Artificial Intelligence - 2018
AI Intro, Uninformed, informed Search

Prepared By;
Vijani Supeshala Piyawardana | BSc (Hons) Computer Science |
Demonstrator - COMP3004L, COMP3008L - UCD |
NSBM Green University Town

Problem Solving using searching

Problem Solving

Search is considered as one of the most commonly used primary intelligent ability. Search can be very primitive or very advanced depending on the problem knowledge.

Any problem solving can be seen as going from one state to another by action.

S1 ----action----> S2

(1+2-5) -----add----> (3-5) -----add----> (-2)

Searching in something to do with states and actions related to a particular problem solving task. The searching involves only when more than one action is possible from a state.

Problem solving by search can be characterized by the concept of states (initial state, goal state), actions, path, state space and the path connecting initial state and the goal.

State space is the graphical representation of all the states and actions to solve the problem. The path from initial state to goal state is known as a solution.

In a diagram, states are denoted by the nodes and actions are denoted by the arcs.

01) Robot navigation

-pic-

Draw the state space for robot navigation from B to T.

-pic-

Robot can have many traversal from B → T.

1. B A P V Q T (a path)
2. B Q T
3. B S T
4. …
5. ….

On the state space, there can be many solutions which are shorter, cheaper, …. Etc than the others.

02) Water jug problem

-pic-

-pic-

It shows a state space for a small problem would be still very large.

03) Farmer, wolf, goat and cabbage problem.

-pic-

-pic-

04) 8-Puzzle

-pic-

-pic-

Search also can be used for designing something. Ex: design a circuit.

There are so many applications of search.

1. Route Finding on a map
2. TSP - Travelling Salesman Problem
3. VLSI layout design
4. Automatic assembling

Search Algorithms

Search algorithms are required to explore a state space for solving a problem. There can be more than one solution for a given problem, therefore we need systematic or efficient algorithm to explore the state space.

In order to implement search algorithms we must have two concerns.

1. Set of nodes to be visited next. (open list)
2. Set of nodes already visited. (closed list)

Two lists can be maintained.

Evaluating a search algorithm, 4 strategies :-

1. Completeness

When there is a solution, if the algorithm can find it, we say that algorithm is complete.

2. Optimality

If the algorithm can find highest quality solution, we say that it is optimal.

3. Space Complexity

How much memory, the amount of memory required for executing the algorithm. In other words memory requirement of the algorithm.

4. Time Complexity

Time taken to execute the algorithm. In computational terms we normally interested in time and space complexity.

Strategies

Uninformed : Only a systematic way (blind search)
Informed : Efficient way (heuristic search)

Uninformed search

- These algorithms are not given any information (no of steps, path costs) about the problem other than its definition.
- Can only distinguish a goal state from a non goal state.
- No bias to go towards the desired goal.
- Can expand nodes more or less randomly.
    - In a map, expand all neighbouring nodes and add them to the fringe without prioritising.
    - Open list : nodes yet to be expanded (fringe)
    - Closed list : set of nodes explored

Uninformed search algorithms

1. Breadth First
2. Uniform Cost
3. Depth First
4. Depth Limited
5. Iterative Deepening
6. Bidirectional

Breadth First Search

-pic-

In breadth first search we exploring the state space layer by layer, until we get the goal node or return the failure.

Let’s apply open and closed list notation to demonstrate how the algorithm works.

In this search algorithm, at each node you’re to consider the children of that node. At one point of time you might find goal node.

Open	Closed
=[P]	=[ ]
=[P S T U]	=[P]
=[S T U X M]	=[P S]

=[T U X M A B]	=[P S T]
=[U X M A B L J]	=[P S T U]

1. We can sketch the steps of the algorithm as follows;
2. Consider the top most node in the open list.
3. Get the children of that node.
4. Add the children to the end of the open list.
5. Remove the top most node from open list and add to closed list.
6. Check whether the removed node is the goal node, if so, stop.
7. Otherwise go to step 1.

