AI Assignment 1

Vikranth Udandarao
2022570

VIKRANTH U
2022570

1-a. A*

$f(n) = \underbrace{g(n)}_{\text{path value}} + \underbrace{h(n)}_{\text{heuristic value}}$

S

$f(S) = g(S) + h(S)$
$= 0 + 8$
$= 8$

Expand S
$S \to A$, $S \to B$, $S \to C$

$f(A) = 3+2 = 5$, $f(B) = 1+1 = 2$, $f(C) = 5+8 = 13$

Expanded - S, B

Expand B

$B \to D$, $B \to F$, $B \to G3$

$f(D) = 7+4 = 11$, $f(F) = 2+3 = 5$, $f(G3) = 13+0 = 13$

Expanded - S, B

Since $f(A) = f(F)$ and is minimum now, we choose A [Alphabetically lesser]

Expand A

$A \to G1$, $A \to D$

$f(G1) = 13+0 = 13$, $f(D) = 7+4 = 11$

Expanded - S, B, A

→ Expand F

Expand F

F → D

$f(D) = 4+4$
$= 8$

Expanded - S, B, A, F

Expand D

D → E, D → G2

$f(E) = 6+1$
$= 7$, $f(G2) = 9+0$
$= 9$

Expanded - S, B, A, F, D

Expand E

E → G1

$f(G1) = 8 + 0$
$= 8$

Expanded - S, B, A, F, D, E.

Path is S → B → F → D → E → G1.

and cost is 8.

Expanded Node	Frontier	Explored	Total Path Cost
S	B-2, A-5, C-13	S	0
B	A-5, D-9, F-6, G3-13, C-13	S, B	1
A	D-9, F-6, G1-13, G3-13, C-13	S, B, A	3
F	D-8, G2-9, G1-13, C-13, G3-13	S, B, A, F	5
D	E-7, G2-9, G1-13, C-13, G3-13	S, B, A, F, D	3
E	G1-8, G2-9, C-13, G3-13	S, B, A, F, D, E	6
G1	Goal Reached	S, B, A, F, D, E, G1	8

6- uniform cost search only depends on path value

S

g(S) = 0

Expand S

S → A , S → B , S → C

g(A) = 0+3 = 3
g(B) = 0+1 = 1
g(C) = 0+5 = 5

Expand B [least cost] → Expanded - S

B → D , B → F , B → G3

g(D) = 1+4 = 5
g(F) = 1+2 = 3
g(G3) = 1+12 = 13

Expanded - S, B

Expand A [next least cost and alphabetically smaller]

A → G1 , A → D

g(G1) = 3+10 = 13
g(D) = 3+4 = 7

Expanded - S, B, A

Expand F

$F \rightarrow D$

$g(D) = 3 + 1 = 4$

Expanded - S, B, A, F

Expand D

$D \rightarrow E$, $D \rightarrow G2$

$g(E) = 4 + 2 = 6$
$g(G2) = 4 + 5 = 9$

Expanded - S, B, A, F, D

Expand C

$C \rightarrow G3$

$g(G3) = 5 + 11 = 16$

→ Expanded - S, B, A, F, D, C

Expand E

$E \rightarrow G1$

$g(G1) = 6 + 2 = 8$ -

G1 is goal node and we have reached it with path cost of 8.

Expanded - S, B, A, F, D, C, E.

Path is $S \rightarrow B \rightarrow F \rightarrow D \rightarrow E \rightarrow G1$.

Expanded Node	Frontier	Explored	Total Path Cost
S	B-1, A-3, C-5.	S	0
B	A-3, F-3, C-5, D-5, G3-13	S, B	1
A	F-3, C-5, D-5, G1-13, G3-13	S, B, A	3
F	D-4, C-5, G1-13, G3-13	S, B, A, F	3
D	C-5, E-6, G1-13, G2-9, G3-13	S, B, A, F, D	4
C	E-6, G1-13, G2-9, G3-13	S, B, A, F, D, C	5
E	G1-8, G2-9, G3-13	S, B, A, F, D, C, E	6
G1	Goal reached	S, B, A, F, D, C, E, G1	8

c. Iterative Deepening A* Search

$f(S) = 0 + 8 = 8.$

→ Threshold = 8.

Expand S

$S \to A$, $S \to B$, $S \to C$

$f(A) = 3+2 = 5$, $f(B) = 1+1 = 2$, $f(C) = 5+8 = 13$

Expanded - S

We will not explore in increasing order and we will do it from left to right.

Expand A

$A \to G1$, $A \to D$

$f(G1) = 13+0 = 13$, $f(D) = 7+4 = 11$.

Expanded – S, A

Expand B

$B \to D$, $B \to F$, $B \to G3$

$f(D) = 5+4 = 9$, $f(F) = 3+3 = 6$, $f(G3) = 13+0 = 13$.

Expanded – S, A, B

~~Expand A~~

$\cancel{A \to G1}$, $\cancel{A \to D}$

$\cancel{f(G1) = 13+0 = 13}$, $\cancel{f(D) = 7+4 = 11}$

Expand F.

$F \to D$

$f(D) = 4+4 = 8$

$\to$ Expanded – S, A, B, F

Now, we see that there are not nodes and paths left with threshold 8, so we increase it the next minimum one.

