Artificial Intelligence

Informed Search

Chiến lược tìm kiếm kinh nghiệm

Informed (Heuristic) Search

- We have seen that uninformed methods of search are capable of systematically exploring the state space in finding a goal state.
- However, uninformed search methods are very inefficient in most cases.
- With the aid of problem-specific knowledge, informed methods of search are more efficient.

Outline

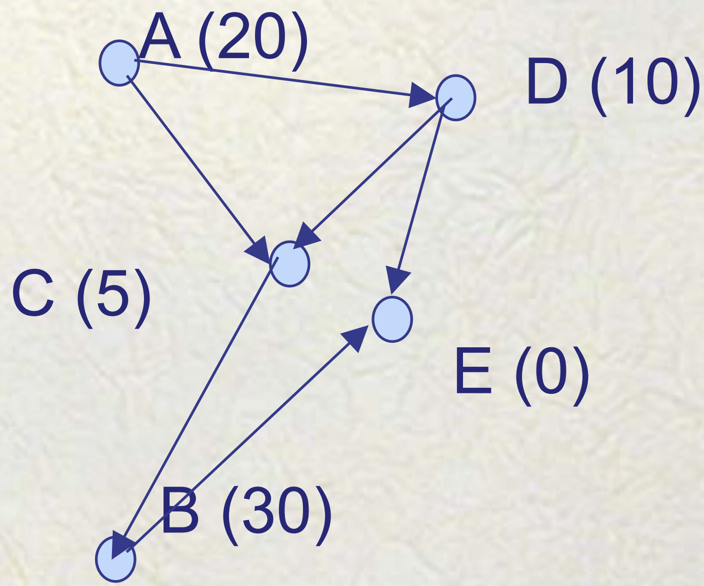
- Heuristics
- Informed Search methods:
  – Greedy Best-first search
  – Beam Search
  – Uniform-cost search
  – A* search

Heuristics

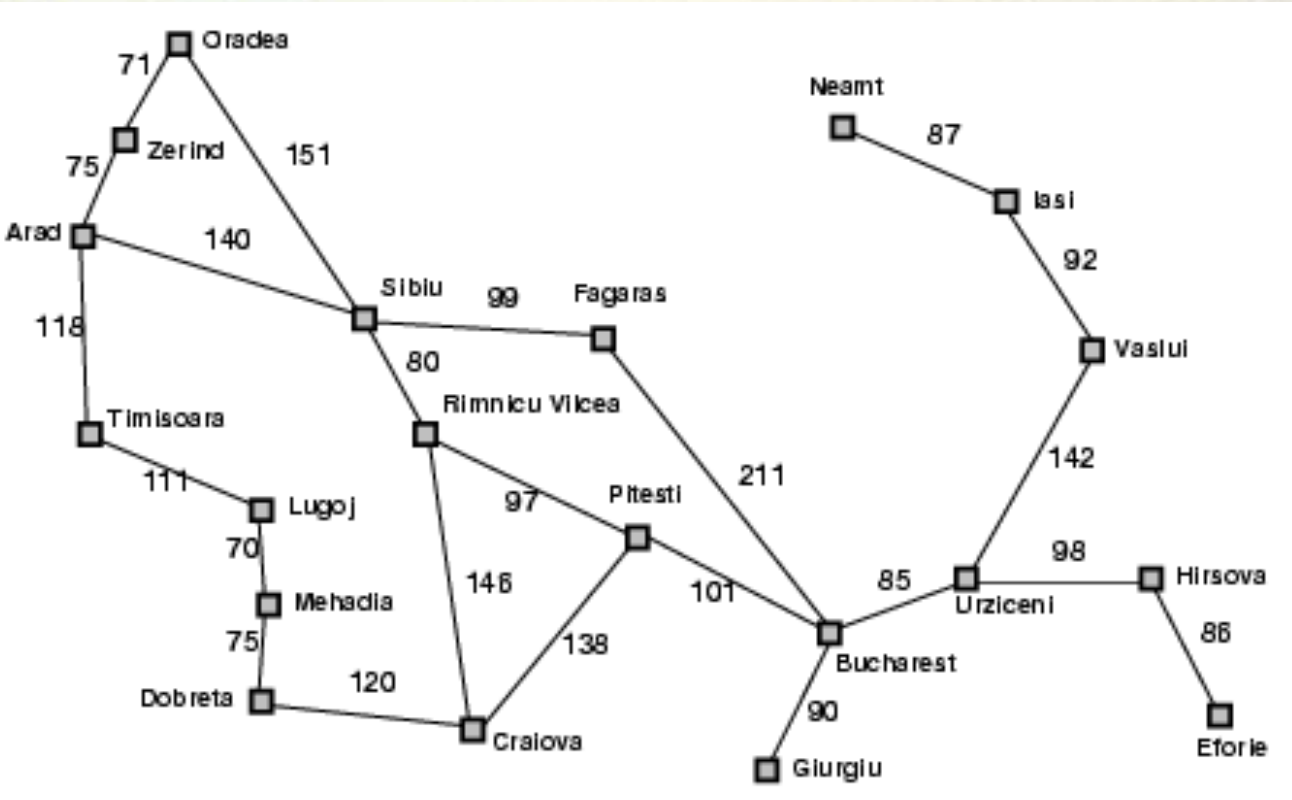
- "Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal."
- Can make use of heuristics in deciding which is the most "promising" path to take during search.
- Evaluation function h(u): a measure to evaluate the *distance* of state u from the goal. e.g: h(u) = 0 if u is the goal state.
- Evaluation functions (or heuristic functions) are problem specific functions that provide an estimate of solution cost.

