University of the Witwatersrand

COMS3005: Advanced Analysis of Algorithms

Peg Solitaire Backtracking Assignment

October 25, 2017

*B*y Bancroft, E. (879192)
*A*nd Chalom, J. (711985)

1 Introduction

The purpose of the assignment is to implement and analyse a version of the Peg Solitaire game and the backtracking algorithm (to play the game).

2 Background

Peg Solitaire is a board game which has a number of holes that can be filled with pegs. We have chosen to use the European style of board which has extra positions on the board 2. In this style most of a grid has peg holes excluding three per corner. The aim of the game is to remove pegs until only one peg remains. This is the position one row directly above the central peg, a row above that and the left most peg in the top row. Moves are made when pegs jump over a peg and are placed in an open position. Then peg which is jumped over is then removed. This move can happen in both horizontal and vertical directions. In the European variant, the game has three possible optimal terminal states. Its is possible to hit sub-optimal states where there are more pegs left on the board but no possible moves left [1].

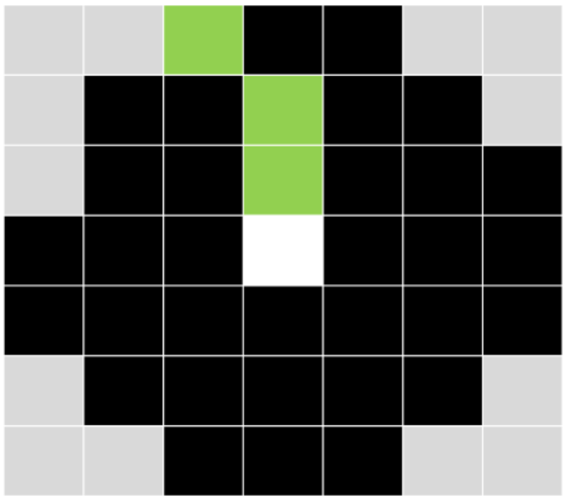
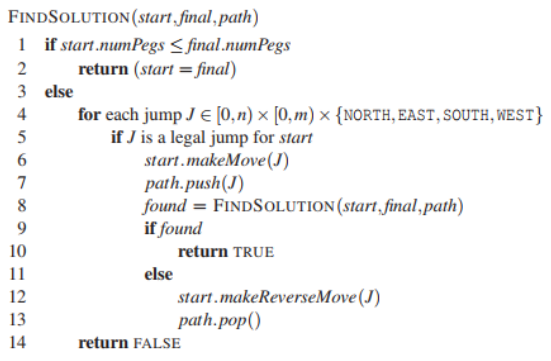


Figure 1: Diagram of an European Peg Solitaire Board Where the Black Squares Represent Peg Positions, Green Are Terminal Positions, Grey Are Not Positions and White is the Central Pixel.

The backtracking algorithm is similar to a brute force approach to finding solutions to problems but is more systematic. It attempts to follow a logical series of decisions in solving these problems and when a block state occurs the algorithm will backtrack to previous decisions and choose different paths until a terminal (complete) state is reached. The full set of solutions to a problem can be found by continuing to run the algorithm until all paths have been searched but that is not always necessary.

2.1 Recursive Algorithm



```
FINDSOLUTION(start, final, path)
if start.numPegs ≤ final.numPegs
    return (start = final)
else
    for each jump J ∈ [0, n) × [0, m) × {NORTH, EAST, SOUTH, WEST}
        if J is a legal jump for start
            start.makeMove(J)
            path.push(J)
            found = FINDSOLUTION(start, final, path)
            if found
                return TRUE
            else
                start.makeReverseMove(J)
                path.pop()
    return FALSE
```

Figure 2: Recursive Algorithm [2]

2.2 Stack Based Algorithm

3 Implementation

4 Theoretical Analysis

5 Results

6 Empirical Analysis

7 Conclusion

8 Group Member Contribution

Member	**Evan Bancroft 879192**	**Jason Chalom 711985**
Game	85%	15%
Back Tracking Algorithm	50%	50%
Complexity Analysis		
Report		

Table 1: Contributions of Group Members By Task

Acknowledgements

All drawn diagrams were drawn using http://draw.io/ and charts were made with Libre Office. All the programming was done in c++ using OpenMP for its timing functions.

References

[1] Douglas Wilhelm Harder. Peg solitaire. https://ece.uwaterloo.ca/~dwharder/aads/Algorithms/Backtracking/Peg_solitaire/.

[2] Charles E. Leiserson. Lab 5: Backtracking search. http://courses.csail.mit.edu/6.884/spring10/labs/lab5.pdf, 2010.