Expand D

$D \rightarrow E$, $D \rightarrow G2$

$f(E) = 6+1 = 7$, $f(G2) = 9+0 = 9$

Expanded - S, A, B, F, D

Expand E

$E \rightarrow G1$

Expanded - S, A, B, F, D, E

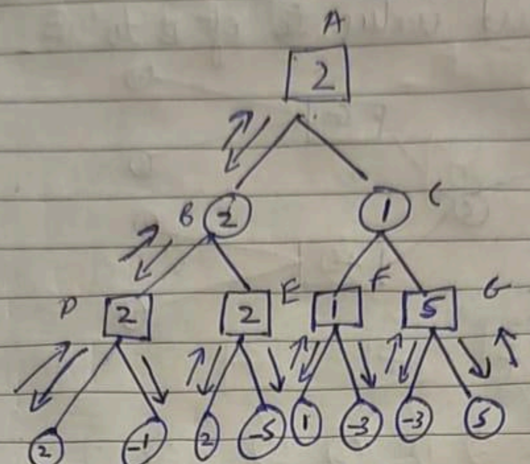
$f(G1) = 8+0 = 8$

Thus, G1 is the goal node reached with a total cost of 8 and the path is $S \rightarrow B \rightarrow F \rightarrow D \rightarrow E \rightarrow G1$

Threshold	Expanded Nodes	Frontier	Explored Nodes	Total Path Cost
8	S	A-5, B-2, C-13	S	0
8	A	B-2, D-5, F-6, G1-13	S, A	1
8	B	D-5, F-6, G1-13	S, A, B	3
8	F	D-5, E-7, G2-9, G1-13	S, A, B, F	3
8	D	E-7, G2-9, G1-13	S, A, B, F, D	5

8	E	G1 - 8	S, A, B, F, D, E	6
8	G1	Goal Reached	S, A, B, F, D, E, G1	8

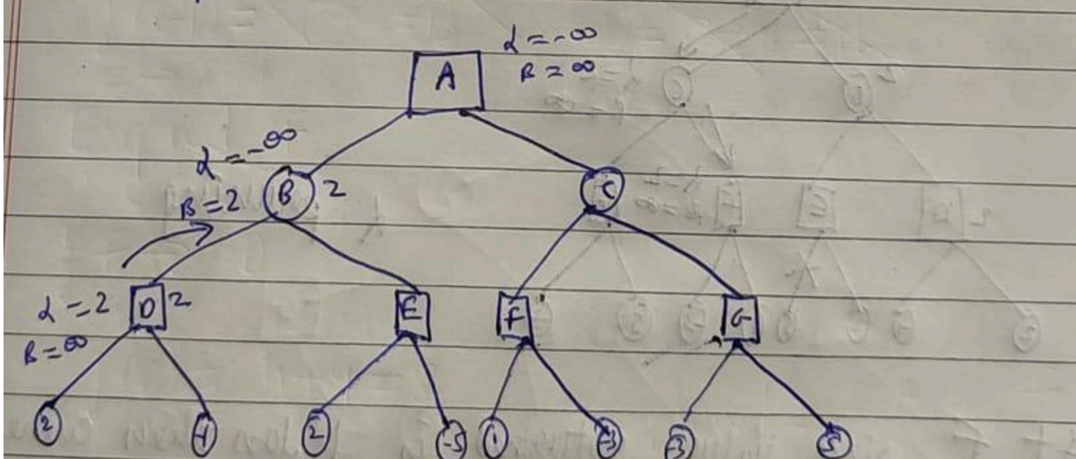
2a-



This is the best play at every level for both the players.

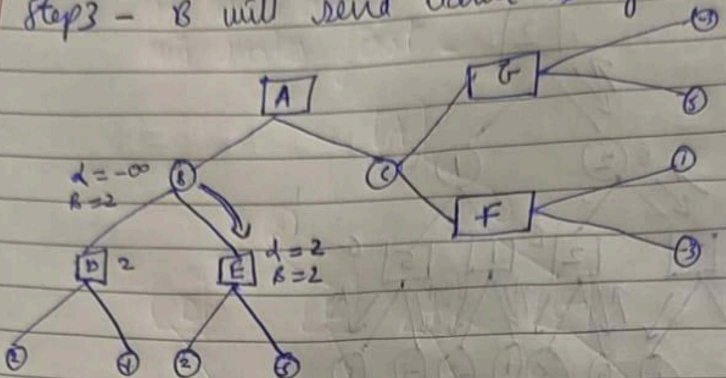
Alpha-Beta Pruning

Step1 - Max player starts from Node A [$\alpha = -\infty$, $\beta = \infty$]. These values will be passed to B where $\alpha = -\infty$, $\beta = \infty$ and then it gets passed to D.



Step 2 - At D, α will be calculated as max's turn so $\alpha = 2$ and thus $D = 2$. It will the backtrack to B and B value will change.

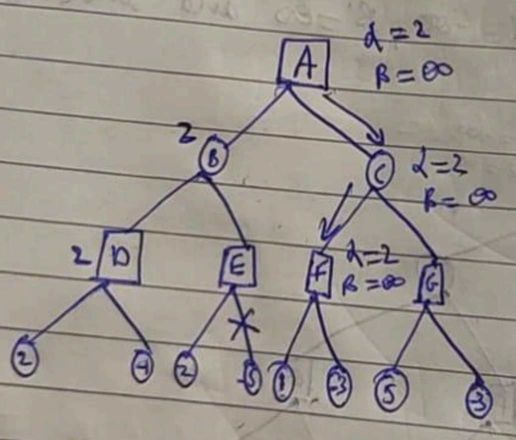
Step 3 - B will send value of B to E.



We prune the right child of E as $\alpha = 2 \geq \beta = 2$.

Step 4 - We again backtrack to B and then to A. At A, we choose maximum. So α will change to 2.

