Bi-directional A* Algorithm and its applications in shortest path finding in road networks

Project Code: BPV02
Supervisor: Viswanath P
Group Members:

P. Sonia (IS201501038)
D. Swathi Reddy (IS201501015)

Objectives

- **Finding the shortest path in a dynamically varying Graph.**

- We propose to explore the Bi-directional A* algorithm and its variants [1].
- Task-1: To implement BFS, Uniform cost search, A* search and its bi-directional variants.
- Task-2: To study the effect of dynamically adding/deleting an edge to/from the graph.
- Task-3: To study the effect of both nodes and edges being added or deleted.

[1] Holte, Robert C., et al. "MM: A bidirectional search algorithm that is guaranteed to meet in the middle." *Artificial Intelligence* 252 (2017): 232-266.

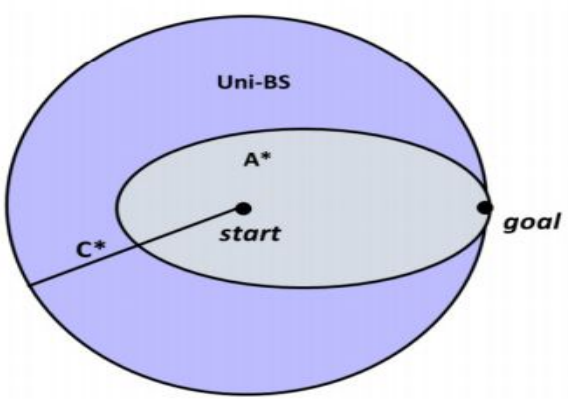
Scope of the proposed work

- AI and Graph Algorithm techniques are used.
- To speedup the search process, we propose to use

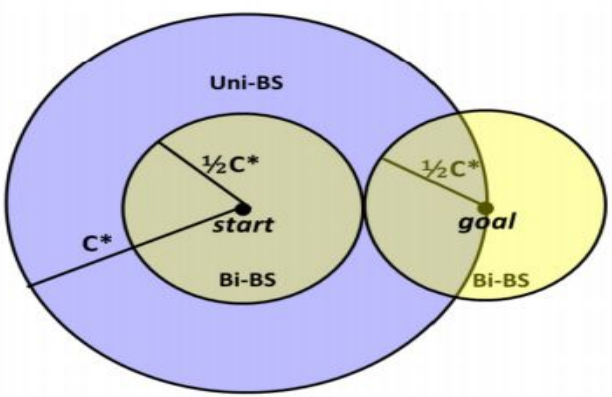
  Case Based Reasoning: New problems are solved by reusing and if necessary adapting the solutions to similar problems that were solved in the past.

  Graph Indexing: Storing reusable information related to the graph so that shortest path finding can be done quickly.

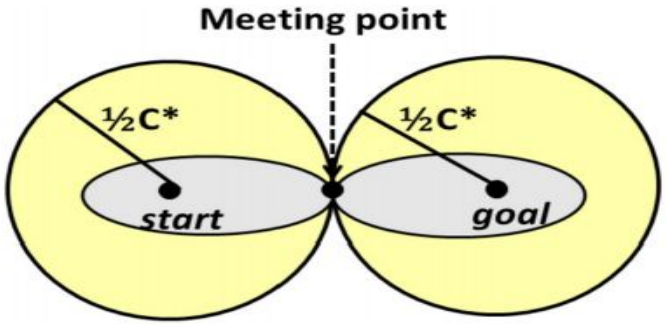
Comparison of expansion of search space between different algorithms. [1]



Uni BFS & A*



Uni BFS & Bi-BFS



Comparing Bi-BS & MM (the Bi-A*) algorithm

Here, C* is the cost of the optimal path solution from source to goal.

