Artificial Intelligence (CS7IS2)

Assignment 1

Catalin Gheorghiu (22305257)

1 Methodology

1.1 Maze Characteristics

I make use of the *mazelib* Python package (`https://github.com/john-science/mazelib`) to generate all the mazes solved in this assignment. My preferred generation algorithm is Cellular Automaton, as the imperfect mazes it creates often have loops, open rooms and overall multiple ways of reaching the exit. This complexity is a good challenge for the solving algorithms, offering more insight into the different search strategies and recommended paths. The generative method normally has 4 parameters, but since I am only investigating square mazes for convenience, only 3 relevant ones remain: *SIZE* (denoting both length and width), *complexity* and *density.* I apply my solving algorithms to three different values of the *SIZE* parameter, from small to large – 5, 10, and 25 – creating grid-like mazes whose dimensions are 11x11, 25x25, and 51x51 tiles respectively. The other two parameters are fixed to balance maze difficulty and the tendency to create wall-free areas. Figure 1 depicts a seeded large maze; settings are specified in the caption. The walls are black, available tiles for pathing are white, and the automatically generated entrance and exit are green and red, respectively.

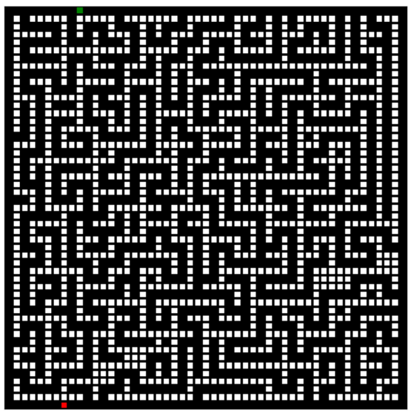

Figure 1: Large Cellular Automaton maze (seed 42, width and height 25, complexity 1, density 0.5).

1.2 Comparisons

When comparing algorithms, I will first show (and discuss if interesting) a visualisation of each algorithm's solution to this specific maze, with different coloring between search algorithms and Markov Decision Processes (MDPs) as seen in figure 2. Search algorithms such as the Breadth-First Search (BFS) shown in the left panel depict the solution path in purple and the tiles – or nodes, in graph terminology – visited in cyan. MDP-based algorithms such as the Policy Iteration (PI) shown in the right panel color each node according to a heat map of their value in the final iteration; the brighter and closer to cream the node, the higher the value. Intuitively, indigo represents the lowest values. The solution path is shown in the highest-value nuance, overriding the tiles original colors to allow a more clear solution contrasting to search algorithms.

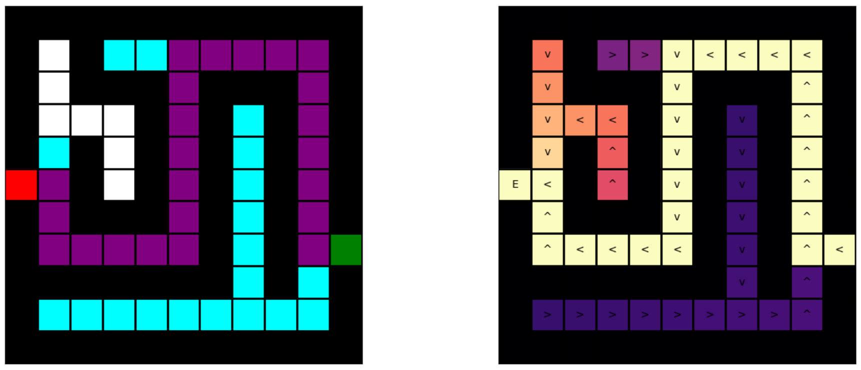

Figure 2: Sample solutions of a search algorithm (here BFS) and an MDP (here PI) to a small maze.

The qualitative comparison is followed by a quantitative one, where for each of the three sizes, I randomly generate 10 mazes and solve each with the five required algorithms: BFS, Depth-First Search (DFS), A*, PI, and Value Iteration (VI). Performance metrics are collected for each algorithm run to be aggregated in tables of means and standard deviations in order to reduce the influence of randomness. These metrics differ between comparison groups, and I have chosen them with time efficiency, computational complexity and efficacy in achieving the objective in mind.

- Search-Search: wall clock time (in seconds), number of visited nodes, path length.
- MDP-MDP: wall clock time, iterations.
- Search-MDP: wall clock time, path length.