Evaluation Function
Hàm đánh giá

- Travelling problem: The evaluation function take the value of the straight-line from one city to the destination city.



Evaluation Function



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

Evaluation Function

Eight-puzzle problem:

- The number of misplaced tiles, or
- Total sum of distances of a tile and its desired location.

4	3	1
	6	5
8	2	7

1	2	3
8		4
7	6	5

Evaluation Function

- The number of misplaced tiles: 9

- Total sum of distances of a tile and its desired location: 3 + 1 + 2 + 1 + 1 + 1 + 1 + 2 + 2 = 14

4	3	1
	6	5
8	2	7

1	2	3
8		4
7	6	5

Evaluation Function

- There are many ways to estimate the solution cost for an evaluation function.
- Evaluation functions might not be optimal.
- The quality of an evaluation function plays an important role in the effectiveness of the informed search.

Informed Search

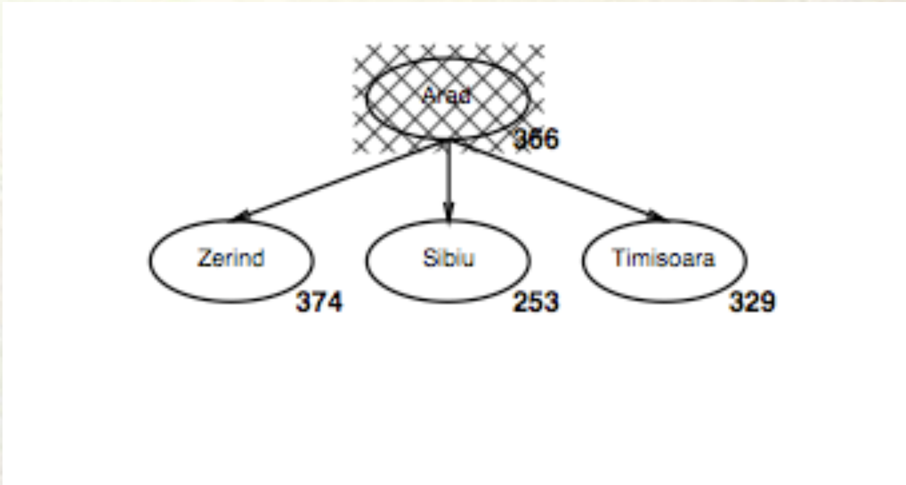
1. Task specification by identifying state space and actions.
2. Identify an evaluation function.
3. Design a strategy to choose which node to expand next.

Greedy Best-First Search

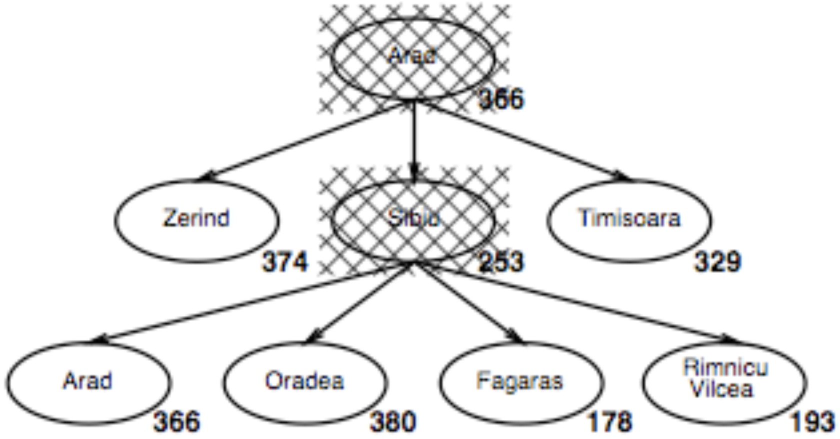
- Tìm kiếm tốt nhất đầu tiên
- Best first Search that selects the next node for expansion using the evaluation function h(u).
- Greedy search minimises the estimated cost to the goal; it expands whichever node u that is estimated to be closest to the goal.

