Knights Tour

Ashwin Karthikeyan
Hriday Chheda
Yifan Ruan

January 6, 2024

1 Abstract

SAT solvers have made impressive advances in recent years in scalability, enabling increasingly large combinatorial problems to become tractable - both decision and counting versions. An important decision that can greatly affect this scalability is the encoding method used to generate the CNF formula used. While there is a body of work studying this problem for regular SAT, it is not well studied within the domain of model counting. In this work, we examine how different encodings perform on the approximate counting version of a well-known combinatorial problem, the Knight's Tour, and compare relative performances with the decision version.

2 Introduction

When using SAT solvers to tackle problems, one must encode the problem into a propositional formula in Conjunctive Normal Form (CNF). For any given problem, there are many different ways to do so. It is known that the choice of such an encoding method can have great effects on the performance of a solver on any given problem. In the past few decades, there have been a variety of encodings that have been proposed and studied. However, despite recent advancements in approximate model-counting, not much work has been done to investigate how encodings can affect performance in this problem setting.

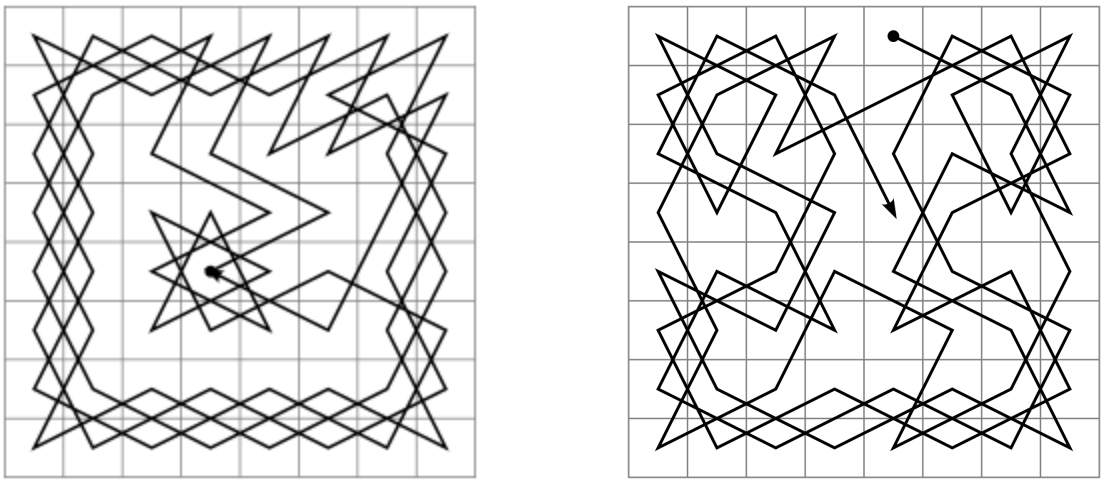
We focus on Knight's tour, a well-known subclass of the Hamiltonian cycle path finding problem. There has been work examining the optimality of different encodings for the decision version of this problem [1]. In this work, they find that the binary encoding with preprocessing is the clear winner for the decision version of the problem.

2.1 Problem Statement

The Knight's Tour is the problem of finding a path a knight can take on the chess board that visits each square on the board exactly once. The chessboard can be formulated as a graph where each square is a node and an edge exists between any pair of squares that can be visited consecutively by a knight, which frames the problem as a subclass of the Hamiltonian path problem.

A path is said to be closed when the starting and ending square are neighbors. Otherwise, it is said to be open.

This problem can be generalized to differently shaped boards - we will focus on the $n \times n$ case, which presents a good benchmark for testing the performance and scalability of different encoding methods. It is well-known that odd instances of the closed knight's tour problem have no solutions because such boards have different numbers of white and black squares, and each square is only adjacent to squares of the other color.



(a) A closed tour

(b) An open tour

Figure 1: Examples of paths a knight can take

3 Method

3.1 Direct Encoding

Let V_{ij} be a variable that indicates whether square i on the $n \times n$ chess board (numbered left to right from 0 to $n^2 - 1$) is the j^{th} node on the Hamiltonian path starting at a given start node. Let $N(i)$ denote the set of neighbouring squares for square i. For example, if $i = 0$, $N(i) = \{10, 17\}$.
An obvious choice for encoding the knight's tour problem as a CNF formula under the given setup is to take the conjunction of the following subformulas:

1) $\forall j \in \{0, \ldots, n^2 - 1\}, \left(\bigvee_{i=0}^{n^2-1} V_{ij} \right)$