- In that notation, closed list shows the order in which the nodes are visited when we reach the goal (Traversal).
- The no of nodes stored in the memory when the algorithm is executed comes from the total no of nodes in open and closed lists at the last step.
- In this case there are 10 nodes in the memory.
- Note that when the goal node is at the level 1, nodes in the level 2 should also go to the memory.
- (One level below the goal node)
- Therefore this algorithm is very bad in memory management.
- Even though 10 nodes are in the memory, only 4 nodes are been processed, all their children adding to open list, removing from the open list and doing the goal test. (Time taken actions, tasks = 3)
- In the closed list we can see the nodes which have been processed by the CPU. (Time taken)

- For breadth first search, time complexity is not so bad.

Question

If the above state space is explored by a computer which requires 1kB for storing the node and can process 100 nodes per second, find the time taken and memory usage of the algorithm.

When the goal node is reached, 10 nodes in the memory.
Therefore, memory requirement is 10 x 1kB = 10 kB.
No of nodes processed = 4.
Therefore time taken to execute = (1/100)x4 = 0.04s.

Note that, behavior of the search algorithm is very much determinant of whether the children nodes are added to the open list;

1. To the end (Breadth first search)
2. To the front (Depth first search)
3. Somewhere in the middle (Heuristic search)

Features of Breadth First Search

- Since the algorithm execute layer by layer, if there is a goal it should find the goal – Complete.

- Since the algorithm detect goal nodes in the shallower level, before going to deeper level, this algorithm is Optimal.

- Space complexity of BFS is really bad, because it requires nodes below the goal level also in the memory.

- Suppose we have a state space with average no of children per node as b (the Branching Factor), if this is explored by BFS, at level 1 we have b nodes, at level 2 we have b^2 node, so on…

- If the goal is at level d, total no of nodes in the memory are,

  b + b^2 + b^3 + ….. + b^d + (b^(d+1) - b)

  Nodes from the d+1 level, without exploring the goal node.

-pic-

- Therefore the space complexity of BFS is O(b^(d+1))

- Time complexity
- Since the goal node is found in the d th level, processing is limited to that level, although nodes from level d+1 are in the memory, without any further processing.
- Therefore time complexity is O(b^d), order of b to the power d.

-pic-

Even for a shallower state space, memory complexity is a big matter, for BFS, but not the time complexity.
However if the state space is deeper, time complexity is also be a problem.

Uniform Cost Search

UCS is a version of BFS where nodes are associated with some cost function. In the execution at a particular level, the lowest cost node is visited first.

Open list	Closed list
=[p]	=[ ]
=[P U T S]	=[P]

Apart from that, features of UCS are the same as that of BFS.

Depth FIrst Search

In DFS, we execute the searching deep down, along the branch and come back, go deep down, and so on until you get the goal node or return a failure.

-pic-

Open list Closed list

Using DFS we found a goal with a traversal [P S X M T A B U] with 8 steps.

This algorithm also returns the deeper level goal node before a goal node at a shallower level. For example : goal node in level 4 found first before one at the level 2.

One interesting part of this algorithm is that if there is a loop in a leftmost branch, the algorithm will not be able to exit, even the goal node is available in the space.
-pic-
-pic-

Therefore in general DFS is not complete. However by chance, DFS can also be complete and optimal.
If the goal node is in the leftmost branch, it can be optimal, if there are no loops, it is complete too.

Features of DFS:

- Not Complete.

- Not Optimal.

- Space Complexity O(bd), because at a particular node only the immediate children nodes are required into the memory. We call that in BFS, when we deal with nodes in the right side, nodes in the left hand side, even 1 level below should be in the memory.

- Time Complexity $O(b^d)$, order b to the power d, because in the worst case scenario, goal node may be the terminal node of the rightmost branch. (At level d)

When comparing DFS and BFS, BFS represent is computationally rather weak due to excessive memory required.
Therefore expansion to search algorithm had been done by considering DFS, at the basics.

Depth Limited Search

In this case DFS is applied for a pre defined depth, shorter than the actual depth of the space. Sometimes by chance if the goal is within the limit we may find the solution.
Even we search for something in a book, we have a tendency to read upto a certain depth, rather than reading the whole book to find something.

Iterative Deepening Depth First Search

In this algorithm, DFS is first applied for depth 1, if the answer is not found, increase the depth for 2,3,4,... etc and apply DFS.
In this case, as we go on iteration, shallower level will be expanded more than 1 time. One might think this is a wasteful action.

