IS-ZC444: ARTIFICIAL INTELLIGENCE

Lecture-04: Problem Solving by Search (contd..)

Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

Sept 12, 2020 FLIPPED (WILP @ BITS-Pilani Jul-Nov 2020)

Recap

AI problem can be formulated using five components

1. The **initial state**
2. Set of **actions**
3. **Transition model**
4. **Goal test**
5. **Path cost**

A *goal-based agent*, called **problem-solving agent** is deployed

that uses state (cumulative) of the world to search for the solution.

- Problems: Vacuum Cleaner, 8-Puzzle, Knuth conjuncture, Route finding, Touring problem: visit each city, TSP: touring with single visit of cities, VLSI layout, Robot navigation, Automatic assembly sequencing
- Searching for Solution (search tree): DFS, BFS

Search

Uninformed or blind search:

strategies have no additional information about the stated beyond what is provided in the problem definition

- BFS: breadth first search generates $O(b^d)$ nodes for evaluation where b is branching factor and d is depth of the tree
- DFS: same number of nodes but order of evaluation is different (frontier is *stack*[a])

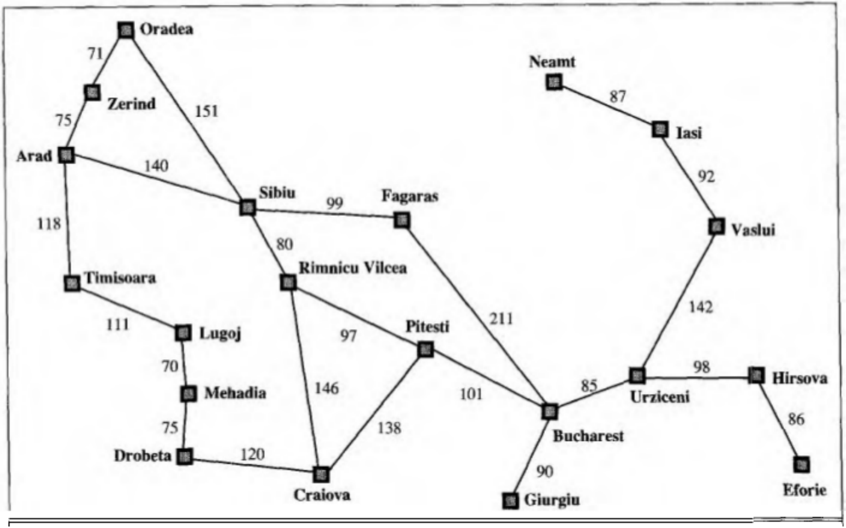
[a] BFS uses *queue*

Informed or heuristic search:

strategies know which non-goal state is more promising then other

Memory requirement and time taken is a major factor

Map of Romania



[3] Uniform-cost Search

Let's not use *stack* or *queue* as frontier

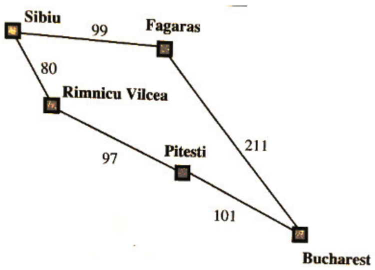
- instead use a list, that is **sorted** by using a *path cost* function $g(n)$, where n is a node in the list

Also

- Goal teat is applied to a node when it is selected for expansion
- Value of $g(n)$ is updated if a better path is explored

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

[3] Uniform-cost Search



- To go from Sibiu to Bucharest, successors are Vilcea and Fagaras (cost 80 and 99).
- Least cost node Vilcea is expanded the adds Pitesti (cost 80+97=177).
- Least node is now Fagaras so it is expanded to add Bucharest (cost 99+211=310)

However goal node is generated but, algorithm would go on expanding Pitesti that again adds/modifies Bucharest, with cost 80+97+101=278.

In general, uniform-cost search is optimal

Space/time complexity $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ where C^* is cost of optimal solution and ϵ is the least cost of each action

