a*, can be represented as:

  h(n) ≤ c(n,a,n') + h(n')

- If *h* is consistent, we have

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

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



It's the triangle inequality !

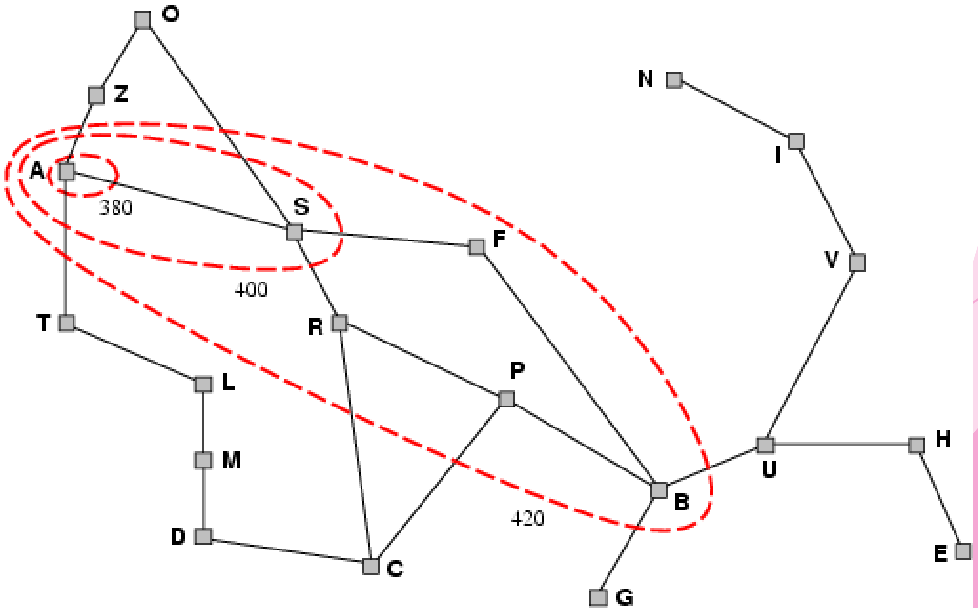
- Theorem:

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

keeps all checked nodes in memory to avoid repeated states

Optimality of A*

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



Properties of A*

- <u>Complete?</u> Yes (unless there are infinitely many nodes with f ≤ *f(G)* , i.e. path-cost > ε)
- <u>Time/Space?</u> Exponential

  except if: $|h(n) - h^*(n)| \leq O(\log h^*(n))$
- <u>Optimal?</u> Yes
- *<u>Optimally Efficient</u>*: Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)

Memory Bounded Heuristic Search: Recursive BFS

- How can we solve the memory problem for A* search?
- Idea: Try something like depth first search, but let’s not forget everything about the branches we have partially explored.
- We remember the best f-value we have found so far in the branch we are deleting.

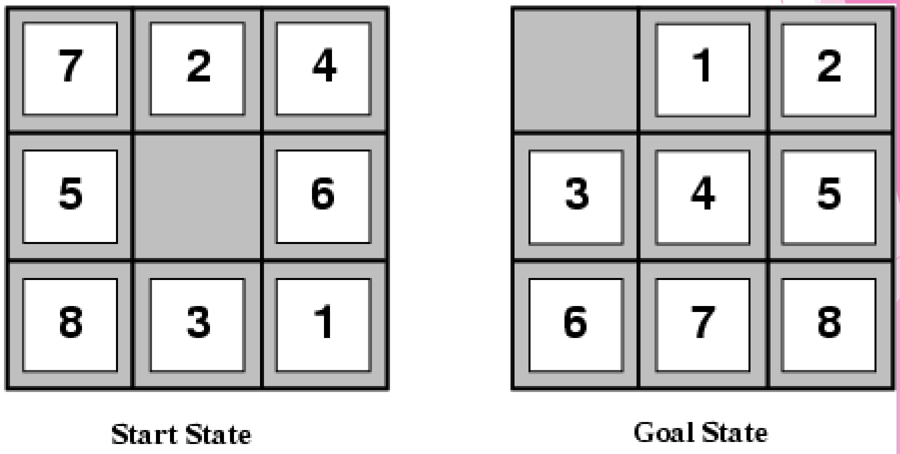
Admissible heuristics

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)

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

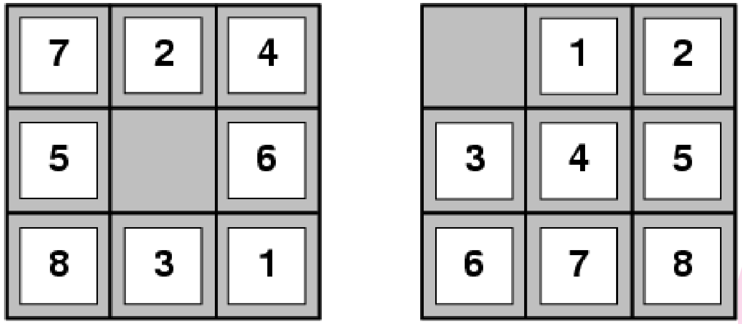


Admissible heuristics

E.g., for the 8-puzzle:

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

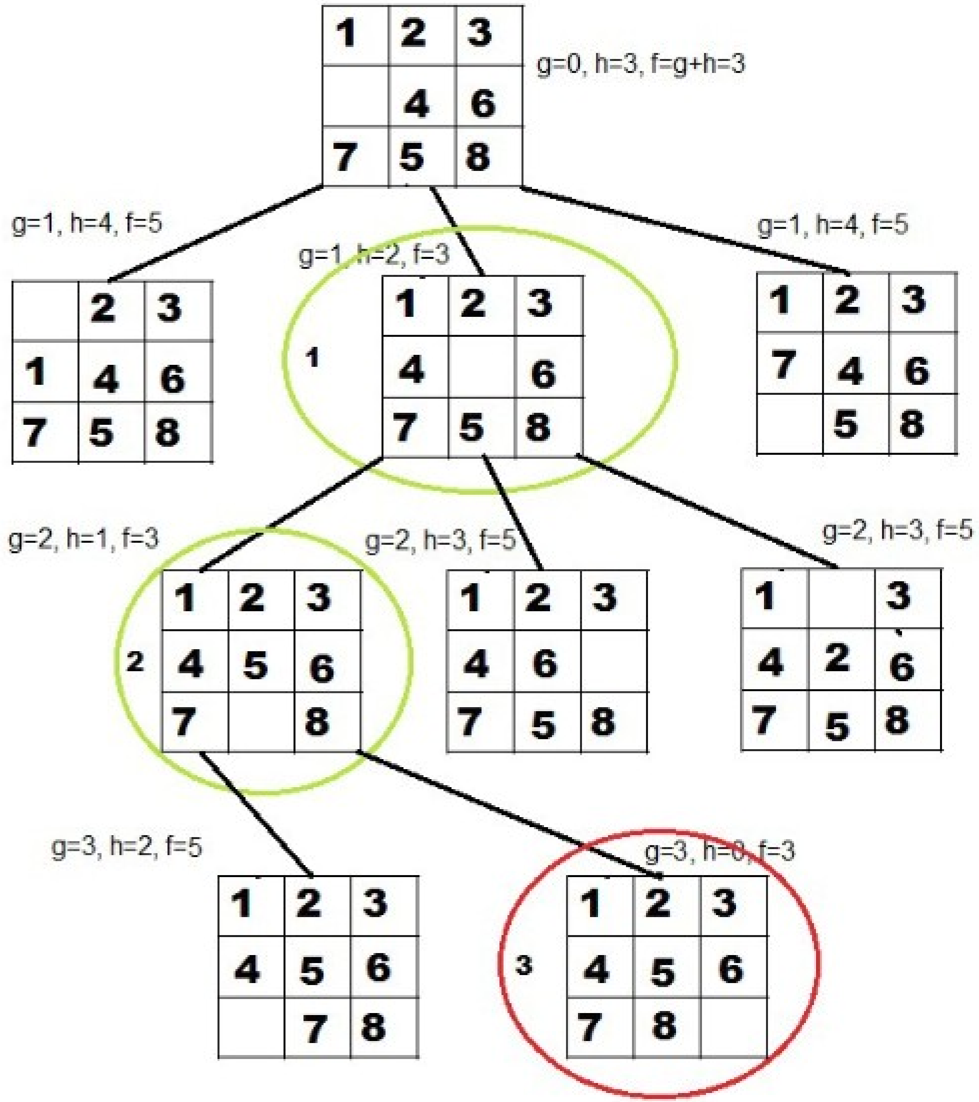
(i.e., no. of squares from desired location of each tile)



Start State Goal State

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

Example



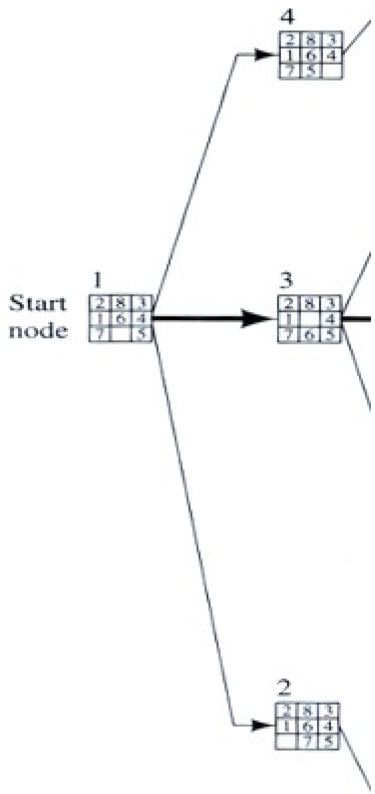
1 2 3
4 6
7 5 8
g=0, h=3, f=g+h=3
g=1, h=4, f=5
g=1, h=2, f=3
g=1, h=4, f=5
2 3
1 4 6
7 5 8
1
1 2 3
4 6
7 5 8
1 2 3
7 4 6
5 8
g=2, h=1, f=3
g=2, h=3, f=5
g=2, h=3, f=5
2
1 2 3
4 5 6
7 8
1 2 3
4 6
7 5 8
1 3
4 2 6
7 5 8
g=3, h=2, f=5
g=3, h=0, f=3
1 2 3
4 5 6
7 8
3
1 2 3
4 5 6
7 8

