IAJ - Pathfinding

Grupo: 2

Aluno(s): João Azevedo (96877) - João Costa (99985) - Óscar Ferrão (113182)

Heuristics and Tie Breaking

To begin our analysis, we compared the performance of different heuristic functions using a basic implementation of the A* algorithm. For consistency, in all tests we fixed the number of nodes processed per frame at 30 and set the tie-breaking weight to 0.5. The results are based on two separate paths: the first path and second path, both of which were executed three times to generate an average time for each heuristic.

	Techniques	Max Open Nodes	Total Nodes	Total Time
Path 1	Zero Heuristic	43	9310	6.6953
	Euclidean Distance	57	7082	4.6921
	Manhattan Distance	80	6697	4.2812
	Euclidean Dist + TieBreakingWeight	63	6437	4.1591
	Manhattan + TieBreakingWeight	81	5793	3.8123

Table 1: Heuristics Comparison for path 1

	Techniques	Max Open Nodes	Total Nodes	Total Time
Path 2	Zero Heuristic	58	16710	8.1070
	Euclidean Distance	121	11816	5.6712
	Manhattan Distance	176	8400	4.8796
	Euclidean Dist + TieBreakingWeight	169	9215	4.3430
	Manhattan Dist + TieBreakingWeight	170	4642	3.5472

Table 2: Heuristics Comparison for path 2

IAJ - Pathfinding

Grupo: 2

Aluno(s): João Azevedo (96877) - João Costa (99985) - Óscar Ferrão (113182)

As shown the Zero Heuristic performed the worst, as it lacks guidance, resulting in excessive node exploration and the longest processing times. The Euclidean Distance heuristic showed significant improvement by focusing the search more effectively, though still exploring a moderate number of nodes. Manhattan Distance further optimized the search by aggressively guiding it along grid lines, reducing both the total nodes processed and execution time. Introducing tie-breaking improved both the Euclidean and Manhattan heuristics, with Manhattan Distance + Tie Breaking performing the best overall, minimizing node exploration and achieving the fastest times. This is especially effective in my grid, which has many closed spaces where the grid-aligned approach of Manhattan Distance excels by quickly navigating tight corridors and reducing unnecessary exploration.

DataStructures

We started by analyzing the impact of different data structures on the performance of the A* algorithm. Our first implementation used an Unordered List for both the Open and Closed sets. This yielded an execution time of 3.44 seconds, with a per-node time of 40 ms. The Open set reached a maximum size of 170 nodes, while the Closed list had 1,349 nodes at its peak.

Next, we tried a combination of a Dictionary for the Closed set and an Unordered List for the Open set. This improved performance, reducing the execution time to 3.08 seconds and the per-node time to 7.05 ms. The maximum size of the Open set remained 170, while the Closed dictionary held 1,349 nodes.

Finally, we implemented a NodePriorityHeap for the Open set and a Dictionary for the Closed set. Surprisingly, this combination increased the execution time to 3.8 seconds with a per-node time of 8 ms. The Open set grew to 217 nodes, and the Closed dictionary expanded to 2,076 nodes.

The results suggest that using a dictionary for the Closed set significantly improves performance, reducing the time per node. However, while priority heaps are typically more efficient for managing priority queues (as in the Open set), in this case, it led to a slower overall execution time compared to the unordered list. This could be due to the additional overhead of maintaining the heap structure, especially when the Open set size remains relatively small. Further investigation into tuning or alternative heap implementations might be needed to fully optimize performance in this context.

IAJ - Pathfinding

Grupo: 2

Aluno(s): João Azevedo (96877) - João Costa (99985) - Óscar Ferrão (113182)

Algorithm Performance

NodesPerFrame = 30; TieBreakingWeight = 0.5; heuristic = Euclidean distance

Paths	Techniques	Max Open Nodes	Total Nodes	Total Time
Path 1	A*	57	7095	4.3452
	A* + Tiebreak	63	6436	3.9768
	NodeArray A*	63	6438	3.9015
	Gateway	63	6441	4.42
Path 2	A*	121	11843	7.1333
	A* + Tiebreak	169	9215	5.6824
	NodeArray A*	169	9217	5.4562
	Gateway	169	9215	6.5326
Path 3	A*	85	9396	5.6800
	A* + Tiebreak	122	6030	3.5582
	NodeArray A*	122	6035	3.5125
	Gateway	122	6033	4.2205

Table 3: Comparison of all algorithm performances

A* consistently worked well on all paths tested, but it tended to search through more nodes and took longer to execute, especially in Path 2. Even though it is a reliable algorithm, its significant node exploration can result in decreased efficiency when dealing with longer or more complicated routes. The modified version of A* for tiebreaking demonstrated continual enhancements in reducing the total nodes explored and in decreasing the overall time needed. Adding a weight for breaking ties improved the search process and led to slightly quicker performance on all routes.

The NodeArray version of A* showed similar performance to A* with tiebreaking, displaying slight yet noticeable enhancements, especially for longer routes. Although the improvements were small, they indicate that NodeArray A* could be a suitable choice for situations where a combination of speed and effectiveness is needed.

The Gateway algorithm also demonstrated promising results, particularly for paths with fewer obstacles or more straightforward layouts. In such scenarios, Gateway proved to be efficient, potentially outperforming the other techniques by minimizing unnecessary node exploration. Its suitability for routes that are more direct or less complex makes it an excellent candidate for tasks requiring both speed and lower computational load.

Ultimately, integrating tie-breaking weights into A* aids in minimizing redundant node examination, enhancing its efficiency for intricate routes. The A* algorithm with NodeArray provides slight enhancements compared to regular A* and is a feasible choice for improving performance. Yet, the Gateway algorithm, particularly in less complicated, more direct paths, offers a highly efficient alternative and could be the best solution for large or time-sensitive pathfinding tasks.

Optional

For the optional optimization section of our report, we implemented BidirectionalAStarPathfinding. This method enhances the efficiency of the standard A* algorithm by conducting simultaneous searches from both the start and goal nodes. The searches continue until they meet at a common node, where both closed lists intersect, allowing for a more efficient pathfinding process.

	Techniques	Max Open Nodes	Total Nodes	Total Time
Path 1	BiDirectionalAStarPathfinding	73	2855	1.3673
Path 2	BiDirectionalAStarPathfinding	123	2755	1.2731

Table 4: BiDirectionalAStarPathfinding performance

As shown in the table, the number of **max open nodes**, **total nodes searched**, and **total time** are significantly improved compared to traditional A* and other algorithms. This is due to the reduction in search space, leading to improved time complexity and better memory efficiency.