

AI-EX-I Solutions

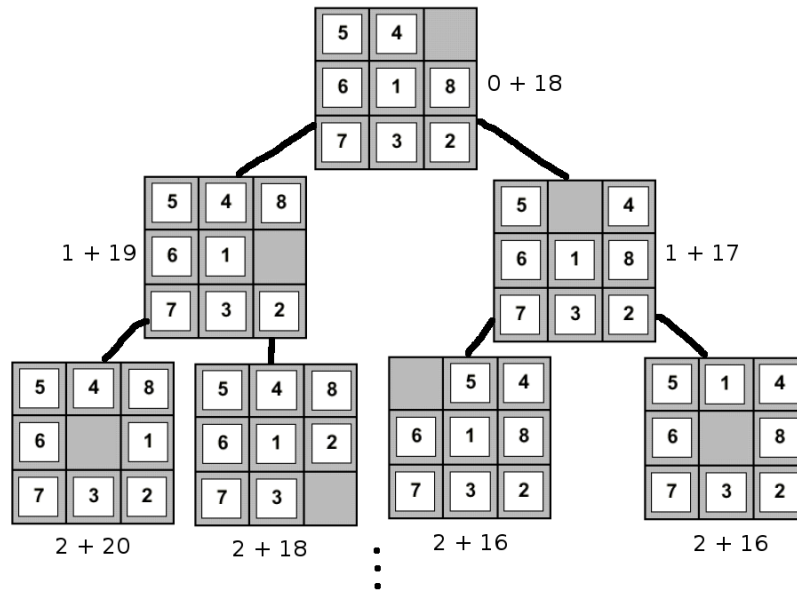
1. Pathfinding

- (a) I. Create a queue Q containing grid positions (x, y) sorted by a desirability function $f(x, y) = g(x, y) + h(x, y)$ where $g(x, y)$ are the actual costs to reach position (x, y) (1 per move) from the start position and $h(x, y)$ is a heuristic for how far away (x, y) is from the disc position.
- II. Loop
1. If Q is empty, return failure.
 2. Remove first position (x, y) from Q .
 3. If (x, y) is the disc position, return the path leading from the initial node to n .
 4. Mark position (x, y) as visited to avoid loops.
 5. Generate the successors of (x, y) by checking whether $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, $(x, y-1)$ are unvisited squares in the maze and add them in order of their desirability value to Q .
- An admissible heuristic h would be to count the valid moves required for the monster to move along the direct path (ignoring the maze) from its current position to the disc position.
 - This is admissible as the monster has to make at least this many moves to reach the disc and hence it does not overestimate the real costs.
- (b) • A^* is guaranteed to find an optimal solution with an admissible, consistent heuristic. A^* is also optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A^* .
- Greedy best-first search is not guaranteed to find an optimal solution as it only considers an estimate for the distance from the current node to the goal, not the costs to reach the current node.
 - So greedy best-first is not better than A^* .
 - In order to be sure that we always find the optimal solution, the heuristic for A^* has to be consistent: for any action $A : x \rightarrow x'$ we have $h(x) \leq \text{costs.of}(A) + h(x')$ / $f(x)$ is non-decreasing along any path generated by A^* . This is the case for the above heuristic as it is a metric.
- (c) • The aim is to find the position with the “longest shortest path” from the monster start position. For this all nodes have to be visited in order of their distance (measured in moves of the monster) from the monster start position.
- As the costs for each move of the monster is constant a breadth-first search with checking for repeated positions can find the optimal disc position—it is the last node visited (or any other node on the same level in the breadth-first search tree). (Uniform-cost is fine as in this case it is equivalent to breadth-first; the general solution is Dijkstra’s algorithm, but this is not required; depth-first strategies will not succeed).

2. The 8-Puzzle

- (a) Represent each state as a 3×3 matrix/array. Problem is defined as a single-state problem by the following elements:
- Initial state: matrix as shown in question.
 - Goal test: verify if current state is goal state as shown in question.
 - Successor function: can move horizontal or vertical adjacent tiles into empty spot giving 4, 3 or 2 successors for a state (no need to list all of these explicitly)
 - Path costs: e.g. one for each operation / move as we try to find minimum number of moves.

- State space expanded from goal state (numbers are “path costs” + “heuristic II” from (b) - careful, verify the values!).



- (b) A trivial heuristic would be $h(x) = 0$, this one is always admissible, but also does not provide any knowledge about the problem (this is not what this question is about!).

Heuristic I: count the number of tiles which are not in the correct position.

This heuristic is admissible as it underestimates the costs. Each tile not in the correct position has to be moved at least once.

Heuristic II: for each tile count the minimum number of moves required to move it to the correct position ignoring any other tiles (Manhattan distance, e.g. tile 5 in the start state is 4 moves away from its position) and add up all these distances.

This heuristic is admissible as it requires at least that many moves to get each tile to the correct location.

- (c) Greedy best-first expands the node with the smallest heuristic value, i.e. it only considers an estimate of how expensive it is to get from the current node to the goal, not how expensive it is to get to the current node.

A* expands the node with the smallest estimate of the total costs, adding the costs to get to the current node and the cost estimate for getting from the current node to the goal. An admissible, consistent heuristic guarantees to find the optimal solution.

The way the graph above is traversed can be indicated by a sequence of nodes indicating the sequence in which they are expanded according to the search strategy.