Exercise

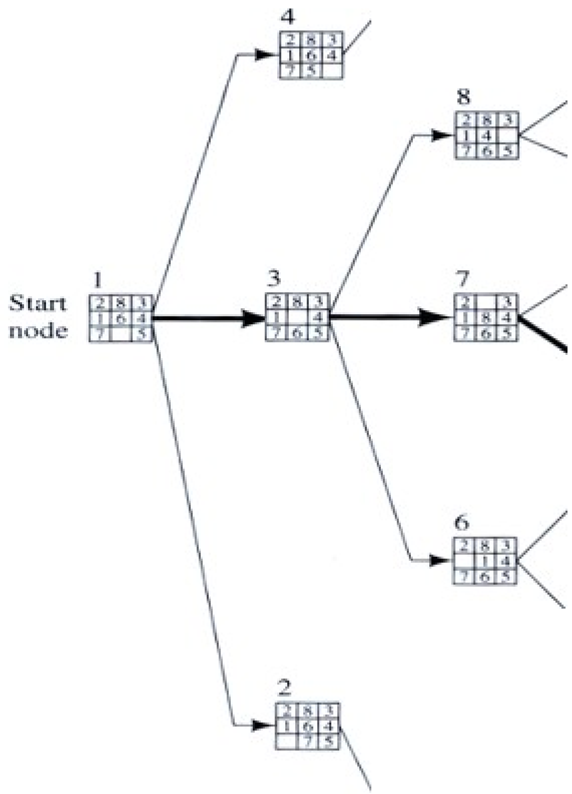
2	**8**	**3**
1	6	4
7		5

1	**2**	**3**
8		4
7	6	5

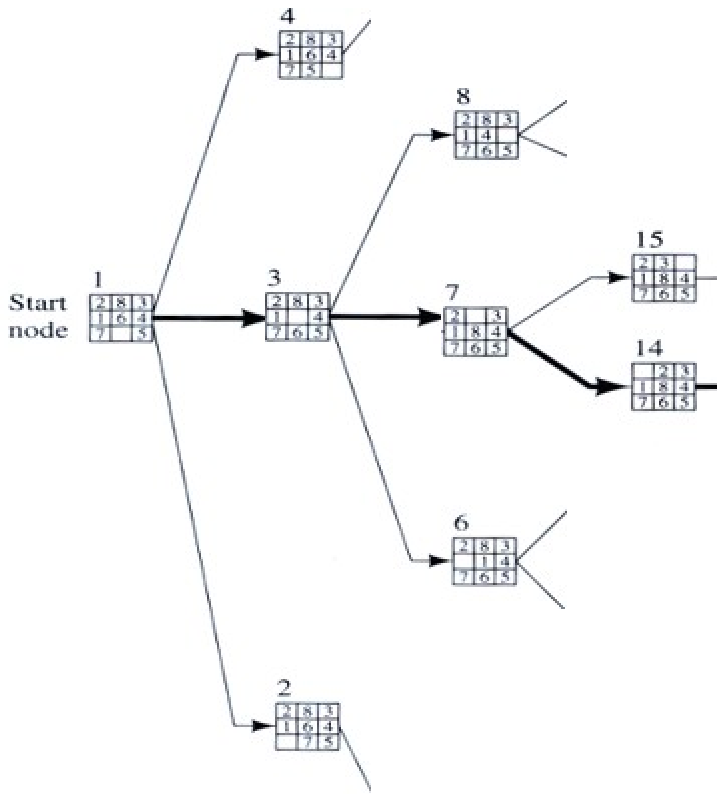
Initial State	**Goal State**



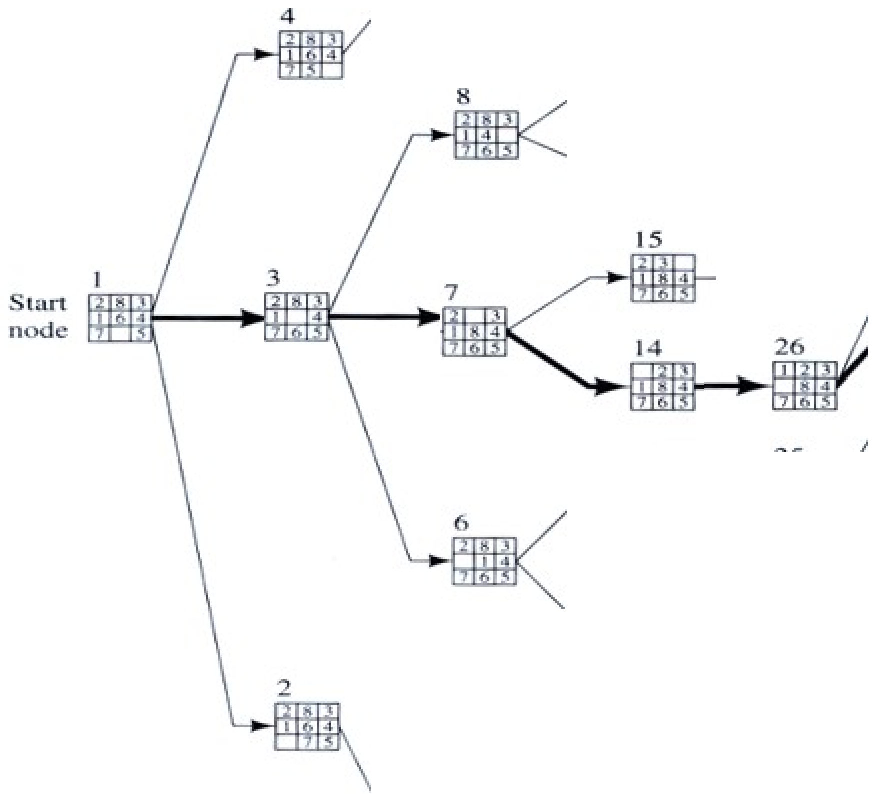
Start
node
1
2
3
4



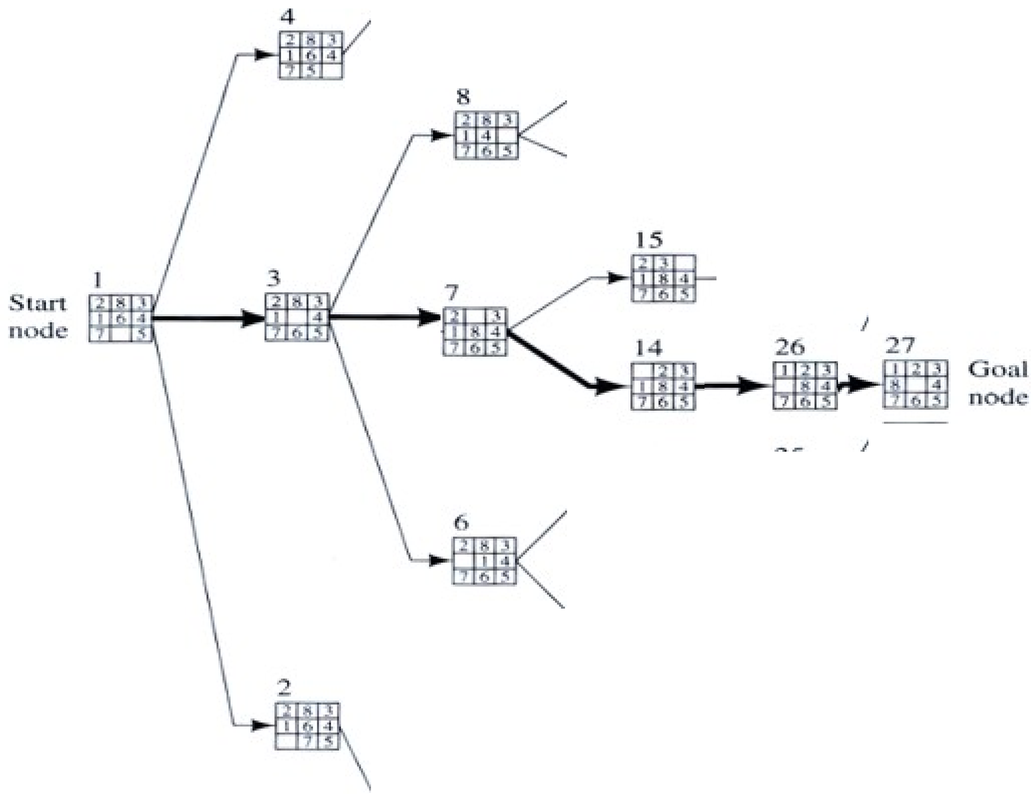
Start
node
1
2 8 3
1 6 4
7 5
4
2 8 3
1 6 4
7 5
3
2 8 3
1 4
7 6 5
2
2 8 3
1 6 4
7 5
8
2 8 3
1 4
7 6 5
7
2 3
1 8 4
7 6 5
6
2 8 3
1 4
7 6 5



Start
node
1
2
3
4