Workdone: In Task-1

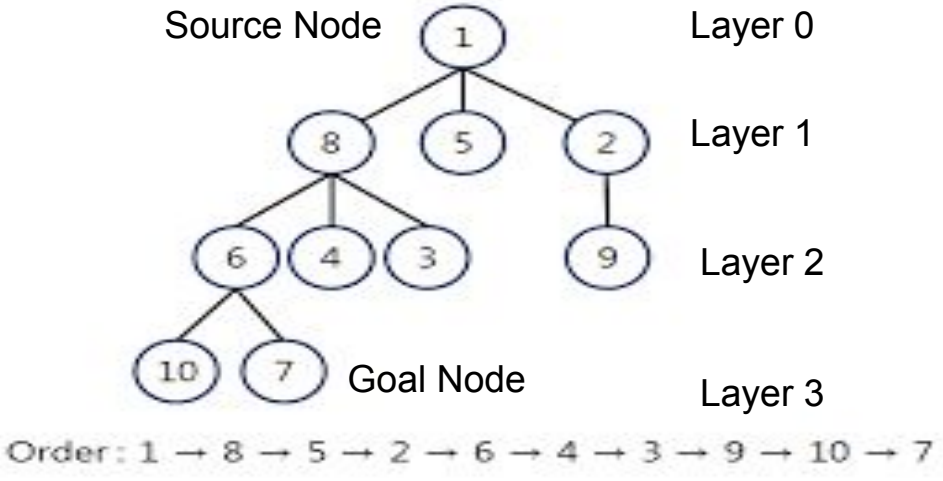
So far, we did,

1. Uni-BFS (Breadth First Search)
2. UCS (Uniform Cost search)
3. Bi-BFS (the Bi-directional BFS algorithm)
4. A* algorithm.
5. MM (the Bi-directional A* algorithm)

Breadth First Search (BFS)

As the name BFS suggests, you are required to traverse the graph breadth wise as follows:

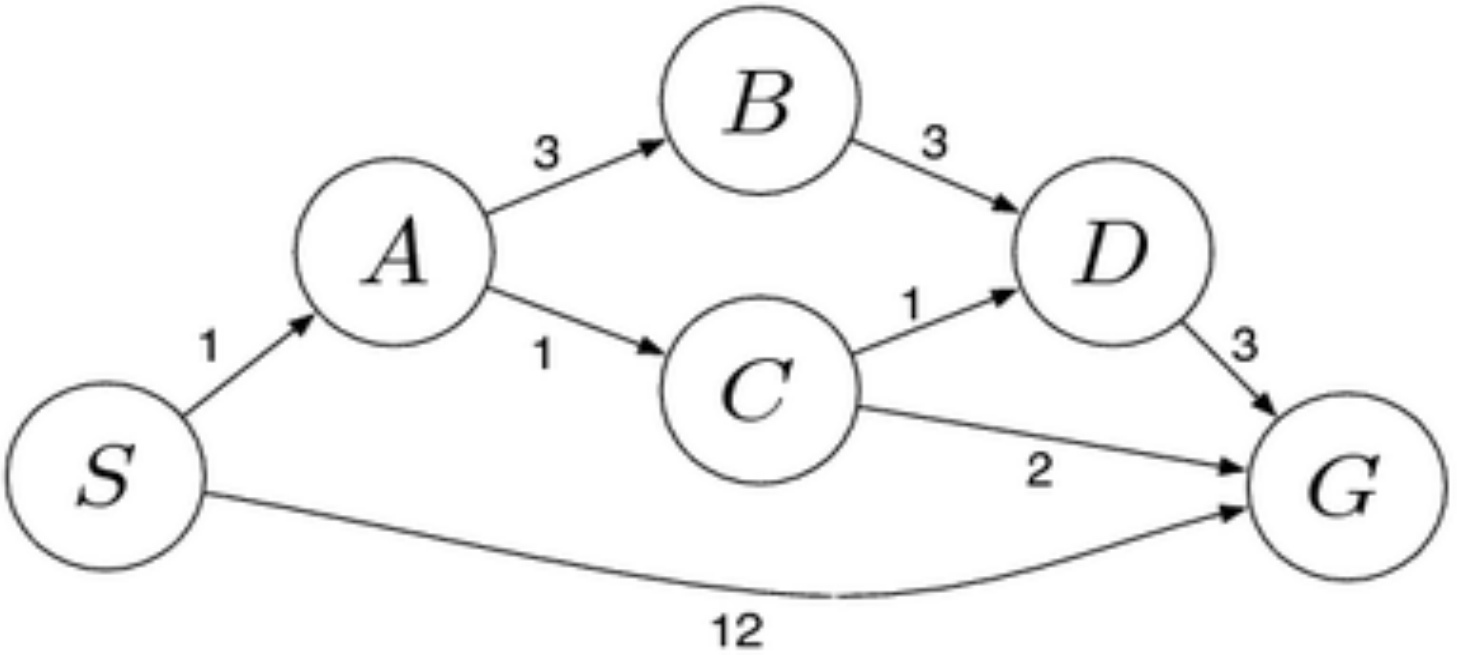
- **Root case** : The traversal queue is initially empty so the the root node must be added before the general case.
- **General case**: Process any items in the queue, while also expanding their children, stop if the queue was empty. The general case will halt after processing the bottom level as leaf nodes have no children.



