Assignment – Artificial Intelligence (COMP3008L)

State Space Search Assignment

Team name	6droids
14208971	Tharkana D Kodagoda
14208893	Sahitha Nelanga H De Silva
14208910	H W Srimal Priyanga Fonseka
14209059	Dilina Namal Weerasinghe
14209074	P W Poorni Yasodara
14209759	Kavindu Yudeesha Lakshan Narathota

1	2	3
4	5	6
7	8	-

Start state

5	1	2
6	-	3
4	7	8

End state

Answer for Q 1.

Searching Algorithm **Breadth-First Search**
Heuristic Function *Not Applicable*

1.1. Depth = 10
1.2. Explored Number of States = 421
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10)

Searching Algorithm **Depth-First Search**
Heuristic Function *Not Applicable*

1.1. Depth = 1000 > Can't identify exactly, Browser crashes before reaching the goal state
1.2. Explored Number of States = 1000 > Can't identify exactly, Browser crashes before reaching the goal state
1.3. Path = [dx = in depth x]
6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > ……..

Searching Algorithm **Iterative Deepening Search**
Heuristic Function *Not Applicable*

1.1. Depth = 10
1.2. Explored Number of States = 454
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10)

Searching Algorithm **A* Search**

Heuristic Function *Euclidean Distance*

1.1. Depth = 10
1.2. Explored Number of States = 19
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10)

Heuristic Function *Manhattan Distance (City-Block distance)*

1.1. Depth = 10
1.2. Explored Number of States = 11
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10

Heuristic Function *Tiles Out-of-place*

1.1. Depth = 10
1.2. Explored Number of States = 11
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10

Searching Algorithm **Greedy Search**

Heuristic Function *Euclidean Distance*

1.1. Depth = 10
1.2. Explored Number of States = 11
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10)

Heuristic Function *Manhattan Distance (City-Block distance)*

1.1. Depth = 10
1.2. Explored Number of States = 11
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10)

Heuristic Function Tiles Out-of-place

1.1. Depth = 10
1.2. Explored Number of States = 12
1.3. Path = [dx = in depth x]
8(d1) > 7(d2) > 4(d3) > 5(d4) > 6(d5) > 3(d6) > 2(d7) > 1(d8) > 5(d9) > 6(d10)

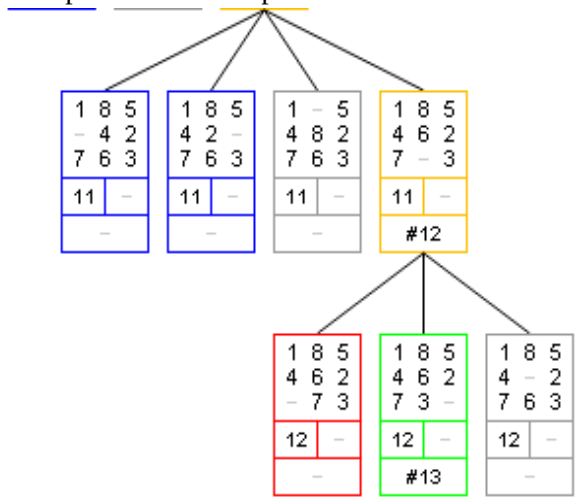
Answer for Q 2.

Same Final Solution = Yes.

Optimal solution found on Greedy Search Algorithm. When using Greedy search, both Euclidean Distance & Manhattan Distance (City-Block distance) Heuristic Functions reached the goal state with 11 explored states & only Tiles Out-of-place Heuristic Function took 12 explored states. Although it's better than every other Uninformed and Informed Search Algorithms. Also A* search with Manhattan distance heuristic function found optimal solution

Answer for Q 3.

On Depth-First Search path of cost 12



Therefore Propose End State is

Goal state:

1	8	5
4	6	2
7	3	–

Other algorithms perform on this end state as below:

Searching Algorithm **Breadth-First Search**
Heuristic Function *Not Applicable*

Depth = 12 > more than 12

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > 8(d10) > 6(d11) > 3(d12) > …..

To reach this end state, this algorithm takes more time than the Depth-First search algorithm.

Searching Algorithm **Iterative Deepening Search**
Heuristic Function *Not Applicable*

Depth = 12

Maximum Depth = 12

Explored Number of States = 12

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > 8(d10) > 6(d11)

Searching Algorithm **A* Search**

Heuristic Function *Euclidean Distance*

Depth = 12

Explored Number of States = 46

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > (d10) > (d11)

Heuristic Function *Manhattan Distance (City-Block distance)*

Depth = 12

Explored Number of States = 17

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > 8(d10) > 6(d11)

Heuristic Function *Tiles Out-of-place*

Depth = 12

Explored Number of States = 63

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > 8(d10) > 6(d11)

Searching Algorithm **Greedy Search**

Heuristic Function *Euclidean Distance*

Depth = 12

Explored Number of States = 20

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > 8(d10) > 6(d11)

Heuristic Function *Manhattan Distance (City-Block distance)*

Depth = 23

Explored Number of States = 48

Path = [dx = in depth x]

6(d1) > 5(d2) > 8(d3) > 6(d4) > 5(d5) > 3(d6) > …. > 5(d21) > 2(d22) > 3(d23)

Heuristic Function Tiles Out-of-place

Depth = 12

Explored Number of States = 228

Path = [dx = in depth x]

6(d1) > 3(d2) > 2(d3) > 5(d4) > 8(d5) > 6(d6) > 3(d7) > 2(d8) > 5(d9) > 8(d10) > 6(d11)

Answer for Q 4.