6
7
8
14
15

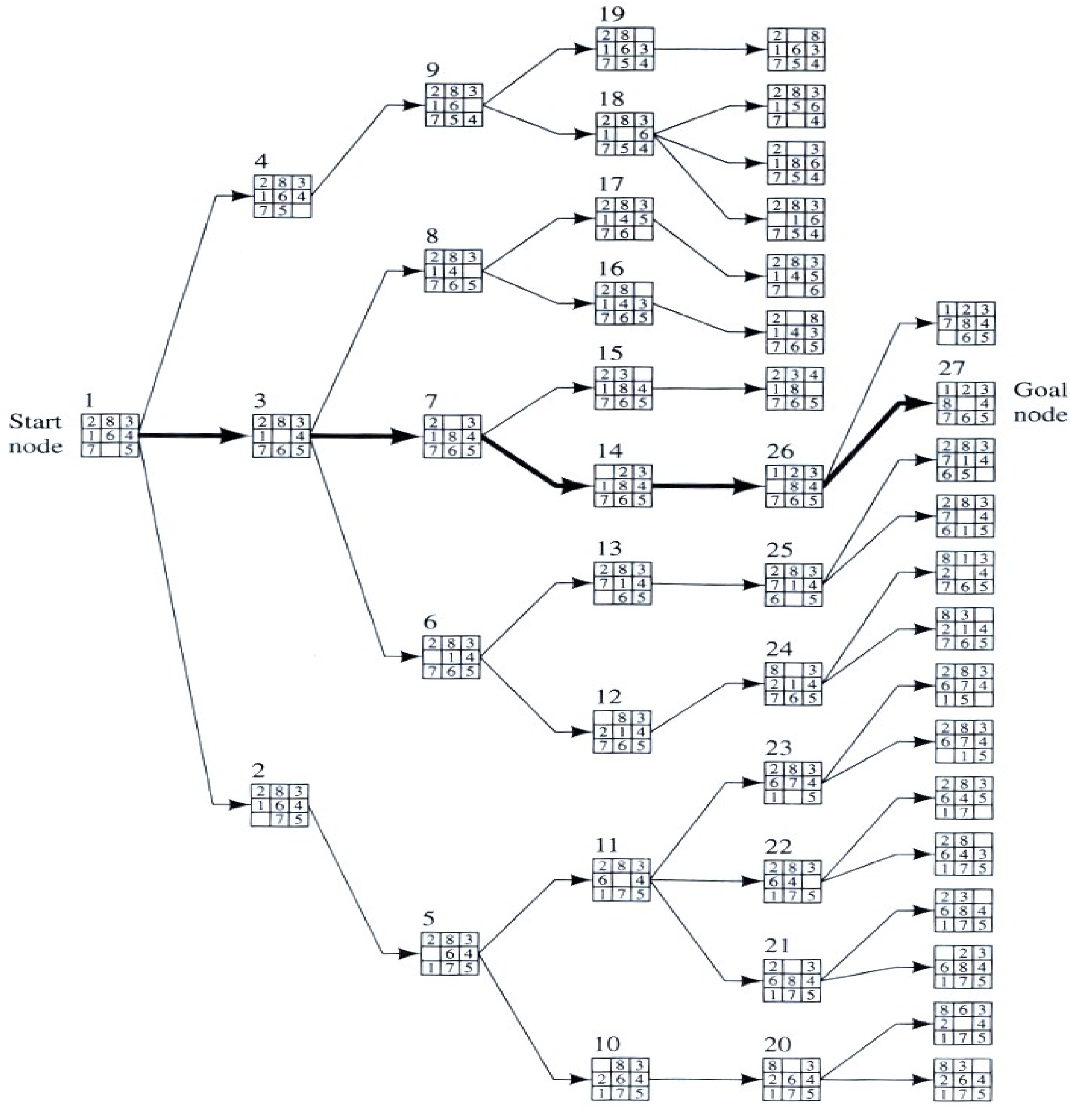


Start node
1
2 8 3
1 6 4
7 5
4
2 8 3
1 6 4
7 5
3
2 8 3
1 4
7 6 5
2
2 8 3
1 6 4
7 5
8
2 8 3
1 4
7 6 5
7
2 3
1 8 4
7 6 5
6
2 8 3
1 4
7 6 5
15
2 3
1 8 4
7 6 5
14
2 3
1 8 4
7 6 5
26
1 2 3
8 4
7 6 5



Start node
Goal node

Example BFS


Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

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: it is guaranteed to expand less nodes.

