CS 520: Introduction to Artificial Intelligence

Prof. Louis Steinberg

Lecture 3:

uninformed search

uninformed search

Lectures on Search

Review

- **Formulation of search problems.**
  - **State Spaces**
- **Uninformed (blind) search algorithms.**
- **Informed (heuristic) search algorithms.**
- **Constraint Satisfaction Problems.**
- **Game Playing Problems.**

Review

State spaces: formulating problems as graph search

- **States**
- **Operators**
- **Start and goal states**
- **Costs**
  - **Cost of search**
  - **Cost of path from start to goal**
  - **Cost of goal**

Review

- **Some classical problems**
  - **As examples of formulation**
  - **Because they are often referred to**
- **Problems**
  - **8 puzzle (also 15 puzzle)**
  - **N-queens**
  - **Water jugs**
  - **Towers of Hanoi**

Review

- The same problem can be formalized in different ways.
  - Better if fewer nodes in space
  - Better if fewer paths to same node
- A non-classical problem: vacuum cleaner
  - An example of a non-observable environment

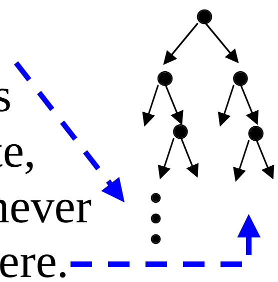
Lectures on Search

- Formulation of search problems.
- Uninformed (blind) search algorithms.
- Informed (heuristic) search algorithms.
- Constraint Satisfaction Problems.
- Game Playing Problems.

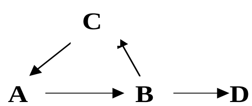
Searching a Graph

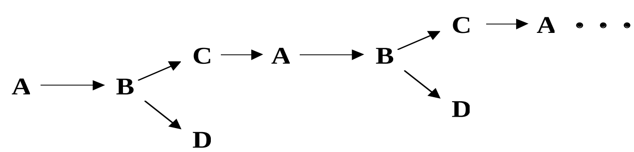
- Simple tree search
  - Depth first, breadth first, constant cost
- Problem: infinite trees

If this path is infinite, DFS never gets here.



- Problem: cycles look like infinite trees



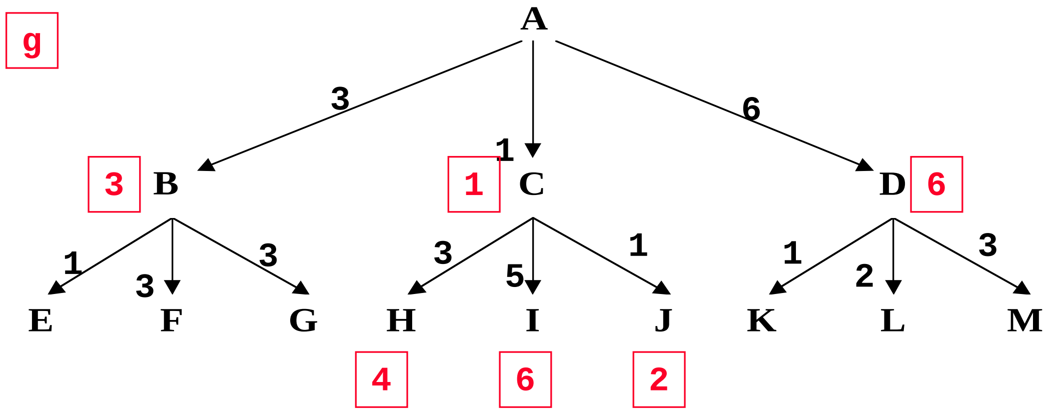


Iterative Deepening

- Completeness and shortest-first order of BFS
- Space cost of DFS
- Time cost not much more than B/DFS

Uniform Cost

- Keep a list of open nodes (aka frontier)
- g(n) = distance from root
- Loop expanding node with minimal g



Lectures on Search

- Formulation of search problems.
  - State Spaces
- Uninformed (blind) search algorithms.
- Informed (heuristic) search algorithms.
- Constraint Satisfaction Problems.
- Game Playing Problems.