In the inter-MDP comparison, I will also investigate the impact of changing the noise, decay, and the update tolerance by looking at line graphs of iteration count over a reasonable range for each metric. Combinations of high-low noise and decay are analysed first on a fixed, low tolerance, then the impact of tolerance is assessed alone under the best settings of noise and decay. These favourable values will also be used when comparing the performance of MDPs to search algorithms. Qualitative analysis of the MDP algorithms will be integrated in this investigation.

2 Search - Search Comparison

2.1 Qualitative Inspection

Each algorithm in figure 3 shows distinct characteristics. The different amount of visited cells is immediately obvious. BFS visits most of the nodes in the maze, especially towards the entrance in the north. DFS and A* look at significantly fewer nodes, but whether they differ in this regard remains for the quantitative inspection to determine. It is noteworthy, however, that A* usually visits nodes between a given position on the path and the exit, which reflects in its west bias as both the entrance and exit are well towards the western edge. As I use Manhattan distance as the heuristic for forward costs, this intuitively makes sense and makes it easy to identify A*. In the grid world with no diagonal movement, Manhattan distance is a good integer alternative to Euclidean distance – respecting the triangle rule – and is guaranteed to be at most as large as the optimal part, making the heuristic consistent. Turning to the solution path, BFS is guaranteed to find the shortest solution, and since A* mirrors it, I gain confidence in the efficacy of the latter. DFS finds a more roundabout route, occasionally exploring entire rooms such as the two 3x3s by the exit. The price for inspecting fewer options seems to be a worse path.

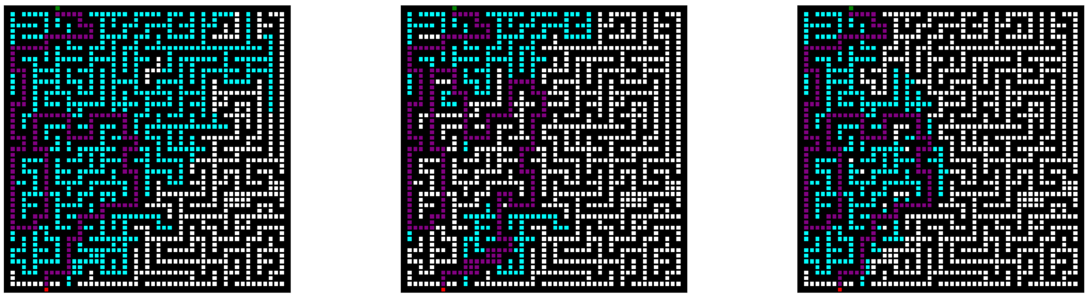
Figure 3: Large maze solution of BFS (left), DFS (middle) and A* (right).

2.2 Quantitative Inspection

Table 1 contains the mean values of running time, number of off-path nodes visited and path length over 10 random mazes of each size. Standard deviations are provided in brackets, except for running times; these are very low even for the largest maze size and tell nothing that the other standard deviations do not already, therefore I have omitted them.

Table 1: Average performances of the search algorithms over different metrics and maze sizes. Standard deviations in brackets.

		BFS	DFS	A*
SIZE = 5	Running Time	$< 10^{-5}$	$< 10^{-4}$	$< 10^{-4}$
	Off-Path Visits	18.9 (5.12)	19.9 (10.43)	12.7 (6.75)
	Path Length	16 (3.82)	16.8 (4.14)	16 (3.82)
SIZE = 12	Running Time	$1.2 * 10^{-3}$	$3.4 * 10^{-3}$	$1.1 * 10^{-3}$
	Off-Path Visits	146.9 (58.06)	115.2 (78.02)	83 (50.94)
	Path Length	52.8 (19.71)	53.6 (19.65)	52.8 (19.71)
SIZE = 25	Running Time	$6.6 * 10^{-3}$	$3 * 10^{-2}$	$1.1 * 10^{-2}$
	Off-Path Visits	920.7 (150.42)	619.8 (330.55)	582.4 (216.41)
	Path Length	128.6 (27.5)	159 (38.84)	129 (28.18)

In terms of running time, I believe the values are too low across sizes and algorithms to tell a significant story. All search algorithms seem very efficient, but in relative terms, it may be noteworthy that a fivefold increase in maze surface (from SIZE=12 to SIZE=25) caused a proportional increase in the running time of BFS, compared to the tenfold increase in the running times of DFS and A*. The recursions in DFS and queue sortings in A* are likely at fault for this difference.