2) $\forall j \in \{0, \ldots, n^2 - 1\}, 0 \leq s \leq t \leq (n^2 - 1), \left(\neg V_{sj} \vee \neg V_{tj} \right)$

3) $\forall i \in \{0, \ldots, n^2 - 1\}, \left(\bigvee_{j=0}^{n^2-1} V_{ij} \right)$

4) $\forall j \in \{0, \ldots, n^2 - 1\}, 0 \leq s \leq t \leq (n^2 - 1), \left(\neg V_{is} \vee \neg V_{it} \right)$

5) $\forall i \in \{0, \ldots, n^2 - 1\}, \forall j \in \{0, \ldots, n^2 - 2\}, \left(\neg V_{ij} \bigvee_{k \in N(i)} V_{k(j+1)} \right)$

6) V_{p0}. Where p is the start node. Note: If we want to make this a closed tour, we can add the following clause: $\left(\bigvee_{k \in N(p)} V_{k(n^2-1)} \right)$

Subformulas (1) and (2) ensure that for all positions j on the Hamiltonian path, exactly one square takes that position. Subformulas (3) and (4) ensure that for all squares i on the board, the square appears exactly once on the Hamiltonian path. Subformula (5) ensures that if the knight is on square i at the j^{th} node on the Hamiltonian path, then the knight performs on of its valid moves to get to the next square, and that square is the $j + 1^{th}$ node on the Hamiltonian path. Finally, subformula (6) ensures the desired start node (optionally the desired tour type: closed/open). Note that a heuristic that we use to compute closed tours is picking the left top corner square as the starting square and picking on of its neighbours as the last square (the neighbours of the corner square are reflections of each other).

3.2 Distance Encodings

We can consider each square on the board as a vertex and edges between two squares if there is a valid knight move between the two. This way we can create a graph, G where the vertex set is the squares on a chess board and edge set is the pair of squares between which there is a valid knight transition. Finding a knights tour on the chess board is equivalent to finding a Hamiltonian Cycle in G. Consider a board with dimensions $\sqrt{n} \times \sqrt{n}$, then the number of squares are n and equivalently the number of vertices in G are n. The distance encoding uses a matrix H of size $(\sqrt{n} \times \sqrt{n})$ of Boolean variables where, H_{ij} represents that the edge (i, j) is on the Hamiltonian cycle (equivalently the knights tour). Only H_{ij} with valid knight transitions need to be assigned a variable because if there is no valid knight transition from square i to square j then we know that (i, j) cannot be in the Hamiltonian cycle. In other words, if (i, j) is not a valid knight move then $H_{ij} = 0$. Therefore, the number of variables in H, is twice the number of edges in G.
The following two constraints follow directly from the fact that H represents a Hamiltonian cycle:

$$\forall i \in \{1, ..., n\}, \sum_{j=1}^{n} H_{ij} = 1 \tag{1}$$

$$\forall j \in \{1, ..., n\}, \sum_{i=1}^{n} H_{ij} = 1 \tag{2}$$

The constraints (1) and (2) simply ensure that for every vertex (square) there is exactly one incoming and outgoing edge on the Hamiltonian cycle.
While H could satisfy constraints (1) and (2), it could still have subcycles and not be a Hamiltonian cycle. To enforce that no sub-cycles are allowed, the distance encoding adds position based constraints such that each vertex is mapped to a unique position on the Hamiltonian cycle. Let $p(i)$ be the position of vertex i, $s(p)$ denote the successor of position p, and $s^k(p)$ be the k^{th} successor of p. The following constraints ensure that the cycle starts at vertex 1 and ends at vertex 1:

$$\forall i \in \{2, ..., n\},\ H_{1i} \Rightarrow p(i) = s(1) \tag{3}$$

$$\forall i \in \{2, ..., n\},\ H_{i1} \Rightarrow p(i) = s^{n-1}(1) \tag{4}$$

Constraint (3) and (4) ensure that for all neighbours of the start vertex in the hamiltonian cycle that at least one of them is the second vertex of the cycle and at least one of them is the last vertex of the cycle.
The following constraints ensure that each vertex is positioned successively in the cycle:

$$\forall i \in \{2, ..., n\}, j \in \{2, ..., n\}, i \neq j, H_{ij} \Rightarrow p(j) = s(p(i)) \tag{5}$$

Constraint (5) ensures that if edge (i, j) is a part of the Hamiltonian cycle then vertex j should be the successor of vertex i.
Constraint (1)-(5) ensure that H is a Hamiltonian cycle and this is the structure of the distance based encoding for the knights tour (equivalently Hamiltonian cyle) problem. There are multiple ways to implement the successor function and its corresponding variables. One way is to introduce a boolean variable corresponding to every possible vertex-position combination (like the direct encoding), we describe this as the unary encoding below. Another attempt is to consider introducing only enough boolean variables to capture the binary representation of positions corresponding to each vertex. We describe this as the binary encoding below.

