

Assignment 3

The Knight's Tour problem as a CSP

The knight in chess is the piece that is shaped like a horse's head. The knight can move in an "L-shaped" pattern, advancing two squares in one direction and then having the option to turn either right or left. If we display all the current possible moves for the knight, the resulting pattern is as follows (see Figure 1).

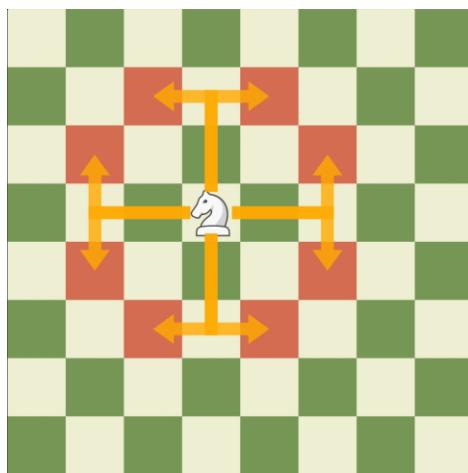


Figure 1 All knight's possible moves

The knight can move to the square marked in red and then repeat the process on the new square. The knight's tour must visit all 64 squares exactly once, starting from (0,0).

- The first goal of this assignment is to implement a Knight's Tour solver using a backtracking algorithm.

Problem Formulation

1. Variables

$$X_0, X_1, \dots, X_{63}$$

where:

- X_k = the board position of the knight at move k
- Each variable represents the knight's location after the k^{th} move.

2. Domains

$$D(X_k) = \{(i, j) \mid 0 \leq i < 8, 0 \leq j < 8 \text{ and } (i, j) \notin \{X_0, \dots, X_{k-1}\}\}$$

where:

- Initially, $X_0 = (0,0)$ (starting position)
- For $k > 0$, domain is all unvisited squares reachable by a knight move from X_{k-1}

3. Constraints

1. Knight Move Constraint

- $X_k \in \text{LegalKnightMoves}(X_{k-1})$
2. All-Different Constraint
$$X_i \neq X_j, \forall i \neq j$$
 3. Board Boundaries
$$0 \leq X_k(\text{row}) < 8, 0 \leq X_k(\text{col}) < 8$$

```
def backtracking(assignment):

    if len(assignment) == 64:
        return assignment

    current_x, current_y = assignment[-1]
    visited = set(assignment)
    successors = successor_fct(current_x, current_y, visited) # positions that respect the three constraints

    for x, y in successors:
        assignment.append((x, y))
        result = backtracking(assignment)
        if result is not None:
            return result
        assignment.pop()

    return None

assignment = [(0, 0)]
solution = backtracking(assignment)
```

- The second goal of this assignment is to implement a Knight's Tour solver using a backtracking algorithm enhanced with CSP heuristics:
 - MRV (Minimum Remaining Values): choose the next square with the fewest onward moves.
 - LCV (Least Constraining Value): prefer moves that restrict future choices the least.

```
def backtracking(assignment):

    if len(assignment) == 64:
        return assignment

    current_x, current_y = assignment[-1]
    visited = set(assignment)
    successors = successor_fct(current_x, current_y, visited) # positions that respect the three constraints

    successors = MRV(successors)
    successors = LCV(successors)

    for x, y in successors:
        assignment.append((x, y))
        result = backtracking(assignment)
        if result is not None:
            return result
        assignment.pop()

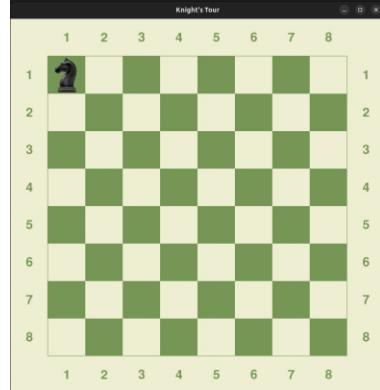
    return None
```

```
assignment.pop()
```

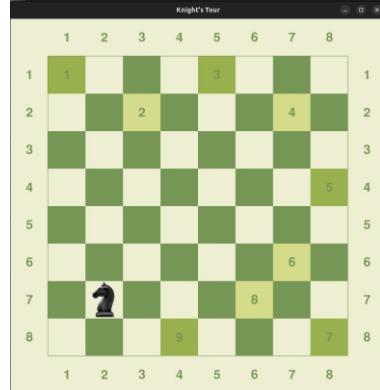
```
return None
```

```
assignment = [(0, 0)]
```

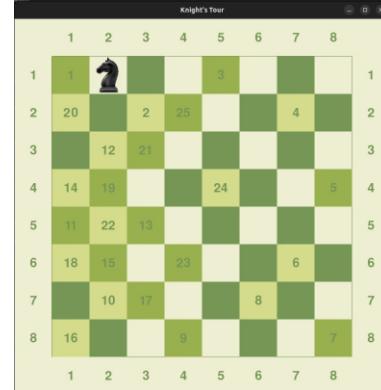
```
solution = backtracking(assignment)
```



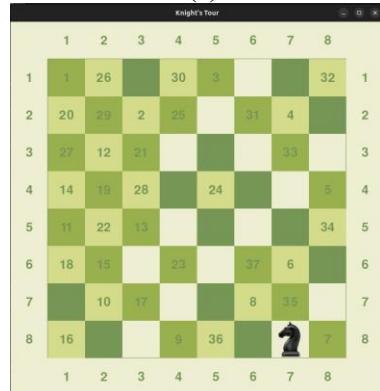
(a)



(b)



(c)



(d)



(e)



(f)

Figure 2 visualization of a solution of the knight's tour problem using the backtracking algorithm