Looking at off-path visits, small mazes paint a diffuse picture that becomes sharper with larger mazes. While sizes 12 and 25 clearly establish BFS as the widest-spreading algorithm, analysing the smallest maze size would indicate DFS to be more expansive. The standard deviation of DFS elucidates the mystery, as it is consistently larger than the deviations of the other two algorithms' off-path visits. This suggests that even though DFS sometimes performs visibly better than BFS, its high volatility makes it highly maze-dependent – some mazes may take DFS significantly longer to solve. Neither BFS nor DFS seem to match the performance of A* however, as this algorithm consistently achieves the lowest number of off-path visits with a comparable standard deviation to the lower volatility of BFS.

Finally, looking at path length, DFS performs unequivocally worse than the other two. This is hard to spot for small and medium mazes where the performance is only somewhat worse, but the average path lengths in the large maze is a whole standard deviation away from the averages of the other algorithms. The remaining two algorithms are hard to distinguish by this metric even at the largest size, as A* seems to almost always find a shortest path of the same length as the theoretical optimal found by BFS.

In conclusion, A* shares the greatest strength of BFS (optimal length) while looking at about 40% fewer nodes thanks to the forward cost heuristic. This makes it preferable to BFS, especially is there are costs associated with visiting nodes. And while DFS is the most inconsistent of the three, it may outperform the other two on the right kind of maze, especially if it is small or medium in size.

3 MDP - MDP Comparison

3.1 Hyperparameter Analysis

All following qualitative assessments are based on the large, seeded maze shown in figure 1. The end of this subsection shows the value heat map for the maze, bound to be identical between policies as the best hyperparameter setting ought to get the two to converge to the same optimal solution.

Tolerance

Figure 4 shows just how differently the two algorithms are affected by the update tolerance. For policy iteration, this parameter decides when policy evaluation has stablisised, as I use a dynamic programming approach to solve the Bellman equations. In other words, each iteration begins with repeated policy evaluation until the largest value modification between consecutive evaluations falls below this update tolerance. For value iteration, it decides when the algorithm converged altogether. The following discussion is carried out in terms of final iteration count.

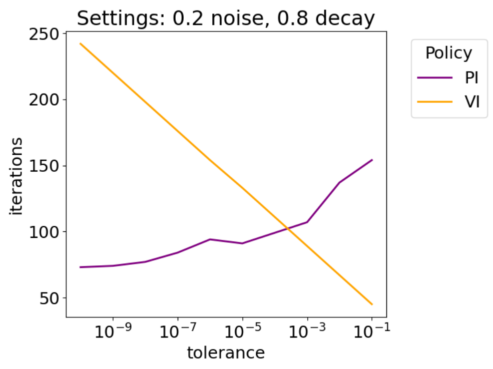

Figure 4: Impact of update tolerance on the number of iterations performed by the different policies; other hyperparameters fixed.

Value iteration scales linearly and inversely with the update tolerance ϵ. High values of *epsilon* cause value iteration to terminate very quickly since the tendency to convergence applies downward pressure on value changes, whereas low values of *epsilon* make it so that even one node that continually changes value can force further iterations. A version of the algorithm where average value change is the convergence criterion may prevent this phenomenon.

Policy iteration scales up with tolerance, but non-linearly. It does, however, change at a much slower rate compared to value iteration, implying lower volatility. The reason why higher tolerance leads to more updates is likely that policies are evaluated badly due to the iterative policy evaluation terminating early, and the policy improvements thus make little progress per iteration towards convergence.

Noise

Figure 5 shows how noise and decay interact to influence the number of iterations the different MDP algorithms need to converge under a fixed tolerance. It is immediately obvious that VI is insensitive to noise, however the impact of decay appears much larger on VI iterations that it does on PI, once again emphasising the volatility of the former. As lower decay diminishes the influence that the exit's positive incentive has on policy and pathing, it is intuitive that values change by less in an iteration. As a result, VI stabilises much faster under 0.5 decay than under 0.9.

