Name: Yussef Mohamed Ameen Mohamed ID: 202304504
[AI102 – Group A21 – Individual] Under supervision: Dr. sahar ghanem and Eng. esraa.habiba

Introduction

The goal of this project is to analyze and compare six fundamental search algorithms: Breadth-First Search (BFS), Depth-First Search (DFS), Greedy Best-First Search, Uniform Cost Search (UCS), Hill-Climbing, and A* Search. These algorithms are applied to solve the classic 8-puzzle problem. The 8-puzzle consists of a 3x3 grid with tiles numbered 1 through 8 and a blank space. The objective is to rearrange the tiles to match a goal configuration by sliding them one at a time into the blank space. By implementing and analyzing these algorithms, we aim to understand their strengths, weaknesses, and suitability for solving this type of problem.

In this project, we will explore the BFS and DFS algorithms, tracing their steps from initial states to the goal state, and compare their performance and suitability for solving this type of problem.

Algorithm Descriptions

Breadth-First Search (BFS)

BFS is an algorithm for traversing or searching tree or graph data structures. It starts at the root and explores all nodes at the present depth level before moving on to the nodes at the next depth level. BFS uses a queue to keep track of nodes to visit and ensures that the shortest path in terms of edge count is found in an unweighted graph.

Depth-First Search (DFS)

DFS is an algorithm for traversing or searching tree or graph data structures. It starts at the root (or an arbitrary node) and explores as far as possible along each branch before backtracking. DFS uses a stack to keep track of nodes to visit.

Greedy Best-First Search

Greedy Best-First Search uses a heuristic to guide the search. It expands the node that appears to be closest to the goal based on the heuristic value. This algorithm is not guaranteed to find the shortest path.

Uniform Cost Search (UCS)

UCS is an algorithm similar to BFS but takes path cost into account. It expands the least costly node first. UCS is optimal and complete, ensuring the least-cost path is found.

Hill-Climbing

Hill-Climbing is an optimization algorithm that continuously moves towards the direction of increasing value (uphill) to find the peak of the solution space. It can get stuck in local maxima.

A* Search

A* Search combines the strengths of BFS and Greedy Best-First Search by using both path cost and a heuristic to guide the search. It expands the node with the lowest cost function (f(n) = g(n) + h(n)),

Goal 1: Starting from Initial State `[2, 8, 3, 1, 6, 4, 7, 5, 0]`

Goal 2: Starting from Initial State `[2, 8, 3, 0, 5, 4, 1, 7, 6]`

Goal 3: Starting from Initial State `[2, 8, 3, 1, 4, 0, 7, 6, 5]`

Goal 1: BFS Trace

Initial State:

```
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

Visited	Queue	s	Neighbor in Graph[s]	Neighbor	Output
[]	[]				
[283164750]	[[283164750]]	[283164750]	[[283104765], [283165470]]	Not goal	
[283104765]	[[283165470]]	[283104765]	[[203184765], [283140765]]	Not goal	[283164750]
[283165470]	[[203184765], [283140765]]	[283165470]	[[283615470], [283164057]]	Not goal	[283164750] [283104765]
[203184765]	[[283140765], [283615470], [283164057]]	[203184765]	[[023184765], [213804765]]	Not goal	[283164750] [283104765] [283165470]
[283140765]	[[283615470], [283164057], [023184765], [213804765]]	[283140765]	[[283104765], [283145760]]	Not goal	[283164750] [283104765] [283165470] [203184765]
[283615470]	[[283164057], [023184765], [213804765], [283145760]]	[283615470]	[[283165470], [283610457]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765]
[283164057]	[[023184765], [213804765],	[283164057]	[[283164507], [283164570]]	Not goal	[283164750] [283104765] [283165470]

	[283145760], [283610457]]				[203184765] [283140765] [283615470]
[023184765]	[[213804765], [283145760], [283610457], [283164507], [283164570]]	[023184765]	[[203184765], [123804765]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057]
[213804765]	[[283145760], [283610457], [283164507], [283164570], [123804765]]	[213804765]	[[203814765], [213840765]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057] [023184765]
[283145760]	[[283610457], [283164507], [283164570], [123804765], [203814765], [213840765]]	[283145760]	[[283145706]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057] [023184765] [213804765]
[283610457]	[[283164507], [283164570], [123804765], [203814765], [213840765], [283145706]]	[283610457]	[[283610475], [283614057]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057] [023184765] [213804765] [283145760]
[283164507]	[[283164570], [123804765], [203814765], [213840765], [283145706], [283610475], [283614057]]	[283164507]	[[283164570], [283164075]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057] [023184765] [213804765] [283145760] [283610457]

[283164570]	[[123804765], [203814765], [213840765], [283145706], [283610475], [283614057], [283164075]]	[283164570]	[[283164507], [283164750]]	Not goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057] [023184765] [213804765] [283145760] [283610457] [283164507]
[123804765]	[[203814765], [213840765], [283145706], [283610475], [283614057], [283164075]]	[123804765]	[[103824765]]	Goal	[283164750] [283104765] [283165470] [203184765] [283140765] [283615470] [283164057] [023184765] [213804765] [283145760] [283610457] [283164507] [283164570]

Goal 2: BFS Trace

Initial State:

