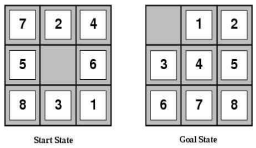
Informed/Heuristic Search

Note: this material was originated from the slides provided by Prof. Padhraic Smyth

Outline

- Limitations of uninformed search methods
- Informed (or heuristic) search uses problem-specific heuristics to improve efficiency
  - Best-first
  - A*
  - Techniques for generating heuristics
- Can provide significant speed-ups in practice
  - e.g., on 8-puzzle
  - But can still have worst-case exponential time complexity

Limitations of uninformed search



- 8-puzzle
  - Avg. solution cost is about 22 steps
  - branching factor ~ 3
  - Exhaustive search to depth 22:
    - 3.1 x 10^{10} states
  - E.g., d=12, IDS expands 3.6 million states on average

  [24 puzzle has 10^{24} states (much worse)]

Best-first search

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

- Implementation:
  - Order the nodes in fringe by f(n) (by desirability, lowest f(n) first)

- Special cases:
  - uniform cost search (from last lecture): f(n) = g(n) = path to n
  - greedy best-first search
  - A^* search

- Note: evaluation function is an <u>estimate</u> of node quality

  => More accurate name for "best first" search would be "seemingly best-first search"

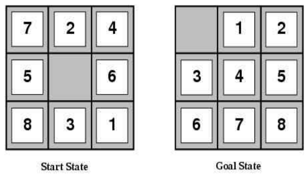
Heuristic function

- Heuristic:
  - Definition: "using rules of thumb to find answers"

- Heuristic function h(n)
  - Estimate of (optimal) cost from n 현재상태 n to goal
  - h(n) = 0 if n is a goal node
  - Example: straight line distance from n to Bucharest
    - Note that this is not the true state-space distance
    - It is an estimate – actual state-space distance can be higher

  - Provides problem-specific knowledge to the search algorithm

Heuristic functions for 8-puzzle



- 8-puzzle
  - Avg. solution cost is about 22 steps
  - branching factor ~ 3
  - Exhaustive search to depth 22:
    - 3.1×10^{10} states.
  - A good heuristic function can reduce the search process.

- Two commonly used heuristics
  - h_1 = the number of misplaced tiles
    - $h_1(s)=8$
  - h_2 = the sum of the distances of the tiles from their goal positions (Manhattan distance).
    - $h_2(s)=3+1+2+2+2+3+3+2=18$

Greedy best-first search

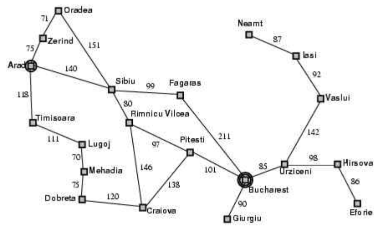
- Special case of best-first search
  - Uses h(n) = heuristic function as its evaluation function
  - Expand the node that appears closest to goal

Romania with step costs in km

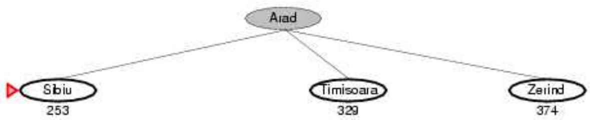
Straight–line distance to Bucharest

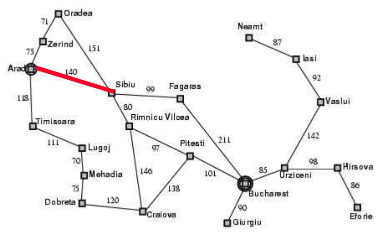
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