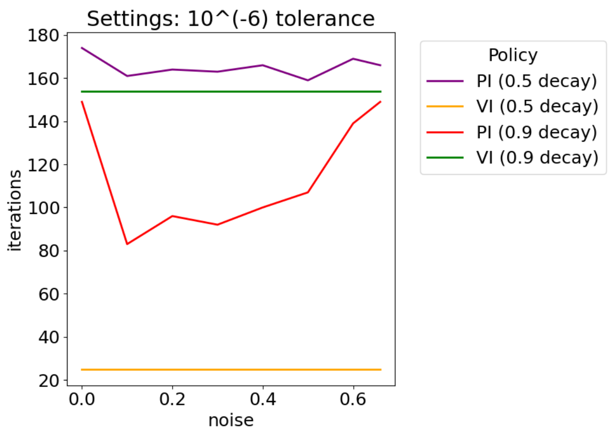

Figure 5: Impact of noise under different decay settings on the number of iterations performed by the different policies; other hyperparameters fixed.

Policy iteration, on the other hand, sees the opposite effect, mirroring the discussion of noise. Lower decay is less volatile but leads to more iterations because of bad policy updates. The sharp decrease in iterations when noise changes from 0 to 0.1, however, is quite interesting. It makes sense for the iteration count to be low under low noise and high when policy becomes an unreliable indicator of the route an agent would take; and yet, determinism greatly increases the number of iterations, for both levels of decay. It may be that the scarcity of the transition matrix under determinism no longer includes good-value alternative paths when computing the utilities of various nodes, reducing the number of available policies in the multi-solution setting of Cellular Automaton and thus requiring more iterations to converge on the optimal policies that remain.

Decay

Finally, figure 7 clearly shows that decay has a much greater impact on VI than it does on PI, while also emphasising the lack of impact noise has on VI. Following the previously discussed intuition, high decay increases VI iterations due to the greater influence exerted by the exit node that ends up affecting more nodes and ultimately creating more value differences. PI shows a clear trend of achieving lower iterations as decay increases and noise decreases; more interestingly, decay seems to bring the iteration count down faster than noise, especially towards higher decay. This mirrors the observations of the *Noise* discussion.

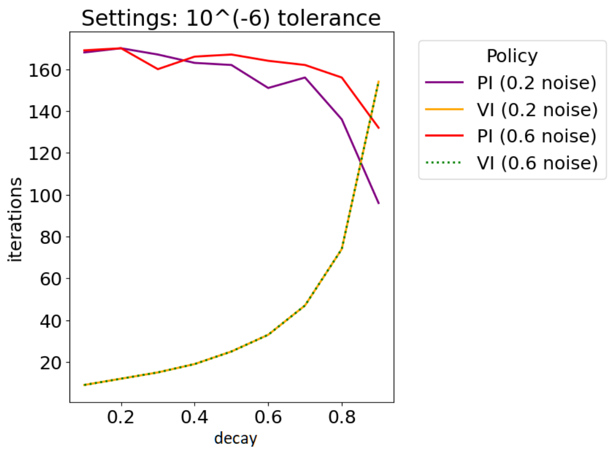

Figure 6: Impact of decay under different noise settings on the number of iterations performed by the different policies; other hyperparameters fixed.

To conclude the hyperparameter discussion, I would put together two key insights. Firstly, at optimal settings, PI tends to go through significantly fewer iterations than VI, as it uses the discrete set of policy modifications to determine convergence rather than the continuous set of value modifications. Secondly, VI is highly responsive to changes in update tolerance and decay while being insensitive to noise, whereas PI is more stable against both tolerance and decay at the cost of noise sensitivity – especially in the presence of high decay. Following this analysis, I proceed with the following settings for quantitative analysis and comparison to search algorithms: 0.9 decay, 10^{-6} update tolerance, and 0.2 noise, to be changed to 0 specifically for the search algorithm comparison.

3.2 Quantitative Analysis

Before looking at time and iteration averages over randomly simulated mazes, it is important to assess the quality of the previously listed, preferred hyperparameters. Figure 7 shows the solution to the seed 42 large maze shown in figure 1 as a set of policies and node values superimposed on the maze. The differences between values become significant about 15 tiles away from the exit, and yet visual inspection would show the policy of every tile to indeed point the agent towards the shortest path to the exit. This intuition will be confirmed when the path is drawn for the comparisons in the next section.

