Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

Knight's Tour Solver

Peter Sivák

Degree Course: Informatics
Class: 3
Subject: Artificial Intelligence
Supervisor: prof. Ing. Pavol Návrat, PhD.
Year: 2011/2012

1 Assignment

The task is to go through the chessboard with legal moves of knight so that each square of the chessboard is visited exactly once. The solution should be proposed to be able to solve the problem for square chessboards of different sizes (at least from size 5x5 to 20x20) and to be able to start the tour through the chessboard on arbitrary initial square.

There exists very good and simple heuristic for solving this problem called *Warnsdorff's rule*. Implement this heuristic to *depth-first search* algorithm and for chessboard 8x8 find one (first) right solution for at least 10 different initial points. Algorithm with the heuristic should be designed and implemented to run also for chessboards of different sizes than just 8x8.

2 Solution

We have implemented this problem in *C++* language as a *graphical user interface* application. User can choose the *chessboard size* and the *initial position* of knight on the chessboard. Then he can choose *searching for a solution* and if a solution has been found, the application shows the solution and if solution does not exist, the application will show the appropriate message. User can stop the searching for a solution anytime, for example when it takes too long. In Figure 1 there is a screenshot from the start of the application.

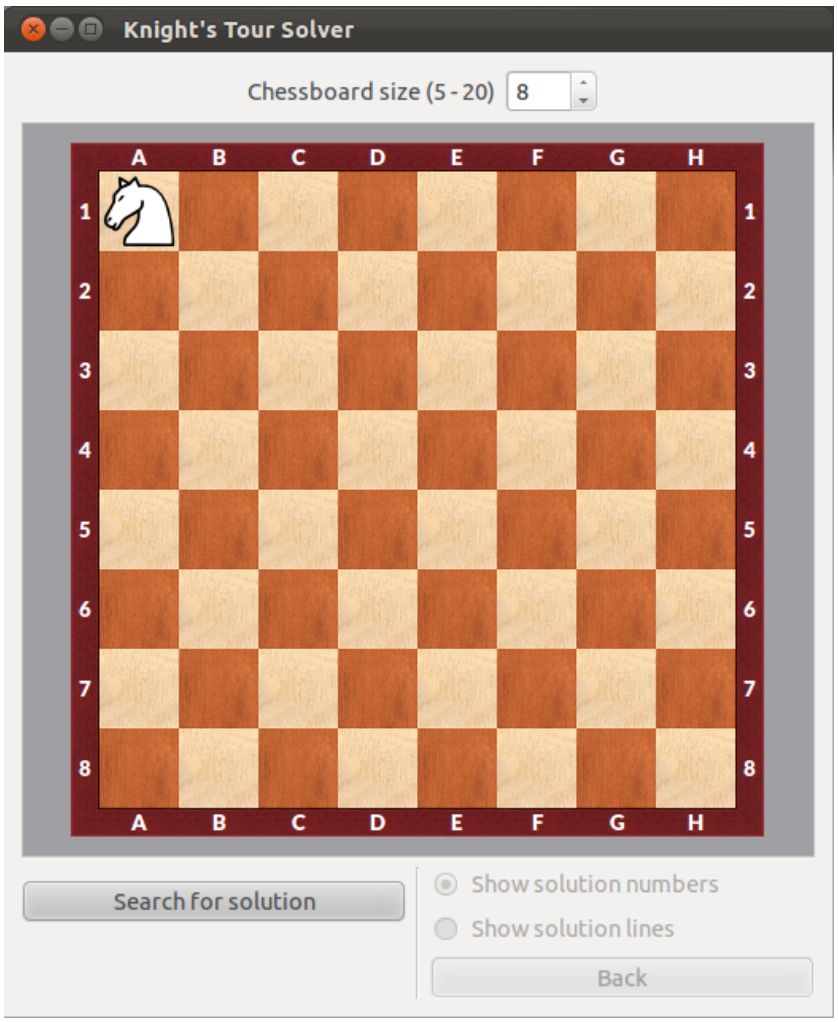

Figure 1: *Screenshot of the application*

3 Data Structures and Algorithms

We have divide our program into several classes:

1. **Main** - Starts the application.
2. **MainWindow** - Represents the application window.
3. **ChessboardScene** - Is responsible for visualizing the chessboard.
4. **Solver** - Is responsible for solving the knight's tour problem and represents the main part of the application.
5. **Node** - Represents a node in a searching tree.

We have implemented the solution searching algorithm using depth-first search:[1]

```
function DEPTH-FIRST-SEARCH(problem, PUSH-TO-STACK)
        returns solution or failure
    static: stack, stack containing generated and undeveloped
                   nodes, empty at the beginning
            node, node of searching tree

    stack <- CREATE-STACK(CREATE-NODE(INITIAL-STATE[problem]))
    loop do
        if stack is empty then return failure
        node <- POP(stack)
        if SOLUTION-TEST[problem] applicated on STATE(node) is
            successful then return CHOOSE-SOLUTION(node)
        stack <- PUSH-TO-STACK(EXPAND(node, OPERATORS[problem]),
            stack)
    end
```

4 Testing

We have tested our application on various configurations, so that with various chessboard sizes and knight's initial positions. We tried to turn off the mentioned heuristic and the solution time were incomparably slower. Using the heuristic, the solution searching of some specific configuration took a second but turning off the heuristic, the solution searching of the same configuration could take million years, so the heuristic increase the solution searching speed incomparably. When some solution searching has taken a long time, we cancelled the searching manually.

5 Conclusions

In this application we have implemented knight's tour solver using depth-first search algorithm.

The advantage of this algorithm is low memory costs because maximum depth of search tree is $d \cdot f$ where d is maximum depth of the tree where $d = n^2 - 1$ where n is a chessboard size and f is the branching factor or asymptotically expressed $O(n)$ which is *linear* space complexity. The disadvantage of this algorithm is worst-case searching time which asymptotically equals $O(f^d)$ which is exponential time complexity.

The application can be easily extended to solve larger chessboard sizes because it is designed to solve chessboards of size mxm.

We have implemented this application using *Qt Creator* integrated development environment so it can run on multiple target platforms including *Windows*, *Linux* and *Mac*.

References

[1] Pavol Návrat a kol. *Umelá inteligencia*. Vydavateľstvo STU v Bratislave, 2007.