3.2.1 Unary Encoding

The unary encoding introduces a boolean variable, $U_{s,p}$, for each square, s, and path position, p, where $U_{s,p}$ indicates that the square s is visited at position p in the path.

As such, the successor function is encoded as follows (as described in [1]):

$$\forall i \in \{2, ..., n\} : H_{1i} \Rightarrow U_{i2}$$

$$\forall i, j \in \{2, ..., n\}, i \neq j, \forall p \in \{2, ..., n-1\} : H_{ij} \wedge U_{ip} \Rightarrow U_{j(p+1)}$$

and for the closed tour case to enforce ending at an adjacent square to the starting square:

$$\forall i \in \{2, ..., n\}, H_{i1} \Rightarrow U_{in}$$

3.2.2 Binary Adder Encoding

The binary encoding uses a log-encoded domain variable for position P_i for each vertex i where the domain is the set of all possible positions for vertex i. For example, if we consider the 8×8 chess board then the knights tour has 64 squares and 64 possible positions on the tour. Then for each vertex (square), we need $log_2(64) = 6$ boolean variables to encode the possible positions of the vertex. Where the 6 boolean variables represent the coefficients of the binary expansion of the position.
Using this log-encoding we can encode (3)-(5) as the following (assuming $P(1) = 0$):

$$\forall i \in \{2, ..., n\},\ H_{1i} \Rightarrow P(i) = 1 \tag{6}$$

$$\forall i \in \{2, ..., n\},\ H_{i1} \Rightarrow P(i) = n - 1 \tag{7}$$

$$\forall i \in \{2, ..., n\}, j \in \{2, ..., n\}, i \neq j, H_{ij} \Rightarrow p(j) = p(i) + 1 \tag{8}$$

Now all that is left to define is how we encode $p(j) = p(i)+1$ from constraint (8) in CNF. Consider $P(i)$ is encoded by variables $\langle X_{m-1}, ..., X_1, X_0 \rangle$ and $P(j)$ is encoded by variables $\langle Y_{m-1}, ..., Y_1, Y_0 \rangle$. We want to encode:

$$\begin{array}{rr} & X_{m-1}...X_1X_0 \\ + & 1 \\ \hline & Y_{m-1}...Y_1Y_0 \end{array}$$

This can be encoded using:

- $Y_0 = \neg X_0$ (2 clauses)
- $(X_0 \Rightarrow Y_1 = \neg X_1) \wedge (\neg X_0 \Rightarrow Y_1 = X_1)$ (4 clauses)
- $\forall i \in \{2, ..., m-1\}, (\neg Y_{i-1} \wedge X_{i-1} \Rightarrow Y_i = \neg X_i) \wedge (Y_{i-1} \vee \neg X_{i-1} \Rightarrow Y_i = X_i)$ (6 clauses for each i)

The above logic is essentially encoding the carry in adding one to a binary number. Additionally we know that:

- $(Y_i = \neg X_i) \equiv ((Y_i \vee X_i) \wedge (\neg Y_i \vee \neg X_i))$
- $(Y_i = X_i) \equiv ((Y_i \vee \neg X_i) \wedge (\neg Y_i \vee X_i))$

We can use all of the above to encode $P(j) = P(i) + 1$ in CNF.
The binary encoding, uses constraint (1), (2) and (6) - (8) to ensure that H is a hamiltonian cycle. The number of variables included in H is $\mathcal{O}(nd)$ where d is the maximum degree in G and n is the number of squares. The number of binary position variables is $\mathcal{O}(nlog_2(n))$ because for every position we add $log_2(n)$ variables. Therefore, the number of variables in the binary encoding is in $\mathcal{O}(nlog_2(n))$. The number of clauses is dominated by constraint (8), which is $\mathcal{O}(n \times d \times log_2(n))$, the $\mathcal{O}(log_2(n))$ comes from the number of clauses needed for $P(j) = P(i) + 1$ and we add these many clauses for every possible edge (i, j) in both directions, hence we get at most $n \times d$ of such additions.

3.3 Preprocessing

As discussed in [1], we experimented with using a limited amount of preprocessing to reduce the search space for the SAT solver. However, we were only able to get this working with the unary encoding.

