# 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** (the set V of vertices of the graph is fixed,but the set E of edges can change(added or removed)).
- **Case Based :** New problems are solved by reusing and if necessary adapting the solutions to similar problems that were solved in the past.
- **Graph Indexing**

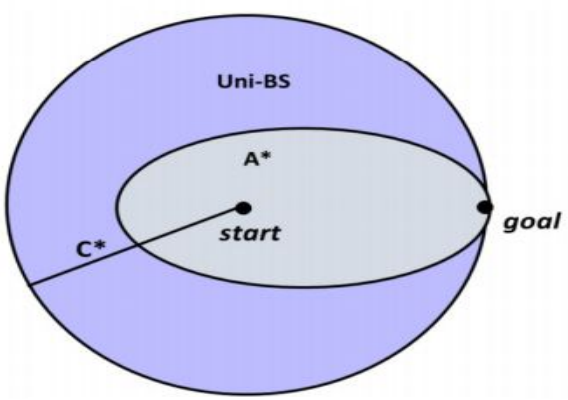
## Scope:

- AI Techniques
- Graph Algorithms

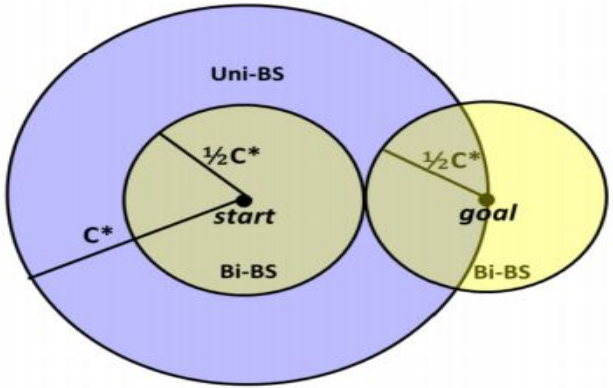
# Past Research Results

- Bi-HS (Bi-directional Heuristic Search) is fundamentally different than Uni-HS.
- Uni-HS (Uni-directional Heuristic Search), in general, is better than Uni-BS (Uni-directional Breadth First Search).
- Bi-BS is better than Uni-BS. The corresponding statement does not hold for Bi-HS.
- Existing Bi-HS systems expanding more nodes than Bi-BS instead of fewer [1].
- Existing Bi-HS systems expand nodes that are further than (½) C* from both start and goal. (Here, C* is the cost of the optimal solution)
- No known Bi-HS algorithm is guaranteed to meet in the middle under all circumstances except the MM technique given in [1].
- It is a new Bi-HS algorithm that for a given admissible heuristic, is guaranteed to meet in the middle and to return the optimal solution.

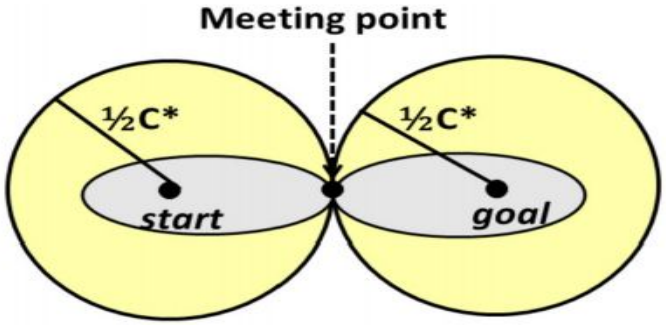
The main purpose of the heuristic function is to focus the search towards the goal. Let C* is the cost of the optimal solution, i.e. the distance from start node to goal node. Following Figures explains this.




Comparing Uni-BS & A*




Comparing Uni-BS & Bi-BS




Comparing Bi-BS & MM algorithm

# Shortest path

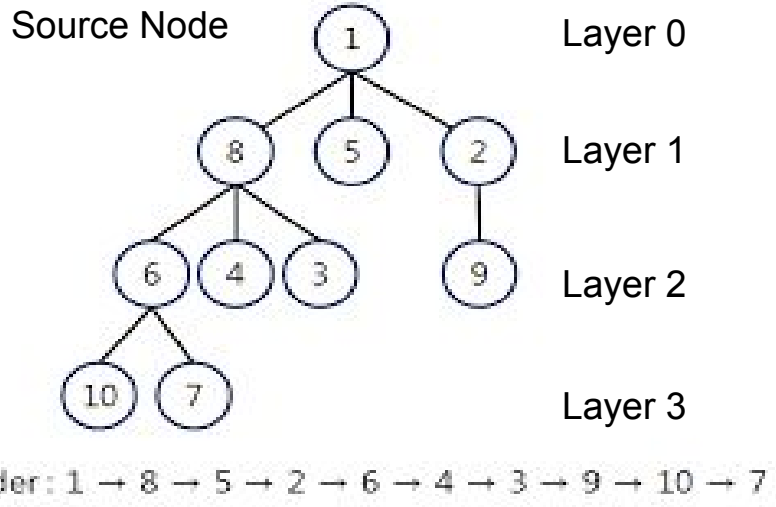
In a least-cost path problem over a state space the task is to find a least-cost path from an initial state (start) to a goal state (goal).