3. A* Algorithm

- (i) Breadth-first search expands the shallowest unexpanded node first. I.e. for A* we set $h(n)$ to 0 and $c(e)$ to 1 for each edge (or set $g(n)$ to the depth of the node).
- (ii) Uniform-cost search expands the least-cost unexpanded node first. I.e. for A* we set $h(n)$ to 0 and $g(n) = \sum_{e \in \text{Path}(n)} c(e)$ (or set $c(e)$ to the actual costs).
- (iii) Depth-first search expands the deepest unexpanded node first. So we set $c(e)$ (and with that $g(n)$) to 0 and assign $h(n)$ such that it is larger for nodes that are found earlier: we initialise a global counter X with a very large value and assign X to all of its unvisited successors. After each assignment X is decreased by one. Hence, the earlier a node is found, the higher its $f(n) = 0 + h(n)$ value.

4. Games

- (a) • 9, 99, and 1 do not have to be evaluated.

- After the left-most min node is visited, the max node is at least -3 , which means after the middle min node evaluates -5 , we know it will not be selected by the max node and hence the remaining states do not have to be evaluated. Similarly, after the right min node evaluates to -4 , we know it will not be selected by the max node, and we can stop evaluating the states.
- (b)
- Expected value of
 - **a** is $10/2 - 3/2 = 3.5$
 - **b** is $-5/3 + 9/3 + 99/3 = 34.33$
 - **c** is $-4/2 + 1/2 = -1.5$
 - So **b** gives the best expected result.
 - In order to adjust minimax for the random opponent the min node should return the expected value, i.e. $\sum_{a_i} 1/n * \text{Max}(a_i)$ for all n actions a_i instead of the minimum value.
 - Pruning is in general not possible anymore. Min nodes need to compute weighted averages and for this require the results of all successors. Max nodes still choose a maximum value, but no minimum value, only weighted averages are available to decide which maximum value would not be used. (If we know the bounds of the evaluation function we can still track an $[\alpha, \beta]$ range and determine when the average will lie outside.)

5. Zero-Sum and Non-Zero-Sum Games

- (a)
- α - β pruning processes min and max nodes differently:
 - it keeps track of α , the best score for max along the path to the current state: at a max node update α whenever a successor produced a larger value.
 - it prunes further successors of max nodes if the current max score for the node becomes larger or equal than β
 - it keeps track of β , the best score for min along the path to the current state: at a min node update β whenever a successor produced a smaller value.
 - it prunes further successors of min nodes if the current min score for the node becomes less or equal than α .
 - By keeping track of α and β it is possible to eliminate nodes that would not be chosen by a parent node because a better max or min score has already been found. So the result for zero-sum games does not change.
- (b)
- Nodes f, i, x, p, q, r are not examined.
 - The node ordering is not optimal for α - β pruning. E.g. if we first process w on the right branch, we would not have to process v and x with its successor: w gets a value of -5 , which would mean the min node z does not return a larger value than -5 , but at the root we already have a value of 5 , so we can stop processing any further successors of z. (Example sufficient to show non-optimality, there are other examples in the left sub-tree with min not choosing t or u after getting their first max value with similar explanation).
- (c)
- Minimax will work if it's setup right for the evaluation scores: return a vector of the two evaluation scores for each player and each node picks its evaluation score and selects the maximum—essentially a maximax algorithm (or select minimum if the opponent's score are still better if they are lower).
 - α - β pruning is not possible. It is based on the idea that what is good for the player is bad for its opponent. E.g. MIN knows it does not have to explore a certain branch, because it can force MAX to go down another branch, which is better for MIN than the pruned branch. For non-zero sum games this is not guaranteed.
- (d) **4-Queens Problem**
- i.
- Variables: Q_1, Q_2, Q_3, Q_4 for row number of queens.
 - Domain: $Q_l \in D_l = \{1, 2, 3, 4\}$ for $l = 1, 2, 3, 4$.

- Constraints: $Q_l \neq Q_k$ for $l \neq k$ (not in same row), $|Q_l - Q_k| \neq |l - k|$ for $l \neq k$ (not in same diagonal).

ii. Backtrackig with forward checking:

- Assign $Q_1 = 1$. This eliminates the following assignments marked with -:

	1	2	3	4
Q_1	x	-	-	-
Q_2	-	-		
Q_3	-		-	
Q_4	-			-

- Assign $Q_2 = 3$. This eliminates the following assignments marked with +:

	1	2	3	4
Q_1	x	-	-	-
Q_2	-	-	x	+
Q_3	-	+	-	+
Q_4	-		+	-

- Q_3 has no valid value, so backtrack and change Q_2 .

- $Q_2 = 4$. This eliminates the following assignments marked with +:

	1	2	3	4
Q_1	x	-	-	-
Q_2	-	-	-	x
Q_3	-		-	-
Q_4	-	+		-

- $Q_3 = 2$. This eliminates the following assignments marked with *:

	1	2	3	4
Q_1	x	-	-	-
Q_2	-	-	-	x
Q_3	-	x	-	-
Q_4	-	+	*	-

- Q_4 has no valid value, so backtrack and change Q_3 . There is no alternative for Q_3 left, so change Q_2 . We have checked all alternatives for Q_2 , so change Q_1 .

- Assign $Q_1 = 2$. This checking eliminates the following assignments marked with -:

	1	2	3	4
Q_1	-	x	-	-
Q_2	-	-	-	
Q_3		-		-
Q_4		-		

- Assign $Q_2 = 4$. This eliminates the following assignments marked with +:

	1	2	3	4
Q_1	-	x	-	-
Q_2	-	-	-	x
Q_3		-	+	-
Q_4		-		+

- Assign $Q_3 = 1$. This eliminates the following assignments marked with *:

	1	2	3	4
Q_1	-	x	-	-
Q_2	-	-	-	x
Q_3	x	-	+	-
Q_4	*	-		+

- Assign $Q_4 = 3$. We have a solution.