Slide 24

When exploring using IDS for the goal at level d,
$N(IDS) = (d)b + (d-1)b^2 + (d-2)b^3 + \ldots + (1)b^d$

Using BFS
$N(BFS) = b + b^2 + b^3 + \ldots + b^d + (b^{(d+1)} - d)$

Due to involvement of the term $b^{(d+1)}$ in DFS
$N(IDS) < N(BFS)$

Example :
For $b = 10$ and $d = 5$
$N(IDS) = 123{,}450$
$N(BFS) = 1{,}111{,}100$

IDS is complete because it increases the depth by exploring level by level. If there is a goal node, it should be found.

IDS is also optimal, because if the goal node is found at a shallower level, we do not increase the depth to find, goal node in deeper level.

Since we apply DFS in IDS, space complexity and time complexity are also less.

This algorithm combines good features of BFS (Complete, Optimal) and good features of IDS (less time/ space complexity)

However IDS may take longer time due to repeated exploring of shallower level.

Bidirectional Search

Bidirectional Search execute two search processes from the start node to the goal, and from goal node to the start in the opposite direction. If there is a solution, two parties should be meeting somewhere in the middle.

In this algorithm, one party explore half of the depth of the space. Therefore time complexity will be very much reduced.

$(b^d)/2 \gg b^{(d/2)}$

This search may have problem in the opposite direction traversal from the goal, when a node is rooted from more than one parent nodes.

So far we discussed static state space, which are trees, but search can be more complicated if the state space is a graph and it may also be dynamically changing.

Comparison

-table-

Informed Search

Under informed search, we go beyond the systematic search and try to make the search more efficient by introducing some additional knowledge such as, common sense, experience, etc.

They have some idea of where to look for the solution. Problem specific knowledge.

Example

- When we are searching for a leak in a bathroom we do the search in more closer to the joints rather than in straight pipes.
- The kind of knowledge useful for improving search is called heuristic.
- Search with additional knowledge is the place where we apply artificial intelligence.
- In informed search when we are at a particular node, we evaluate all the children nodes by some function to decide which node to be visited next.
- Evaluation function f(n) is generally heuristic based or heuristic function h(n).

What is a heuristic?

It is a serving to discover or find out. In informed search, we use heuristics to identify the most promising search path, or it is used as a guide that leads to better overall performance in getting to the goal.

A heuristic function

A heuristic function at a node n is an estimate of the optimum cost from the current node to a goal, denoted by h(n).
h(n) : estimated cost of the cheapest path from node n to a goal node.

Example:

1. We want to find the shortest / cheapest path from Colombo to Kandy in a map;

   -pic-
   x: estimate of the distance of the path
   y: actual path

The straight line distance is underestimate of the actual path from S to G. So heuristic for G is the straight line distance.

2. 8 puzzle

-pic-

One heuristic is h1: the no of tiles out of place, 2,8,1,6,7
h1(n) = 5 , So at least 5 moves are needed to goal.

Another heuristic is h2: the manhattan distance that means the sum of the distance from each tile not in its correct position, from its goal.

2	move 1 position
8	move 2 position
6	move 1 position

7	move 1 position
1	move 1 position

h2(n) = 1+2+1+1+1 = 6 is the estimated actual no of moves, we need at least 6 moves to achieve goal.

Admissible heuristic

Let h(n) be the heuristic function on node n and h*(n) be the actual cost (actual distance) to reach a goal from node n.
So, the heuristic function is admissible if it never overestimates the actual cost to goal.
If h(n) <= h*(n) for all nodes, the heuristic function is admissible.
Also an admissible heuristic function is always optimistic.

Consistent heuristic (monotone)

Let n be a node and n' its successor and h(n) be the heuristic function on node n, that estimates the distance (cost) from n to goal. So, h(n) the heuristic function is consistent, if it is always at most equal to the estimated distance (cost) from n' to the goal, plus the step cost of reaching n' from n
If h(n) <= c(n,n') + h(n') for all nodes, the heuristic function h(n) is consistent, and also a consistent heuristic is always admissible.

Instead of exploring the search tree blindly (one node at a time), the nodes that we could go to are ordered according to some evaluated function f(n) , that determines which node is probably the "best" to go to next. This node is then expanded and the process is repeated. This type of searching is called "Best First Search".

