IM1202 Knight Tour Project Report

Stijn de Preter (852726504)
Arjan Broer (850166428)

November 17, 2025

Abstract

This document presents a comprehensive overview of the research conducted for the IM1202 project. It includes an introduction to the problem, methodology, related work, experiments, results, and conclusions. The focus of the project is on solving the Knight Tour problem using Answer Set Programming (ASP). Key findings and insights are discussed throughout the document.

Contents

1 Introduction

The Knight's Tour problem is a classic problem in computer science. It involves moving a knight piece on a chessboard such that it visits every square exactly once. The challenge lies in the unique movement capabilities of the knight, which moves in an "L" shape: two squares in one direction and then one square perpendicular, or one square in one direction and then two squares perpendicular. More formally the knight's tour can be defined as a graph for an $n \times n$ chessboard defined by $G = (V, E)$ where $V = (i, j)|1 \leq i, j \leq n$ and $E = ((i, j), (k, l))||i - k|, |j - l| = 1, 2$. [4]

Research questions that we discuss in this report include:

- How can the Knight's Tour problem be logically modeled so that it is solvable with Answer Set Programming (ASP)?
- Which different logical models can be used to formulate the problem?
- How does the size of the chessboard affect the complexity and solution methods?
- How can heuristics and multi-shot solving techniques be applied to improve the efficiency of solving the Knight's Tour problem with Answer Set Programming (ASP)?

To limit the scope of the project and allow for comparisons, we will focus on finding five results for the problem on a standard board (8x8 squares). We look at Open tours meaning: The knight visits every square exactly once, but the ending square is possibly not a knight's move away from the starting square. If no performance difference is observed, the board can be expanded until a noticeable difference in performance is achieved. The starting point will also be fixed at a corner of the chessboard.

2 Methodology

There are several methods to solve this problem. The following two methods are available (sources still to be found/added):

- Method 1: A sequence is used. This means that we search for the next square of a chessboard from a specific square.
- Method 2: "Connected" fields are searched for for all squares. This searches for combinations of all the fields where they are connected.

Other strategies may also exist.

3 Related Work

In [4] the author discusses an efficient algorithm for finding knight's tours on larger chessboard. The scalability of the algorithm is proven by showing that boards can be split up in smaller boards. Following a structure of knight steps in the corners creates a structure that enables the boards to be connected. Solving the problem for the smaller board and copying this solution to stitch together the larger board allows for an efficient solution. The article [1] describes the open and closed knight's tour problem and provides a solutions in a CLINGO listings. The listing for the open knight's tour problem is designed by defining the $n \times m$ chessboard, the knights possible steps and the constraint of visiting a square only once. The solution to be found should be exactly $n \times m$ steps long following the set rules. This will guarantee a solution to the open knight's tour problem.

Other strategy for having the same rules, but the approach is to provie a rule that each square is reachable from another square. [Citation needed]

4 Experiments and Results

4.1 Methods to solve the knight problem

There are several methods to solve the knight problem. In this section two methods are discussed.

4.1.1 Method 1

This approach models the problem as a sequence of steps (or time points). The first step starts at position (x=1,y=1). The second step might be at (x=2,y=3). Each such combination of coordinates and step number is called a visit: at step "z", the knight occupies square (x,y). Finally, a constraint ensures that every square on the board is visited exactly once within a total number of steps equal to the number of squares.

```
% Board size
#const n = 8.

% Board
square(1..n, 1..n).

% Knight moves
move(X1,Y1,X2,Y2) :- square(X1,Y1), square(X2,Y2), 1 = |X1-X2|, 2
    = |Y1-Y2|.
move(X1,Y1,X2,Y2) :- square(X1,Y1), square(X2,Y2), 2 = |X1-X2|, 1
    = |Y1-Y2|.

% Steps
step(1..n*n).

% Initial position at step 1
visit(1,1,1).

% For each step > 1, the knight must move from the previous square
      to a new square via a legal knight move.
% The choice rule ensures exactly one move is made at each step.
{ visit(X2,Y2,Z2) : visit(X1,Y1,Z1), move(X1,Y1,X2,Y2), Z2 = Z1 +
    1 } = 1 :- step(Z2), Z2 > 1.

% Constraint: no square may be visited more than once.
:- visit(X,Y,Z1), visit(X,Y,Z2), Z1 != Z2.

```

```

#show visit/3.
```

Listing 1: Methode 1

4.1.2 Method 2

The second method is based on reachability. For each square on the board, a possible predecessor square is selected (except for the starting square). The final solution only includes configurations that satisfy two conditions:

- Every square must be reachable from the start (no isolated loops or disconnected regions, such as two squares pointing only to each other).
- Each square can have at most one outgoing edge, ensuring the structure forms a single continuous path rather than branching.