[4] Depth-limited Search

In DFS, do not go beyond the depth-limit ℓ

Time and space complexities would be $O(b^\ell)$ and $O(b\ell)$

- Idea is that going beyond **diameter** is not needed

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
     cutoff_occurred? ← false
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        result ← RECURSIVE-DLS(child, problem, limit − 1)
        if result = cutoff then cutoff_occurred? ← true
        else if result ≠ failure then return result
     if cutoff_occurred? then return cutoff else return failure
```

[5] Iterative Deepening Depth-first Search

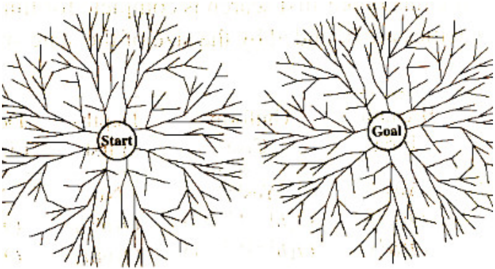
It is a general strategy that applies *Depth-limited Search* multiple number of times (gradually increasing depth-limit ℓ). First 0, then 1 then 2 and so on.... until the goal is found

- Memory requirement is $O(bd)$
- Iterative generation of same states seem wasteful but it is not as most of the nodes are at leaf (try hybrid if needed)
- Time complexity is $O(b^d)$

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
    for *depth* $= 0$ **to** ∞ **do**
        result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
        if *result* $\neq$ cutoff **then return** *result*

[6] Bidirectional Search

Can we run two search? one in forward and another in backward. In a hope that they would meet in middle.



- Expected search time is $b^{d/2} + b^{d/2}$ that is much less than b^{d}

If a problem has solution at depth 6, and each direction runs BFS, and the search meets at depth 3. For b=10, total nodes generated is 2220 as compared to 1111110 for standard BFS.

Comparing uniformed search strategies

Criterion	BFS	Uniform Cost	DFS	Depth Limited	Iterative Deepening	Bidirectional
Complete?	Y	Y	N[1]	N	Y	Y
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Y	Y	N	N	Y	Y

Let

- *b*: branching factor, *d*: depth of shallowest solution, *m*: depth of search tree
- A **complete** strategy finds a solution if it is there

[1] if depth is infinite

Informed Search: [7] Greedy best-first Search

For expansion, nodes are selected on an **evaluation function** $f(n)$

- A **heuristic function** $h(n)$ takes a node as input and its output depends only on the state of the node. (cheapest path)
- Heuristic function provides additional knowledge, $h(n) = 0$ for goal

Greedy best-first Search

expands the node closest to goal.

For example let straight-line-distance (SLD) to Bucharest is known.

Arad=366, Mehadia=241, Bhcharest=0, Neamt=234, Craiova=160, Drobeta=242, Eforie=161, Fagaras=176, Giurgiu=77, Hirsova=151, Iasi=226, Lugoji=244, Oradea=380, Pitesti=100, Rimnicu Vilcea=193, Sibiu=253, Timisoara=329, Urziceni=80, Vaslui=199, Zerind=374

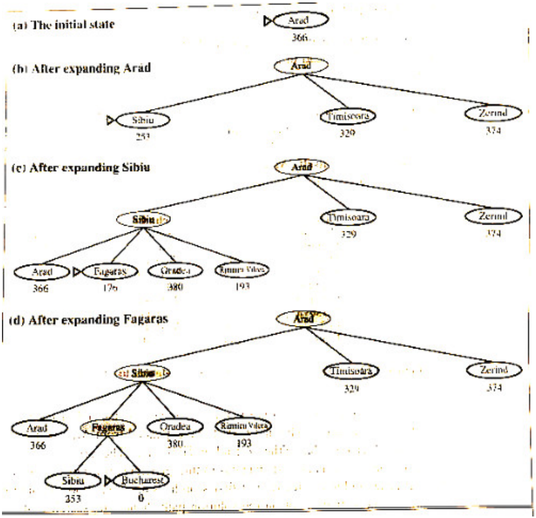
- We can use SLD to consider nodes for expansion. However, it may not be optimal [2].

---

[2] Consider getting from Iasi to Fagaras. The hurestic suggest that Nemt be expanded first because it is closest to Fagaras, but it is a dead end.

Greedy best-first Search

Arad=366, Mehadia=241, Bhcharest=0, Neamt=234, Craiova=160, Drobeta=242, Eforie=161, Fagaras=176, Giurgiu=77, Hirsova=151, Iasi=226, Lugoji=244, Oradea=380, Pitesti=100, Rimnicu Vilcea=193, Sibiu=253, Timisoara=329, Urziceni=80, Vaslui=199, Zerind=374



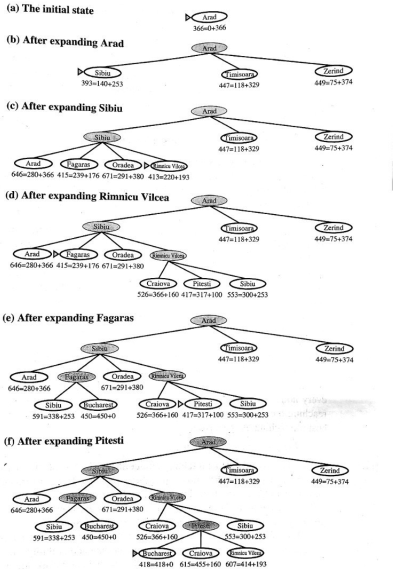
A* Search (A star)

Use both, path-cost and heuristic

$$f(n) = g(n) + h(n)$$

- A* is both optimal and complete
- It expands nodes in order of increasing f value
- No other optimal algorithm is guaranteed to expand fewer nodes than A*

A* Search (A star)



Thank You!

Thank you very much for your attention!

Queries ?

(Reference[3])

[3] Book - *AIMA*, ch-03, Russell and Norvig.