1.

a) A*

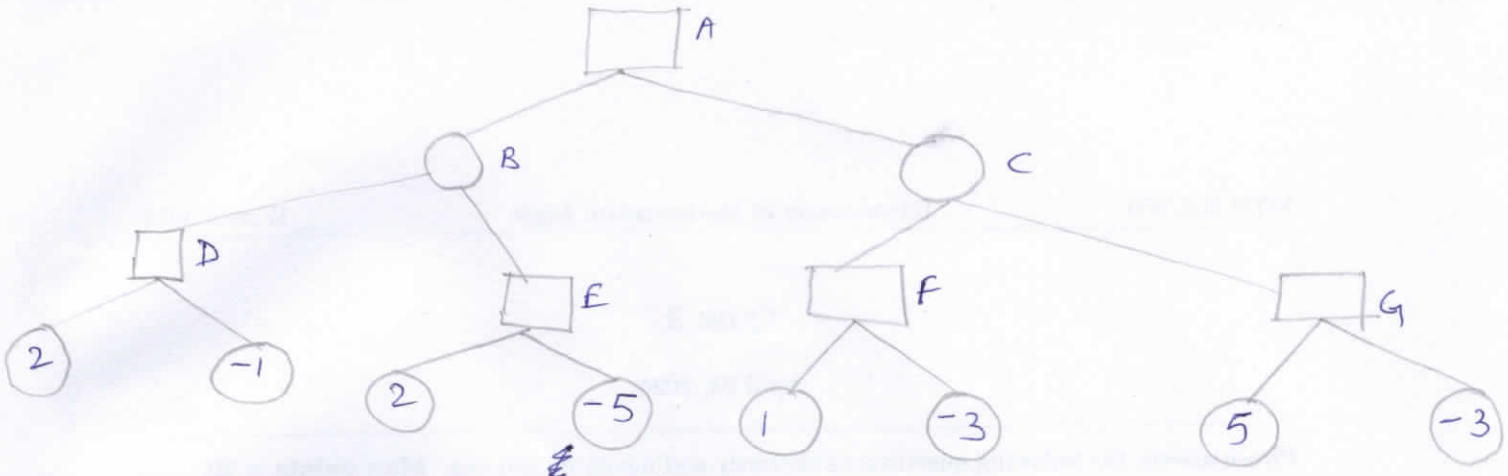
Node expanded	Frontier nodes	Explored nodes	g(n)	h(n)	f(n) = g(n) + h(n)
S	A, B, C	S	0	8	8
B	A, D, F, G3, C	S, B	1	1	2
A	G1, D, F, G3, C	S, B, A	3	2	5
F	G1, D, G3, C	S, B, A, F	3	3	6
D	G1, E, G2, G3, C	S, B, A, F, D	4	4	8
E	G1, G2, G3, C	S, B, A, F, D, E	6	1	7
G1	G2, G3, C	S, B, A, F, D, E, G1	8	0	8