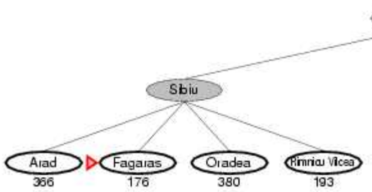


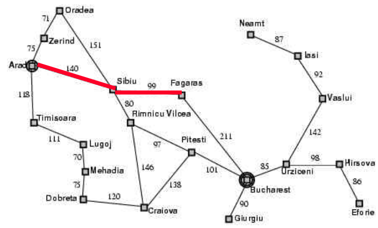
Greedy best-first search example



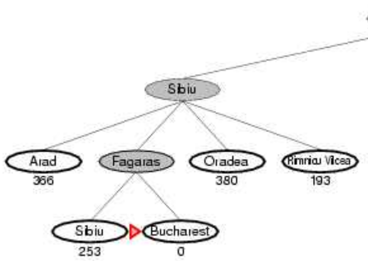


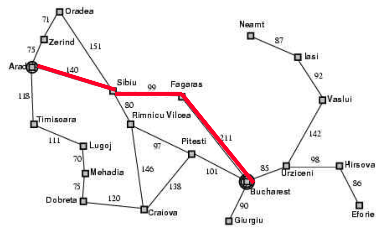
Greedy best-first search example



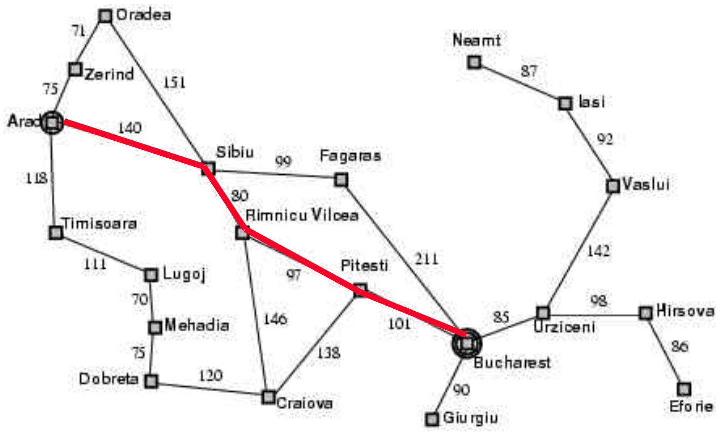


Greedy best-first search example





Optimal Path



Properties of greedy best-first search

- Complete?
  - Not unless it keeps track of all states visited
    - Otherwise can get stuck in loops (just like DFS)
- Optimal?
  - No – we just saw a counter-example
- Time?
  - *$O(b^m)$*, can generate all nodes at depth m before finding solution
  - m = maximum depth of search space
- Space?
  - *$O(b^m)$* – again, worst case, can generate all nodes at depth m before finding solution

A* Search

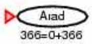
- Expand node based on estimate of total path cost through node

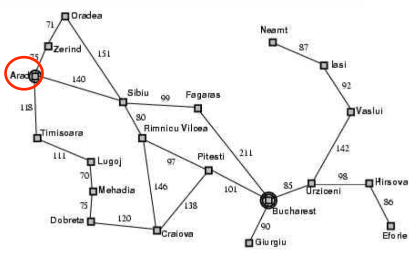
- Evaluation function *f(n) = g(n) + h(n)*
  - *g(n)* = cost so far to reach *n*
  - *h(n)* = estimated cost from *n* to goal
  - *f(n)* = estimated total cost of path through *n* to goal

- Efficiency of search will depend on quality of heuristic h(n)