Best First Search

Generalization of Breadth First Search.
An evaluation function f(n) is applied to each node which measures distance from goal. Nodes are stored in a priority queue sorted according to f(n) values.
Choose best node to expand first, which has lowest f(n) value. Two types of best first searches;

1. Greedy best first search
2. A* search

Greedy Best First Search

- In Greedy Best First Search we are so greedy reaching to the goal rather than any other concern.
- Expand the node that appears to be the closest to the goal.
- Therefore we define f(n) = h(n) where h(n) is a goal based heuristic.
- This algorithm ignores the cost of getting to n, with the lowest h value, greedily trying to achieve a low cost solution.

-map-

Consider the given map and find the route from Arad to Bucharest.

Here we use the heuristic function as the straight line distance from Bucharest to any city.

-graph-

Now we terminate the search because we found the goal node with the lowest f(n) value.

From the state space, traversal from Arad to Bucharest is;

Arad → Sibiu → Fagaras → Bucharest

The distance covered in this route is;
140 + 99 + 211 = 450

From the map, we can see another route from Arad to Bucharest covering the distance,
140 + 80 + 97 + 101 = 418

This shows that the Greedy Search would not find the optimal path.

We can show that Greedy Search is always not complete (even if the answer is there, it cannot be found)

Suppose you want to go from Lasi to Fagaras, (Fagaras is the goal) on the map.
To apply Greedy search now we need straight line distance from Fagaras to other cities. We assume;

-pic-

According to the map we can see that, one can go from Lasi to Fagaras. This means there is a solution. Now we are going to prove greedy search cannot find this solution.

-pic-

Here we will get a loop through Lasi and Neamt, without been able to pick Vaslui. Therefore the algorithm cannot find the route Fagaras.

Therefore Greedy search is not complete or optimal.

In order to go out of the loop we notice that Greedy search may be improved by introducing extra term to f(n) considering already incursed cost at a node.

A* Search

- A form of a best first search which uses most sophisticated heuristic function based on knowledge.
- As greedy search ignores the cost getting to n, how when exploring nodes that costs a lot to get, but seem to closer to goal?

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

S------------------n-----------------G

f(n) : estimated cost of cheapest solution that posses through n.
g(n) : already known exact : cost from initial state to n.
h(n) : heuristic based : heuristic function estimated lowest cost from node to goal.

- Finding a good heuristic is the core of the function.
- Better heuristic : fewer nodes will be expanded in searches.
- If h(n) is good, f(n) may be a good estimate of final cost to get from start to goal.
- A* is optimal and complete if h(n) is admissible : h(n) never overestimate the cost to reach the goal.
- h(n) is optimistic, since the cost of solving the problem is less than it actually is.
- Finds the solution with shortest no of expanding.

If we had a bad heuristic, A* is still exponential. Condition for A*, to find a solution in less than exponential time:

$$| h(n) - h^*(n) | <= O\log(h^*(n))$$

Example

Consider the following search space where the start state is S and the goal state is G.

-continue-

Example

Let’s apply the A* algorithm to find the path from Arad to Bucharest.

-graph-

So far the lowest f(n) value is with Fagaras.

-pic-

Now we have found the goal Bucharest but it is not with the lowest f(n) value, that is 450. Therefore algorithm doesn’t terminate. Now we continue to expand on lowest f(n) node that is Petesti.

-pic-

Now we found goal node Bucharest again with the lowest f(n) value 418, on the state space.

Let’s write the traversal generated by A* algorithm.

Arad → Sibiu → Rimnicu → Fagaras → Petesti → Bucharest

However the traversal may not be same as the route on the map.

Arad → Sibiu → Rimnicu → Petesti → Bucharest → is the route on the map.

This algorithm has discovered the optimal path from Arad to Bucharest.

Question

Considering the following problem for travelling from Lasi to Fagaras explain that A* is complete.

-pic-

We have already shown that there is a loop when we apply Greedy Search. If we can show, A* can break this loop and jump into Vaslui explanation is adequate. (enough)

-graph-

It shows that A* is complete and optimal. This is the best search algorithm ever.
Note that A* can be seen as combination of Greedy search and Uniform cost search, where we consider the cost incurred in the search.
Assume that we apply A* with iterative deepening search.