~~b)~~

b) Uniform cost seach

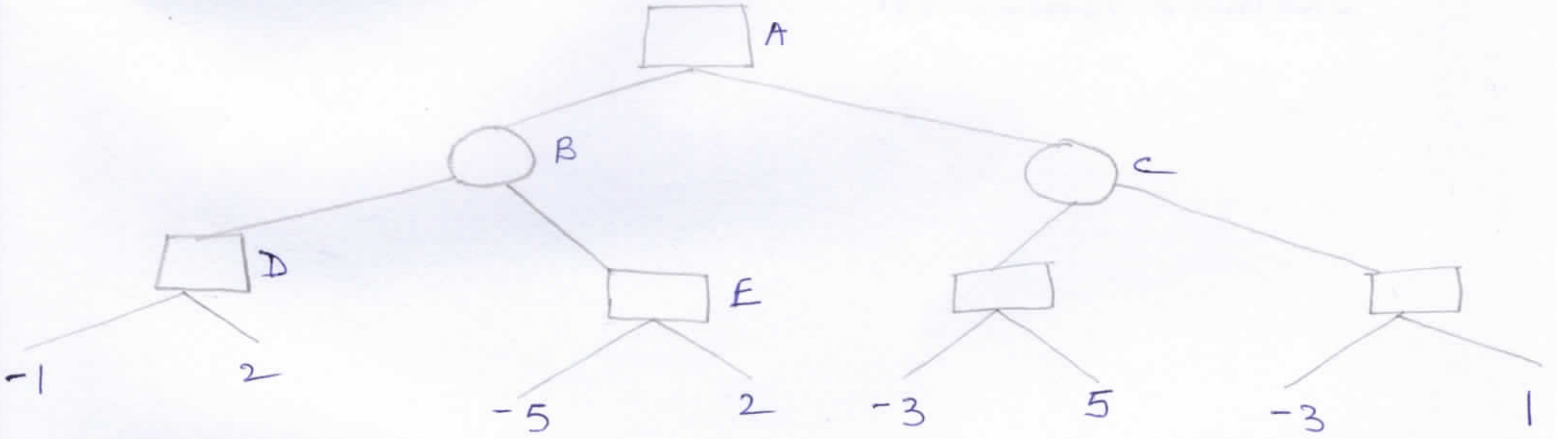
Node explored	Frontier nodes	Explored nodes	f(n) = g(n)
S	A, B, C	S.	0
B	A, D, F, G3, C	S, B	1
A	G1, D, F, G3, C	S, B, A	3
F	G1, D, G3, C	S, B, A, F	3
~~C~~	~~G1, D, G3~~		~~5~~
D	G1, E, G2, G3, C	S, B, A, F, D	~~5~~ 4
~~E~~	~~G1, G2, G3~~		~~6~~
C	G1, E, G2, G3	S, B, A, F, D, C	5
E	G1, G2, G3	S, B, A, F, D, C, E	6
G1	G2, G3	S, B, A F D, C, E, G1	8

(b) Best case



The nodes are arranged such that ~~any~~ corresponding to its minimiser (or maximiser) status node at any level finds the min (or max) value node at the next level from ~~the~~ its leftmost branch (or subtree). This effectively prunes most of the tree giving best case time complexity of $O(b^{d/2})$ as evaluation is done from left to right.

(c) Worst case



The nodes are arranged such that any node ~~at any level~~ non-optimal subtrees are evaluated first, leading to evaluation of all nodes. This does not prune any subtree (or branch) giving worst case complexity of $O(b^d)$ as evaluation is done from left to right.

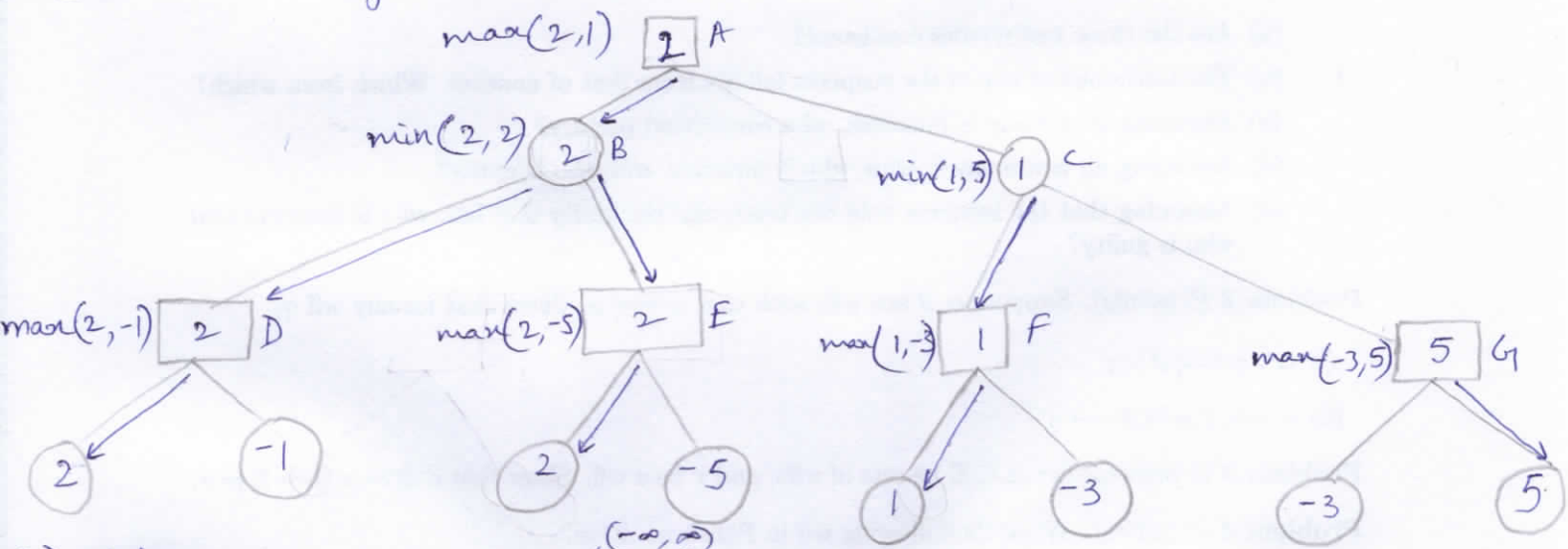
(c) If the node arrangement is optimal, the number of ~~leaf~~ nodes evaluated is $O(b \times 1 \times b \ldots . \times (b \text{ or } 1))$ depending d is odd or even. $= O(b^{d/2})$ We need to evaluate all the maximiser's moves but only need to consider the best move of minimiser which is the leftmost branch of the tree.