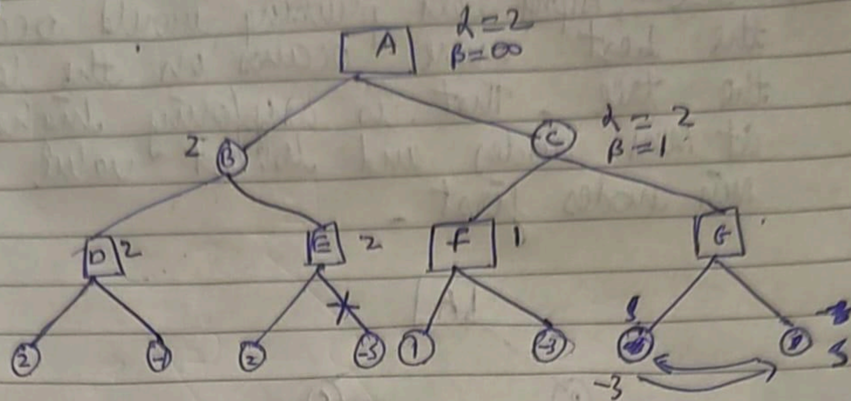
Step 5 - We pass this to C which passes the value to F.



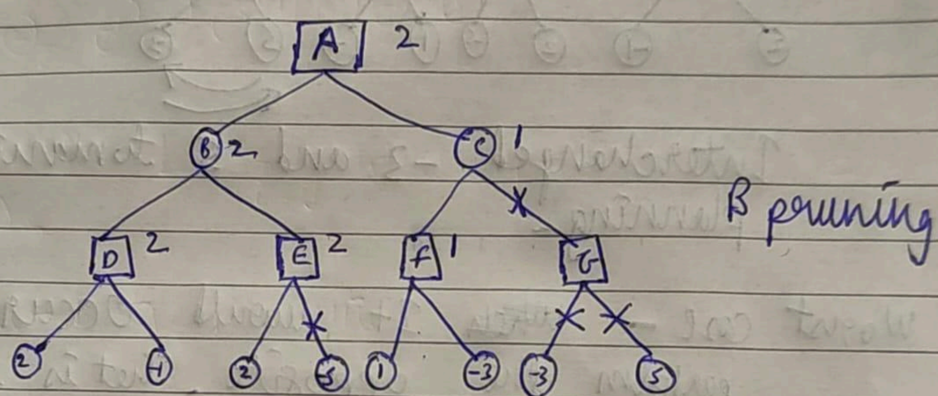
α pruning

At F, since value returned is 1 less then current alpha value, No change is made to α. 1 is returned to C. So β value at C will become 1.

Step 6 -



Since, $\alpha \geq \beta$, we prune G and tree returns 2.

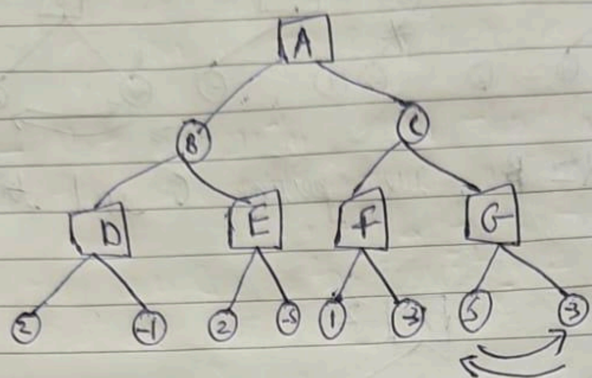


α pruning is applied by maximising player to cut off minimising level.

β pruning is applied by minimising player to cut off maximising level.

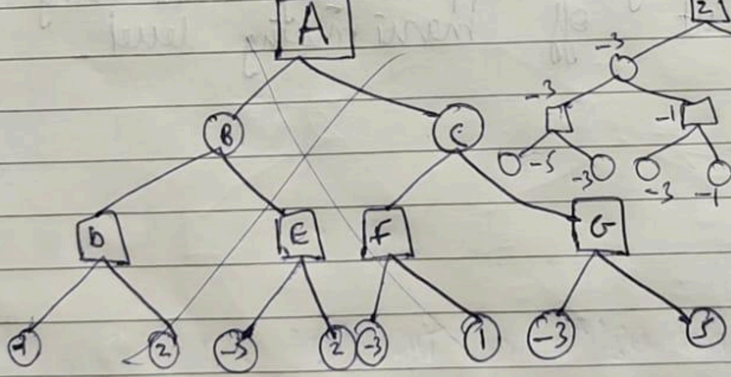
6- B

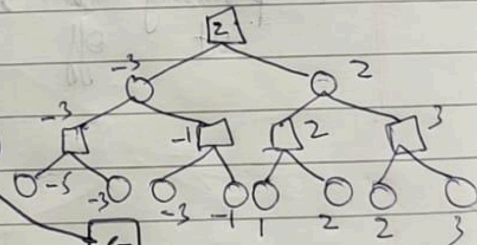
Best case — Alpha beta pruning would occur when the best move occurs on the left side of the tree. That is exploring highest values at max nodes and lowest value nodes at min nodes first.



Interchanged -3 and 5 to maximise planning.

Worst case - ~~then~~ It would occur when we perform the opposite. That is for every lowest value node at max node will be explored first and highest value node at min ~~node~~ node will be explored first.





