INFO9015–1 Logic for Computer Science

Project Report — Knight's Tour SAT solving

Teacher:
Pascal Fontaine

Author:
Valérian Wislez s200825

Teaching Assistant:
Tom Clara

Academic Year 2025–2026

1 Introduction

This report describes the way each question of the project has been solved as well as the results obtained. A more comprehensive view on the submitted source code can be found at https://github.com/valerian20024/knight_tour_SAT [1].

2 Naive solution

Let M and N be the dimensions of the board and $T = M \cdot N$ the total number of moves. For every cell (i, j) $(0 \leq i < M,\ 0 \leq j < N)$ and every time step t $(0 \leq t < T)$ we introduce a Boolean variable representing the knight's position at a given timestep:

$$x_{i,j,t}$$

The first attempt at solving the problem involves the following set of constraints:

1. A cell can only be occupied at one timestep.[2] That is, there must be at least one time step for visiting a cell, and visiting a cell at a previous time step t_1 enforces to not revisit it at time step $t_2 > t_1$. Formally,

$$\bigvee_{t=0}^{T-1} x_{i,j,t} \tag{1}$$

$$\neg x_{i,j,t_1} \vee \neg x_{i,j,t_2} \tag{2}$$

2. For a given timestep, the knight can only occupy one cell. Similarly to the previoius constraints, one can force the knight to choose, for each timestep, at least one cell on the board. Then, forbid it to choose another cell for this time step. Formally,

$$\bigvee_{i=0}^{M-1} \bigvee_{j=0}^{N-1} x_{i,j,t} \tag{3}$$

$$\neg x_{i_1,j_1,t} \vee \neg x_{i_2,j_2,t} \tag{4}$$

3. Legal moves of the knight restrict its movement. Chess knights can move form a given time step t in 8 different (L-shaped) directions to another cell in $t + 1$ (provided it does not land outside the chessboard). Let L be the set of available legal moves of the knight at t. Formally,

$$\neg x_{i,j,t} \bigvee_{(i',j') \in L} x_{i',j',t+1} \tag{5}$$

One can go even a step further to help the SAT solver by specifying the reverse constraints. That is, if a given cell has been reached at $t + 1$, then the knight must have perform one of its legal moves at t and thus it was standing on another cell at t. Let L' be the moves available to the knight at time $t + 1$. Formally,

$$\neg x_{i',j',t+1} \bigvee_{(i,j) \in L'} x_{i,j,t} \tag{6}$$

[1] The repository has been made public *after* the submission deadline.

[2] To ensure only one constraint, one can simply ensure "at most one" and "at least one".

Each set of constraints are implemented in dedicated functions in the code.

Naive solutions[3] is able to solve 8×8 board in about 1.5 to 1.7 [s], depending on the knight start position. It can also solve up to 13×13 boards in less than 10 [s] but gets overwhelmed when reaching 15×15 boards (it takes more than a minute to find the solution)

3 Efficient solution

While the naive approach can solve decently sized chessboard in reasonable time on a mid-range laptop, it must use a quadratic number of clauses for encoding the "exactly one" family of clauses. A more efficient solution would benefit from having less clauses for reducing the search space of the solver. After doing some research about different encodings for the "at-most-one" family of constraints (refined to "exactly one" in this problem), the sequential counter encoding seemed to be one of the most effective (yet simple enough) [1]. The sequential counter works by assigning a set of new, *auxiliary variables* a_k for $k \in [0, n]$, whose size grows *linearly* with the size of the problem. These variables act as a chain that tracks whether a solution has been found previously. If it is the case, then the solver should know it cannot find any more solution. It will thus propagate the information through the a_k instead of enumerating all pairwise possibilities.

Formally, abstracting away the positions and time steps for a simpler mono-indexed set of variables, the key clauses are :

$$\bigvee_i x_i \qquad \text{There is at least one solution.} \tag{7}$$

$$\neg x_k \vee a_k, \qquad \text{A solution triggers the } a\text{, for } k \in [0; n-2] \tag{8}$$

$$\neg a_{k-1} \vee a_k, \qquad \text{Propagating the information through the } a_k\text{'s.} \tag{9}$$

$$\neg x_k \vee \neg a_{k-1}, \qquad \text{A new solution requires no previous solution.} \tag{10}$$

$$\tag{11}$$

This technique can be applied to both the one cell only and one time step only constraints.

However, it appears that the sequential counter encoding is not always better than the pairwise encoding, as reported by [2]. It has actually a high variance depending on the problem instance. It is sometimes better, sometimes worse than pairwise encoding. This can be solved by implementing a more clever version of the sequential encoding, which always beats the pairwise encoding. However, as the naive approach was sufficient for the scope of the project, sequential encoding has not been developed further.

Indeed, because of

4 Counting solutions

In order to count all solutions for a given board, one simply needs to enumerate all the models found by the SAT solver, for each couple $(i_0, j_0) \in M \times N$. Using the PySAT library, it is easy to do so by calling the `enum_models` method after solving the problem and simply count them.

Solving the 3×4 chessboard takes about 1 [s], finding 16 solutions, while the 5×5 chessboard takes 2 [s], finding 1728 solutions. Comparing theses results with the ones showed in [3], one can see that it's the right number of solutions. Unfortunately, it will not be possible to dare solving for larger boards in reasonable time on this hardware.

[3] on a Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz

5 Counting up to symmetry

The easiest way to count up to symmetry was to enumerate all the solutions, apply symmetries and check whether or not the solution is identical to another solution. This enumeration takes into account every possible solutions for every possible starting position for a given chessboard size, as for the previous counting technique. While this might not be the most effective nor elegant solution, it was fast enough for solving boards up to 5×5 in a few seconds. Solving the 3×4 chessboard takes about $1,2$ [s], finding 12 solutions, while the 5×5 chessboard takes 6.1 [s], finding 1568 solutions.

6 Finding additional constraints

First we enumerate all tours that satisfy the basic encoding. Each tour is converted into a path, i.e. a sequence of positions indexed by time step t, and duplicate paths are discarded. From the remaining distinct tours, we randomly select one to be the unique tour that will survive. This guarantees fairness because different runs or different starting cells typically choose different winners. Finally, we iterate over the other tours in arbitrary order: for each alternative that is still compatible with the clauses (constraints) added so far, we locate the earliest time step $t \geq 1$ at which it diverges from the chosen tour and force the knight to occupy the chosen tour's position at that step by adding the unit clause $X_{i,j,t}$. Because a new clause is introduced only when an alternative would otherwise remain satisfiable, every added clause is necessary: removing it would immediately allow that alternative (and possibly others) back.

References

[1] Steffen Hölldobler and Van Hau Nguyen. An efficient encoding of the at-most-one constraint. Technical Report KRR Report 13-04, Technische Universität Dresden, Knowledge Representation and Reasoning Group, Dresden, Germany, 2013. Accessible at: https://iccl.inf.tu-dresden.de/w/images/d/d7/Hn-tr-2013.pdf. Accessed: 2025-11-13.

[2] Joao Marques-Silva and Inês Lynce. Towards robust cnf encodings of cardinality constraints. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, Providence, RI, USA, 2007. Springer-Verlag. Available at https://sat.inesc-id.pt/~ines/publications/cp07.pdf. Accessed: 2025-30-11.

[3] Wikipedia contributors. Problème du cavalier. https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_cavalier#D%C3%A9nombrement_des_solutions, 2025. Accessed: 2025-11-13.