(c) Iterative deepening A^*

Threshold 1 : $f(n) = 8$

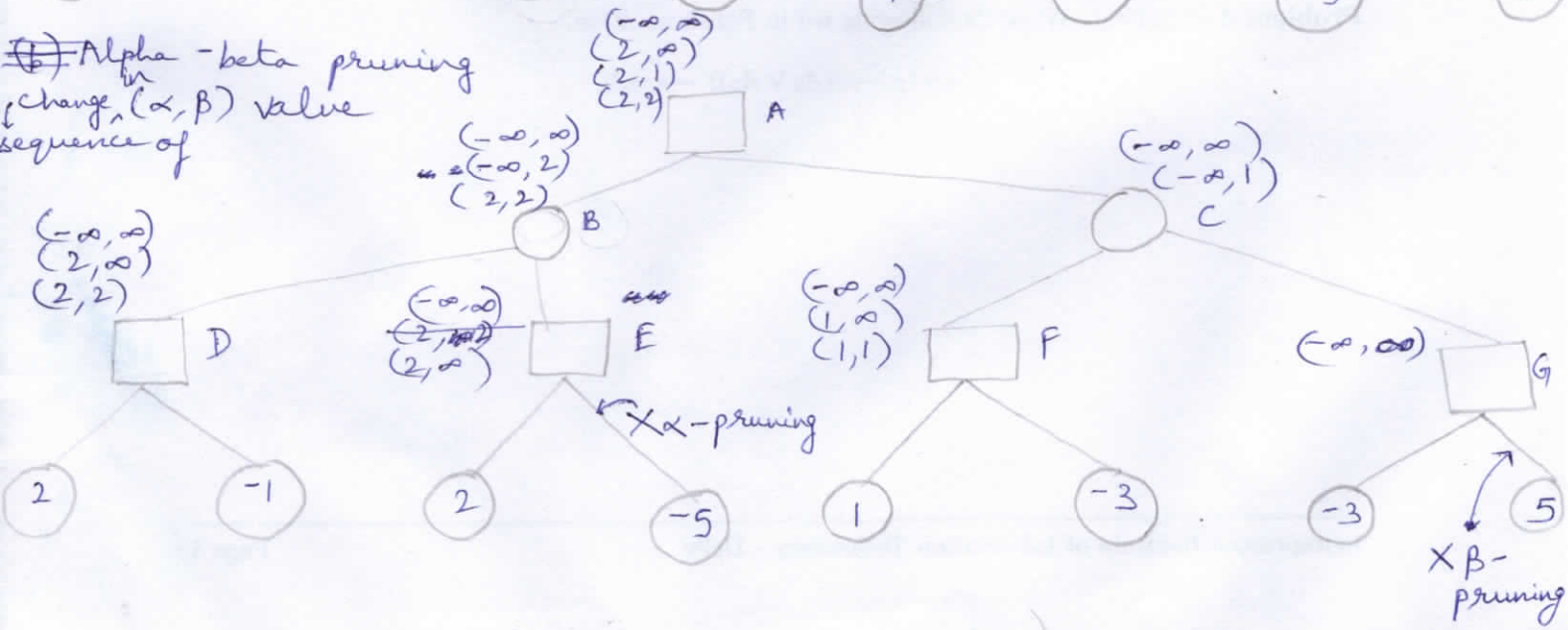
Node expanded	Frontier nodes	Explored nodes	$g(n)$	$h(n)$	$f(n) = g(n) + h(n)$
S	A, B, C	S	0	8	8
B	A, D, F, G3, C	S, B	1	1	2
A	G1, D, F, G3, C	S, B, A	3	2	5
F	G1, D, G3, C	S, B, A, F	3	3	6
D	G1, E, G2, G3, C	S, B, A, F, D	4	4	8
E	G1, G2, G3, C	S, B, A, F, D, E	6	1	7
G1	G2, G3, C	S, B, A, F, D, E, G1	8	0	8