Left branch has least value at max node.

mycompanion

6- $O(b^{d/2})$ is the best case improving an average alpha-beta pruning because with an average of branching factor of b and search depth of d, the maximum ~~number~~ number of leaf nodes evaluated would be $O(b \times b \times \ldots \times b)$ d times $= O(b^d)$. Similar to minimax search. If the move order is optimal, the leaf node evaluated is about $O(b \times 1 \times b \times 1 \times \ldots \times 1)$ for even depth. That is ~~basically~~ all the first player moves must be suited to find the best one, but for each, only the best second player's move is needed to consider all but the first (and best) player move. This ensures no other second player moves need to be calculated.

3. a.

```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A1>  python .\code_2022570.py
Enter the start node: 1
Enter the end node: 2
Iterative Deepening Search Path: [1, 7, 6, 2]
Bidirectional Search Path: [1, 7, 6, 2]
A* Path: [1, 27, 9, 2]
Bidirectional Heuristic Search Path: [1, 27, 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), (1
11, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47,
48)]
```

```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A1> python .\code_2022570.py
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), (1
11, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47,
48)]
```

```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A1> python .\code_2022570.py
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), (1
11, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47,
48)]
```

```
PS C:\Users\vikra\OneDrive\Desktop\CSE643-AI\A1> python .\code_2022570.py
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, 5, 97, 28, 10, 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), (1
11, 108), (111, 110), (106, 111), (75, 106), (55, 56), (12, 57), (53, 54), (95, 96), (53, 95), (14, 53), (14, 99), (47,
48)]
```

b.

3-6-

	TC1	TC2	TC3	TC4
Start Node	1	5	12	4
End Node	2	12	49	12
IDS Path	[1,7,6,2]	[5,97,98,12]	None	[4,6,2,9,8,5,97,98,12]
BDS Path	[1,7,6,2]	[5,97,98,12]	None	[Same for both]

For -

never

TC1 - Yes, ~~a~~ IDS & BDS will always return ~~the~~ same path for ~~nodes~~ start and end nodes 1 and 2 as this is always the least cost and best path and only this one path is there.

mycompanion

TC2 - ~~Yes~~ No, IDS & BDS will always return the same paths for start & end nodes 5 and 12 as this is always the least cost and best path.

TC3 - Both IDS & BDS will always return None as there does not exist any paths between 12 & 47.

TC4 - Not necessarily, even though they returned the same path, there might be many other least costs and best paths. If multiple paths of same depth exist, then different paths can be formed.

IDS explores deeper nodes first whereas BDS ~~explores~~ BFS on both sides.
Also IDS doesn't guarantee finding shortest path as efficiently as BDS.

c./e. path_results.csv has all the paths for all the four algorithms - IDS, BDS, A*, BHDS

Iterative Deepening Search:

- **Time:** IDS is slower because it repeatedly explores the graph with increasing depth limits.
    - 11240 seconds
- **Memory:** It is supposed to use less memory as it is depth first storing a single path but while running the python code, we got high memory usage.
    - 39008 KB

Bidirectional Breadth-First Search:

- **Time:** BDS is faster because it simultaneously explores the graph from both ends.
    - 3.13 seconds

- **Memory:** It is supposed to use more memory as it has 2 frontiers and priority queues to store 2 simultaneous paths but while running the python code, we got lesser memory usage compared to IDS.
    - 2300 KB

A1 > performance.txt

```
--- IDS Performance ---
Memory used: 39008 KB
Time taken: 11240.016671657562 seconds

--- BDS Performance ---
Memory used: 2300 KB
Time taken: 3.136460065841675 seconds

--- A* Performance ---
Memory used: 2040 KB
Time taken: 7.463478326797485 seconds

--- BHDS Performance ---
Memory used: 1844 KB
Time taken: 5.247537136077881 seconds
```

A* Search:

- **Time:** A* is faster than IDS but slower than BDS and BHDS because it uses a heuristic function to guide its search towards the goal. However, it explores more nodes than BHDS since it is not bidirectional. The time complexity depends heavily on the quality of the heuristic function.
    - 7.46 seconds
- **Memory:** A* generally uses more memory than BHDS because it stores all visited nodes and the priority queue required for evaluating the next best node. It uses more memory compared to BHDS but less than IDS.
    - 2040 KB

Bidirectional Heuristic Depth Search:

- **Time:** BHDS is one of the fastest algorithms due to its use of both bidirectional search and a heuristic function, allowing it to explore both ends of the path simultaneously while prioritizing nodes likely to lead to the goal. This significantly reduces the number of nodes explored.
    - 5.25 seconds
- **Memory:** BHDS has the lowest memory usage because it leverages both bidirectional search and heuristics, efficiently reducing the number of nodes stored and evaluated.

- 1844 KB

Algorithm	Time (seconds)	Memory (KB)
IDS	11240	39008
BDS	3.13	2300
A*	7.46	2040
BHDS	5.25	1844

e.

e-

	TC1	TC2	TC3	TC4
Start Node	1	5	12	4
End Node	2	12	49	12
A* Path	[1,27,9,2]	[5,97,28,10,12]	None	[4,6,27,98,5,97,28,10,12]
BHDS Path	[1,27,6,2]	[5,97,98,12]	None	[4,34,33,71,32,31,3,5,97,28,10,12]

for-

TC1 - The paths are different as A* search expands the graph outward from start node while BHDS expands from both start and end node and thus may find a different meeting point.

TC2 – The paths are different but both are valid shortest paths as BHDS finds a different meeting point which might return better shorter path.

TC3 – Both algorithms will return None as no path exists between start and end node.

TC4 – Same as TC1&TC2, the paths are different but both are valid shortest paths.