Answer for Q 5.

When we select Manhattan distance & Euclidean distance instead of Tiles Out of Place algorithm selected node is same as tiles out of place algorithm. in this both Explored Number of States become 4

Answer for Q 6.

6.1. Selected function : calculateEuclideanDistance()

@Description :
This is the method of Heuristic Function Euclidean distance. This function take two parameters and compare it to find the matching tiles in the puzzle between Start state and End state. then finally returns the distance between the tiles positions.

@Params :
a: tile values in puzzle initial state
b: tile values in puzzle goal state

@Return :
Distance between the tiles positions

6.2. Five difference Start & End States

A.

Start

Initial state:

1	2	3
4	5	6
7	–	8

End

Goal state:

1	2	3
4	–	8
7	5	6

B.

Start

Initial state:

1	2	3
4	–	6
7	5	8

End

Goal state:

2	1	3
4	5	8
7	–	6

C.

Start

Initial state:

1	–	3
4	2	6
7	5	8

End

Goal state:

1	2	3
4	–	7
5	8	6

D.

Start

Initial state:

–	1	3
4	2	6
7	5	8

End

Goal state:

1	2	3
4	6	7
5	8	–

E.

Start

Initial state:

4	1	3
–	2	6
7	5	8

End

Goal state:

1	2	–
4	6	7
5	3	8

6.3. Answers on running standard Greedy Best search and A* algorithms, using the correct Euclidean distance.

A. Running on first pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 16 > More than 16

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 3(d15) > ….

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 35 > More than 35

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 5(d15) > ….

B. Running on second pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 15 > More than 15

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 1(d15) > ….

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 37

Explored Number of States = 400

Path = [dx = in depth x]

5(d1) > 8(d2) > 6(d3) > …. > 5(d37)

C. Running on third pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 17 > More than 17

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 7(d15) > ….

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 40 > More than 40

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 2(d40) > ….

D. Running on fourth pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 16

Explored Number of States = 450

Path = [dx = in depth x]

1(d1) > 2(d2) > 6(d3) > …. > 6(d13) > 4(d14) > 5(d15) > 8(d16)

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 28

Explored Number of States = 109

Path = [dx = in depth x]

1(d1) > 2(d2) > 5(d3) > …. > 6(d25) > 4(d26) > 5(d27) > 8(d28)

E. Running on fifth pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 19

Explored Number of States = 857

Path = [dx = in depth x]

4(d1) > 1(d2) > 3(d3) > …. > 2(d16) > 7(d17) > 6(d18) > 2(d19)

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 66 > More than 66

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

4(d1) > …. > 7(d66) > ….

6.4. Answers on running standard Greedy Best search and A* algorithms, using the modified distance algorithm.

A. Running on first pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 18 > More than 18

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 7(d18) > ….

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 40 > More than 40

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 8(d40) > ….

B. Running on second pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 19 > More than 19

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 6(d19) > ….

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 42

Explored Number of States = 541

Path = [dx = in depth x]

5(d1) > 8(d2) > 6(d3) > …. > 4(d541)

C. Running on third pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 21 > More than 21

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 8(d21) > ….

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 480 > More than 48

Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state

Path = [dx = in depth x]

5(d1) > …. > 6(d48) > ….

D. Running on fourth pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 24

Explored Number of States = 620

Path = [dx = in depth x]

1(d1) > 2(d2) > 6(d3) > …. > 7(d20) > 4(d21) > 3(d22) > 8(d23)

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 34

Explored Number of States = 163

Path = [dx = in depth x]

1(d1) > 2(d2) > 6(d3) > …. > 6(d159) > 4(d160) > 3(d161) > 8(d162)

E. Running on fifth pair

Searching Algorithm: A* Search

Heuristic Function: Euclidean Distance

Path length for pair = 27

Explored Number of States = 1027

Path = [dx = in depth x]

4(d1) > 1(d2) > 3(d3) > …. > 2(d1024) > 7(d1025) > 6(d1026) > 2(d1027)

Searching Algorithm: Greedy Best search

Heuristic Function: Euclidean Distance

Path length for pair = 72 > More than 72
Explored Number of States = 1000 > More than 1000 Can't identify exactly, browser crashes before reaching the goal state
Path = [dx = in depth x]
4(d1) > …. > 3(d72) > ….

6.5. A consistent (or monotone) heuristic function is a function that estimates the distance of a given state to a goal state, and that is always at most equal to the estimated distance from any neighboring vertex plus the step cost of reaching that neighbor.

A heuristic function is said to be admissible if it never overestimates the cost of reaching the goal

When we going through the above results and compare original function and modified function, we can decide our function is admissible and consistent.