- 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

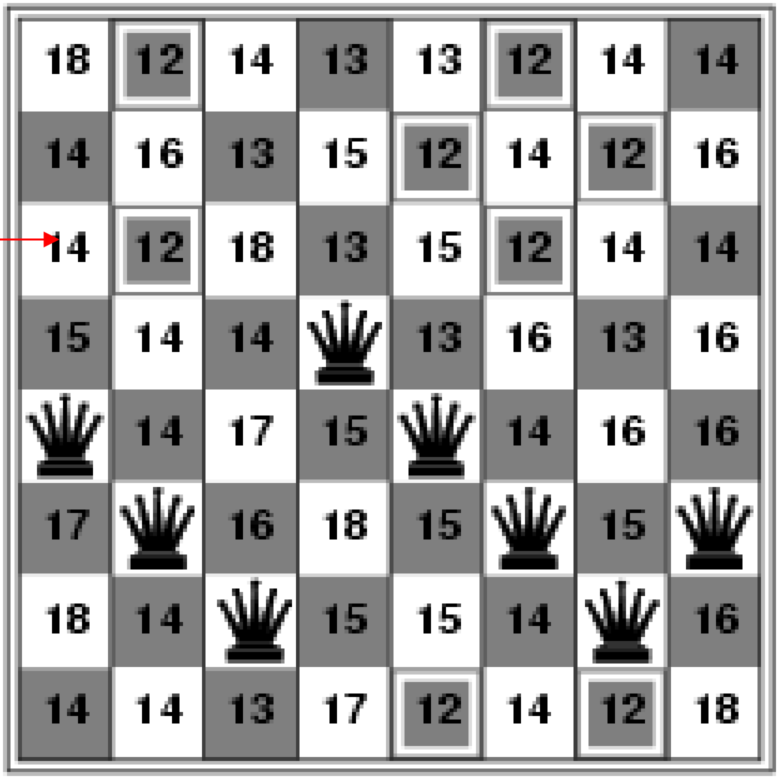
Relaxed problems

- 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

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.
- Very memory efficient (only remember current state)
- Key advantages:
  - Use less memory
  - Often find reasonable solutions in large state spaces.

Hill-climbing search: 8-queens problem



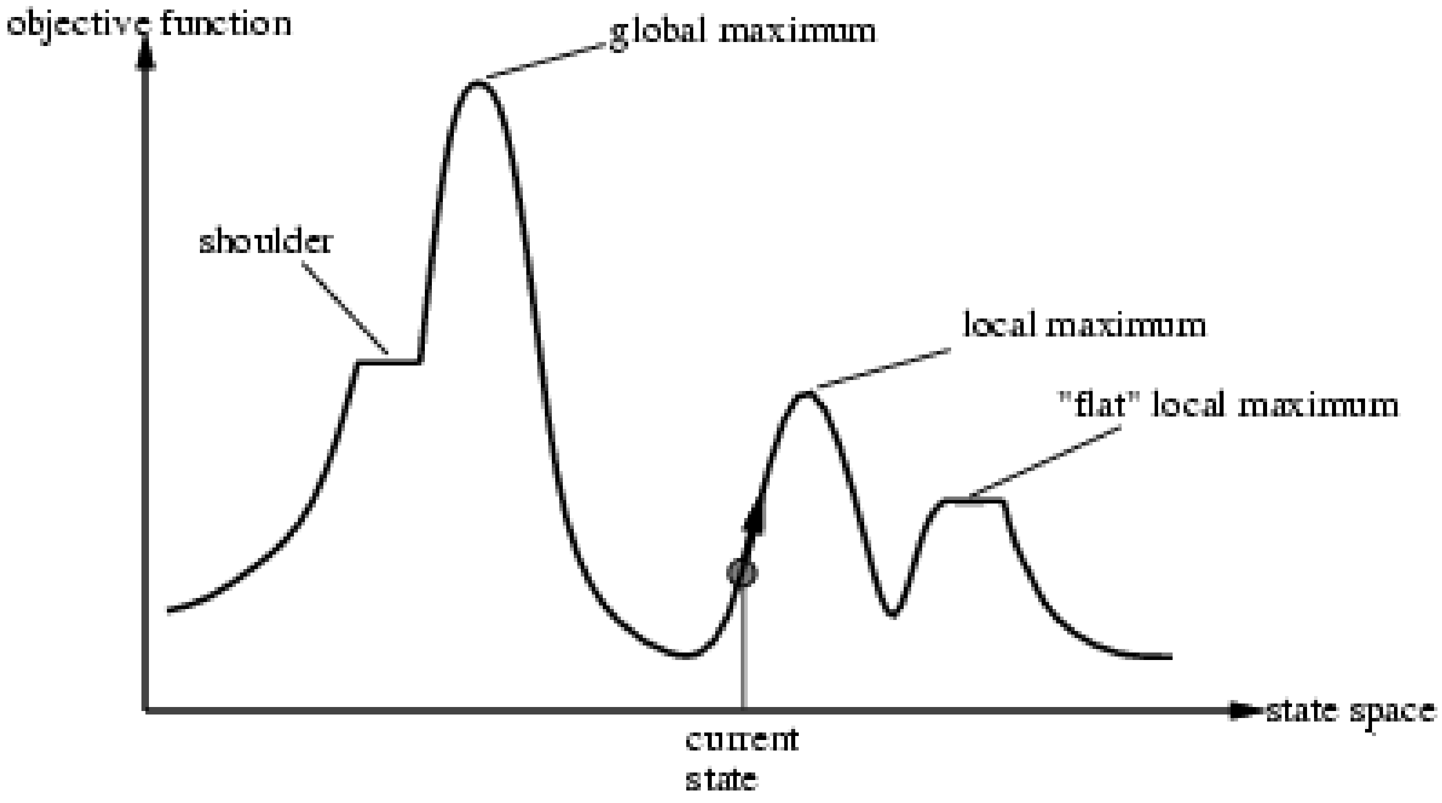
Each number indicates h if we move a queen in its corresponding column

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

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

Hill-climbing search

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



Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency.