In both A* and Bi-directional A* search algorithms, the goal is to find the shortest path from the start node to the goal node. However, these algorithms can produce different paths due to the nature of their search strategies. Let's analyze their performance for the public test cases:

Test Case 1:

- **Start Node**: 1, **Goal Node**: 2
  - *A Search Result**: `[1, 27, 9, 2]`
  - *Bi-directional A Search Result**: `[1, 27, 6, 2]`
  - Both algorithms return different paths. A* search expands the graph outward from the start node, while Bi-directional A* expands from both the start and goal, potentially meeting in the middle at a different point. Both paths are valid shortest paths. Therefore, **the paths can be different** based on the expansion strategy but will still yield an optimal solution.

Test Case 2:

- **Start Node**: 5, **Goal Node**: 12
  - *A Search Result**: `[5, 97, 28, 10, 12]`
  - *Bi-directional A Search Result**: `[5, 97, 98, 12]`
  - The paths are different, but both are valid shortest paths. The bi-directional search finds a different meeting point between the start and goal, leading to a shorter second half of the path. Again, both algorithms return optimal paths, but the path itself can vary.

Test Case 3:

- **Start Node**: 12, **Goal Node**: 49

- *A Search Result**: `None`
- *Bi-directional A Search Result**: `None`
- Both algorithms correctly return `None`, meaning no path exists between the start and goal nodes. In cases where no path exists, both algorithms will return identically.

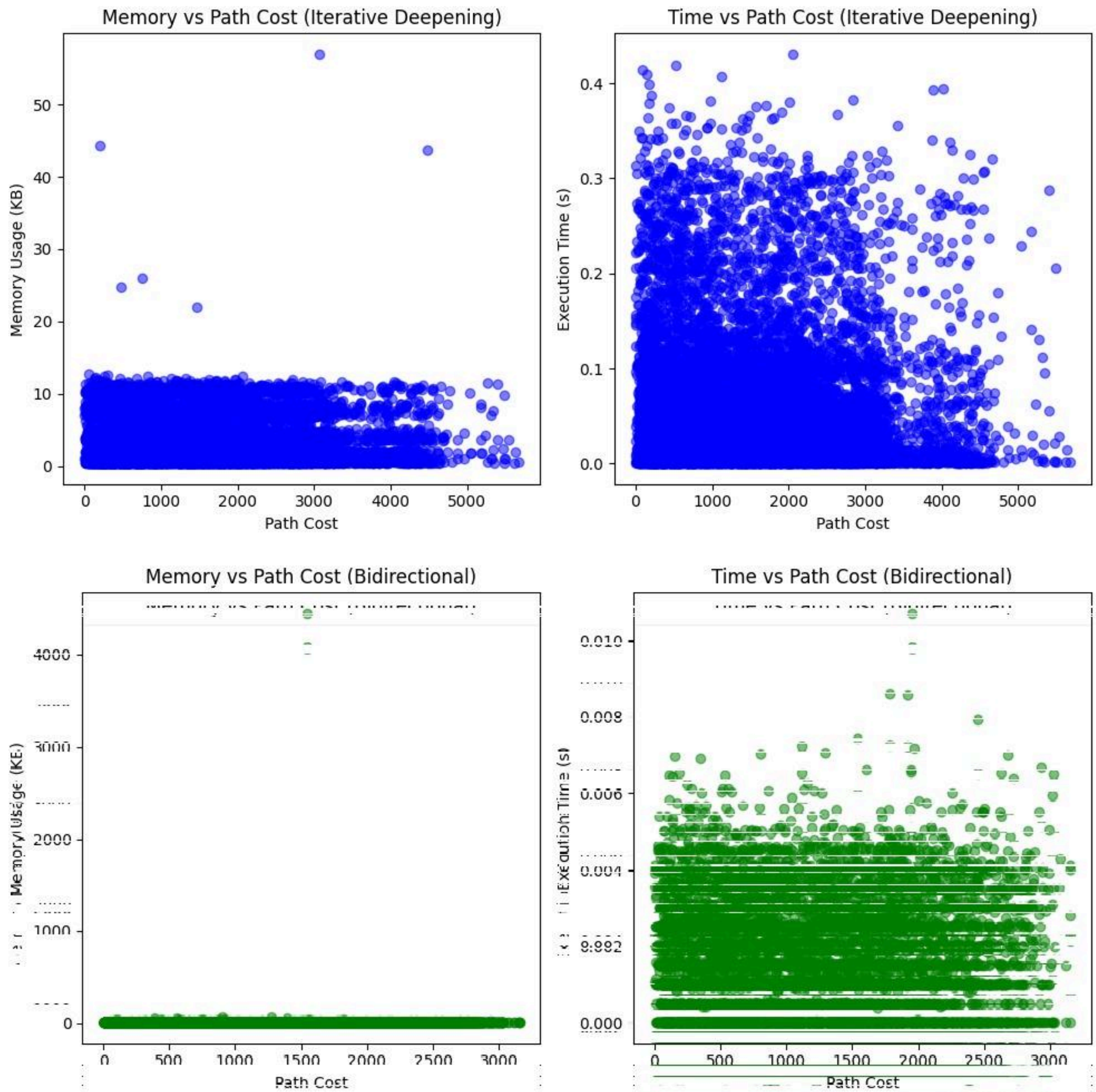
Test Case 4:

- **Start Node**: 4, **Goal Node**: 12
  - *A Search Result**: `[4, 6, 27, 9, 8, 5, 97, 28, 10, 12]`
  - *Bi-directional A Search Result**: `[4, 34, 33, 11, 32, 31, 3, 5, 97, 28, 10, 12]`
  - **Analysis**: The paths are different, but both are valid shortest paths. Again, the meeting point in the bi-directional search leads to a different path, but both are optimal.