Greedy best-first search example

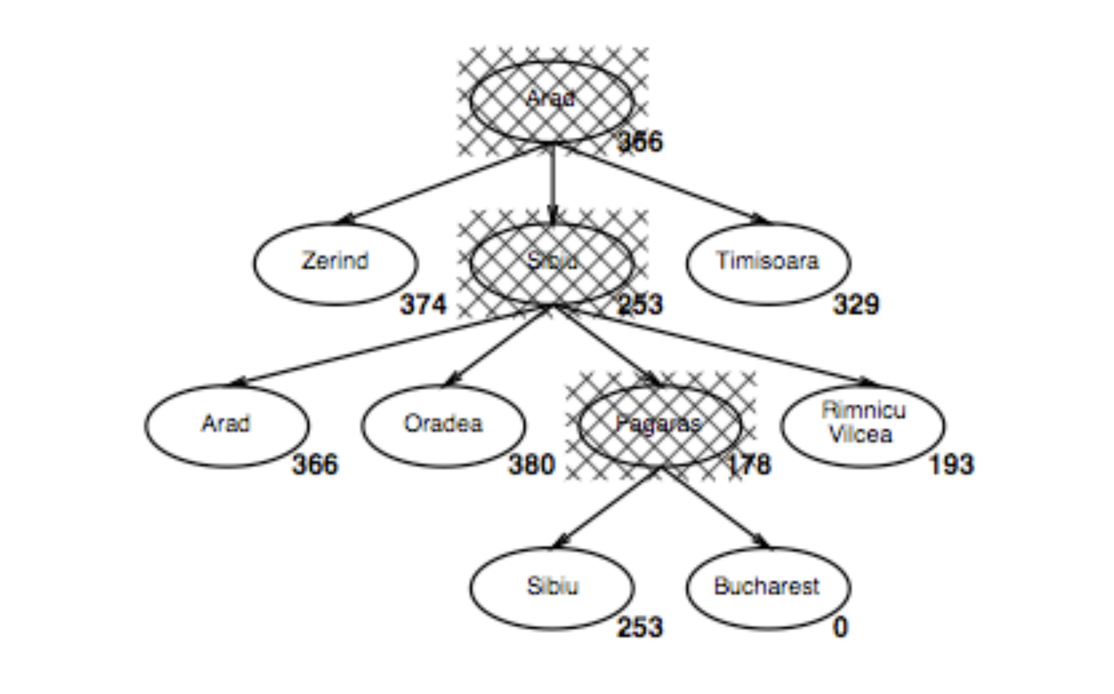
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



Greedy Best First Search

1. Initialize queue L containing only the initial state.
2. **Loop do**

   2.1 **If** (L is empty) **then**

   {search failed; exit}

   2.2 Take the first node u from beginning of L;

   2.3 **If** (u is a goal) **then**

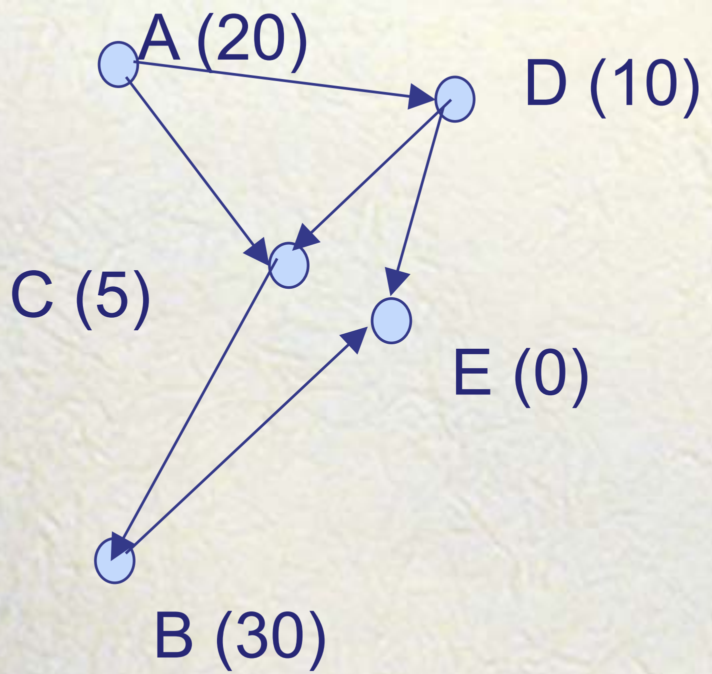
   {goal found; exit}

   2.4 **For** (each node v adjacent to u) **do**

   {Put v to L so that L is sorted in increasing order of the evaluation function}

Greedy Best first search

Find a path from A to E



- Find E

- **L**: A - A
- **L**: C, D - C
- **L**: D, B - D
- **L**: E, B - E
- Found E

Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →

  Complete in finite space with repeated-state checking
- Time? $O(b^m)$, m is the maximum depth in search space
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

A good heuristic function can reduce time and memory cost substantially.