Uniform Cost Search (UCS)

- UCS is used for finding the shortest path in a weighted graph, which does not involve the use of heuristics.
- At any given point in the execution, the algorithm never expands a node which has a cost greater than the cost of the shortest path in the graph.
- Uniform cost search determines a path to the goal state that has the lowest weight, whereas the breadth-first search determines a path to the goal state that has the least number of edges.
- Uniform Cost Search can also be used as Breadth First Search if all the edges are given a cost of 1.
- The main difference between Dijkstra and UCS is that, in Dijkstra we find the shortest path from source node to every other node in the graph. Whereas, in UCS we find the shortest path from source node to goal node only.

UCS Example



UCS Contd...

Initialization: { [S , 0] }

Iteration1: { [S->A , 1] , [S->G , 12] }

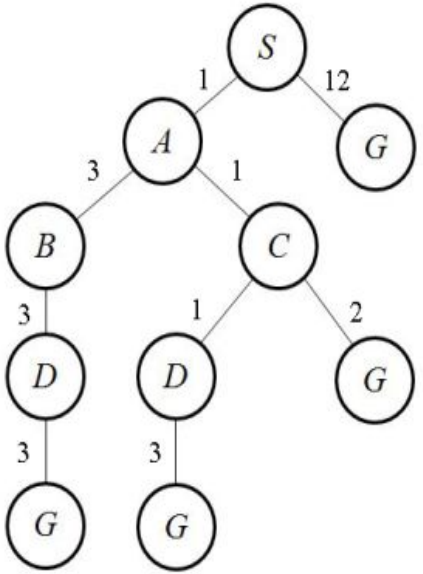
Iteration2: { [S->A->C , 2] , [S->A->B , 4] , [S->G , 12] }

Iteration3: { [S->A->C->D , 3] , [S->A->B , 4] , [S->A->C->G , 4] , [S->G , 12] }

Iteration4: { [S->A->B , 4] , [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->G , 12] }

Iteration5: { [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->A->B->D , 7] , [S->G , 12] }

Iteration6 gives the final output as S->A->C->G.



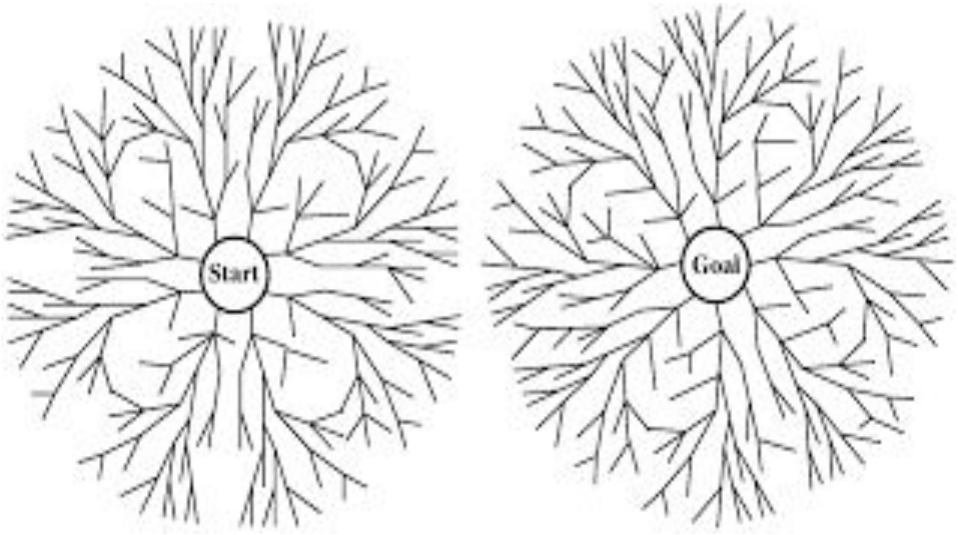
Bidirectional BFS

Bidirectional BFS is a graph search algorithm which finds the smallest path from source to goal node. It runs in two directions –

