CS 188 Spring 2021 Introduction to Artificial Intelligence

Exam Prep 1 Solutions

Course Overview

Here are some questions for you:

1. What is AI?
2. What can AI do?
3. What do you want to learn from this course?

There are two types of discussion sections:

1. Regular Discussion
2. Exam Prep

There are 5 graded components:

1. Programming Assignments (25%)
2. Electronic Homework Assignments (10%)
3. Written Homework Assignments (10%)
4. Midterm (20%)
5. Final exam (35%)

Q1. *n*-Queens

Max Friedrich William Bezzel invented the "eight queens puzzle" in 1848: place 8 queens on an 8×8 chess board such that none of them can capture any other. The problem, and the generalized version with n queens on an $n \times n$ chess board, has been studied extensively (a Google Scholar search turns up over 3500 papers on the subject).

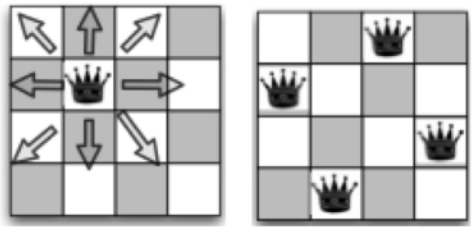

Figure 1: Queens can move any number of squares along rows, columns, and diagonals (left); An example solution to the 4-queens problem (right).

(a) Formulate n-queens as a search problem. Have each search state be a board, where each square on the board may or may not contain a queen. To get started, we'll allow boards in our state-space to have any configuration of queens (including boards with more or less than n queens, or queens that are able to capture each other).

Start State: An empty board

Goal Test: Returns True iff n queens are on the board such that no two can attack each other

Successor Function: Return all boards with one more queen placed anywhere. Another possibility (see part d) - place queens left to right (i.e. in the first column, then the second column, etc.)

(b) How large is the state-space in this formulation? There are n^2 squares, each of which may or may not contain a queen. Therefore there are 2^{n^2} possible states, or 1.8×10^{19} for 8-queens.

(c) One way to limit the size of your state space is to limit what your successor function returns. Reformulate your successor function to reduce the effective state-space size. The successor function is limited to return legal boards. Then, the goal test need only check if the board has n queens.

(d) Give a more efficient state space representation. How many states are in this new state space? A more effective representation is to have a fixed ordering of queens, such that the queen in the first column is placed first, the queen in the second column is placed second, etc. The representation could be a n-length vector, in which each entry takes a value from 1 to n, or "null". The i-th entry in this vector then represents the row that the queen in column i is in. A "null" entry means that the queen has not been placed.

Since each of the n entries in the vector can take on $n+1$ values, the state space size is $(n+1)^n \approx 4.3 \times 10^7$ for $n = 8$.

To further limit the state space, we can require that queens are placed on the board in order (in this case, left to right). Now we know that if k queens have been placed on the board, the first k entries in the state space are non-null and the last $n-k$ entries are null. This creates a total state space size of $\sum_{k=0}^{n} n^k = \frac{n^{n+1}-1}{n-1} \approx 1.9 \times 10^7$ for $n = 8$.

Finally, by combining this idea with the successor function in part (c), we can further limit the *effective* state size.

Q2. Search

For this problem, assume that all of our search algorithms use tree search, unless specified otherwise.

(a) For each algorithm below, indicate whether the path returned after the modification to the search tree is guaranteed to be identical to the unmodified algorithm. Assume all edge weights are non-negative before modifications.

(i) Adding additional cost $c > 0$ to every edge weight.

	Yes	No
BFS	●	○
DFS	●	○
UCS	○	●

(ii) Multiplying a constant $w > 0$ to every edge weight.

	Yes	No
BFS	●	○
DFS	●	○
UCS	●	○

(b) **For this part**, two search algorithms are defined to be **equivalent** if and only if they expand the same states in the same order and return the same path. **Assume all graphs are directed and acyclic.**

Assume we have access to costs c_{ij} that make running UCS algorithm with these costs c_{ij} equivalent to running BFS. How can we construct new costs c'_{ij} such that running UCS with these costs is equivalent to running DFS?

○ $c'_{ij} = 0$	○ $c'_{ij} = 1$	○ $c'_{ij} = c_{ij}$
● $c'_{ij} = -c_{ij}$	○ $c'_{ij} = c_{ij} + \alpha$	○ Not possible

Breadth-First Search expands the node at the shallowest depth first. Assigning a constant positive weight to all edges allows to weigh the nodes by their depth in the search tree. Depth-First Search expands the nodes which were most recently added to the fringe first. Assigning a constant negative weight to all edges essentially allows to reduce the value of the most recently nodes by that constant, making them the nodes with the minimum value in the fringe when using uniform cost search. Hence, we can construct new costs c'_{ij} by flipping the sign of the original costs c_{ij}.