Beam Search

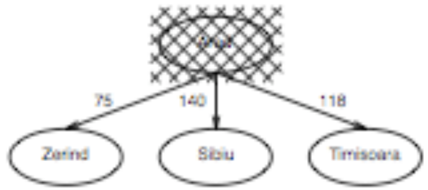
- Similar to greed best first search but only consider expanding k nodes at the next step i.e. the queue has a maximal size of k.
- Pros: better time complexity
- Cons: do not consider all paths, so might fail to find a solution i.e. not complete.

Uniform-Cost Search

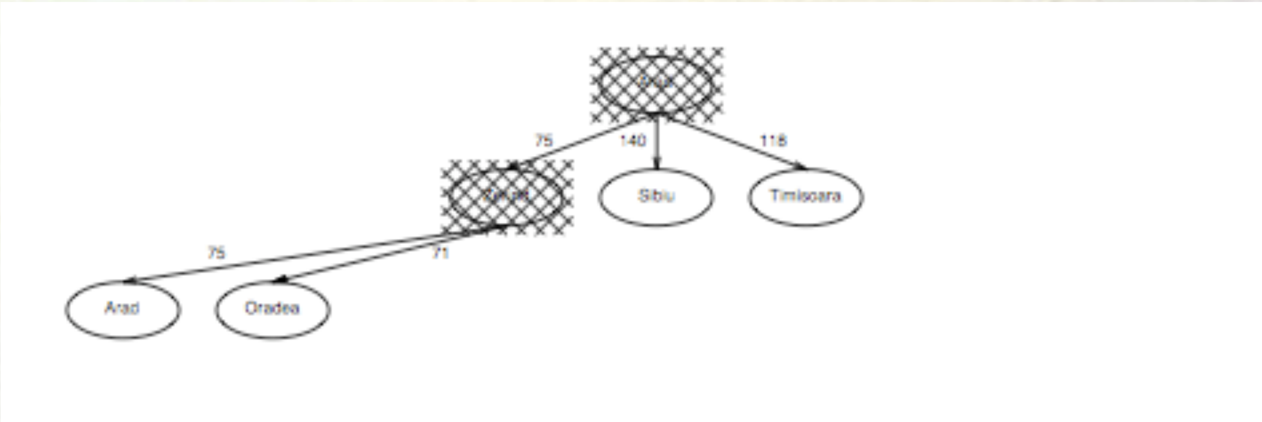
- Expand root first, then expand least-cost unexpanded node.
- Implementation: insert nodes in order of increasing path cost.
- Reduces to breadth-first search when all actions have same cost.
- Find the cheapest goal provided path cost is monotonically increasing along each path (i.e. no negative-cost steps)