Informed Search

- Incorporate problem-specific knowledge into the search strategy.
- Only useful if extra knowledge exists.
- One common form:

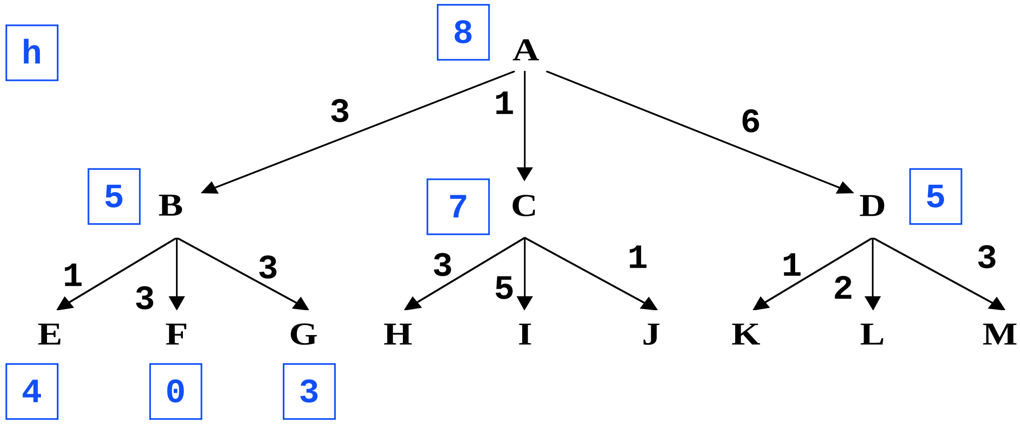
  f(n) = estimate of distance to goal

Best-First Search

- Expand the nodes in order of their f value.
- Also known as “Greedy” search

```
Initialize open to contain root
Loop while not(found or empty(open))
  new = expand(head (open))
  open =
     sort-by-value(append(rest(open), new))
```

Example



The Sixteen Puzzle

1	10	3	8
15	6	7	9
4	2		5
13	14	12	11

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- **Number of misplaced tiles: h(n) = 9**

