COMS 4733 Homework 2

Jaisel Singh

October 4, 2025

1 Problem 1: Discrete Search Algorithms

1.1 Search Algorithm Comparison

Notation:

- b = branching factor (average number of successors per node)
- d = depth of the shallowest goal node
- m = maximum depth of the search tree

Note: All algorithms use graph search (visited set) on finite state space.

Algorithm	**Complete?**	**Cost-Optimal?**	**Time Complexity**	**Space Complexity**
DFS	Yes	No	$O(b^m)$	$O(bm)$
BFS	Yes	No	$O(b^d)$	$O(b^d)$
Dijkstra	Yes	Yes	$O(b^d)$	$O(b^d)$
A* (admissible h)	Yes	Yes	$O(b^d)$	$O(b^d)$

Table 1: Comparison of search algorithms

1.1.1 Justification for Selected Entries

Dijkstra's Algorithm - Time Complexity: $O(b^d)$

Dijkstra's algorithm explores nodes in order of increasing cost from the start node. In the worst case, it must explore all nodes up to the depth d of the optimal solution before guaranteeing it has found the cheapest path.

Consider a tree structure where:

- At depth 0: 1 node (start)

- At depth 1: b nodes
- At depth 2: b^2 nodes
- $\vdots$
- At depth d: b^d nodes

The total number of nodes explored is:

$$1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

Each node is processed once (removed from the priority queue), and for each of its neighbors, we may perform a priority queue operation. With an efficient priority queue implementation (binary heap), these operations take $O(\log n)$ time where n is the number of nodes in the queue.

Therefore, the overall time complexity is $O(b^d)$ in the context of search algorithms.

Dijkstra's Algorithm - Space Complexity: $O(b^d)$

Dijkstra's algorithm uses a priority queue to store nodes that have been discovered but not yet fully explored. In the worst case, before reaching the goal at depth d, the priority queue may contain all nodes at the frontier of the search.

Since Dijkstra's explores in a breadth-first manner (ordered by cost rather than depth), it must maintain all nodes at the current cost level. In a tree with branching factor b and goal at depth d, the maximum number of nodes in the priority queue at once is proportional to b^d (the number of nodes at depth d).

Additionally, with graph search (using a visited set to avoid cycles), we must store all visited nodes, which is also $O(b^d)$ in the worst case.

Therefore, the space complexity is $O(b^d)$.

1.2 Search Tree Expansion

1.2.1 Task 1: BFS Expansion Order

Using lexicographic ordering (smallest x first, then smallest y) for tie-breaking, the BFS expansion order is:

(0,3) 5	**(1,3)** 8	**(2,3)** 12	**G (3,3)** 14
(0,2) 2	**X (1,2)** -	**(2,2)** 9	**(3,2)** 13
S (0,1) 0	**(1,1)** 3	**(2,1)** 6	**(3,1)** 10
(0,0) 1	**(1,0)** 4	**(2,0)** 7	**(3,0)** 11

Complete expansion sequence:

$0 : S(0,1) \to 1 : (0,0) \to 2 : (0,2) \to 3 : (1,1) \to 4 : (1,0) \to 5 : (0,3) \to 6 : (2,1)$
$\to 7 : (2,0) \to 8 : (1,3) \to 9 : (2,2) \to 10 : (3,1) \to 11 : (3,0) \to 12 : (2,3)$
$\to 13 : (3,2) \to 14 : G(3,3)$

1.2.2 Task 2: Dijkstra's Algorithm

Dijkstra's algorithm expands nodes in the **same order** as BFS for this problem.

Explanation:

BFS and Dijkstra's expand nodes identically because all edges have uniform cost (cost = 1). Dijkstra's algorithm uses a priority queue to explore nodes in order of increasing cumulative cost from the start. Since each move costs 1, a node at distance d from the start has total cost d. This matches BFS exactly, which explores nodes layer-by-layer by distance.

When multiple nodes have the same priority (same cost in Dijkstra's, same depth in BFS), both algorithms apply the lexicographic tie-breaking rule. Since cost equals depth in this uniform-cost scenario, both algorithms break ties identically and produce the same expansion order.

Therefore, Dijkstra's expansion order is identical to the BFS grid shown above.