Uniform Cost Search

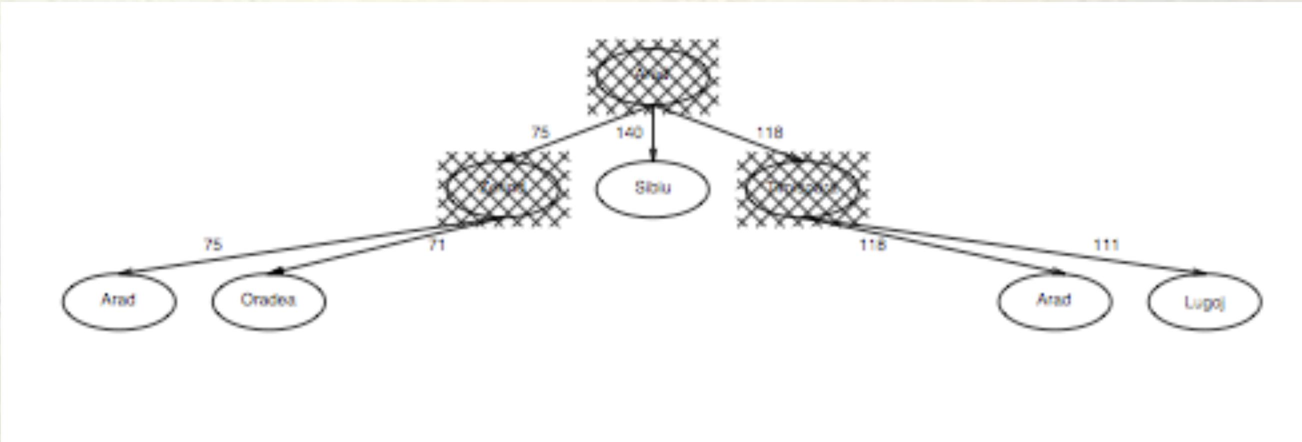
Uniform Cost Search



Uniform Cost Search



Uniform Cost Search



Properties of Uniform Cost Search

- Complete? Yes, if step cost >0 or b is finite
- Time? $O(b^m)$, m is the maximum depth in search space
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? Yes

Can we still guarantee optimality but search more efficiently, by giving priority to more promising nodes?

A* Search

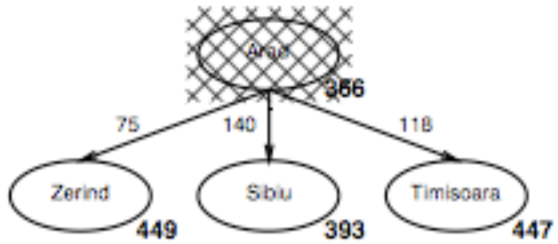
- A* Search uses evaluation function f(n) = g(n) + h(n)
  - g(n): cost from initial node to node n
  - h(n): estimated cost of cheapest path from n to goal.
  - f(n): estimated total cost of cheapest solution through n.
- Greedy best first search minimises h(n)
  - Efficient but not optimal or complete
- Uniform-cost search minimizes g(n)
  - Optimal and complete but not efficient

A* Search

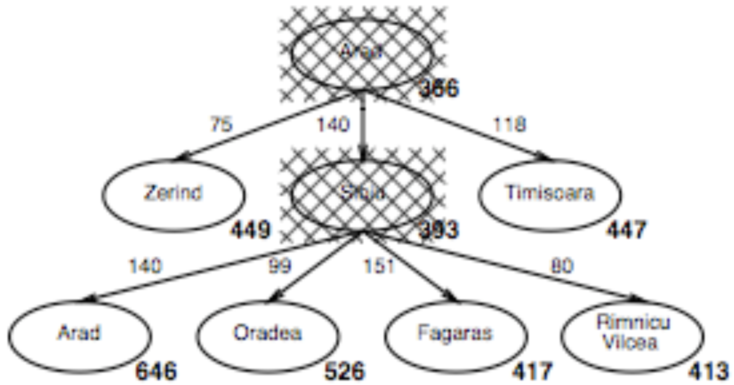
- A* search minimizes f(n) = g(n) + h(n)
  - Idea: preserve efficiency of Greedy Search but avoid expanding path that are already expensive
- Question: Is A* search optimal and complete?
- Yes! Provided h(n) is *admissible*- it never overestimates the cost to reach the goal.