A* can be applied together with iterative deepening by setting values for f(n) as the depth limit.
Here evaluation function means depth of evaluation.

Exam questions

Question 1 (a) Distinguish between informed and non-informed search.

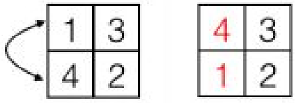
(2 marks)

(b) In informed search, what does it mean to say a heuristic is *admissible* and *consistent*.

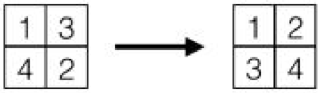
(4 marks)

(c) Describe the A* and greedy best-first informed search algorithms. **(6 marks)**

(d) Consider a puzzle consisting of a 2×2 board with 4 tiles numbered $1, 2, 3, 4$. In each move, a pair of tiles may be swapped in either the horizontal or vertical direction. For example, a vertical swap is as follows:



Using such pairwise swaps, the goal is to change the tiles from the given start state to the goal state as shown below:



Use Algorithm A* and greedy best-first algorithm with the heuristic function

$h(x) =$ the number of tiles not in their final position

to find a sequence of moves that takes you from the start to the goal. **(8 marks)**

Question 1 (a) Distinguish between depth-first search and breadth-first search.

(4 marks)

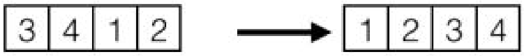
(b) How does iterative deepening improve over depth-first search? Give an argument to explain why its time complexity is the same as standard depth-first search, asymptotically.

(4 marks)

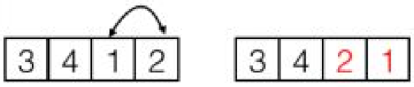
(c) Describe the A* algorithm, explaining clearly the meaning of *heuristic* function. What does it mean to say a heuristic is *admissible* and *consistent*?

(4 marks)

(d) Consider a search problem to sort a horizontal row of tiles, where each move in the search swaps a pair of adjacent tiles. The following shows the start and end states:



and an example move is as follows:



Find the number of moves required to reach the goal state, using the A* algorithm with a heuristic function of the form

$h(x) = d(1) + d(2) + d(3) + d(4)$

where $d()$ is the number of positions each tile is from its goal position e.g. in the start state $d(1) = 2$, since it is in position 3 and its goal position is 1.

(6 marks)

Draw the search tree, clearly showing the value of the total cost $f(x)$ at each node x.

(2 marks)

Question 1 (a) i. Explain the meaning of a *heuristic* function in informed search algorithms.

(2 marks)

ii. What does it mean to say a heuristic is *admissible* and *consistent*?

(2 marks)

iii. Describe how a heuristic function is used in the Greedy Best First informed search algorithm.

(3 marks)

iv. Describe the A* search algorithm.

(3 marks)

(b) Consider a search problem on the grid shown below. A robot starts at the tile $(1, 1)$ and must navigate to the destination tile $(4, 1)$. At each stage, the robot can move in one of at most four directions – either up (`U`), down (`D`), right (`R`) or left (`L`), but cannot move onto a black tile. The cost of a path is the number of tiles traversed over that path. For a tile at location $x = (r, c)$, the robot uses the heuristic function

$$h(x) = (4 - r) + (c - 1)$$

to estimate the distance to the goal e.g. from tile $x = (1, 1)$, $h(x) = (4 - 1) + (1 - 1) = 3$.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
■	(2,2)	(2,3)	■	(2,5)	(2,6)
(3,1)	■	■	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	■

i. Write down two possible sequences of moves that will successfully navigate the robot from the source to the destination (e.g. a starting sequence of three moves could be (`R,R,R...`). What is the cost of each path? **(3 marks)**

ii. Is the path that the Greedy Best First algorithm finds an optimal path? Explain. **(2 marks)**

iii. Draw the first few levels of the search tree for the A* algorithm, clearly showing the value of the cost $f(x)$ at each node in the tree and hence the path chosen by the A* algorithm up to the point at which tile $(2, 5)$ is reached. **(5 marks)**

Question 1 (a) i. Distinguish between depth-first and breadth-first search. Explain why breadth-first search is less memory efficient than depth-first search.