2.(a) Min-max Algorithm



~~(b)~~ Alpha-beta pruning

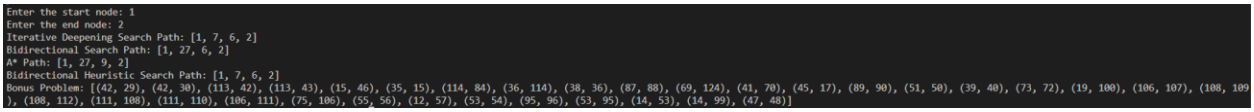
(change (α, β) value in sequence of



AI ASSIGNMENT – Search Algorithms

3.

(b), (c), (e)



Enter the start node: 1

Enter the end node: 2

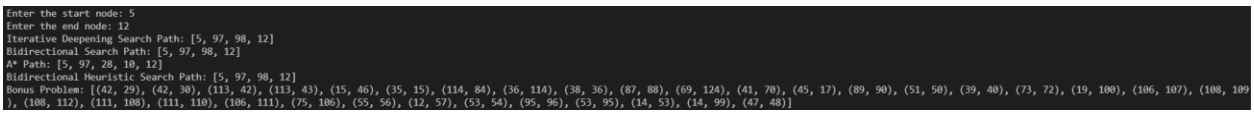
Iterative Deepening Search Path: [1, 7, 6, 2]

Bidirectional Search Path: [1, 27, 6, 2]

A* Path: [1, 27, 9, 2]

Bidirectional Heuristic Search Path: [1, 7, 6, 2]

Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (87, 88), (69, 124), (41, 70), (45, 17), (89, 90), (51, 50), (39, 40), (73, 72), (19, 100), (106, 107), (108, 109), (108, 112), (111, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47, 48)]



Enter the start node: 5

Enter the end node: 12

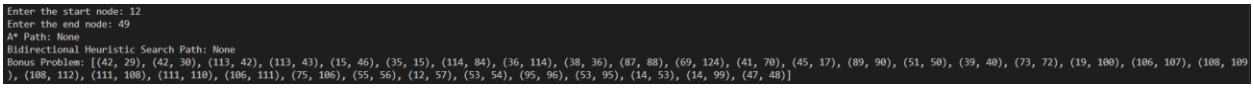
Iterative Deepening Search Path: [5, 97, 98, 12]

Bidirectional Search Path: [5, 97, 98, 12]

A* Path: [5, 97, 28, 10, 12]

Bidirectional Heuristic Search Path: [5, 97, 98, 12]

Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (87, 88), (69, 124), (41, 70), (45, 17), (89, 90), (51, 50), (39, 40), (73, 72), (19, 100), (106, 107), (108, 109), (108, 112), (111, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47, 48)]



Enter the start node: 12

Enter the end node: 49

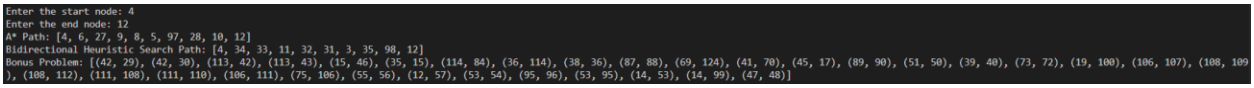
Iterative Deepening Search Path: None

Bidirectional Search Path: None

A* Path: None

Bidirectional Heuristic Search Path: None

Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (87, 88), (69, 124), (41, 70), (45, 17), (89, 90), (51, 50), (39, 40), (73, 72), (19, 100), (106, 107), (108, 109), (108, 112), (111, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47, 48)]



