CS-F407: Artificial Intelligence

09 Search Techniques

Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

August 30, 2024 M/W/F - 3:00 pm 6101 @ BITS-Pilani [Aug-Dec 2024]

http://ktiwari.in/ai

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*

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

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])

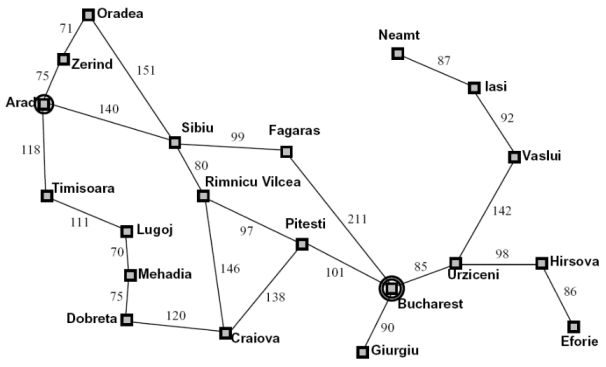
[a] BFS uses *queue*

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

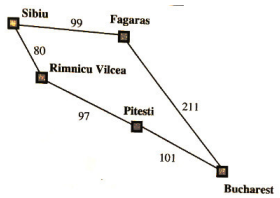
- instead use a list, that is **sorted** by using a *path cost* function $g(n)$, where n is a node in the list

Also

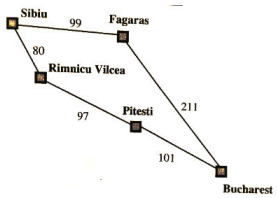
- Goal teat is applied to a node when it is selected for expansion
- Value of $g(n)$ is updated if a better path is explored

```
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

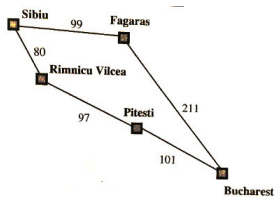


[3] Uniform-cost Search



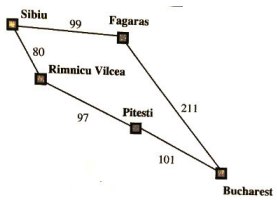
- To go from Sibiu to Bucharest, successors are Vilcea and Fagaras (cost 80 and 99).

[3] Uniform-cost Search



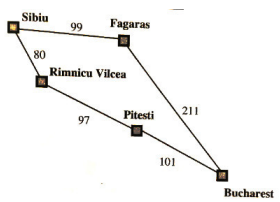
- To go from Sibiu to Bucharest, successors are Vilcea and Fagaras (cost 80 and 99).
- Least cost node Vilcea is expanded the adds Pitesti (cost 80+97=177).

[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)

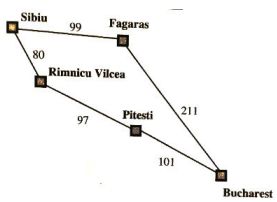
[3] Uniform-cost Search



- 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.

[3] Uniform-cost Search

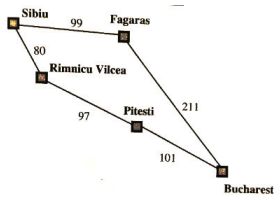


- To go from Sibiu to Bucharest, successors are Vilcea and Fagaras (cost 80 and 99).
- Least cost node Vilcea is expanded the adds Pitesti (cost 80+97=177).
- Least node is now Fagaras so it is expanded to add Bucharest (cost 99+211=310)

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

[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($problem$, ℓ) **returns** a node or *failure* or *cutoff*
  $frontier \leftarrow$ a LIFO queue (stack) with NODE($problem$.INITIAL) as an element
  $result \leftarrow failure$
  while not IS-EMPTY($frontier$) **do**
    $node \leftarrow$ POP($frontier$)
    if $problem$.IS-GOAL($node$.STATE) **then return** $node$
    if DEPTH($node$) $>$ ℓ **then**
      $result \leftarrow cutoff$
    else if not IS-CYCLE($node$) **do**
      for each $child$ **in** EXPAND($problem$, $node$) **do**
        add $child$ to $frontier$
  return $result$

[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

[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 node or *failure*
  for $depth = 0$ **to** ∞ **do**
    $result \leftarrow$ DEPTH-LIMITED-SEARCH($problem$, $depth$)
    if $result \neq cutoff$ **then return** $result$

Thank You!

Thank you very much for your attention!

Queries ?

(Reference[1])

[1] Book - *AIMA*, ch-02+03, Russell and Norvig.