The idea is to not include variables that will never be true. Since it is highly time-consuming to identify whether a there exists a path of length l from a start square to every other square for every l in range $1, n \times n$, we follow [1] by calculating the shortest path to each square on the board and omitting variables that encode impossible combinations of square and position.

For instance, if the minimum path length from the starting square to square i is m, then the variables U_{ij} under the unary encoding where $j \leq m$ and $j > n^2 - m$ are impossible and can thus be omitted from the propositional formula.

We found that preprocessing somewaht improved solve times for the unary encoding, so we used it in our experiments.

4 Experiments and Results

The base code used for this section can be found here: https://github.com/yifanr/knights . Modifications to the code required for experiments are described in the corresponding subsections.

4.1 Size of Encodings

Size of board	Direct Encoding		Unary Encoding		Binary Encoding	
	# of Variables	# of Clauses	# of Variables	# of Clauses	# of Variables	# of Clauses
6×6	1296	46692	1291	22784	370	5605
8×8	4096	262208	4055	127236	714	12161
10×10	10000	1000100	9855	482250	1269	24669
12×12	20736	2986128	20375	1433980	2024	43385
14×14	38416	7529732	37699	3610940	2808	62021

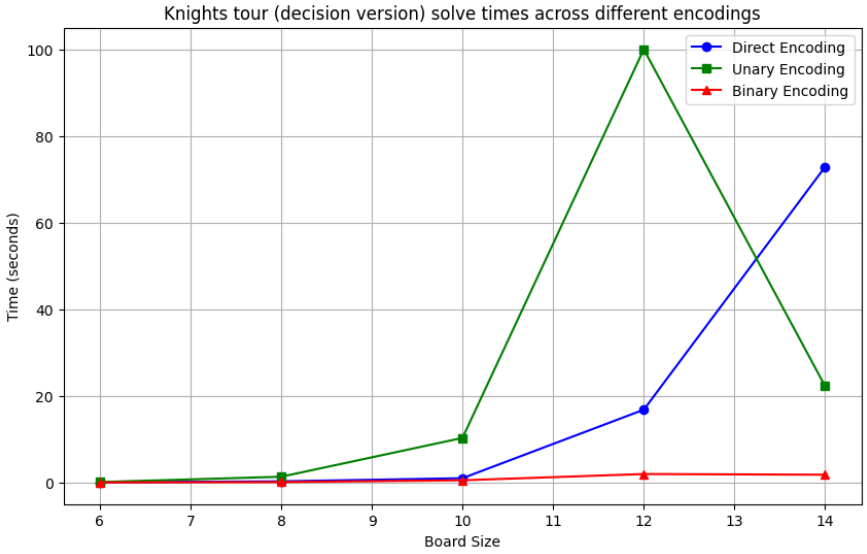
Table 1: Comparison of size of different encodings across various board sizes

As expected, the binary encoding has the smallest number of variables and clauses followed by the unary encoding followed by the largest number of variables and clauses in the direct encoding.
However, note that the difference in number of variables for unary and direct is very small but the number of clauses are roughly halved going from direct to unary.

4.2 Solve time (decision version)

Next, we compare the solve times of the decision version of the knights tour problem across the three encodings.

We used the SAT solver Cadical153 from the python pysat library on a Macbook Air (2023) machine with M2 chip for this experiment. As expected, the binary encoding (whose encoding size is much smaller) has the lowest solve time with a less than 2 second solve time for all instances up to a 14×14 board size. This is expected because the number of variables and number of clauses for the binary encoding are significantly lesser than the direct or unary. We were also expecting the solve time for unary to be lower than direct but to our surprise, the direct encoding is usually faster than the binary encoding (except on the last instance). We expected it to be the opposite especially since the number of clauses are approximately halved in the unary encoding when compared to the direct encoding. We are unsure why this is the case but our hunch is that the learning mechanism of the solvers might be less efficient in unary encodings due to the nature of the conflicts encountered. While we have no way of guaranteeing this could be the reason, it could be an interesting direction for future work.



4.3 Counting Knights Tour

Note that minor adjustments to the encodings above allow us to use these encodings for open and closed tours. For the direct encoding, we can perform open tour counting by following not including the clause $\left(\bigvee_{k\in N(p)} V_{k(n^2-1)}\right)$. For unary and binary encodings, we add a constraint that allows every node that is not the start node to have a valid move to the start node. This combined with the constraints for path length ensure that the returned satisfying assignment for the CNF formula is indeed a hamiltonian path. Additionally, we remove any preprocessing that doesn't allow for open tours to be included.