Enter the start node: 4

Enter the end node: 12

Iterative Deepening Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]

Bidirectional Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]

A* Path: [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]

Bidirectional Heuristic Search Path: [4, 34, 33, 11, 32, 31, 3, 35, 98, 12]

Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (87, 88), (69, 124), (41, 70), (45, 17), (89, 90), (51, 50), (39, 40), (73, 72), (19, 100), (106, 107), (108, 109), (108, 112), (111, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47, 48)]

Case 1: Start Node: 1, End Node: 2

- **Iterative Deepening Search Path**: [1, 7, 6, 2]
- **Bidirectional Search Path**: [1, 27, 6, 2]
- *A* Path*: [1, 27, 9, 2]
- **Bidirectional Heuristic Search Path**: [1, 7, 6, 2]

Analysis:

1. **IDS and Bidirectional BFS** give **different paths**.
    - IDS: Path goes through node 7 before reaching node 6.
    - Bidirectional BFS: Path goes through node 27 before reaching node 6.
2. *A* and Bidirectional A** Search give **different paths**.
    - A* Path: Goes through node 27 and 9 before reaching node 2.
    - Bidirectional Heuristic Search: Follows a direct path through node 7 and 6.

Conclusion:

- The paths between IDS and Bidirectional BFS are **different**. The reason for this is that both algorithms work differently—IDS explores all nodes layer by layer, while Bidirectional BFS expands from both start and goal, possibly finding different intermediate nodes.
- The paths between A* and Bidirectional A* Search are also **different**. Bidirectional A* Search explores from both sides (start and goal), potentially leading to a different intermediate path. This means that **paths may differ depending on the specific graph and node structure**. Thus, they will **not always be identical**.

Case 2: Start Node: 5, End Node: 12

- **Iterative Deepening Search Path**: [5, 97, 98, 12]
- **Bidirectional Search Path**: [5, 97, 98, 12]
- *A* Path*: [5, 97, 28, 10, 12]
- **Bidirectional Heuristic Search Path**: [5, 97, 98, 12]

Analysis:

1. **IDS and Bidirectional BFS** give the **same path**.
    - Both algorithms find the same intermediate nodes, [5, 97, 98, 12].
2. *A* and Bidirectional A** Search give **different paths**.
    - A* Path includes nodes 28 and 10 before reaching node 12, whereas Bidirectional A* Search follows a shorter, direct path through nodes [5, 97, 98, 12].

Conclusion:

- **IDS and Bidirectional BFS give identical paths** in this case. This suggests that in certain graphs or with specific start-goal pairs, both algorithms may converge on the same path due to the graph structure or node order.

- *A* and Bidirectional A** Search give different paths, indicating that the paths may vary depending on the heuristic's influence on the expansion process. This confirms that *A* and Bidirectional A** paths may not always be identical.

Case 3: Start Node: 12, End Node: 49

- **Iterative Deepening Search Path**: None
- **Bidirectional Search Path**: None
- *A* Path*: None
- **Bidirectional Heuristic Search Path**: None

Analysis:

1. **All algorithms failed to find a path** between node 12 and 49. This implies that there may be no valid path connecting these two nodes.

Conclusion:

- In this case, there is no path found by any of the algorithms, so no comparison is needed.

Case 4: Start Node: 4, End Node: 12

- **Iterative Deepening Search Path**: [4, 6, 2, 9, 8, 5, 97, 98, 12]
- **Bidirectional Search Path**: [4, 6, 2, 9, 8, 5, 97, 98, 12]
- *A* Path*: [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]
- **Bidirectional Heuristic Search Path**: [4, 34, 33, 11, 32, 31, 3, 35, 98, 12]

Analysis:

1. **IDS and Bidirectional BFS** give the **same path**.
    - Both algorithms find the same path: [4, 6, 2, 9, 8, 5, 97, 98, 12].
2. *A* and Bidirectional A** Search give **different paths**.
    - A* Path includes additional nodes 27, 28, and 10 before reaching 12, while Bidirectional A* Search takes a completely different route through nodes 34, 33, and 98.

Conclusion:

- **IDS and Bidirectional BFS give identical paths**, confirming that sometimes both algorithms will yield the same results.