```
[2, 8, 3]
[0, 5, 4]
[1, 7, 6]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

Goal 2: BFS Trace

Visited	Queue	s	Neighbor in Graph[s]	Neighbor	Output
[]	[]				
[283054175]	[[283054175]]	[283054175]	[[283405175], [283154075]]	Not goal	
[283405175]	[[283154075]]	[283405175]	[[283045175], [283415075]]	Not goal	[283054175]

[283154075]	[[283045175], [283415075]]	[283154075]	[[281534075], [283514075]]	Not goal	[283054175] [283405175]
[283045175]	[[283415075], [281534075], [283514075]]	[283045175]	[[203845175], [283405175]]	Not goal	[283054175] [283405175] [283154075]
[283415075]	[[281534075], [283514075], [203845175]]	[283415075]	[[283145075], [283415705]]	Not goal	[283054175] [283405175] [283154075] [283045175]
[281534075]	[[283514075], [203845175], [283145075], [283415705]]	[281534075]	[[281035475], [281534705]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075]
[283514075]	[[203845175], [283145075], [283415705], [281035475], [281534705]]	[283514075]	[[283510475], [283514705]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075]
[203845175]	[[283145075], [283415705], [281035475], [281534705], [283510475], [283514705]]	[203845175]	[[203405875], [203854715]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075]
[283145075]	[[283415705], [281035475], [281534705], [283510475], [283514705], [203405875], [203854715]]	[283145075]	[[283145705]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075] [203845175]
[283415705]	[[281035475], [281534705], [283510475], [283514705], [203405875], [203854715], [283145705]]	[283415705]	[[283415075], [283415075]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075] [203845175] [283145075]
[281035475]	[[281534705], [283510475], [283514705],	[281035475]	[[281035475]]	Not goal	[283054175] [283405175] [283154075]

	[203405875], [203854715], [283145705], [283415075], [283415075]]				[283045175] [283415075] [281534075] [283514075] [203845175] [283145075] [283415705]
[281534705]	[[283510475], [283514705], [203405875], [203854715], [283145705], [283415075], [283415075], [281035475]]	[281534705]	[[281534715]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075] [203845175] [283145075] [283415705] [281035475]
[283510475]	[[283514705], [203405875], [203854715], [283145705], [283415075], [283415075], [281035475], [281534715]]	[283510475]	[[283510475]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075] [203845175] [283145075] [283415705] [281035475] [281534705]
[283514705]	[[203405875], [203854715], [283145705], [283415075], [283415075], [281035475], [281534715], [283510475]]	[283514705]	[[283514715]]	Not goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075] [203845175] [283145075] [283415705] [281035475] [281534705] [283510475]
[203405875]	[[203854715], [283145705], [283415075], [283415075], [281035475], [281534715],	[203405875]	[[103824765]]	Goal	[283054175] [283405175] [283154075] [283045175] [283415075] [281534075] [283514075]

	[283510475], [283514715]]				[203845175] [283145075] [283415705] [281035475] [281534705] [283510475] [283514705]
### Goal 3: BFS Trace					