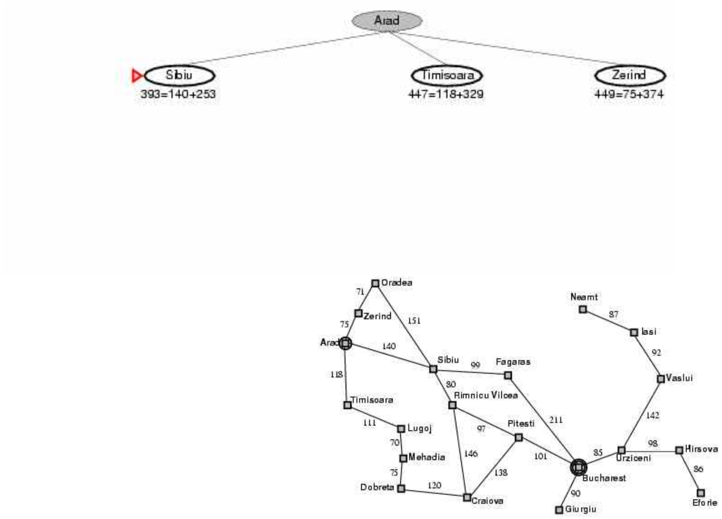
A* search example

shortest path distance : 죽어다 깨도 실제거리보단 작아야한다
Arad

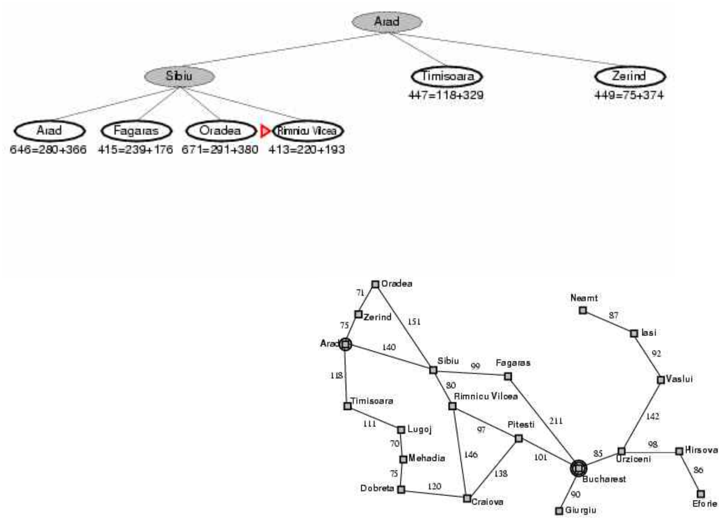




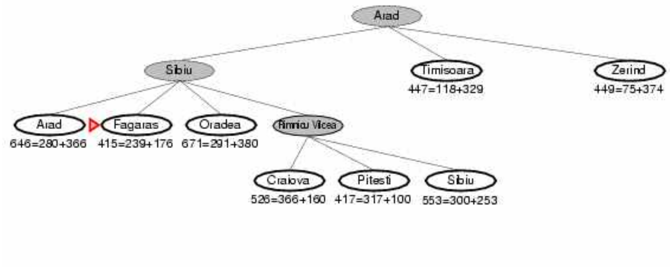
A* search example

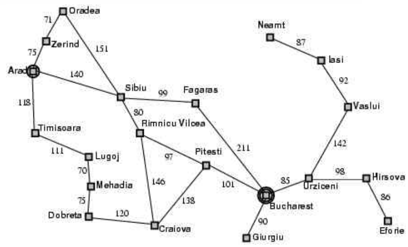


A* search example

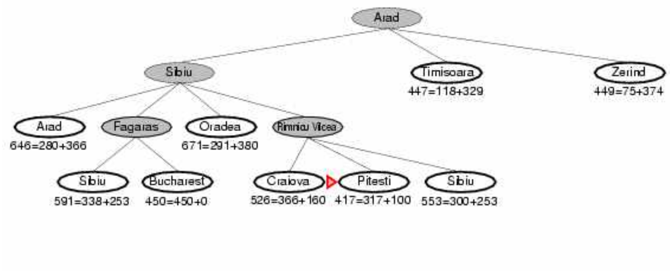


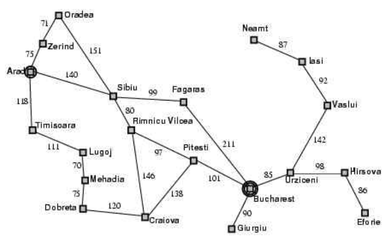
A* search example



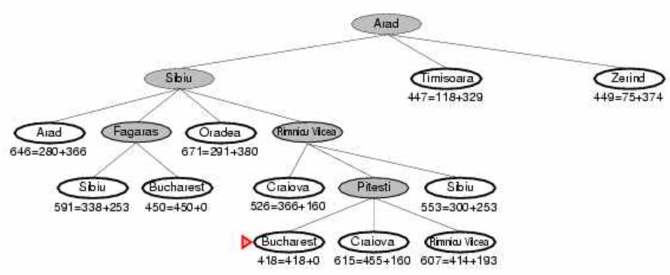


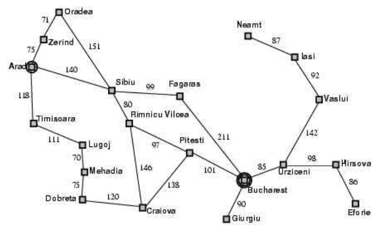
A* search example





A* search example





Admissible heuristics

a 알고리즘은 항상 optimal한 text를 찾는다 (X)
a 알고리즘은 f = g+ h 로 정의하고... 가장 좋은 방법을 찾는 것
a 알고리즘은 그대로 쓰는게 아니고 g + h 휴리스틱 특정한 조건을 가질 때 옵티멀. 항상성이 있다고 한다

- A heuristic *h(n)* is admissible if for every node *n*,
  뭔지는 모르지만~ 실제의 거리보다 작거나 같은 걸 만족.
  그 때, h는 admissible하다~
  $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from *n*.
  휴리스틱 함수를 설정할 때 실제 경로. 실제 나음 경로(거리)가 무엇인지는 모르겠지만 항상 그보다 작거나 같은걸 만족하게 h를 설정

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
  옵티멀하려면 h < h*가 항상 성립해야 한다.
  항상 실제값보다 작거나 같은게 정의가 되면 된다
  실제 비용보다 항상 더 과장해서 나오면 안된다 (optimistic)

- Example: $h_{StraighLineDistance}(n)$ *is admissible*
  - never overestimates the actual road distance

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

best

Properties of A*

- Complete?
  - Yes (unless there are infinitely many nodes with f ≤ *f(G)*)
- Optimal?
  - Yes
  - Also optimally efficient:
    - No other optimal algorithm will expand fewer nodes, for a given heuristic
- Time?
  - Exponential in worst case
- Space?
  - Exponential in worst case