The Sixteen Puzzle

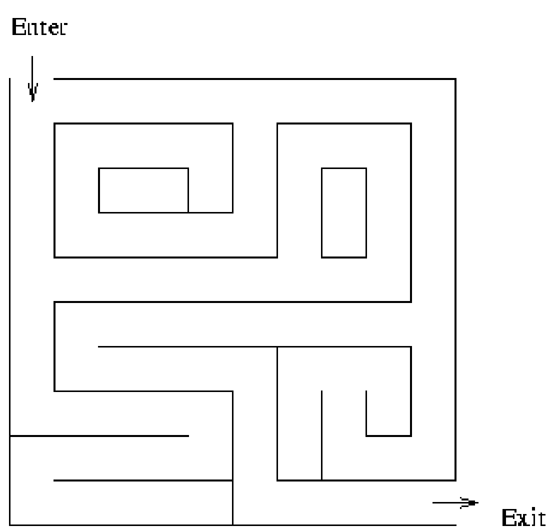
1	10	3	8
15	6	7	9
4	2		5
13	14	12	11

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

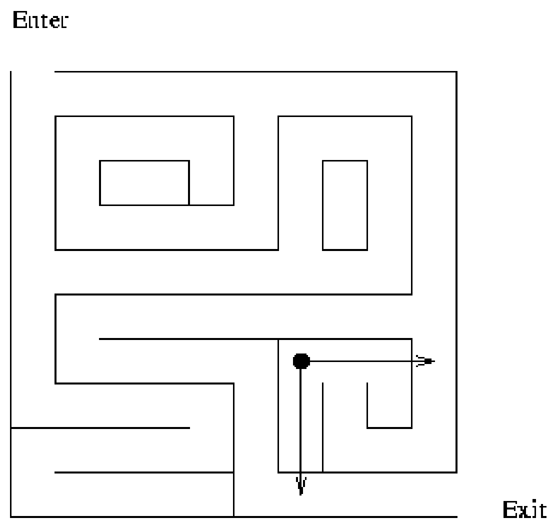
5 steps

- **Sum of Manhattan distances to correct place, h(n)=26**

A Maze Problem

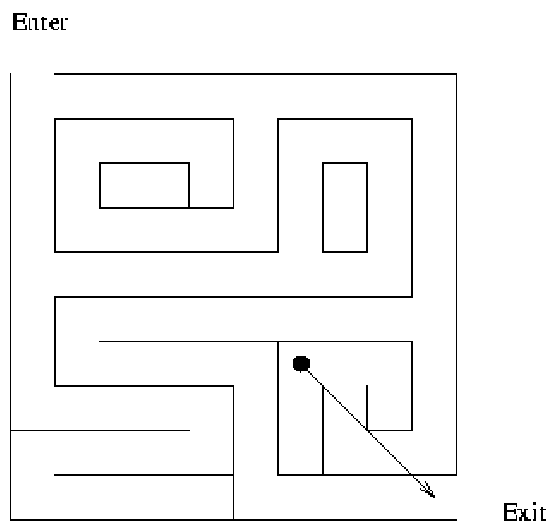


Heuristic for a Maze Problem



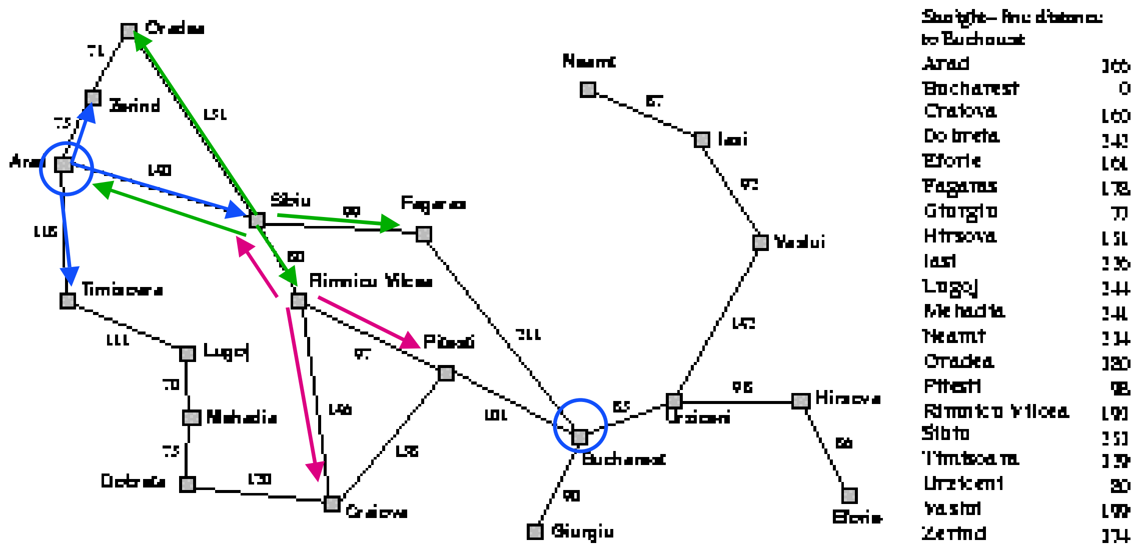
Manhattan Distance: h(s) = North/South distance to exit
+ East/West distance to exit.

Heuristic for a Maze Problem



Euclidean Distance: h(s) = Straight line distance to
exit "as the crow flies".