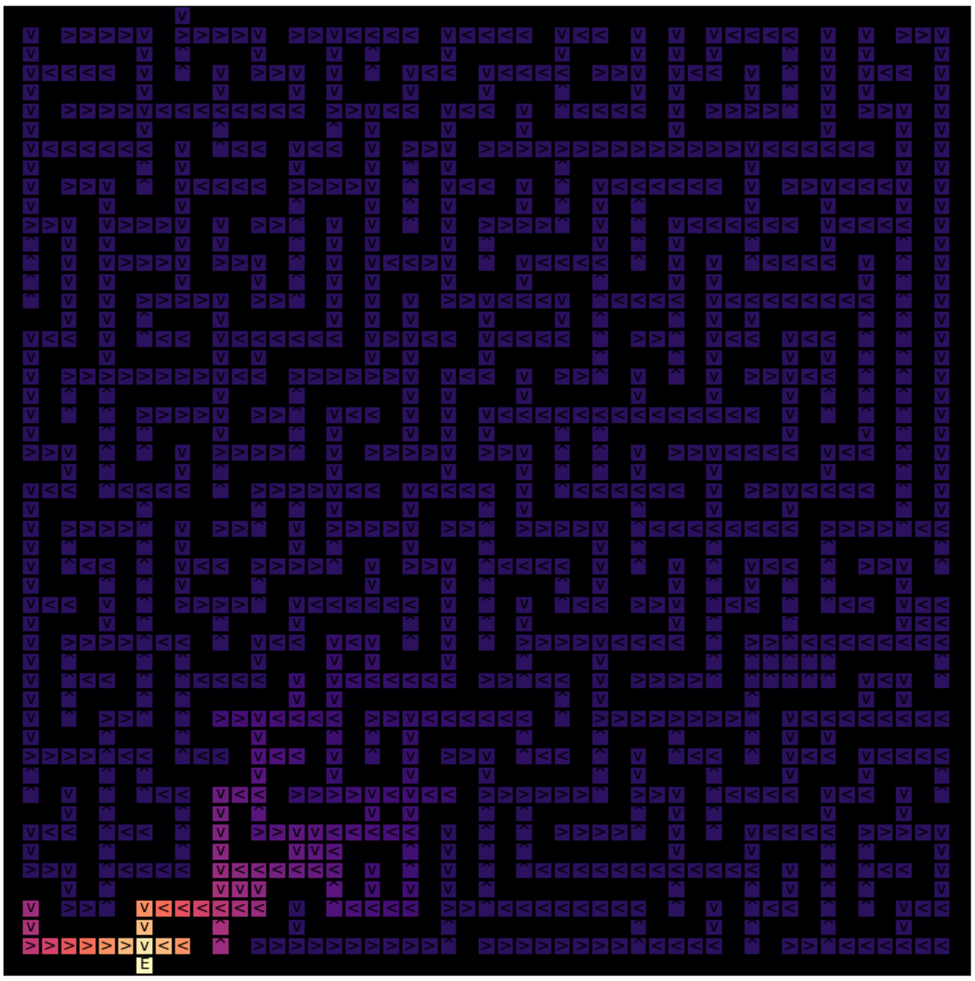

Figure 7: Solution of the seeded large maze, identical between the converged VI and PI. Higher brightness indicates higher node values, while arrows give the policy of the tile.

Table 2 contains the mean values of running time and iteration count over 10 random mazes of each size. Running time obviously increases with maze size, but it is interesting to note that PI sees a significantly sharper increase compared to VI. Furthermore, while larger mazes incur higher iterations for PI, they have no impact on VI; however, in spite of the increase with maze size, the final iteration count of PI is always much smaller than that of VI. Finally, it is worth noting that standard deviations are clearly lower for VI at the medium and large maze size.

Table 2: Average performance of the MDP-based algorithms over different metrics and maze sizes. Standard deviations in brackets.

		PI	VI
SIZE = 5	Running Time	0.33 (0.05)	0.39 (0.008)
	Iterations	9 (2.37)	154 (0)
SIZE = 12	Running Time	10.57 (3.77)	5.58 (1.52)
	Iterations	25.7 (5.85)	154 (0)
SIZE = 25	Running Time	45.34 (13.96)	19.75 (8.01)
	Iterations	46.5 (16.84)	154 (0)

In conclusion, VI is far less sensitive to maze size than PI. This is an advantage from the running time perspective, but a clear disadvantage from an iteration count perspective. It follows that, while PI goes through fewer iterations, a single PI iteration is a lot more complex than a VI iteration, as reflected by the running time. And as the previous subsection has shown, outside of optimal settings, PI is overall more stable to changes in update tolerance or decay, while its reactions to noise can sometimes lead to significantly fewer iterations. For these reasons, if the computation power is available, I believe PI to be preferable to VI.