A* Search Example

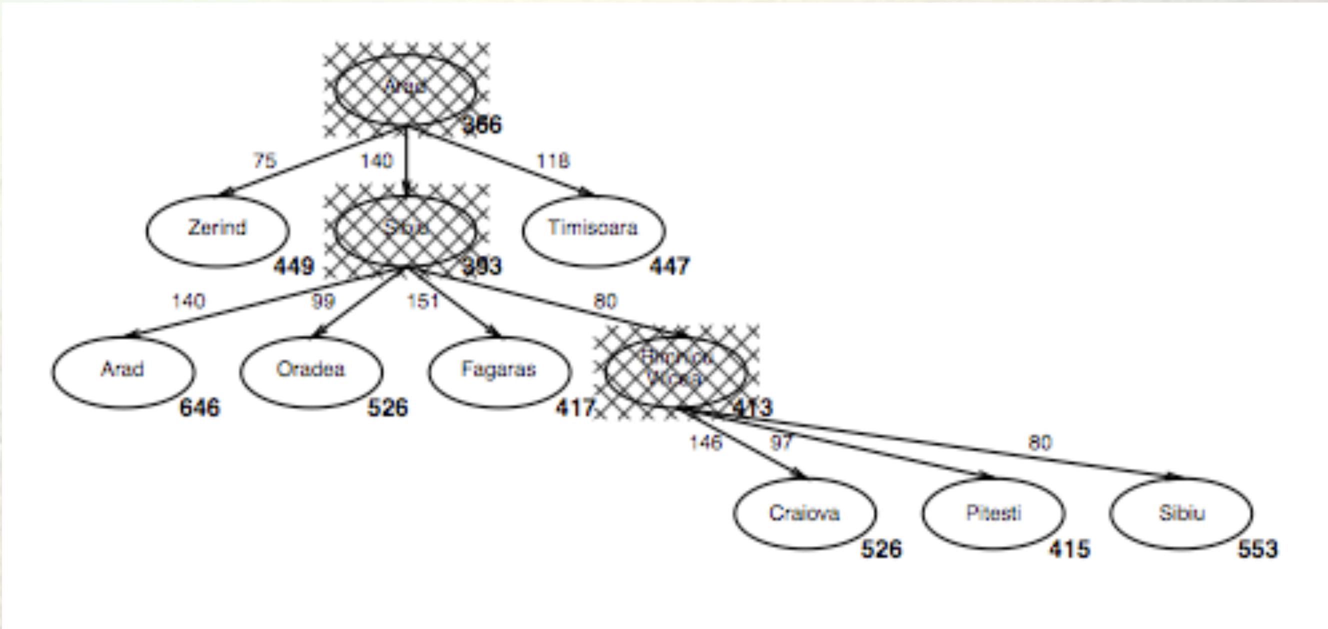
A* Search Example



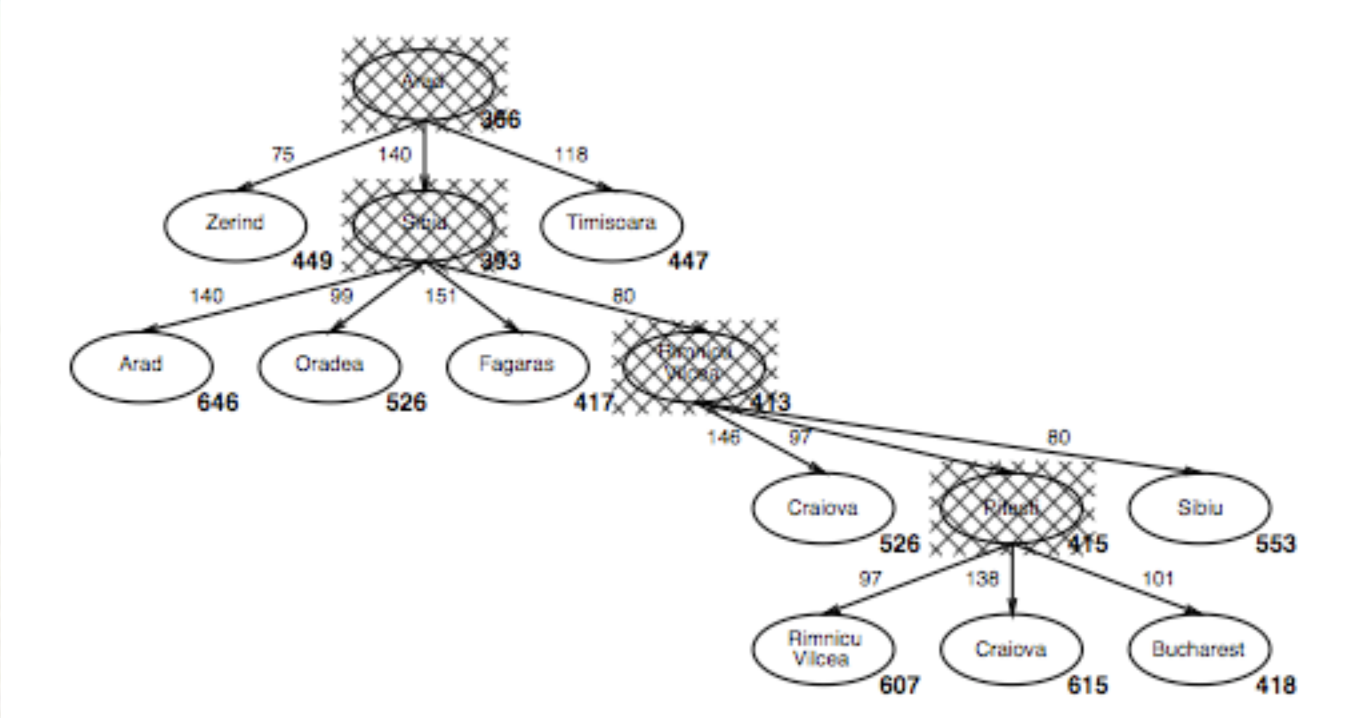
A* Search Example



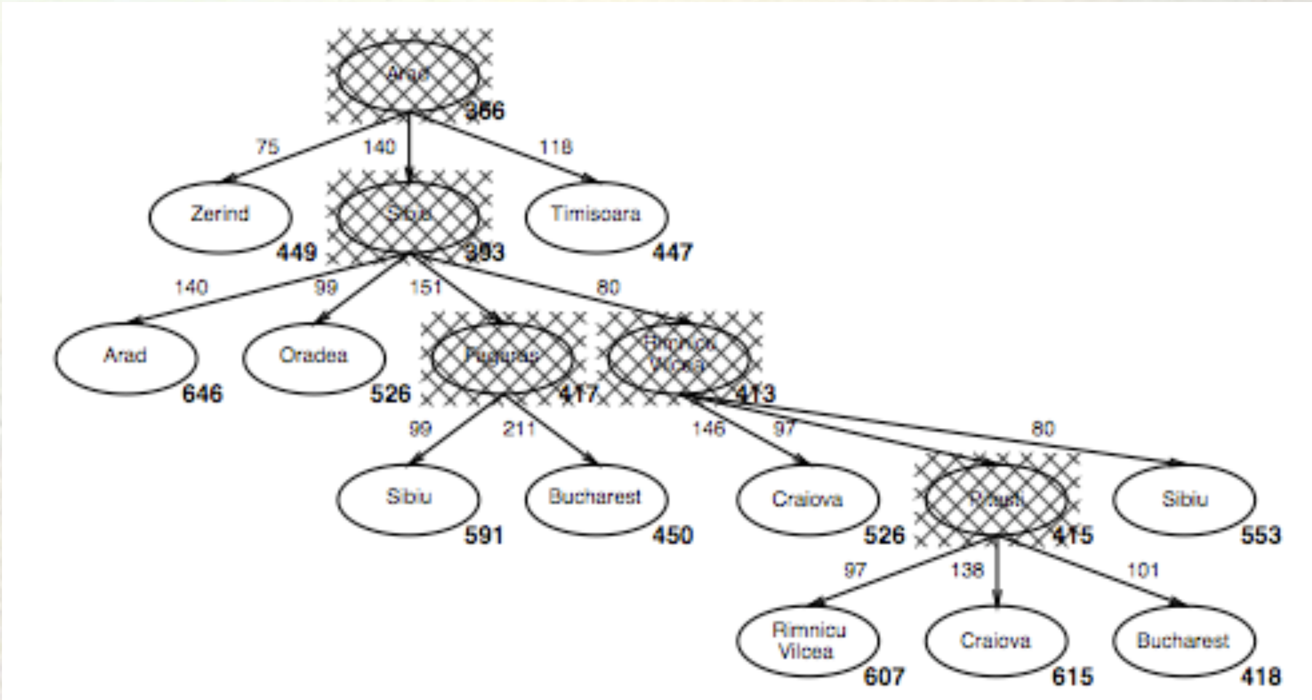
A* Search Example



A* Search Example



A* Search Example



A* Search

1. Initialize queue L containing only the initial state.
2. **Loop do**

   2.1 **If** (L is empty) **then**
   {search failed; exit}

   2.2 Take the first node u from beginning of L;

   2.3 **If** (u is a goal) **then**
   {goal found; exit}

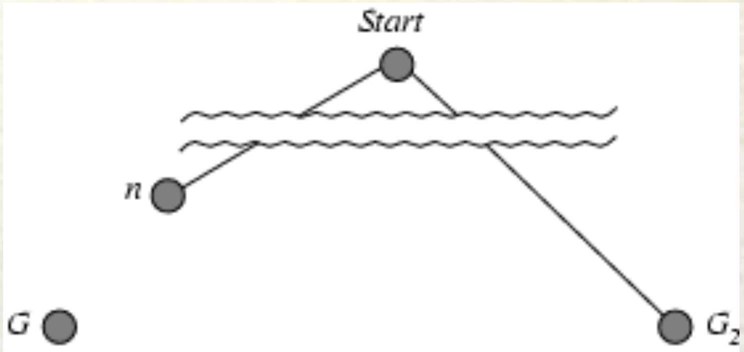
   2.4 **For** (each node v adjacent to u) **do**
   {g(v) := g(u) + k(u,v);
   f(v) := g(v) + h(v);
   Put v to L so that L is sorted in increasing order of the evaluation function f;}

Admisible Heuristics

- Hàm đánh giá chấp nhận được
- An evaluation function h(n) is admissible if h(n) is always optimistic ("lạc quan"): it never overestimates the optimal cost.
- If h(n) is admissible then A* 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*.



- $f(G_2) > f(G)$ 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

Optimality of A* Search

- Since $f(G_2) > f(n)$, A* will never select G_2 for expansion.
- The suboptimal goal node G_2 may be *generated*, but it will never be *expanded*.
- In other words, even after a goal node has been generated, A* will keep searching so long as there is a possibility of finding a shorter solution.
- Once a goal node is selected for expansion, we know it must be optimal, so we can terminate the search.