- This is like smoothing the cost landscape.

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 (however, this may take VERY long)

- 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.

What is GA

- A **genetic algorithm** (or **GA**) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- Genetic algorithms are categorized as global search heuristics.
- Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Key terms

- **Individual** - Any possible solution
- **Population** - Group of all *individuals*
- **Search Space** - All possible solutions to the problem
- **Chromosome** - Blueprint for an *individual*
- **Trait** - Possible aspect (*features)* of an *individual*
- **Allele** - Possible settings of trait (black, blond, etc.)
- **Locus** - The position of a *gene* on the *chromosome*
- **Genome** - Collection of all *chromosomes* for an *individual*

GA Requirements

- a genetic representation of the solution domain, and
- a fitness function to evaluate the solution domain.
- A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way.
- The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation.
- Variable length representations may also be used, but crossover implementation is more complex in this case.
- Tree-like representations are explored in Genetic programming.

General Algorithm for GA

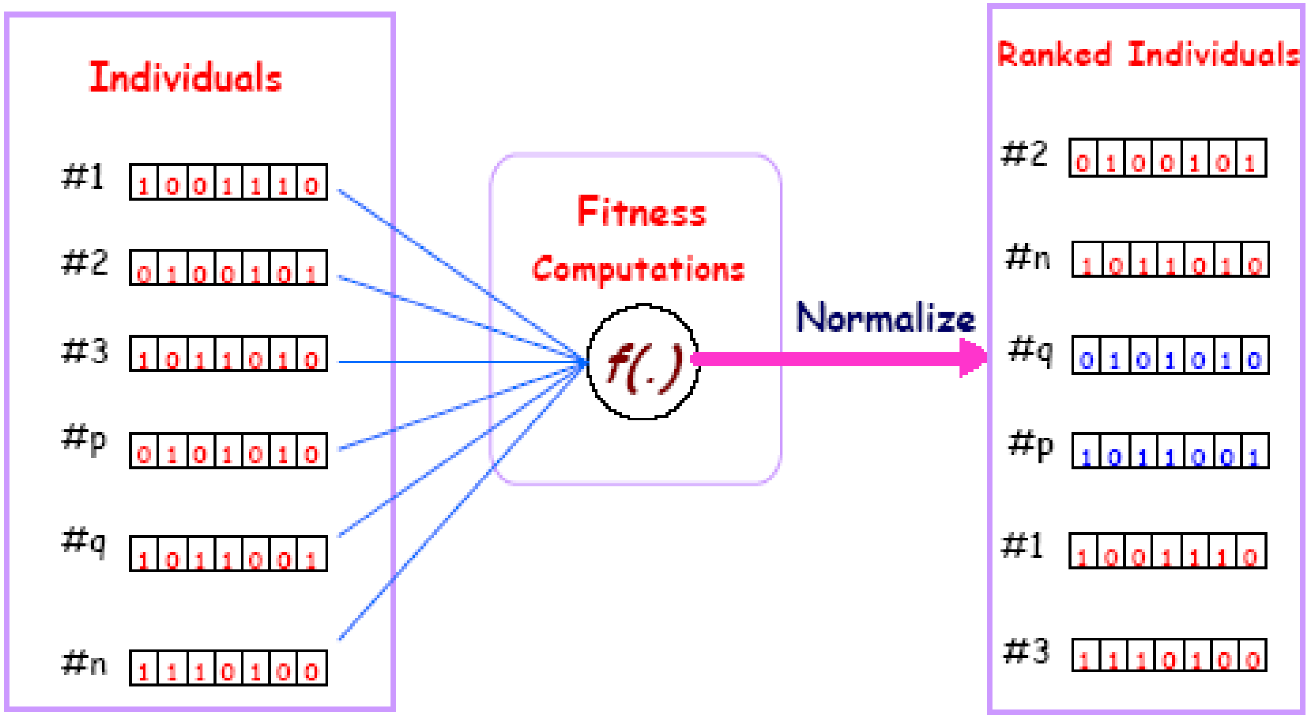
- Initialization
  - Initially many individual solutions are randomly generated to form an initial population.
- Fitness Function
- Selection
  - During each successive generation, a proportion of the existing population is selected to breed a new generation selected through fitness function
- Reproduction
- Crossover
- Mutation

Representation

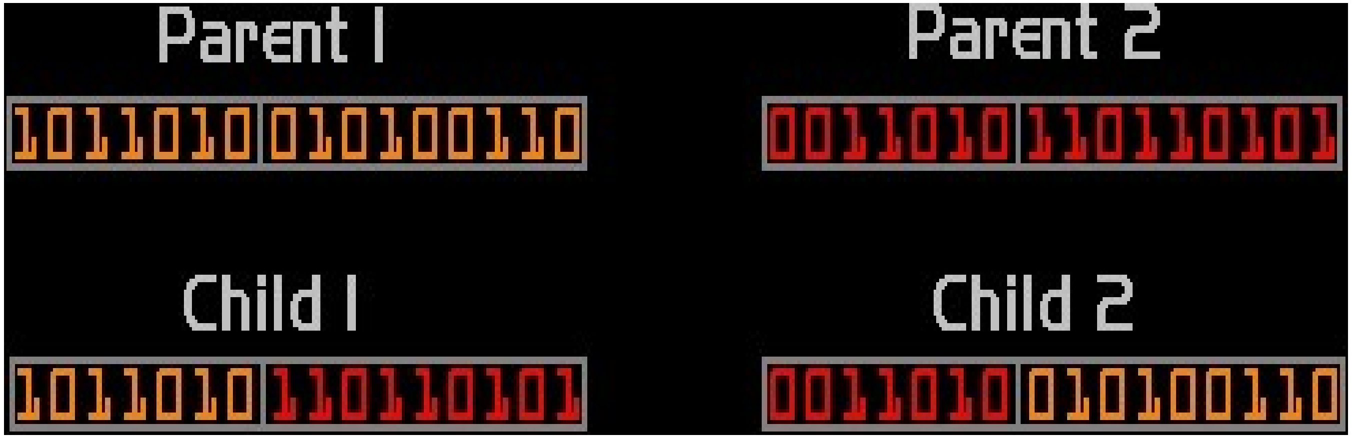
Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