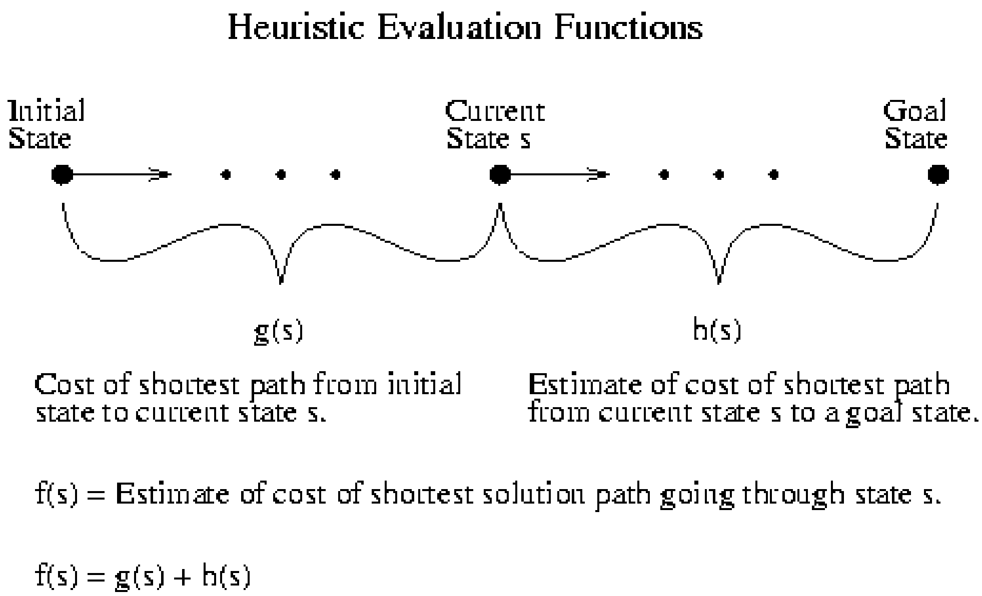
Greedy Search example



Greedy Search example

- **Notes:**
  - **not optimal (best path through R.V. and Pitesti)**
  - **prone to false starts (consider Iasi to Fagaras)**
  - **not complete (if repeated states are not checked)**

A* idea



Monotonic Heuristic Function

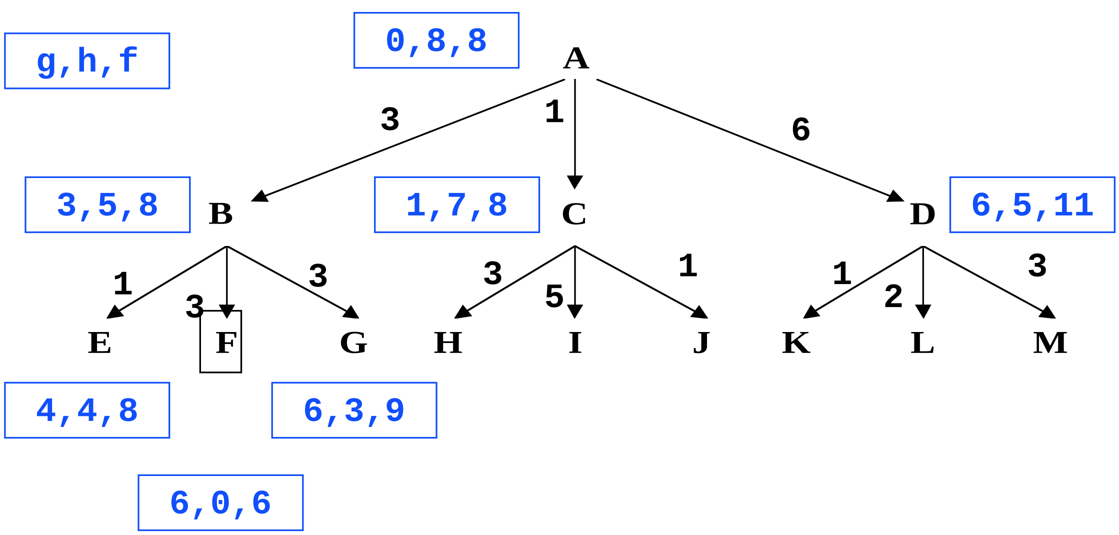
- **A heuristic evaluation function h(s) is said to be ``monotonic'' if f(s)=g(s)+h(s) does not go down along any path in the state space.**

Monotone f

- $f(s_{i+1}) \geq f(s_i)$
- $g(s_{i+1}) + h(s_{i+1}) \geq g(s_i) + h(s_i)$
- $g(s_i) + cost(O(s_i)) + h(s_{i+1}) \geq g(s_i) + h(s_i)$
- $cost(O(s_i)) + h(s_{i+1}) \geq h(s_i)$
- $h(s_{i+1}) \geq h(s_i) - cost(O(s_i))$

- **How to define a monotonic heuristic evaluation function?**
  - $h_{monotonic}(s_{i+1}) = \max[h(s_{i+1}), h(s_i) - cost(O(s_i))]$

