1 Program Overview

This B-15 puzzle solving program utilised two node instances: an initial node and a current-state node. Upon each increase of the threshold, the state node was reset to the initial node.
Utilising only two node instances in memory provided a relatively low memory usage by the program (approximately 64 kilobytes).
Prior to testing the provided puzzle configurations, the program relied on exploring all possible actions from each node. However, this implementation was observed to take over 10 minutes for even the medium-complexity puzzle of 2.puzzle:

$$\begin{bmatrix} 13 & 5 & 4 & 10 \\ 9 & 12 & 8 & 14 \\ 2 & 3 & 7 & 1 \\ 0 & 15 & 11 & 6 \end{bmatrix}$$

Due to processing limitations, it seemed that testing the more complex puzzles was out of the question.
To fix this issue, the logic was corrected by preventing the program from exploring nodes which it had already explored for a given threshold. This sped up the program by many orders of magnitude, with 2.puzzle taking only ≈ 0.3 seconds.

2 Heuristic Improvements

The algorithm the B-15 puzzle solution was observed with two different heuristic functions:

1. Manhattan Distance
2. Manhattan Distance with Last Moves Optimisation

Overall, the heuristic optimisation outperformed the basic Manhattan heuristic when it came to time performance in the more complex examples (puzzles 1, 3, 88). The optimisation also consistently decreased the number of expanded nodes required to obtain a solution. However, this came at the cost of roughly halving the expansion speed (*nodes/sec*) across all test instances.
Although this implementation did not consistently improve performance across all test cases, its decrease in the execution-time of puzzle 88 (arguably the most complex puzzle, expecting roughly $6003M$ expanded nodes) and its heavy reduction in the number of expanded nodes ($1090M$) suggests that it is an efficient optimisation.
This optimisation was informed by the paper 'Finding Optimal Solutions to the Twenty-Four Puzzle' by Richard E. Korf and Larry A. Taylor (1996).

3 Results

3.1 No Optimisation

ID	h(s_0)	Thresholds	Optimal Moves	Expanded Nodes	Expanded/sec	Time (secs)
1	41	41 43 45 47 49 51 53 55 57	57	206,131,932	16,813,273	12.26
2	43	43 45 47 49 51 53 55	55	4,636,976	16,416,979	0.28
3	41	41 43 45 47 49 51 53 55 57 59	59	229,658,354	15,334,415	14.98
4	42	42 44 46 48 50 52 54 56	56	41,689,053	16,457,564	2.53
14	41	41 43 45 47 49 51 53 55 57 59 61	61	1,090,399,330	17,037,135	64.00
88	43	43 45 47 49 51 53 55 57 59 61 63 65	65	6,610,107,370	16,972,586	389.46

3.2 Last Moves Optimisation

ID	h(s_0)	Thresholds	Optimal Moves	Expanded Nodes	Expanded/sec	Time (secs)
1	41	41 43 45 47 49 51 53 55 57	57	77,138,658	8,313,906	9.28
2	43	43 47 49 51 53 55	55	2,937,285	8,724,237	0.34
3	41	41 45 47 49 51 53 55 57 59	59	95,347,992	8,742,682	10.91
4	42	42 44 46 48 50 52 54 56	56	33,152,534	9,238,433	3.59
14	41	41 43 45 47 49 51 53 55 57 59 61	61	1,034,575,816	9,437,904	109.62
88	43	43 47 49 51 53 55 57 59 61 63 65	65	1,879,404,554	8,404,575	223.62