(3 marks)

ii. Describe the A* search algorithm, explaining clearly the meaning of the functions $f(n)$, $g(n)$ and $h(n)$. Explain two properties that a heuristic function should satisfy in order for the A* algorithm to be optimal.

(5 marks)

iii. How is $h(n)$ used in the Greedy Best First informed search algorithm?

(3 marks)

(b) Consider in Figure 1, the map of some cities in Sri Lanka and road routes between them. The roads are labelled with the distance between them in kilometres. Also, the direct point-to-point distance between each city and Kandy is shown in a table. Using the point-to-point distance as an estimate of the distance between a city and Kandy along the road network, answer the following:

i. If the child nodes are added to the queue in alphabetical order (e.g. the children of Colombo are, in order, Badulla, Galle, Gingathhena, Kurunegala, Negombo), what route will the depth-first algorithm find between Colombo and Kandy?

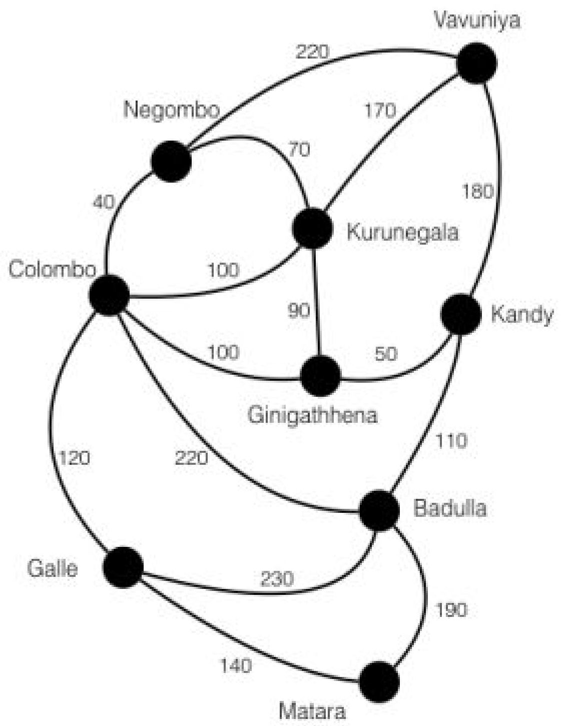
(3 marks)

ii. Show that the Greedy Best First algorithm finds a route between Colombo and Kandy of length 240km.

(3 marks)

iii. Consider the search tree of a run of the A* algorithm shown in Figure 2. By computing the $f(n)$ value for the nodes `a` to `t`, write out the order in which the nodes were expanded in searching for a route between Colombo and Kandy.

(3 marks)



Direct Distance to Kandy

Colombo	90
Negombo	85
Kurunegala	40
Vavuniya	160
Kandy	0
Ginigathhena	50
Badulla	60
Matara	150
Galle	140

Figure 1:

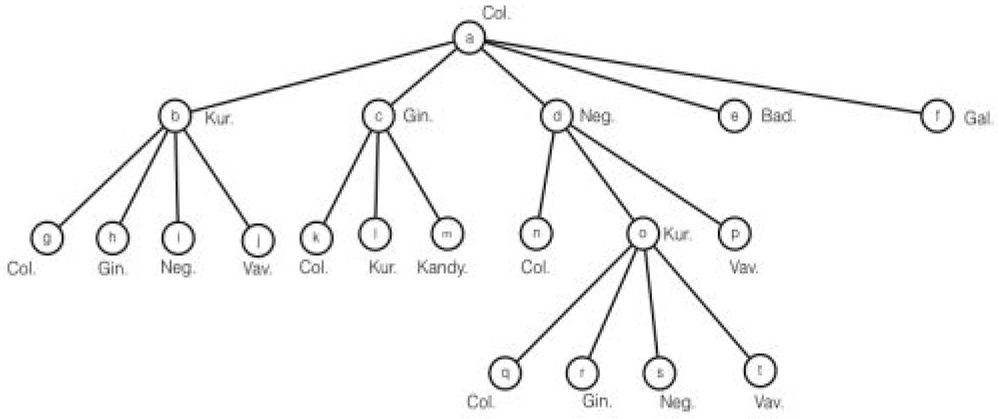


Figure 2:

------------------------------------ end of the tutorial -------------------------------------