4 Search - MDP comparison

4.1 Qualitative Analysis

I randomly chose seed 24 for a fresh maze to compare the most consistent search algorithm (A*), the most volatile, high-risk high-reward search algorithm (DFS) against the MDP methods. Both converge to the same solution, seen in the middle panel of figure 8, and since iterations are not important and I fix the decay and update tolerance at favourable values, I chose VI for the comparison due to its speed. I adjusted the heatmap so that the lowest values are a bit brighter to make the (small) policy arrows more easily visible.

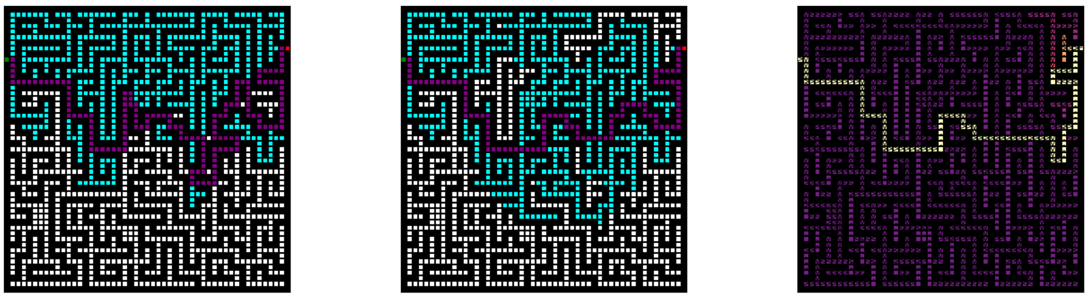

Figure 8: Seed 24 large maze solution for DFS (left), A* (middle) and VI (right).

It is immediately clear that A* and VI find the same solution, which is optimal but not unique for this particular maze. DFS, however, maintains its pattern of taking a longer route; to its credit though, less ndoes are visited compared to A*. Even with the higher speed of VI however, there is no comparing the execution time of the MDP-based algorithm and the search algorithms. The next section contains hard proof of this.

4.2 Quantitative Analysis

Re-sampling another 10 random mazes of each size, I make a final comparison of all 5 algorithms in terms of execution time and path length. In order to use the same mazes across algorithm categories, I randomly select 10 integers between 1 and 200 as seeds at the beginning and use these in both analysis loops for maze generation. The MDP settings are 0 noise, 0.9 decay, 0 living rewards and 10^{-6} update tolerance.

Table 3: Average performance of all algorithms over different metrics and maze sizes. Standard deviations in brackets.

		BFS	DFS	A*	PI	VI
SIZE = 5	Time	$3 * 10^{-4}$	$6 * 10^{-4}$	$4 * 10^{-4}$	0.18 (0.04)	0.39 (0.02)
	Path	18.8 (4.42)	19.6 (4.2)	18.8 (4.42)	18.8 (4.42)	18.8 (4.42)
SIZE = 12	Time	10^{-3}	$4 * 10^{-3}$	10^{-3}	4.03 (1.22)	5.73 (1.51)
	Path	59 (16.54)	60.2 (17.02)	59 (16.54)	59 (16.54)	59 (16.54)
SIZE = 25	Time	0.005 (0.001)	0.025 (0.014)	0.007 (0.006)	31.35 (11.59)	23.5 (9.05)
	Path	114 (21.13)	146.8 (34.59)	115.2 (20.54)	114 (21.13)	114 (21.13)

Rather than deliver new insights, the primary contribution of table 3 is confirming previous intuitions and crystalising a conclusion to the assignment. BFS and A* still perform very similarly, with BFS being slightly faster and slightly more precise when mazes get large at the cost of significantly more visited nodes. The MDP-based algorithms always match the optimal performance of MDP, albeit slower by orders of magnitude. PI and VI do maintain the advantage that they solve the *entire* maze, i.e. an optimal solution is provided wherever one may set the entrance to the maze. PI outperforms VI on small and medium mazes – a change from the MDP comparison, where it only pulled ahead on small mazes, signalling insignificance of the result on medium mazes – but loses clearly when applied to the large mazes. Finally, while the volatility of DFS is not as self-evident for small and medium mazes as I previously found, this algorithm performs markedly worse than its peers on large mazes, in terms of both time and path length.

A case can be made for the usage of each algorithm. A* is a good default, boasting both speed and effectiveness. If there are no costs associated to investigating nodes, BFS has a slight edge over it and is worth choosing. And if a solution for the whole maze is required, PI is the most reliable choice in most conditions. However, if the setting can be optimised, VI is able to achieve the same results faster and with lower complexity, at the cost of significantly more iterations.