Heuristic functions

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

타일의 개수로 남아있는 distance 정의 (낙관적으로)
실제 타일을 몇번 옮겨야하는냐? 적어도 8번 이상

- 8-puzzle
  - Avg. solution cost is about 22 steps
  - branching factor ~ 3
  - Exhaustive search to depth 22:
    - 3.1×10^{10} states.
  - A good heuristic function can reduce the search process.

admissible한 함수가 되려면, 실제 거리보다 작거나 같아야하는게 성립해야한다
h1, h2, h3 .. <= h*
h1<=h2면 h2이 더좋다고 얘기한다

두가지 방법을 쓸 수 있다

- Two commonly used heuristics
  - h_1 = the number of misplaced tiles
    - $h_1(s)=8$
  - h_2 = the sum of the distances of the tiles from their goal positions (manhattan distance).
    - $h_2(s)=3+1+2+2+2+3+3+2=18$ 8개 파일을 중간에 장애물이 없다고 생각하고 옮김.
실제의 거리는 이것보다 작을 수 없다!

h1, h2 모두 ㅁdmissible...

Notion of 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) for 8-puzzle problem

  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

Effectiveness of different heuristics

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

- Results averaged over random instances of the 8-puzzle

Inventing heuristics via “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

- Uninformed search methods have their limits
- Informed (or heuristic) search uses problem-specific heuristics to improve efficiency
  굳이 갈 필요가 없는데 영 아닌 것 같은건 일단 우선순위를 낮춰서 선택을 안한다
  그런 휴리스틱 함수를 쓴다
  - Greedy Best-first search 옵티멀할 필요가 없다
  - A* search
  - Techniques for generating heuristics
- Can provide significant speed-ups in practice
  - e.g., on 8-puzzle
  - But can still have worst-case exponential time complexity