Initial State:

```
[2, 8, 3]
[1, 4, 0]
[7, 6, 5]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

Visited	Queue	s	Neighbor in Graph[s]	Neighbor	Output
[]	[]				
[283140765]	[[283140765]]	[283140765]	[[283104765], [283145760]]	Not goal	
[283104765]	[[283145760]]	[283104765]	[[203184765], [283140765]]	Not goal	[283140765]
[283145760]	[[203184765], [283140765]]	[283145760]	[[283145706], [283145760]]	Not goal	[283140765] [283104765]
[203184765]	[[283140765], [283145706]]	[203184765]	[[023184765], [213804765]]	Not goal	[283140765] [283104765] [283145760]
[283140765]	[[283145706], [023184765], [213804765]]	[283140765]	[[283104765], [283145760]]	Not goal	[283140765] [283104765] [283145760] [203184765]
[283145706]	[[023184765], [213804765], [283104765]]	[283145706]	[[283145076], [283145706]]	Not goal	[283140765] [283104765] [283145760] [203184765] [283140765]
[023184765]	[[213804765], [283104765], [283145076]]	[023184765]	[[023184765], [023184765]]	Not goal	[283140765] [283104765] [283145760]

					[203184765] [283140765] [283145706]
[213804765]	[[283104765], [283145076], [023184765]]	[213804765]	[[203814765], [213840765]]	Not goal	[283140765] [283104765] [283145760] [203184765] [283140765] [283145706] [023184765]
[283104765]	[[283145076], [023184765], [203814765], [213840765]]	[283104765]	[[283104765], [283104765]]	Not goal	[283140765] [283104765] [283145760] [203184765] [283140765] [283145706] [023184765] [213804765]
[283145076]	[[023184765], [203814765], [213840765], [283104765]]	[283145076]	[[283145706]]	Not goal	[283140765] [283104765] [283145760] [203184765] [283140765] [283145706] [023184765] [213804765] [283104765]
[023184765]	[[203814765], [213840765], [283104765], [283145706]]	[023184765]	[[023184765]]	Not goal	[283140765] [283104765] [283145760] [203184765] [283140765] [283145706] [023184765] [213804765] [283104765] [283145076]
[203814765]	[[213840765], [283104765], [283145706], [023184765]]	[203814765]	[[103824765]]	Goal	[283140765] [283104765] [283145760] [203184765] [283140765] [283145706] [023184765] [213804765] [283104765] [283145076] [023184765]

Goal 1: DFS Trace

Initial State:

```
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

DFS Trace:

Visited	Stack	s	Neighbor in Graph[s]	Neighbor	Outp
[]	[]				
[283164750]	[[283164750]]	[283164750]	[[283104765], [283165470]]	Not goal	
[283165470]	[[283104765]]	[283165470]	[[283615470], [283164057]]	Not goal	[283164
[283164057]	[[283104765], [283615470]]	[283164057]	[[283164507], [283164570]]	Not goal	[283164 [283165
[283164570]	[[283104765], [283615470], [283164507]]	[283164570]	[[283164507], [283164750]]	Not goal	[283164 [283165 [283164
[283164750]	[[283104765], [283615470], [283164507], [283164507]]	[283164750]	Goal	[283164750], [283165470], [283164057], [283164570]	

Goal 2: DFS Trace

Initial State:

```
[2, 8, 3]
[0, 5, 4]
[1, 7, 6]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

DFS Trace:

Visited	Stack	s	Neighbor in Graph[s]	Neighbor	Outp
[]	[]				
[283054175]	[[283054175]]	[283054175]	[[283405175], [283154075]]	Not goal	
[283154075]	[[283405175]]	[283154075]	[[281534075], [283514075]]	Not goal	[28305
[283514075]	[[283405175], [281534075]]	[283514075]	[[283510475], [283514705]]	Not goal	[28305 [28315
[283514705]	[[283405175], [281534075], [283510475]]	[283514705]	[[283514715]]	Not goal	[28305 [28315 [28351
[283514715]	[[283405175], [281534075], [283510475], [283514705]]	[283514715]	Goal	[283054175], [283154075], [283514075], [283514705]	

Goal 3: DFS Trace

Initial State:

```
[2, 8, 3]
[1, 4, 0]
[7, 6, 5]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

DFS Trace:

Visited	Stack	s	Neighbor in Graph[s]	Neighbor	Outp
[]	[]				
[283140765]	[[283140765]]	[283140765]	[[283104765], [283145760]]	Not goal	
[283145760]	[[283104765]]	[283145760]	[[283145706], [283145760]]	Not goal	[283140
[283145706]	[[283104765], [283145760]]	[283145706]	[[283145076], [283145706]]	Not goal	[283140 [283145
[283145076]	[[283104765], [283145760], [283145706]]	[283145076]	[[283145706]]	Not goal	[283140 [283145 [283145

[283145706]	[[283104765], [283145760], [283145706], [283145706]]	[283145706]	Goal	[283140765], [283145760], [283145706], [283145076]	