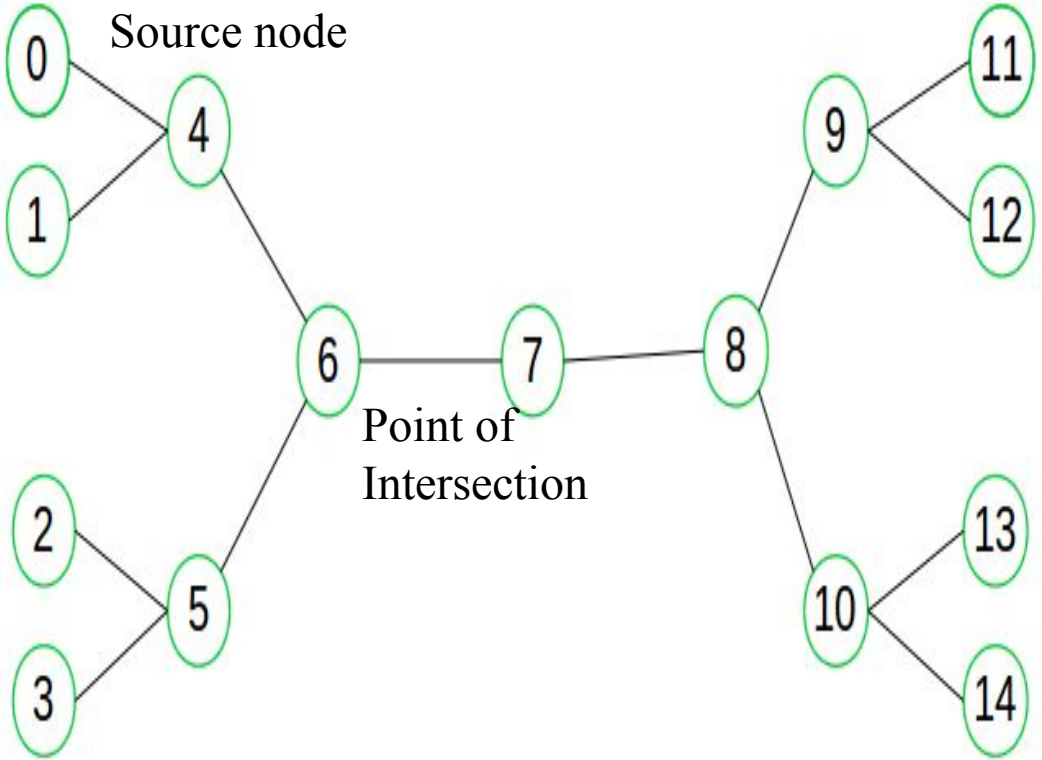
1. Forward search form source/initial vertex towards goal vertex.
2. Backward search form goal/target vertex towards source vertex.

Bi-BFS will have two sub graphs – one starting from initial vertex and other starting from goal vertex. **The search terminates when two graphs intersect.**

Bidirectional BFS Contd...



Bidirectional BFS Contd...



Path exists between source node and goal node.

