
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 paper describes the way each question of the project has been solved. A more comprehensive view on the submitted source code can be found at [3] [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,

[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".

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

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,

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

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

## 3 Efficient solution

While the naive solution works well in general, for boards ranging up to . . . x . . . it has a quadratic number of constraints for encoding the one cell and one time only. A more efficient solution would benefit of having less constraints for reducing the search space of the solver. After doing some research about different encodings [1] for the "at-most-one" family of constraints (refined to "exactly one" in this problem), it appeared the naive encoding is actually known as "pairwise encoding", and the sequential counter seems to be the one most effective encodings there is. The sequential counter works by assigning a set of new, auxiliary variables, whose size grows *linearly* with the size of the problem.

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

However, it appears that the sequential counter encoding is not always better than the pairwise encoding, as reported by [1] Indeed, because of

This is implemented in

## 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.

Comparing the results obtained when solving $3 \times 4$ as well as 5 chessboards with the ones showed in [2]

This is implemented in

## 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. While this might not be the most effective nor elegant solution, it was fast enough for solving boards up to $5 \times 5$ in a few seconds.

This is implemented in

## 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] Holger H. Hoos and Thomas Stützle. Sat solvers for the knight's tour problem. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, pages 723–731. Springer, 2007.

[2] 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.

[3] Wislez Valérian. knight_tour: Project for sat solving the knight's tour problem. https://github.com/valerian20024/knight_tour_SAT, 2025. GitHub repository, Accessed: 2025-11-13.