1 n-Queens

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

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

(1) Formulate \( n \)-queens as a search problem:

Start State: An empty board

Successor Function: Return all boards with one more queen placed anywhere

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

(2) Like many search problems, \( n \)-queens can be parameterized in a number of different ways. Suggest a different formulation (hint: how many successors does your function return?) The successor function is limited to return legal boards. Then, the goal test need only check if the board has \( n \) queens.

(3) How large is the state space in your formulations? There are \( n^2 \) choices for the first queen, \( n^2 - 1 \) choices for the second queen, and so on. But, order doesn’t matter. So we have \( \frac{(n^2-n)!}{(n^n-n!)} \). That’s 4,426,165,368 possible boards for the 8-queens problem.

(4) Why is depth first search preferable to breadth first search for this problem? All of the solutions are at the deepest part of the search tree. BFS expands all possibilities at each successive depth, while DFS considers depth-\( n \) solutions right away. The 8-queens problem has 92 solutions, by the way.
2 8-puzzle

The 8-puzzle (actually, its more complex cousin, the 15-puzzle) was briefly the subject of a gaming craze in the U.S. in 1880 after Sam Lloyd offered $1000 for a solution to what turned out to be an impossible tile arrangement. The puzzle involves sliding tiles until they are ordered correctly. To solve these puzzles efficiently with A* search, good heuristics are important.

![Sam Lloyd’s unsolvable 15-puzzle. Why can’t it be solved?](image)

(1) Create a heuristic for the 8-puzzle based on the number of misplaced tiles. Count the number of tiles out of place, not including the blank tile.

(2) Create a heuristic using Manhattan Distance. Sum the Manhattan (city block) distances between each tile’s current position and its intended position.

(3) Explain why your heuristics are admissible. These heuristics are both relaxations of the original problem so the heuristic estimate is always less than or equal to the actual cost of reaching the goal. Note that if you include the blank tile in the estimate, this heuristic is no longer admissible (think about the case where only 1 tile is out of place).

See nqueens.py and eightpuzzle.py for implementations of the ideas in this handout. Once you’ve written search.py, you can run `python nqueens.py n` and `python eightpuzzle.py`. Modify the main section of these files to try different search algorithms and heuristics. Note: you’ll need util.py (from the search project) in the same directory.