Breadth-first search and its weighted version uniform cost search (Dijkstra's algorithm) are best-first search algorithms designed to solve least-cost path problems.

# Breadth first search

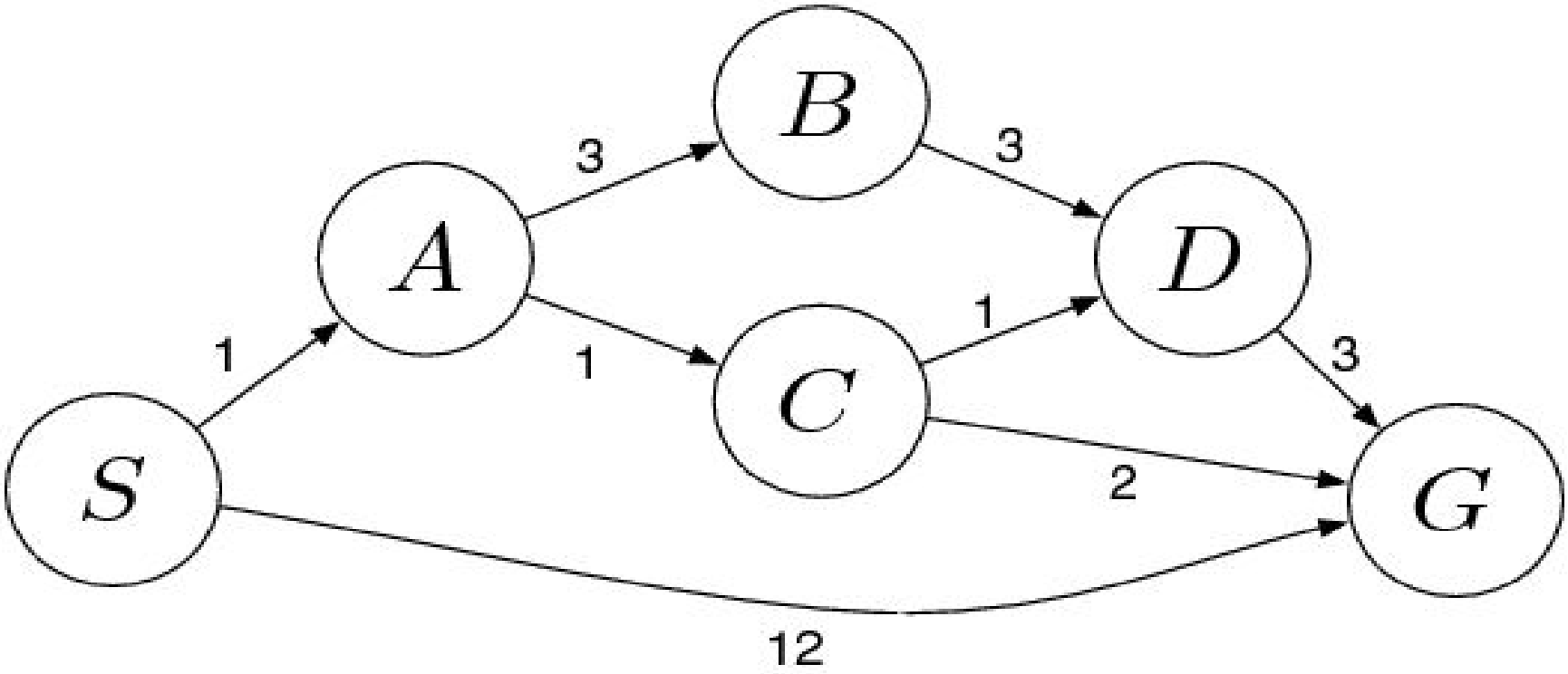
BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise. Thus, exploring the neighbour nodes. You must then move towards the next-level neighbour nodes. As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer



# Uniform Cost Search

- 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.


S
A
B
C
D
G
1
3
1
3
1
3
2
12

**Initialization: { [ S , 0 ] }**

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

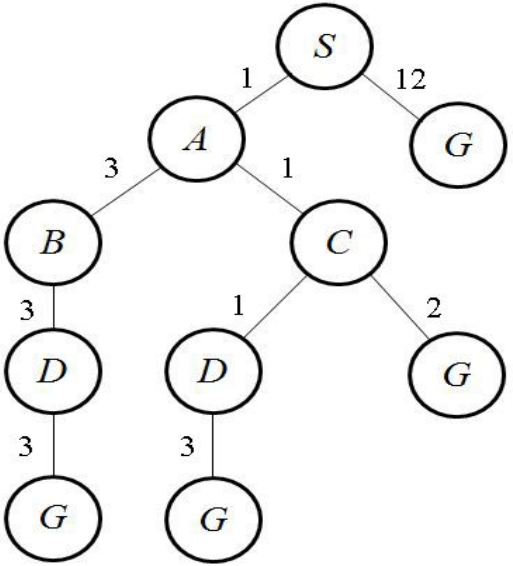
**Iteration2: { [ S->A->C , 2 ] , [ S->A->B , 4 ] , [ S->G , 12] }**