Path: 0-4-6-7-8-10-14

Intersection at:7

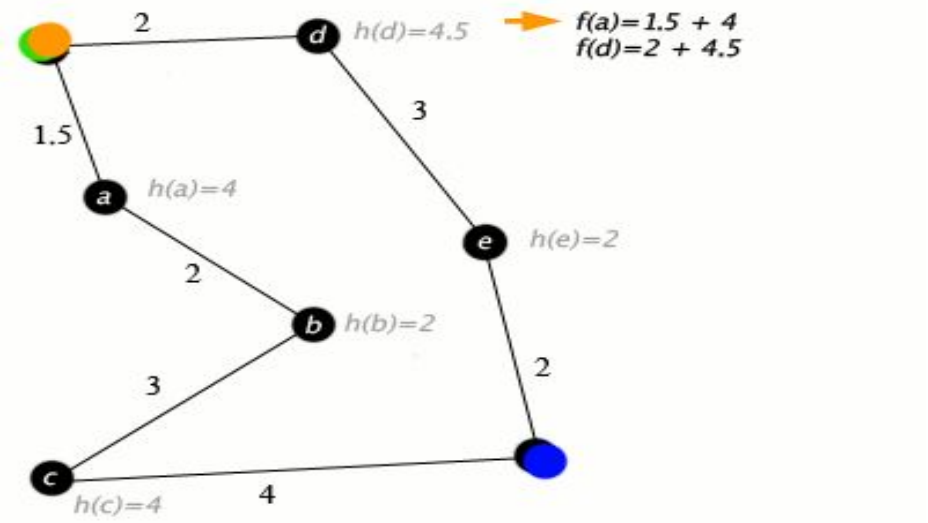
A* algorithm

- At each step of the algorithm, the node with the lowest $f(n)$ value is removed from the open list, the *f* and *g* values of its neighbours are updated accordingly.
- Then, it is added to the closed list, and the neighbors of n are also added to the open list.
- The algorithm continues until the open list is empty.The *f* value of the goal is the length of the shortest path, since *h* at the goal is zero in an admissible heuristic, meaning that it never overestimates the actual cost to get to the nearest goal node.

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

- where *n* is the node on the path, $g(n)$ is the cost of the path from the start node to *n*, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from *n* to the goal.

Working of A* Algorithm



MM (Meet in the middle)

MM, a new Bi-Directional heuristic search algorithm is guaranteed to meet in the middle i.e., its forward search never expands a node n with $gF(n) > (1/2)C*$ and its backward search never expands a node n with $gB(n) > (1/2)C*$ [1].

Here, **C*** is the cost of an optimal solution, i.e. the distance from start to goal.

gF(n) measures the cost of the path from start to n.

gB(n) measures the cost of the path from goal to n.

F= Forward search. B= Backward search.

[1] Holte, Robert C., et al. "MM: A bidirectional search algorithm that is guaranteed to meet in the middle." *Artificial Intelligence* 252 (2017): 232-266.

MM Algorithm

- Basic MM runs an A*-like search in both directions, except that MM orders nodes on the Open list in a novel way.
- The priority of node n on OpenF is: $prF(n) = \max(fF(n), 2gF(n))$
- $prminF$: Minimum priority on OpenF.
- $prminB$: Minimum priority on OpenB.
- $C = min(prminF, prminB)$

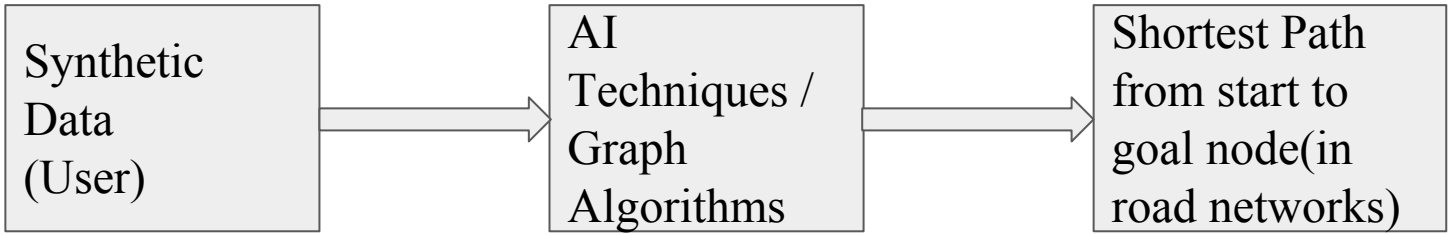
MM Algorithm Contd...

- On each iteration MM expands a node with priority C. When a state s is generated in one direction MM checks whether s is in the Open list of the opposite direction. If it is, a solution (path from start to goal) has been found.

- MM maintains the cost of the cheapest solution found so far in the variable U. U is initially infinite and is updated whenever a better solution is found. MM stops when $U \leq \max\left(C, f \min F, f \min B, g \min F + g \min B + \epsilon\right)$

  Where **Є** is the cost of the cheapest edge in the state space.

Workflow Diagram



REFERENCES

1. Holte, Robert C., et al. "MM: A bidirectional search algorithm that is guaranteed to meet in the middle." *Artificial Intelligence*252 (2017): 232-266.
2. Chen, Jingwei, et al. "Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions." *arXiv preprint arXiv:1703.03868* (2017).
3. Ding, Bolin, Jeffrey Xu Yu, and Lu Qin. "Finding time-dependent shortest paths over large graphs." *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. ACM, 2008.
4. Dennis de Champeaux, Lenie Sint, An improved bidirectional heuristic search algorithm, J. ACM 24 (2) (1977) 177–191.