1.2.3 Task 3: A* with Manhattan Distance Heuristic

Using the Manhattan distance heuristic $h(x, y) = |3 - x| + |3 - y|$ and lexicographic tie-breaking, A* expands nodes in the following order:

Order	**Node**	$g(n)$	$h(n)$	$f(n) = g(n) + h(n)$
0	S (0,1)	0	5	5
1	(0,2)	1	4	5
2	(0,3)	2	3	5
3	(1,1)	1	4	5
4	(1,3)	3	2	5
5	(2,1)	2	3	5
6	(2,2)	3	2	5
7	(2,3)	4	1	5
8	(3,1)	3	2	5
9	(3,2)	4	1	5
10	G (3,3)	5	0	5

Grid with expansion order:

(0,3) 2	**(1,3)** 4	**(2,3)** 7	**G (3,3)** 10
(0,2) 1	**X (1,2)** -	**(2,2)** 6	**(3,2)** 9
S (0,1) 0	**(1,1)** 3	**(2,1)** 5	**(3,1)** 8
(0,0) -	**(1,0)** -	**(2,0)** -	**(3,0)** -

Observations/Notes

- All expanded nodes have $f(n) = 5$, which equals the optimal path cost
- Nodes in row $y = 0$ (bottom row) such as (0,0), (1,0), (2,0), and (3,0) are generated but never expanded because they all have $f = 7 > 5$
- The Manhattan distance heuristic is admissible (never overestimates) and consistent in this 4-connected grid with unit costs
- Because h is admissible and the grid uses 4-connected movement with unit costs, every node on an optimal path satisfies $g + h = 5$ (constant). Therefore, the expansion order among these nodes is determined purely by the lexicographic tie-breaking rule
- A* reaches the goal at step 10, having expanded only 11 nodes (including the start), compared to 15 nodes for BFS

1.3 Heuristic Admissibility

1.3.1 Is Euclidean distance admissible?

Answer: Yes, the Euclidean distance heuristic $h(n) = \sqrt{(x_g - x)^2 + (y_g - y)^2}$ is admissible for this cost model.

Justification:

A heuristic is admissible if it never overestimates the true optimal cost to reach the goal. The Euclidean distance represents the straight-line distance between two points, which is the shortest possible distance in Euclidean space.

With 8-connected movement where diagonal moves cost $\sqrt{2}$ and cardinal moves cost 1, we can analyze the relationship between the heuristic and actual cost:

- Consider moving from $(0, 0)$ to $(3, 3)$:
  - Euclidean distance: $h = \sqrt{(3-0)^2 + (3-0)^2} = \sqrt{18} = 3\sqrt{2} \approx 4.243$
  - Optimal path: Three diagonal moves $(0, 0) \to (1, 1) \to (2, 2) \to (3, 3)$
  - Actual cost: $3\sqrt{2} \approx 4.243$
- In general, for any two points, the Euclidean distance equals the cost of moving in a straight diagonal line (when possible), which is the optimal path in an obstacle-free 8-connected grid
- Since we cannot move "more directly" than the straight line, and diagonal moves have cost exactly $\sqrt{2}$, the Euclidean distance never overestimates the true cost

Therefore, the Euclidean distance heuristic is admissible for this cost model.

1.3.2 Propose a different admissible heuristic

Answer: The Chebyshev distance (L^∞ norm) is an admissible heuristic for this cost model:

$$h(n) = \max(|x_g - x|, |y_g - y|)$$

Justification:

The Chebyshev distance measures the maximum absolute difference in either coordinate, which corresponds to the minimum number of moves required to reach the goal when diagonal moves are allowed.

To show admissibility, we verify that this heuristic never overestimates:

- The Chebyshev distance counts the number of diagonal or cardinal moves needed along the longer dimension
- Example: From $(0, 0)$ to $(3, 3)$:
  - Chebyshev distance: $h = \max(|3 - 0|, |3 - 0|) = 3$
  - Optimal path: Three diagonal moves with cost $3\sqrt{2} \approx 4.24$
  - Since $3 < 4.24$, the heuristic does not overestimate
- More generally, consider moving from (x, y) to (x_g, y_g):
  - Let $\Delta x = |x_g - x|$ and $\Delta y = |y_g - y|$
  - Assume without loss of generality that $\Delta x \geq \Delta y$
  - Optimal strategy: Move diagonally Δy times, then cardinally $(\Delta x - \Delta y)$ times
  - Actual cost: $\Delta y \cdot \sqrt{2} + (\Delta x - \Delta y) \cdot 1 = \Delta x + \Delta y(\sqrt{2} - 1)$
  - Chebyshev distance: $h = \Delta x$
  - Since $\sqrt{2} - 1 \approx 0.414 > 0$, we have $h = \Delta x < \Delta x + \Delta y(\sqrt{2} - 1)$ $=$ actual cost

Therefore, the Chebyshev distance never overestimates the true cost and is admissible.

Alternative: Any scaled Manhattan distance of the form $h(n) = c \cdot (|x_g - x| + |y_g - y|)$ where $c \leq \frac{\sqrt{2}}{2} \approx 0.707$ is also admissible, though it provides a looser bound than Chebyshev distance.