Q3. SpongeBob and Pacman (Search Formulation)

Recall that in Midterm 1, Pacman bought a car, was speeding in Pac-City, and SpongeBob wasn't able to catch him. Now Pacman has run out of gas, his car has stopped, and he is currently hiding out at an undisclosed location.

In this problem, you are on SpongeBob's side, tryin' to catch Pacman!

There are still p of SpongeBob's cars in the Pac-city of dimension m by n. In this problem, **all of SpongeBob's cars can move, with two distinct integer controls: throttle and steering, but Pacman has to stay stationary**. Spongebob's cars can control both the throttle and steering for each step. Once one of SpongeBob's cars takes an action which lands it in the same grid as Pacman, Pacman will be caught and the game ends.

Throttle: $t_i \in \{1, 0, -1\}$, corresponding to {Gas, Coast, Brake}. This controls the **speed** of the car by determining its acceleration. The integer chosen here will be added to the velocity for the next state. For example, if a SpongeBob car is currently driving at 5 grid/s and chooses Gas (1) it will be traveling at 6 grid/s in the next turn.
Steering: $s_i \in \{1, 0, -1\}$, corresponding to {Turn Left, Go Straight, Turn Right}. This controls the **direction** of the car. For example, if a SpongeBob car is facing North and chooses Turn Left, it will be facing West in the next turn.

(a) Suppose you can **only control 1 SpongeBob car**, and have absolutely no information about the remainder of $p - 1$ SpongeBob cars, or where Pacman has stopped to hide. Also, the SpongeBob cars can travel up to 6 grid/s so $0 \leq v \leq 6$ at all times.

(i) What is the **tightest upper bound** on the size of state space, if your goal is to use search to plan a sequence of actions that guarantees Pacman is caught, no matter where Pacman is hiding, or what actions other SpongeBob cars take. Please note that your state space representation must be able to represent **all** states in the search space.

$28mn * 2^{mn}$

There are mn positions in total. At each legal position, there are 7 possible speeds (0, 1, 2, 3, 4, 5, 6), so a factor of 7 is multiplied. In addition, since change of direction depends on orientation of the car, another factor of 4 is multiplied.
The only sequence of actions which guarantees that Pacman is caught is a sequence of actions which visits every location. Thus, we also need to a list of $m * n$ boolean to keep track of whether we have visited a specific grid location, and that is another factor of 2^{mn}

(ii) What is the maximum branching factor? Your answer may contain integers, m, n.

9

3 possible throttle inputs, and 3 possible steering inputs. The list of boolean does not affect the branching factor.

(iii) Which algorithm(s) is/are guaranteed to return a path passing through all grid locations on the grid, if one exists?

- ☐ Depth First Tree Search
- ■ Depth First Graph Search
- ■ Breadth First Tree Search
- ■ Breadth First Graph Search

Please note the list of boolean is in the state space representation, so we can revisit the same grid position if we have to.

(iv) Is Breadth First Graph Search guaranteed to return the path with the shortest number of **time steps**, if one exists?

● Yes ○ No

Breadth First Graph Search is guranteed to return the path with the shortest amount of time, because each edge here represent moving for 1 unit of time.

(b) Now let's suppose you can control **all** p SpongeBob cars at the same time (and know all their locations), but you still have no information about where Pacman stopped to hide

(i) Now, you still want to search a sequence of actions such that the paths of p SpongeBob cars combined **pass through all** $m * n$ **grid locations**. Suppose the size of the state space in part (a) was N_1, and the size of the state space in this part is N_p. Please select the correct relationship between N_p and N_1.

○ $N_p = p * N_1$ ○ $N_p = p^{N_1}$ ○ $N_p = (N_1)^p$ ● None of the above

In this question, we only need one boolean list of size mn to keep track of whether we have visited a specific grid location. So the size of the state space is bounded by $N_p = (28mn)^p 2^{mn}$, which is none of the above.

(ii) Suppose the maximum branching factor in part (a) was b_1, and the maximum branching factor in this part is b_p. Please select the correct relationship between b_p and b_1.

○ $b_p = p * b_1$ ○ $b_p = p^{b_1}$ ● $b_p = (b_1)^p$ ○ None of the above

For example, the case of $p = 2$ means two cars can do all 9 options, so the branching factor is $9^2 = 81$. In general, the branching factor is then b_1^p.