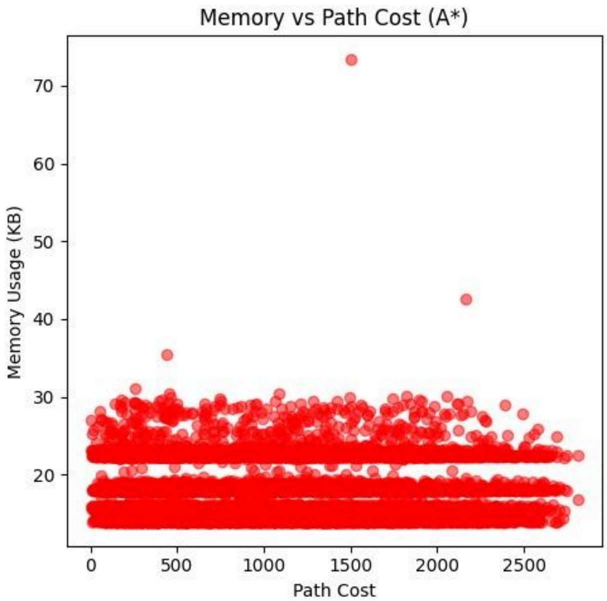
Conclusion:

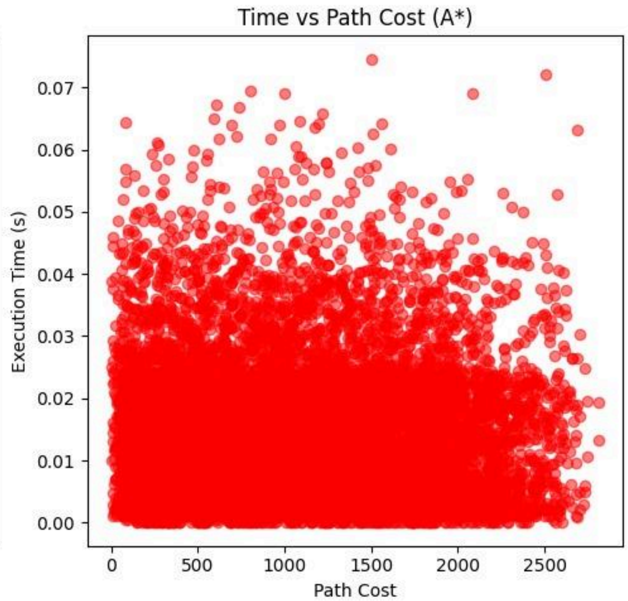
- **Path Difference**: The paths produced by A* and Bi-directional A* search can **differ** because of the way the two algorithms expand the search. A* searches outward from the start node, while Bi-directional A* searches from both the start and goal, potentially meeting at different points in the graph. Therefore, the specific paths may vary based on where the searches meet, but both paths will be optimal (shortest path).
- **Will the paths always be identical?** No, the paths will not always be identical. While both algorithms guarantee an optimal solution (the shortest path in terms of cost), the actual nodes traversed can differ due to the differing expansion strategies of the algorithms.

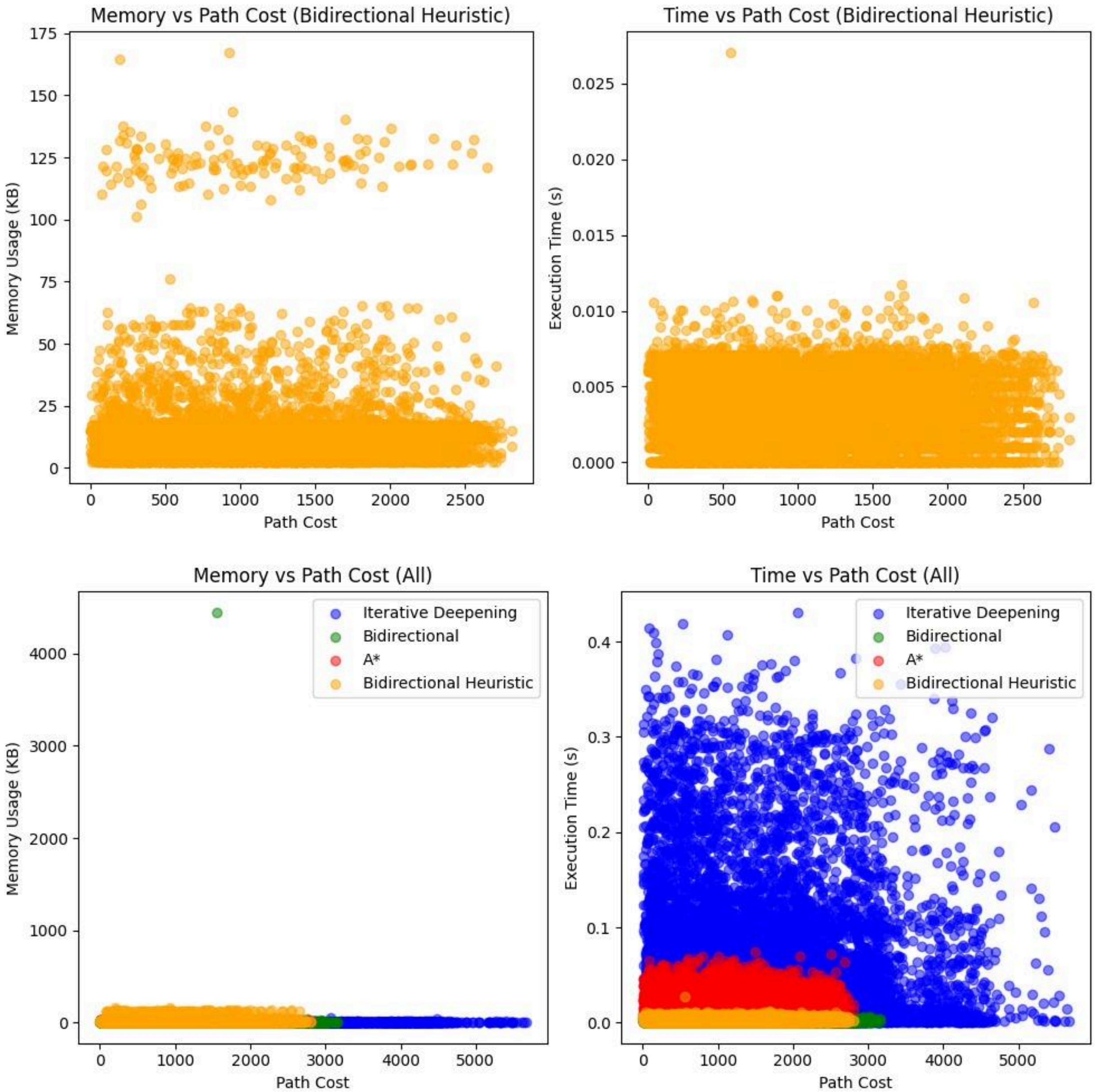
f.