Greedy Best-First Search Trace

Initial State:

```
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

Heuristic: Manhattan Distance

Greedy Best-First Search Trace:

Visited	Priority Queue	Current Node	Neighbors	Goal Reached?
[]	[(h=6, 283164750)]	283164750	[(h=5, 283104765), (h=5, 283165470)]	No
[283164750]	[(h=5, 283104765), (h=5, 283165470)]	283104765	[(h=4, 203184765), (h=5, 283140765)]	No
[283164750, 283104765]	[(h=5, 283165470), (h=4, 203184765), (h=5, 283140765)]	203184765	[(h=3, 103824765), (h=4, 213804765)]	No
[283164750, 283104765, 203184765]	[(h=5, 283165470), (h=5, 283140765), (h=3, 103824765), (h=4, 213804765)]	103824765	[]	Yes

---

Uniform Cost Search (UCS) Trace

Initial State:

```
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

UCS Trace:

Visited	Priority Queue	Current Node	Neighbors	Goal Reached?
[]	[(cost=0, 283164750)]	283164750	[(cost=1, 283104765), (cost=1, 283165470)]	No
[283164750]	[(cost=1, 283104765), (cost=1, 283165470)]	283104765	[(cost=2, 203184765), (cost=2, 283140765)]	No
[283164750, 283104765]	[(cost=1, 283165470), (cost=2, 203184765), (cost=2, 283140765)]	203184765	[(cost=3, 103824765), (cost=3, 213804765)]	No
[283164750, 283104765, 203184765]	[(cost=1, 283165470), (cost=2, 283140765), (cost=3, 103824765), (cost=3, 213804765)]	103824765	[]	Yes

---

Hill-Climbing Trace

Initial State:

```
[2, 8, 3]
[1, 6,

 4]
[7, 5, 0]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

Heuristic: Manhattan Distance

Hill-Climbing Trace:

Visited	Current Node	Neighbors	Goal Reached?
[]	283164750	[283104765, 283165470]	No
[283164750]	283104765	[203184765, 283140765]	No
[283164750, 283104765]	203184765	[103824765, 213804765]	No
[283164750, 283104765, 203184765]	103824765	[]	Yes

A* Search Trace

Initial State:

```
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]
```

Goal State:

```
[1, 0, 3]
[8, 2, 4]
[7, 6, 5]
```

Heuristic: Manhattan Distance

A* Search Trace:

Visited	Priority Queue	Current Node	Neighbors	Goal Reached?
[]	[(f=6, 283164750)]	283164750	[(f=6, 283104765), (f=6, 283165470)]	No
[283164750]	[(f=6, 283104765), (f=6, 283165470)]	283104765	[(f=6, 203184765), (f=6, 283140765)]	No
[283164750, 283104765]	[(f=6, 283165470), (f=6, 283140765), (f=6, 203184765)]	203184765	[(f=6, 103824765), (f=7, 213804765)]	No

[283164750, 283104765, 203184765]	[(f=6, 283165470), (f=6, 283140765), (f=6, 103824765), (f=7, 213804765)]	103824765	[]	Yes

---

Comparison

- **Completeness:**
  - **BFS:** Complete, will always find a solution if one exists.
  - **DFS:** May not be complete, can get stuck in loops or go down a path with no solution.
  - **Greedy:** Not complete, can get stuck in loops or explore inefficient paths.
  - **UCS:** Complete, will always find a solution if one exists.
  - **Hill-Climbing:** Not complete, can get stuck in local maxima.
  - **A*:** Complete, will always find a solution if one exists when using an admissible heuristic.
- **Optimality:**
  - **BFS:** Optimal for unweighted graphs, finds the shortest path.
  - **DFS:** Not optimal, can find longer paths.
  - **Greedy:** Not optimal, does not guarantee the shortest path.
  - **UCS:** Optimal, finds the least-cost path.
  - **Hill-Climbing:** Not optimal, can get stuck in local maxima.
  - **A*:** Optimal when using an admissible heuristic, finds the shortest path.
- **Time Complexity:**
  - **BFS:** (O(V + E)), where (V) is the number of vertices and (E) is the number of edges.
  - **DFS:** (O(V + E)).
  - **Greedy:** (O(V + E)).
  - **UCS:** (O(V + E)).
  - **Hill-Climbing:** (O(V + E)).
  - **A*:** (O(V + E)).
- **Space Complexity:**
  - **BFS:** (O(V)), where (V) is the number of vertices.
  - **DFS:** (O(V)).
  - **Greedy:** (O(V)).
  - **UCS:** (O(V)).
  - **Hill-Climbing:** (O(V)).
  - **A*:** (O(V)).

---

https://drive.google.com/file/d/1n4MtVaK3kq3RgFimZU4k9XASsRZMWXSM/view?usp=sharing