This method differs most from method 1 in that no time/step is kept track of.

```

#const n = 8.

% Board
square(1..n, 1..n).

% Knight moves
knight_move(X1,Y1,X2,Y2) :- square(X1,Y1), square(X2,Y2), 1 = |X1-X2|, 2 = |Y1-Y2|.
knight_move(X1,Y1,X2,Y2) :- square(X1,Y1), square(X2,Y2), 2 = |X1-X2|, 1 = |Y1-Y2|.

% Fix starting square
start(1,1).

% At most one incoming knight move per square except for the start
1 { route(X1,Y1,X2,Y2) : knight_move(X1,Y1,X2,Y2) } 1 :- square(X2,Y2), not start(X2,Y2).


% Each square must be reachable from the start
reachable(X,Y) :- start(X0,Y0), route(X0,Y0,X,Y).
reachable(X,Y) :- reachable(X1,Y1), route(X1,Y1,X,Y), not start(X,Y).

% each square must be reachable (start not included)
:- A < n*n-1, A = #count { X1,Y1 : reachable(X1,Y1) }.

% there must be n*n-1 routes (edges) in total and from each square exactly one outgoing edge
:- N < n*n-1, N = #count { X1,Y1 : route(X1,Y1,_,_) }.

#show route/4.
```

Listing 2: Methode 2

4.2 Performance of the methods

Both methods are asked to generate five answers. The first takes 31.880 seconds to do this, while the second method takes 0.342 seconds. To better test the performance of the second model, the second method is asked to provide 10 000 answers, which takes 3.688 seconds.

To better understand performance, we look at the statistics. Although we cannot find any good documentation of CLINGO's statistics, we can make some assumptions.

- Choices are the number of branches the algorithm had to make.
- Conflicts would be the number of times the solver should backtrack.

Method 2 has significantly lower values than method 1, which may also explain the process time.

4.2.1 Method 1 - Performance

"choices": 1546701.0 "conflicts": 370082.0

4.2.2 Method 2 - Performance

"choices": 27908.0 "conflicts": 23864.0

4.3 heuristics

Heuristics have been explored to potentially solve the problem faster. There is a well-known heuristic for the knight tour: Warnsdorff's algorithm [3]. This heuristic should ensure that the algorithm first searches for combinations, whereby at each step the step is chosen from which the fewest possible future steps are possible.

The heuristic was implemented in method 1 as follows:

```

% Board size
#const n = 8.

% Board
square(1..n, 1..n).

% Knight moves
move(X1,Y1,X2,Y2) :- square(X1,Y1), square(X2,Y2), 1 = |X1-X2|, 2 = |Y1-Y2|.
move(X1,Y1,X2,Y2) :- square(X1,Y1), square(X2,Y2), 2 = |X1-X2|, 1 = |Y1-Y2|.

% Steps
step(1..n*n).
```

```

% Initial position at step 1
visit(1,1,1).

% Define valid target squares for heuristic
valid_target(X3,Y3,Z3,Z2) :- move(X2,Y2,X3,Y3), visit(X2,Y2,Z2),
    visit(X3,Y3,Z3), Z3 > Z2.
heuristic(H,X2,Y2,Z2) :- visit(X2,Y2,Z2), H = #count { X3,Y3 :
    valid_target(X3,Y3,Z3,Z2)}.


% For each step > 1, the knight must move from the previous square
     to a new square via a legal knight move.
% The choice rule ensures exactly one move is made at each step.
{ visit(X2,Y2,Z2) : visit(X1,Y1,Z1), move(X1,Y1,X2,Y2), Z2 = Z1 +
    1 } = 1 :- step(Z2), Z2 > 1.

% Constraint: no square may be visited more than once.
:- visit(X,Y,Z1), visit(X,Y,Z2), Z1 != Z2.

#heuristic visit(X2,Y2,Z2) : heuristic(H,X2,Y2,Z2). [-H@8, true]

#show visit/3.
```

Listing 3: Methode 1 with Warnsdorff's algorithm

The solver takes longer (89.270s) when the heuristic is implemented this way, even though the statistics show less backtracking and fewer choices being reviewed. "choices": 579352.0 "conflicts": 100768.0 This might be explained by the fact that the heuristic needs to be recalculated repeatedly. As a result, each node will take longer. This leads to a better choice of nodes, but at the cost of additional computations (for the heuristic).

4.4 Adding rules for inprovement of the performance

4.5 Multi shot solving

Multi-shot solving was explored to see if it could improve performance. The idea behind multi-shot solving is to break the problem into smaller subproblems that can be solved sequentially. For the knight's tour, this could involve solving for smaller sections of the board and then combining these solutions. This approach for the Knight's tour problem is described in detail in [4]. The technique of multi-shot solving is described in [2]. A base program is created that defines the basic rules for the problem to solve. Then an additional program is created that adds specific constraints or goals for each subproblem. When this approach is applied to the knight's tour problem, the challenge lies in finding and defining appropriate subproblems that can be solved independently while still leading to a valid overall solution. Although the performance of solving the smaller subproblems will be faster, the number of found solutions will be less and thus incomplete compared to single shot solving. For an implementation example, a board is defined of size 10×10. This board is then split into four 5×5 sub-boards. The size is chosen because this minimal board size for an open knight's tour is 5×5. Open knight's tours can only be found on square boards of $n \times n$ with $n \geq 5$ as described in [2].

```
% Work in progress - multi-shot solving for knight's tour
```

Listing 4: Knight's tour multi-shot solving

5 Discussion

6 Conclusion

References

[1] Edeilson Milhomem da Silva Carvallho, Ary Henrique de Oliveira, Glenda Michele Botelho, and Glêndara Aparecida de Souza Martins. Two classic chess problems solved by answer set programming.

[2] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *CoRR*, abs/1705.09811, 2017.

[3] Kartik. Warnsdorff's algorithm for knight's tour problem, July 2025. Accessed: 2025-10-21.

[4] Ian Parberry. An efficient algorithm for the knight's tour problem. *Discrete Applied Mathematics*, 73(3):251–260, 1997.