Memory vs Path Cost (Iterative Deepening)
Time vs Path Cost (Iterative Deepening)
Memory Usage (KB)
Execution Time (s)
Path Cost
Memory vs Path Cost (Bidirectional)
Time vs Path Cost (Bidirectional)


Memory vs Path Cost (A*)
Memory Usage (KB)
Path Cost
70
60
50
40
30
20
0
500
1000
1500
2000
2500


Time vs Path Cost (A*)
Execution Time (s)
Path Cost
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
500
1000
1500
2000
2500



Memory vs Path Cost (Bidirectional Heuristic)
Memory Usage (KB)
Path Cost
Time vs Path Cost (Bidirectional Heuristic)
Execution Time (s)
Path Cost
Memory vs Path Cost (All)
Memory Usage (KB)
Path Cost
Iterative Deepening
Bidirectional
A*
Bidirectional Heuristic
Time vs Path Cost (All)
Execution Time (s)
Path Cost
Iterative Deepening
Bidirectional
A*
Bidirectional Heuristic

f- Efficiency

→ Memory usage vs Path Cost -

1- IDS shows high memory usage at certain points with several outliers reaching over 50KB, despite the expectation of lesser memory usage. This is due to repeated depth first exploration which accumulates memory over recursive calls.

2- BDS uses significantly lesser memory with outliers at around 4000KB. However it maintains low memory footprint for most path costs.

3- A* & BHDS demonstrates lower memory usage compared to IDS & BDS. BHDS is very memory efficient, while A* has few outliers near 20KB.

Time vs Path Cost -

1- IDS has much higher time complexity compared to other algorithms. This is evident from dense distribution of points around high time values.

2- BDS, A* & BHDS show better performance in time performance. BHDS uses a heuristic that improves time & memory while BDS benefits from simultaneously exploration form both ends of path.

3- A* shows variation but remains more efficient than IDS although slower than BHDS & BDS.

→ Path Cost / Optimality

1- IDS explores all possible paths which ensures optimality but at the cost of increased memory & time. IDS is exhaustive but inefficient.

2- BDS is optimal as it uses breadth first search from both ends. However, it may not guarantee lowest path cost unless properly unconstrained.

3- A* guarantees optimality, provided heuristic function is admissible. A* often finds shorter paths but still consumes more memory than BHDS.

4- BHDS leverages both bidirectional search and heuristic resulting in even lower path cost while memory remains ~~these~~ efficient.

Scatter Plot Metrics

1- The scatter plots were generated by plotting Path cost on x axis and comparing it to Memory usage (KB) or Execution Time (seconds) on y axis

2- memory usage is measured based on amount of data stored in data structures like arrays, queues, stacks and explored nodes.

3- Time is measured based on total time taken to explore the graph and reach solution.

Benefits of using informed search Algorithms over uninformed ones:-

-> More efficient in memory & time.

-> Heuristics guide the search, reducing no. of nodes needed to be explored.

Drawbacks -

-> If heuristic is not admissible, A* may not guarantee shortest path. Memory usage might spike if many nodes are expanded based on suboptimal heuristic.

In general, informed search algorithms significantly outperform uninformed search algorithms in terms of memory and time.

1. Efficiency (Space and Time):

- **Memory Usage vs Path Cost**:
  - **Iterative Deepening Search (IDS)** shows a very high memory usage at certain points, with several outliers reaching over 50 KB, despite the expectation of low memory usage. This is due to repeated depth-first exploration, which accumulates memory over recursive calls.
  - **Bidirectional Search (BDS)** uses significantly less memory, with outliers at around 4000 KB. However, it maintains a relatively low memory footprint for most path costs.