Now, we use the encodings listed above to get an approximation on the number of all knight's tours (open and closed) starting at square 0 (or equivalently, square (0,0)) for board size 5×5, and use the unmodified encodings to find the number of closed knight's tours for board size 6×6. The following results were produced with the following code setup:

(5×5) **All encodings:** $\epsilon = 0.8,\ \delta = 0.2$ projected on all variables.
Timeout: 10 secs

(6×6) **Direct:** $\epsilon = 0.8,\ \delta = 0.2$ projected on variables corresponding to the $4^{th}, 8^{th}, 16^{th}, 20^{th}, 24^{th}$ and 32^{nd} nodes of the circuit.
Unary: $\epsilon = 0.8,\ \delta = 0.2$ projected on every 4^{th} variable starting at 1 till 157.
Binary: $\epsilon = 0.8,\ \delta = 0.2$ projected on first 161 variables.
Timeout: 210 secs

(8×8) **Direct:** $\epsilon = 0.8,\ \delta = 0.2$ projected on variables corresponding to the $12^{th}, 24^{th}, 36^{th}, 48^{th}$ and 60^{th} nodes of the circuit.
Unary: $\epsilon = 0.8, \delta = 0.2$ projected on every 16^{th} variable starting at 1 till 305.
Binary: $\epsilon = 0.8, \delta = 0.2$ projected on every 4^{th} variable starting at 1 till 317.
Timeout: 5000

Note that all closed tours are counted no matter what start node we pick. Now, for every satisfying assignment on the closed tour variant, the corresponding closed tour is counted twice as the tour with the opposite direction is a different satisfying assignment. So, we divide the approximate count by 2 (Note: We should not divide by 2 for the direct encoding because the heuristic mentioned at the end of section 3.1 accounts for reflections/the opposite direction). The choice of 161 for Unary and Binary (in 6×6) is because the H matrix contains the first 160 variables. The choice of stopping at 317 is because the H matrix has 336

variables and we wanted to project onto a subset of variables in the H matrix.

The system setup for the results are as follows:

Device: Dell inspiron 15 7567 laptop.
Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz.
RAM: 16GB.

The results are as follows:

Size of Board (All/Closed)	Exact number of knights tours	# with Direct Encoding (Time - secs)	# with Unary Encoding (Time - secs)	# with Binary Encoding (Time - secs)
5×5 (All)	304	288 (< 1)	296 (12)	304 (6)
6×6 (Closed)	9,862	6656 (175)	6,400 (143)	9,216 (41)
8×8 (Closed)	13,267,364,410,532	16,777,216 (1907)	Timeout	755,914,244,096 (4,529)

Table 2: Comparing the performance of approximate counting the number of knight tours using different encodings. (Exact count for 6×6 and 8×8 is from [2] and [3])

For 8×8, closed tours, if the direct encoding was projected onto variables corresponding to the $12^{th}, 24^{th}, 32^{nd}, 36^{th}, 48^{th}$ and 60^{th} nodes of the circuit, then the runtime jumps to 20652 seconds. The approximate count obtained was 385,875,968.

5 Discussion and Future Work

The Binary Adder encoding had the shortest solve time across all encodings explored in this project. Moreover, the Binary Adder encoding scaled well and performed with highest accuracy across all other encodings compared in this project in the approximate counting version of the problem.

We would like to explore the following directions of work for this problem:

1) Testing the effects of preprocessing as mentioned in section 3.3 for different encodings, comparing its effects on the decision and counting versions of the problem.

2) Finding more effective heuristics for picking variables to project on while using ApproxMC.

3) Finding a heuristic for the order of variables for performing BCP for each encoding.

6 References

[1] Zhou, NF. (2020). In Pursuit of an Efficient SAT Encoding for the Hamiltonian Cycle Problem. In: Simonis, H. (eds) Principles and Practice of Constraint Programming. CP 2020. Lecture Notes in Computer Science(), vol 12333. Springer, Cham. https://doi.org/10.1007/978-3-030-58475-7_34

[2] Alwan, K., & Waters, K. (1992). Finding re-entrant knight's tours on n-by-m boards. ACM Southeast Regional Conference: Proceedings of the 30th Annual Southeast Regional Conference; 08-10 Apr. 1992, 377–382. https://doi.org/10.1145/503720.503806

[3] Brendan McKay, D. (1997). Knight's tours of an 8×8 chessboard. [Working/Technical Paper]. Australian National University. http://hdl.handle.net/1885/40759