# 1 Chess Search

## 1.1 Board

**Note**: The boards used for these exercises will be modified to decrease the state space and allow for more tractable examples. Below is an example of how a board may be setup. It is only 4x4, but piece movements are still the same as in chess.

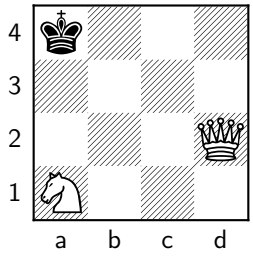


Figure 1: Example board

## 1.2 State Space

Modeling a state space should depend on the objective. Let's assume that black is not allowed to move any pieces, and we want to capture all black pieces using white pieces. White pieces continue to move each turn. In Figure 1, it would take 2 moves for example.

### 1.2.1 Number of Pieces on Board

Maybe we model the state space as the number of pieces on the board. Why might this not be a very useful state space?
Ask students for answers. Looking for something along the lines of, but not limited to:

1. Cannot tell which pieces are which, so there is no clear successor function
2. Cannot check for goal because we don't know how many pieces of each color exist
3. No clear way to derive a successor function

### 1.2.2 Location of Every Piece and Color

A little better than the last. We now can check for a goal by seeing if any black pieces are still on the board. But we still don't know which pieces are which, so

there is no successor function (moves of each piece cannot be inferred without knowing what piece is what).

#### 1.2.3 Location of Piece, With Color and Piece Type

No we have a sufficient state space. We can derive a successor function by trying all possible white piece movements. Additionally, the can check for a goal state by seeing if all black pieces are captured.

### 1.3 Deriving Successor and Goal State Functions

Using the State space from before (position of every piece as well as color, and piece type, lets define a successor function as well as a goal state predicate function.

### 1.4 Goal State

What would be the goal state if the objective of the game is for all black pieces to be captured?

Answer: A state where there are no more black pieces

#### 1.4.1 Successor Function

How might we define a successor? Assume you are also given the dimensions of the board ($N$).

Work through how one state would transform into the next state by moving at most one piece. Some considerations:

1. White pieces should not be able to land on squares with other white pieces (white cannot capture white)
2. Cannot move a piece off of the board
3. Each piece has fixed movement patterns (cannot move to any arbitrary square in next turn)

Consider for example, the following board:

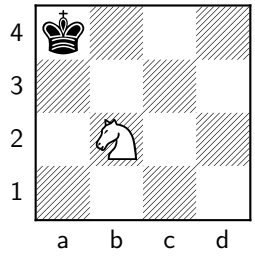


Figure 2: State 1

Lets define the start state for Figure 3 to be:

$$[(black, king, a4), (white, knight, b2)]$$

What are possible successors?

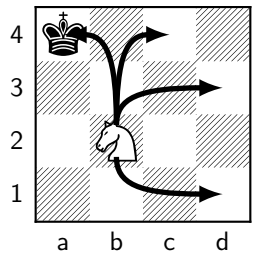


Figure 3: State 1 Successors

Written out, the successor states are:

{[(white,knight,a4)],
[(black,king,a4),(white,knight,c4)],
[(black,king,a4),(white,knight,d3)],
[(black,king,a4),(white,knight,d1)]}

**Are any of these a goal state? Which one(s)?**

### 1.4.2 Example

Walk through the successor tree of the following board with the students to a depth of 1 or 2.

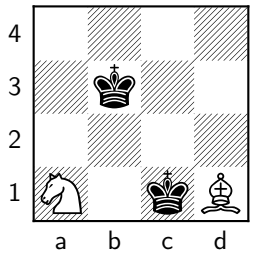


Figure 4: Problem 2

**If your objective was to finish the game in Figure 4 in as few moves as possible, why would you not want to capture king b3 with the bishop?**

Answer: Capturing with the bishop requires at least 4 moves to finish the game. Capturing with the knight only requires 2.

See next page for successors

Below are the successors

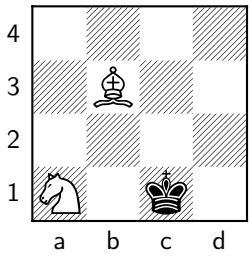


Figure 5: Successor 1

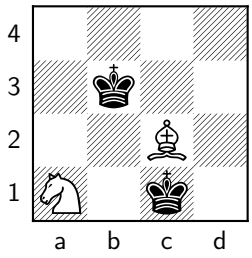


Figure 6: Successor 2

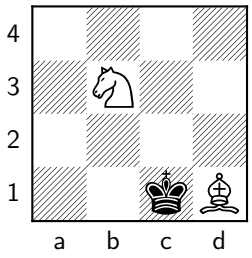


Figure 7: Successor 3

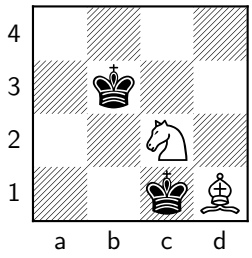


Figure 8: Successor 4

From here, we can see that Successor 3 is one move away from finishing. All other successors are at least 2 moves away.

## 1.5 Search Algorithms

Now we have defined our successor function, and goal predicate function. We want to run a search algorithm on our game. What are some options?

### 1.5.1 BFS

If we wanted to know the minimum number of moves, which search algorithm could we use?

What are some downsides to using BFS?

## 1.6 Iterative Deepening

How might iterative deepening help to address some downsides of BFS?

## 1.7 A*

Let's assume at first that every move has a cost of 1. What would be an admissible heuristic?

One possible answer: Number of black pieces on the board (since at most 1 can be captured per turn)

What if the cost of every move was equal to the manhattan distance of that move? Would are previous heuristic still be admissible?

Answer: Yes

Can anyone come up with a better heuristic that is also admissible? **Note, this is pretty hard and no one may come up with one. Just say that they can think about it in free time. You don't have to wait for someone to come up with a better one.**

## 1.8 Difference Between Greedy and A*

Assume we use the number of black pieces on the board to be the heuristic for each state. Why might greedy fail to find a solution to the following problem efficiently? Walk through some state transitions. Why would A* do better?

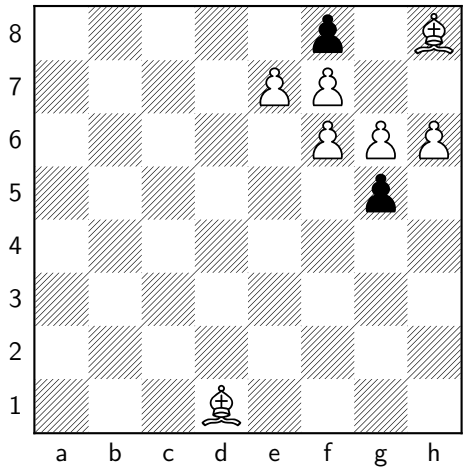


Figure 9: Problem 3

# 2 Directions Problem (Optional)

Depending on how much time is left, this could be a fun exercise. It also is more practical, and very open ended.

## 2.1 Problem Statement

Say you wanted to develop a tool that will help you plan your vacation. There are 10 cities (a,b,c,d,e,f,g,h,i,j) and you want to visit 5 of them (b,e,h,i,j). Not all cities are directly connected, but a path exists from one city to every other city through potentially other cities if not direct.

You want to visit all your top cities with as little travel time as possible. You can pick any start city and any end city (you haven't booked your flight or hotels yet).

## 2.2 State Space

How might we define a state space for this problem? What do we need to keep track of?

## 2.3 Search

What search algorithm could we use? Are some better than others? Why?