A fitness function

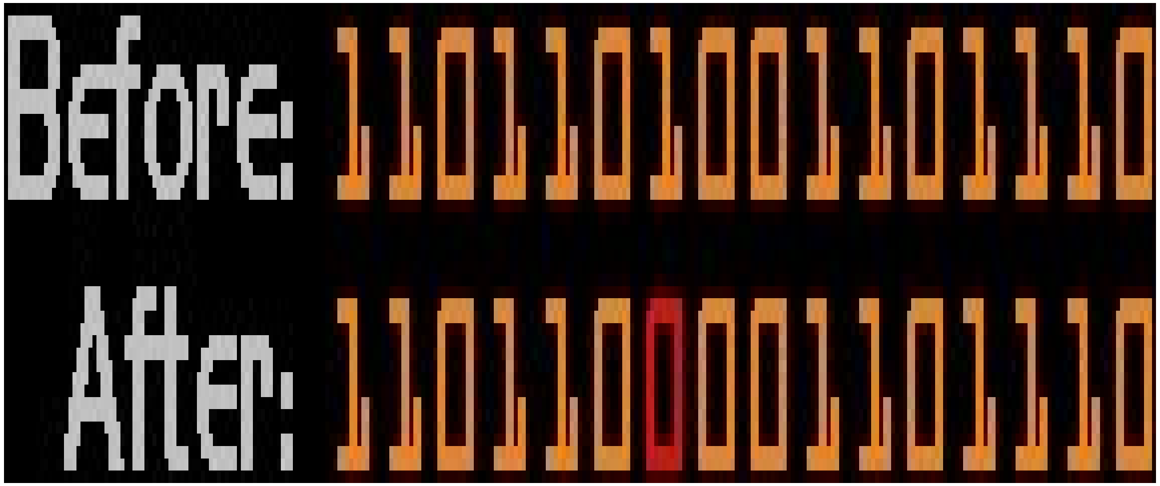


Crossover



Parent 1
1011010 010100110
Parent 2
0011010 110110101
Child 1
1011010 110110101
Child 2
0011010 010100110

Mutation



General Algorithm for GA

- **Termination**
- This generational process is repeated until a termination condition has been reached.
- Common terminating conditions are:
  - A solution is found that satisfies minimum criteria
  - Fixed number of generations reached
  - Allocated budget (computation time/money) reached
  - The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
  - Manual inspection
  - Any Combinations of the above

Aplications

- Routing Problems
- Financial markets
- Manufacturing System
- Engineering Design
- Data clustering
- Neural Nteworks
- Medical Sciences

Example

- $f(x) = \{MAX(x^2): 0 <= x <= 32 \}$
- Encode Solution: Just use 5 bits (1 or 0).
- Generate initial population.

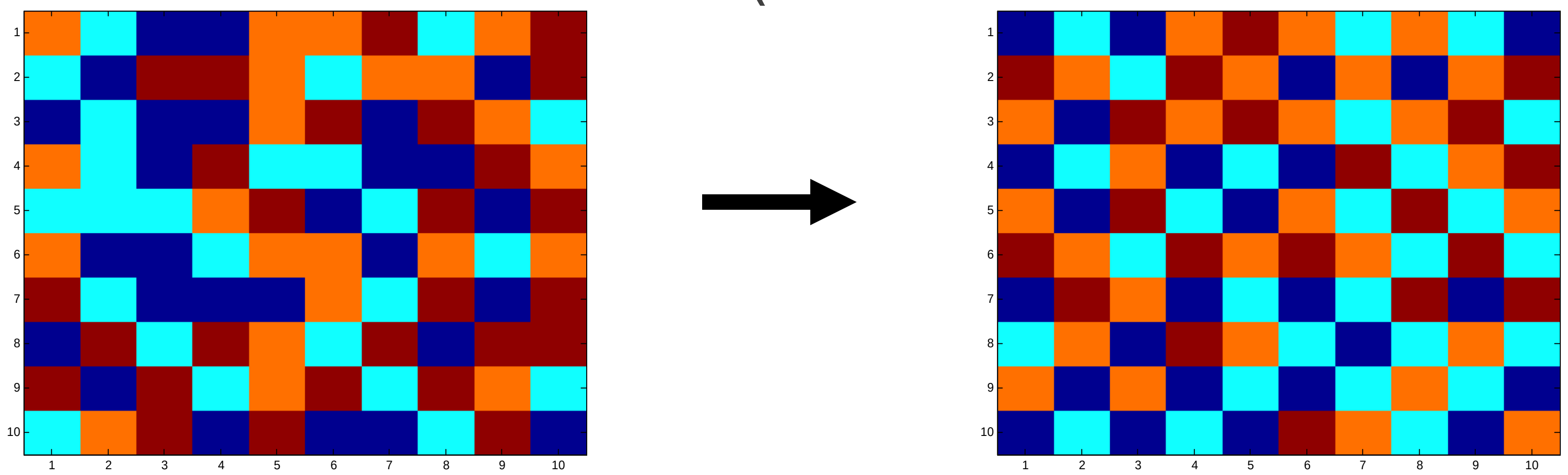
A	0	1	1	0	1
B	1	1	0	0	0
C	0	1	0	0	0
D	1	0	0	1	1

- Evaluate each solution against objective.