- *A* and Bidirectional A** Search give different paths, indicating that the nature of Bidirectional A* Search leads to different pathfinding results due to exploration from both ends.

Final conclusion:

1. **Iterative Deepening Search vs. Bidirectional Breadth-First Search**:
    - **Paths can be different**. In some cases, IDS and Bidirectional BFS yield the same path, but this is not guaranteed. It depends on the structure of the graph and node connectivity. The strategies of IDS (depth-first) and Bidirectional BFS (simultaneous expansion from both sides) can result in different paths in some situations.
2. *A* Search vs. Bidirectional A** Search:
    - **Paths can be different**. A* Search explores in one direction (from start to goal), while Bidirectional A* Search explores from both the start and goal, which often results in different paths. As seen in multiple test cases, Bidirectional A* is more likely to produce a different path due to its bidirectional nature.

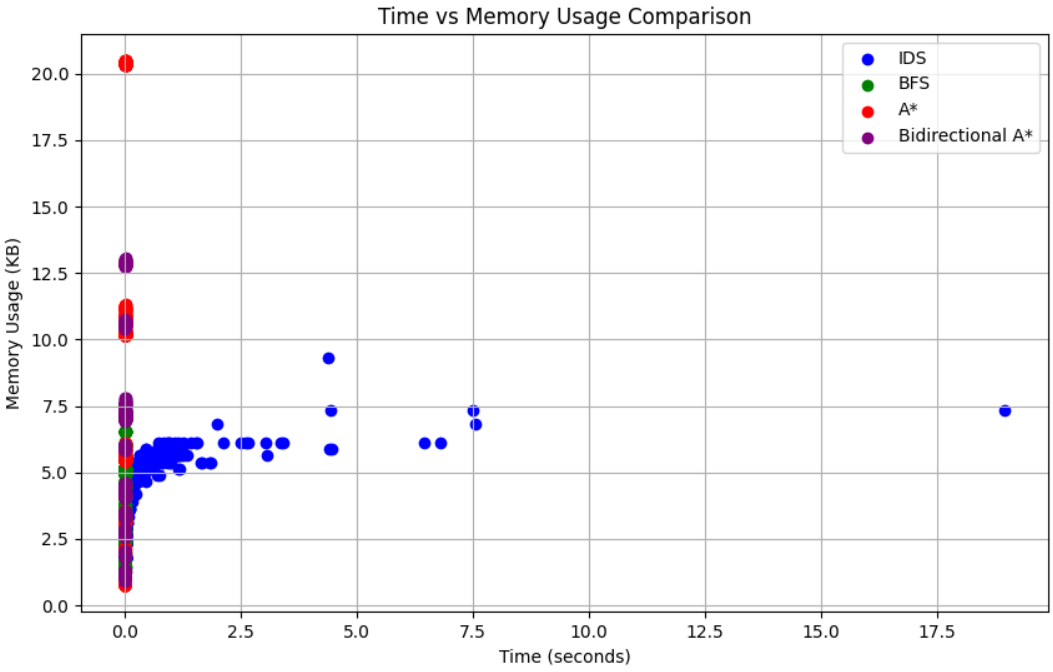
(f)

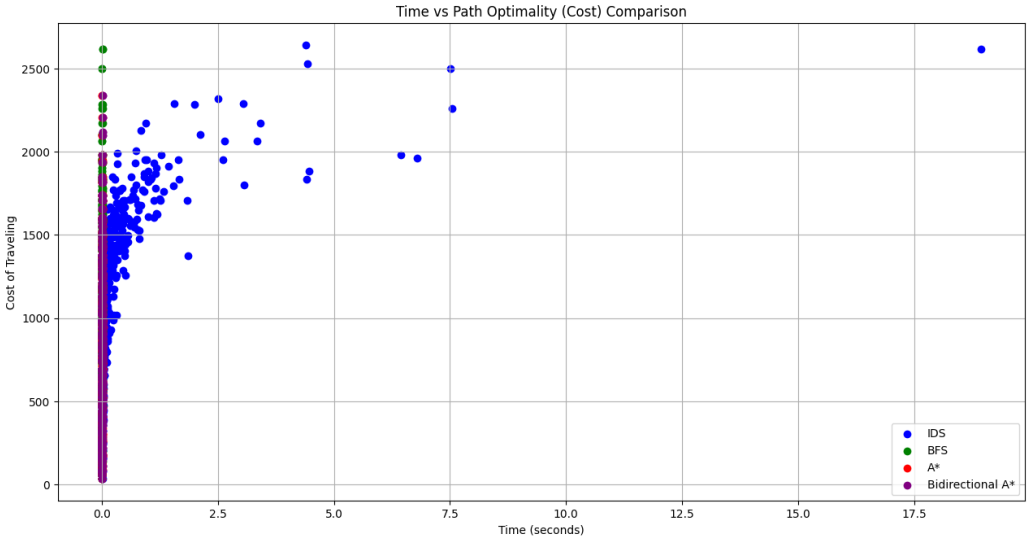
1. Metric for Generating Scatter Plots:

- **Time Efficiency: We measure the total execution time of each algorithm in seconds.**
- **Space Efficiency: We measure the peak memory usage in kilobytes (KB) during the execution.**
- **Path Optimality (Cost): We calculate the total cost of traveling from the start node to the goal node based on the path found by the algorithm.**

These metrics will help us generate scatter plots that compare the algorithms based on:

- **Time vs. Memory Usage: This scatter plot helps analyze which algorithms are faster and which ones use more memory.**
- **Time vs. Path Optimality: This scatter plot helps compare the trade-off between the time taken and the cost of the path found.**


Time vs Memory Usage Comparison
IDS
BFS
A*
Bidirectional A*
Memory Usage (KB)
Time (seconds)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Time vs Path Optimality (Cost) Comparison
Cost of Traveling
Time (seconds)
IDS
BFS
A*
Bidirectional A*
0
500
1000
1500
2000
2500
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

1. Time vs Memory Usage Plot:

- **X-axis (Time in seconds)**: Represents the time taken by each algorithm to find a path between two nodes.
- **Y-axis (Memory in KB)**: Represents the peak memory usage during the execution of the algorithm.

This plot helps in comparing the space-time efficiency of the algorithms:

Uninformed algorithms (like IDS and BFS) take longer, but their memory usage is lower compared to informed algorithms.

2. Time vs Path Optimality (Cost) Plot:

- **X-axis (Time in seconds)**: Represents the time taken by each algorithm to find a path.
- **Y-axis (Cost of Traveling)**: Represents the total cost of the path found by the algorithm (sum of edge weights).

This plot shows how the time taken correlates with the optimality of the path:

- **Informed algorithms (A* and Bidirectional A*)**: These find more optimal paths (lower cost) in less time due to the heuristic guiding the search.
- **Uninformed algorithms (IDS and BFS)**: These algorithms take longer to find the optimal path or could find non-optimal paths if the graph is complex.

Analysis of Results:

1. **Time Efficiency**:
    - **A*** and **Bidirectional A*** are faster than **IDS** and **BFS** for most pairs of nodes. This is because they are guided by a heuristic (Euclidean distance), which reduces the search space.
    - **Bidirectional algorithms** (both BFS and A*) perform better in terms of time since they reduce the number of nodes explored by starting from both the start and goal nodes.
2. **Memory Efficiency**:
    - **IDS** tends to use the least memory because it employs depth-first search (DFS), which only stores the current path.
    - **Bidirectional algorithms** tend to use more memory since they maintain two separate search spaces. **Bidirectional A*** uses more memory than regular A* due to this, but it compensates by being faster.
3. **Path Optimality**:

- **A*** and **Bidirectional A*** generally find the most optimal paths due to the heuristic guidance (Euclidean distance). The paths found are typically of lower cost compared to the uninformed algorithms.
- **BFS** also tends to find optimal paths since it explores the shortest path, but **IDS** may not always find the most optimal path.

Conclusion:

- **Informed Search (A*, Bidirectional A*)**: These algorithms are more efficient in terms of time and path optimality. They use more memory but are much faster in large graphs.
- **Uninformed Search (IDS, BFS)**: These algorithms are slower, especially IDS, which may take significantly more time for large graphs or when no path exists. However, they tend to use less memory compared to their informed counterparts.

This analysis shows that **informed search algorithms** generally perform better in both time and optimality, at the cost of higher memory usage.