**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.**



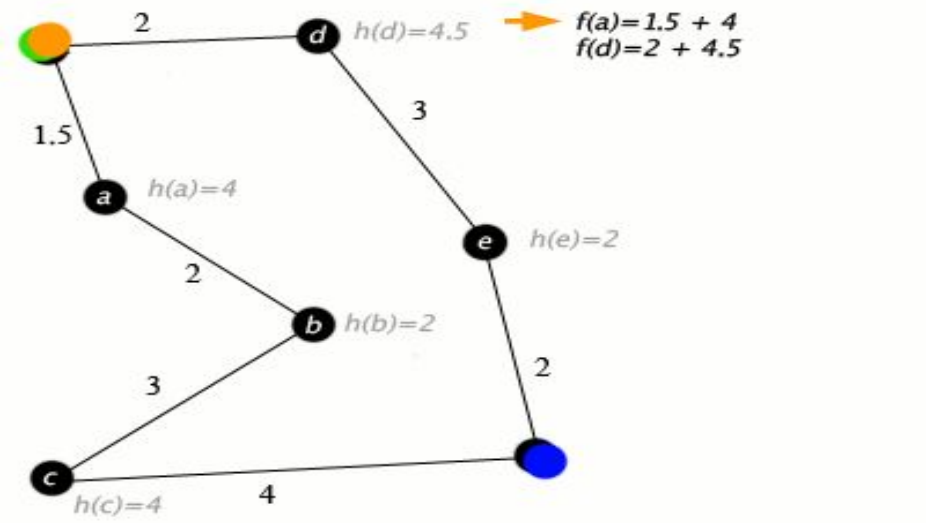
# A* algorithm

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 neighbors are updated accordingly, and add it to the closed list, and the neighbors of n are 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.

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

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. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

# Working of A* Algorithm



# Bidirectional Search

Bidirectional search algorithms interleave two separate searches, a normal search forward from the start state, and a search backward from the goal.

It is well known that adding a heuristic to unidirectional search dramatically reduces the search effort.

# Why meet in the middle?

- Meeting in the middle provides an upper bound on how many nodes a Bi-HS system will expand in the worst case and an upper bound on how much memory it will need.
- In state spaces where the number of states at distance d from start or goal grows exponentially with d, a system that ventures beyond d = (½)C*, whether it be bidirectional or unidirectional, is at risk of expanding exponentially more nodes than a system that meets in the middle.
- Meeting in the middle provides a characterization of nodes that are guaranteed not to be expanded, analogous to the fact that A* is guaranteed not to expand nodes with f(n) > C*.
- Meeting in the middle guarantees that a state expanded in one direction will not be expanded in the other direction unless it is exactly distance (½)C* from both start and goal

# MM (Meet in the middle)

A bidirectional search algorithm meets in the middle if its forward search never expands a node n with $gF(n) > (1/2)C^*$ and its backward search never expands a node n with $gB(n) > (1/2)C^*$. 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.

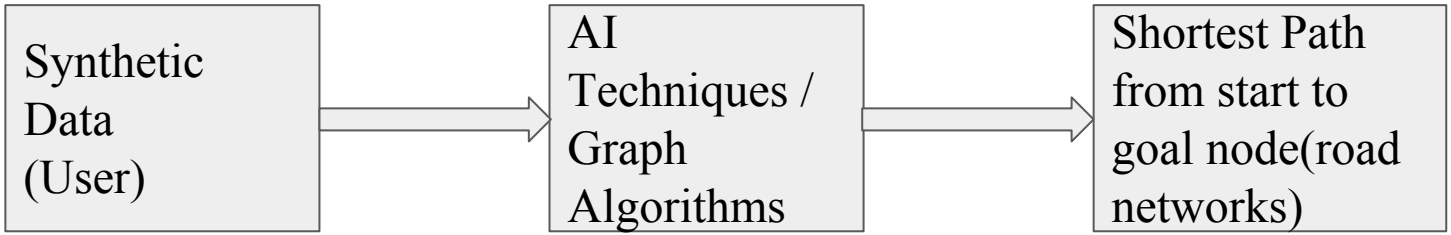
F=Forward search.

B=Backward search.

# 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, prF (n), is: **prF (n) = max(f F (n), 2gF (n))**. prB (n) is defined analogously. We use prminF and prminB for the minimum priority on OpenF and OpenB, respectively, and **C = min(prminF,prminB ).**
- 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≤max(C,fminF,fminB,gminF+gminB +Є)** 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. De Champeaux, D.,Sint, L. An improved bidirectional heuristic search algorithm,Volume 24.