Sol.	**String**	**Fitness**	**% of Total**
A	01101	169	14.4
B	11000	576	49.2
C	01000	64	5.5
D	10011	361	30.9

Checkboard example

- We are given an ***n*** by ***n*** checkboard in which every field can have a different colour from a set of four colors.
- Goal is to achieve a checkboard in a way that there are no neighbours with the same color (not diagonal).



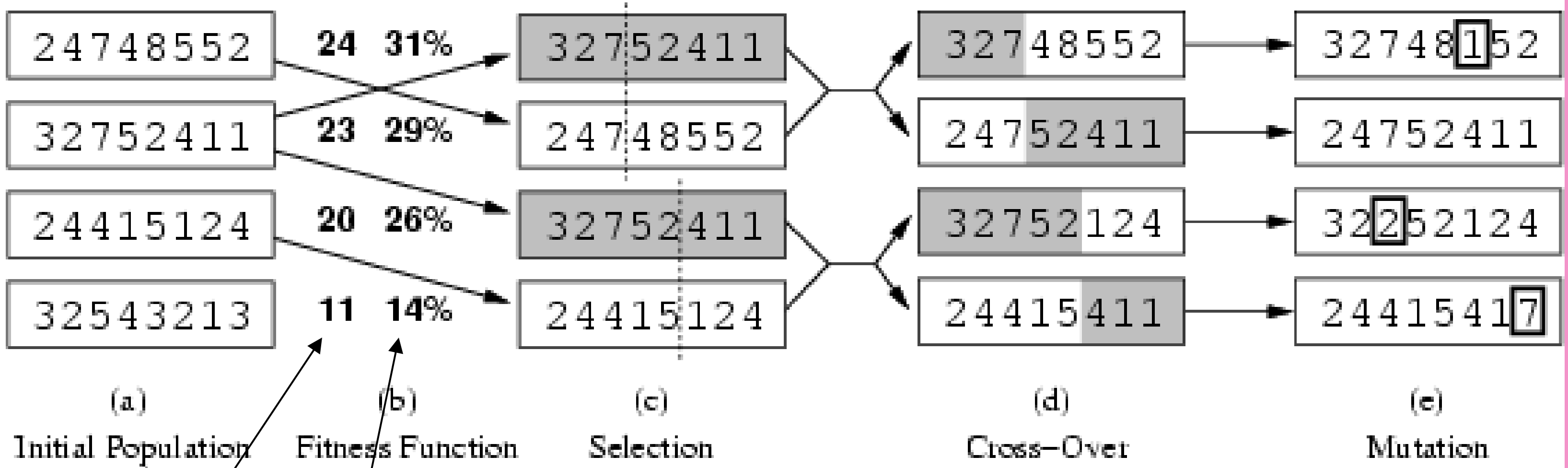
Checkboard example Cont’d

- Chromosomes represent the way the checkboard is colored.
- Chromosomes are not represented by bitstrings but by **bitmatrices**
- The bits in the bitmatrix can have one of the four values 0, 1, 2 or 3, depending on the color.
- Crossing-over involves matrix manipulation instead of point wise operating.
- Crossing-over can be combining the parential matrices in a horizontal, vertical, triangular or square way.
- Mutation remains bitwise changing bits in either one of the other numbers

Checkboard example Cont’d

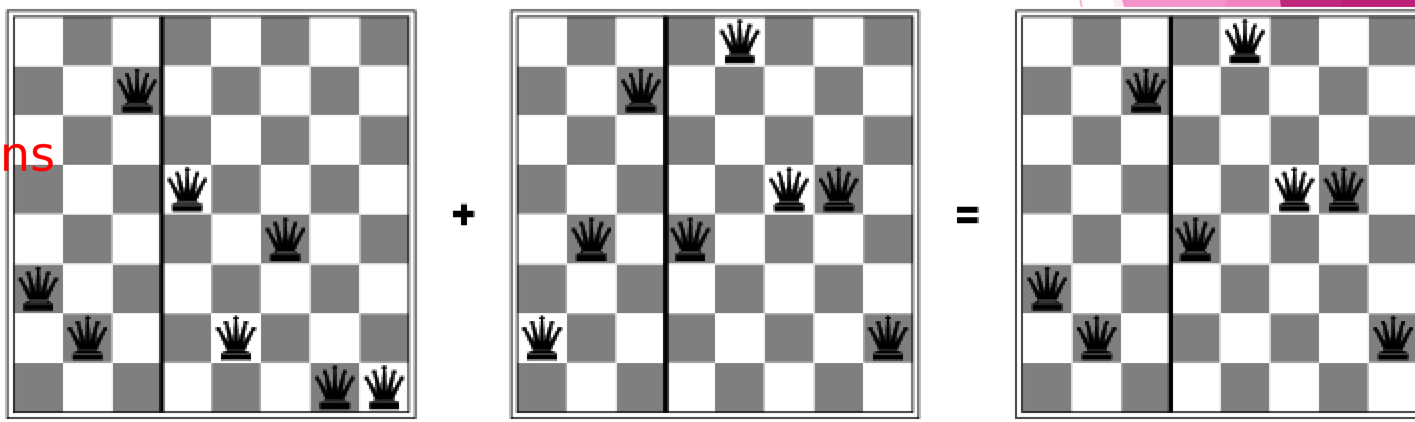
- This problem can be seen as a graph with ***n*** nodes and ***(n-1)*** edges, so the fitness **f(x)** is defined as:

$$\mathbf{f(x) = 2 \cdot (n-1) \cdot n}$$



itness:
#non-attacking queens

probability of being
regenerated
n next generation



- Fitness function: number of non-attacking pairs of queens
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Appendix