    - **A*** and **Bidirectional Heuristic (BHDS)** search algorithms demonstrate consistently lower memory usage compared to IDS and BDS. Particularly, BHDS is very memory efficient, rarely exceeding 25 KB, while A* has a few outliers near 70 KB but generally maintains a low profile.
- **Time vs Path Cost**:
    - **IDS** has much higher time complexity compared to other algorithms. This is evident from the dense distribution of points around the higher time values, reaching up to 0.4 seconds. IDS is inherently slower due to the repeated exploration up to increasing depth limits.
    - **BDS**, **A***, and **BHDS** show much better performance in terms of time, with BDS and BHDS being particularly fast. BHDS uses a heuristic that improves both time and memory, while BDS benefits from simultaneous exploration from both ends of the path.
    - **A*** shows some variation in time as path costs increase but remains more efficient than IDS, although slower than BHDS and BDS.

2. Optimality (Path Cost):

- **Iterative Deepening Search (IDS)** explores all possible paths up to the solution, which ensures optimality but at the cost of increased memory and time. The path costs range significantly, demonstrating that IDS is exhaustive but inefficient.
- **BDS** is optimal in most cases, as it uses a breadth-first search strategy from both ends. However, it may not guarantee the lowest path cost unless properly constrained.
- **A*** guarantees optimality, provided the heuristic function is admissible (does not overestimate the cost to reach the goal). Informed by the heuristic, A* often finds shorter paths faster but still consumes more memory than BHDS.
- **BHDS** leverages both bidirectional search and a heuristic, resulting in even lower path costs while remaining memory efficient.

3. Scatter Plot Metrics:

- The scatter plots were generated by plotting **Path Cost** on the x-axis and comparing it to either **Memory Usage** (KB) or **Execution Time** (seconds) on the y-axis.
- Memory usage is measured based on the amount of data stored in queues, stacks, and explored nodes during the search process.
- Time is measured based on the total time taken to explore the graph and reach the solution.

Benefits of Informed Search Algorithms:

1. **Efficiency in Time**:
    - **Informed algorithms** (A* and Bidirectional Heuristic Depth Search - BHDS) incorporate heuristic functions, which drastically reduce the number of nodes explored, leading to significantly faster execution times.

- As seen in the scatter plots, **BHDS** and **A*** completed the searches in **5.25 seconds** and **7.46 seconds**, respectively, while **uninformed algorithms** like IDS took over **11,240 seconds**. The difference is clear in the time vs. path cost scatter plots, where IDS is consistently slower compared to informed searches.
- **BDS** also performed well, completing in **3.13 seconds**, but informed search algorithms still outperformed it in certain larger graphs or more complex search spaces due to the guidance provided by heuristics.

2. **Memory Efficiency**:
    - **Informed algorithms** often use less memory because the heuristic helps direct the search towards the goal, avoiding exploration of unnecessary nodes.
    - For example, **BHDS** had the lowest memory usage, peaking at **1844 KB**, compared to **IDS**'s **39,008 KB**. The memory vs. path cost scatter plots show that both A* and BHDS had lower, more controlled memory usage compared to uninformed search algorithms like IDS and BDS.
3. **Optimality**:
    - Informed search algorithms, like **A***, guarantee finding the optimal solution as long as the heuristic is **admissible** (i.e., it does not overestimate the cost).
    - This balance of efficiency and optimality makes informed algorithms the preferred choice in many practical scenarios where time and memory are limited, and the quality of the solution is important.

Drawbacks of Informed Search Algorithms:

1. **Dependence on Heuristic Quality**:
    - The performance of informed algorithms is heavily dependent on the quality of the heuristic. If the heuristic is **non-admissible** (overestimating the cost) or **poorly chosen**, the algorithm may end up exploring unnecessary paths, leading to suboptimal performance.
    - For example, while A* performs well in most cases, its memory usage may spike if the heuristic does not direct the search effectively. This can be seen in the outliers in the memory usage scatter plot for A*, where it exceeded **70 KB** in some cases, higher than BHDS.
2. **Heuristic Computation Overhead**:
    - While informed search algorithms like **A*** and **BHDS** are faster, they incur an overhead for calculating the heuristic function at each step. If the heuristic is computationally expensive, it can reduce the overall benefit of using an informed algorithm, particularly in environments where time is critical.
    - In contrast, **uninformed algorithms** like **BDS** don’t require a heuristic and simply explore the space in both directions, making them faster in cases where a good heuristic is difficult to compute.
3. **Higher Memory Usage for Poor Heuristics**:
    - If the heuristic is not well-tuned, informed search algorithms can use significantly more memory than expected. A* can end up exploring many nodes unnecessarily if the heuristic provides poor guidance. This is evident in the few outliers for

memory usage in the scatter plots for A*, where memory usage spiked unexpectedly.
  - Uninformed algorithms, despite being slower, have more predictable memory usage in some cases, as they do not rely on a heuristic. For example, **BDS** consistently used about **2300 KB** of memory.

Informed algorithms (A* and BHDS) show clear advantages in both time and memory efficiency due to the guidance of heuristics, making them ideal for large and complex search spaces where optimality and efficiency are crucial.

Uninformed algorithms (IDS and BDS) are more reliable when no heuristic information is available, but they suffer from inefficiencies in both time and space, making them impractical for most real-world applications.

Conclusion:

- **BHDS** offers the best trade-off between time and memory usage, outperforming the others due to its heuristic and bidirectional strategy.
- **IDS** is the most time-inefficient and consumes more memory than expected, particularly in implementations.
- **A*** provides a reliable heuristic-based search with strong performance but uses slightly more memory compared to BHDS.
- **BDS** performs well with reasonable memory usage, but it's time efficiency is heavily influenced by the size of the graph and the search space.

In general, **informed search algorithms** like A* and BHDS significantly outperform **uninformed algorithms** in terms of memory and execution time while maintaining optimality.

g. 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)]