Properties of A* search

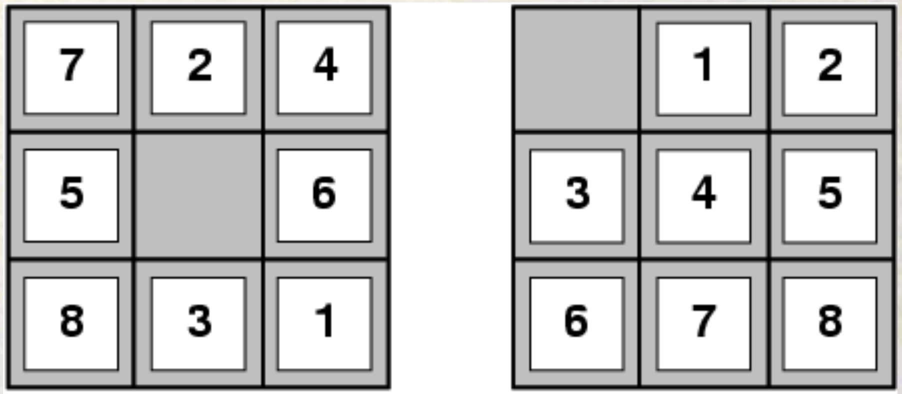
- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

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

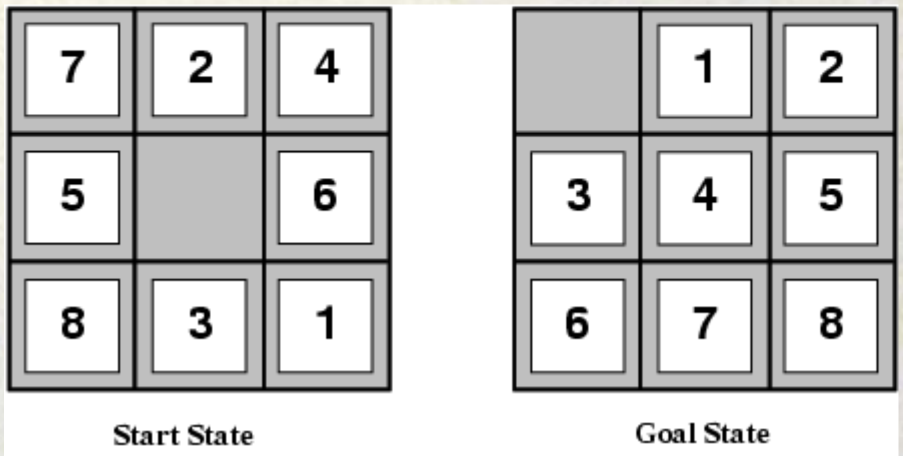
- $h_1(S) = ?$
- $h_2(S) = ?$

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)



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

Dominance
Tính áp đảo

- 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 ~ 54 * 10^9 nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Cách tìm admissible heuristics

- Giảm bớt ràng buộc.
- 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

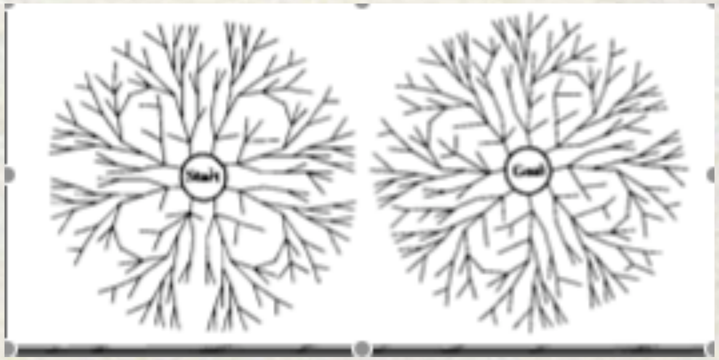
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Composite Heuristic Functions

- Let h_1, h_2,.., h_m be admissible heuristics for a given task.
- Define the composite heuristic:
  - $h(n) = \max (h_1(n), h_2(n), \ldots, h_m(n))$.
- h is admissible
- h dominates h_1, h_2, …, h_m

Bidirectional Search

- Symmetrical problems.
- We can have inverse operators.
- Explicate goal states



Properties of Bidirectional search

- <u>Complete?</u> Yes (if *b* is finite)
- <u>Time?</u> $O(b^{d/2})$
- <u>Space?</u> $O(b^{d/2})$
- <u>Optimal?</u> Yes (if uniform cost per step)

References

- Artificial Intelligence, A modern Approach. Chapter 4.
- AI Illuminated. Chapter 4.