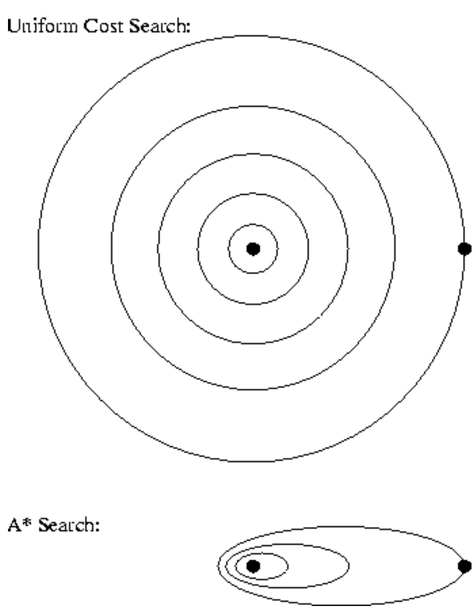
A* example



A* search contours

- **If f^* is the cost of the optimal solution path, then:**
  - **A* expands all nodes with $f(n) < f^*$.**
  - **A* may expand some of the nodes on the ``goal contour'' for which $f(n) = f^*$ before selecting a goal node.**
  - **A* is   optimally efficient for a give heuristic (no other algorithm is guaranteed to expand fewer nodes).**

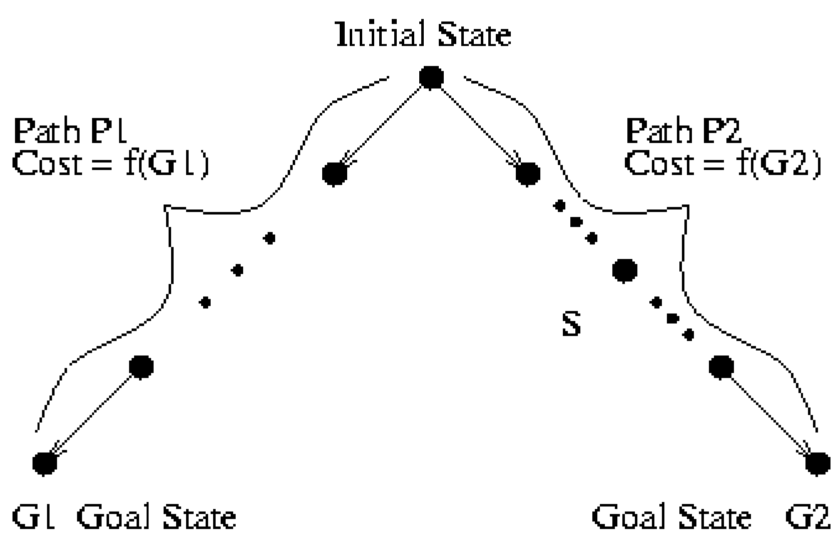
Topographical Interpretation of A* Algorithm



Admissible Heuristic Function

- A heuristic evaluation function h(s) is said to be ``admissible'' if h(s) is always less than or equal to the cost of the shortest path
- from s to a goal state, i.e., h(s) underestimates the cost of reaching a solution from state s.
- Therefore:
  – The value of h(GoalState) is zero.
  – The value of f(GoalState) is the exact cost of a shortest path
  – from the initial state to a goal state.

A* Finds Optimal Solutions if h(s) is Admissible



How complete is A*

- **A* will expand nodes in order of increasing f and thus it will eventually reach a goal state with cost f***
- **Unless there is an infinite number of states with $f(n) \le f^*$.**

Heuristic Performance

- **If N nodes are expanded and the solution depth is d, then the effective branching factor , b* , is the banching factor that a uniform tree of depth d would need to contain N nodes.**
- **$N = 1+b^*+(b^*)^2+ \ldots+(b^*)^d$**
- **h_2 dominates h_1 if, for every node n,**
  $h\_2(n) \ge h\_1(n)$
- **Larger heuristics have smaller branching factors.**
- **If several heuristics exist for a problem, the best way is to use a composite heuristic:**
- **$h(n)=\max(h\_1(n),\ldots,h\_m(n))$**

Comparison of performance for 8-puzzle

	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	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	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

Where do Heuristics Come From?

- **Convert the original problem into a simpler one.**
  - **E.g., A decomposable problem.**
- **Solve the simple problem using a specialized method.**
  - **E.g., By decomposition/recomposition.**
- **Use the solution of the simple problem as a guide to solving the original problem.**
  - **E.g., Let heuristic evaluation function h(s) compute the length of the solution to the simple problem.**