Generalised Euler's knight

Nicolas Beldiceanu, Eric Bourreau, Helmut Simonis, Abderrahmane Aggoun

Generalised Euler's knight

A. Aggoun, N. Beldiceanu, E. Bourreau, H. Simonis
COSYTEC SA, Parc Club Orsay Université
4, rue Jean Rostand, F-91893 Orsay Cedex
cosytec@cosytec.fr

Abstract

Euler, Vandermonde, Dudeney, Schwesk, Berliner, Conrad and many others already considered knight's tours on chessboards. The classical knight's tour problem consist of finding out on a $N \times M$ chessboard a sequence of legal knight moves that visit every cell exactly once and finish by returning to the initial cell. A more challenging question is to generalise the problem to more than one knight. More precisely, we search for a partitioning of the m x n chessboard by a set of C cycles in such a way that each cell belongs to one single cycle. Moreover we impose all the cycles to be balanced. Since a cycle can't have an odd number of cells, we enforce that each cycle visits between 2 x floor(floor((m x n) / c) / 2) and 2 x cell(cell((m x n) / c) / 2) cells. We systematically consider all the boards m x n such that 1<m<11, 1<n<11, (m x n) mod 2=0 and we solve the problems for the number of knights c varying between 1 and floor((m x n) / 2). We show how to express this problem with CHIP and prove that for more than one quarter of the 820 instances there is no balanced solution (there is for instance no solution for 6 knights on a 6x6 board such that each knight visits exactly 6 cells).

Introduction

Mathematical puzzles around knight's path on a chessboard have always excited magazine and curious minds. The most famous problem is from Euler's memory (1759): *"Solution d'une question ingénieuse qui ne parait soumise à aucune analyse", "Solution to an ingenious question which look like without any possible analysis"*. Diderot encyclopaedia refer it to an old Indian problem where a knight must travel on each case of a 8x8 chessboard and come back to the start without pass two times on the same place. Euler, and may other mathematicians (Gianutio, De Moivre), solve it with constructive procedures. Berge models the problem also with the graph theory [Ber73], ALICE [Lau76] solve it with Artificial Intelligence and Constraint Reasoning without any choice point to find the first solution. We generalise this problem when a $N \times M$ chessboard must be cover by C knights. Systematic tests have shown those maximum-balanced solutions between length of knight's paths impose difficulty to find existence of solution. We have most of a third of problems without well-balanced solutions. If we relax the balanced condition, some problems always haven't any solutions. But if we increase both size of the chessboard, solutions are more often available. We hope this work will help to perfectly known which have and which have not solutions.

The rest of the paper is organised as follows. Next section explains how to model the problem like a graph problem with limitation. Then we explain how we solve it with the CHIP system and finally we give a table to illustrate the results.

Model

We will model the knight's path like a walk on graph. The goal is to find a hamiltonien cycle on the move's graph [Ber73]. Let's consider a 4x4 chessboard, we associate to it a graph G=(V,E) where V is vertices set (the chessboard cases) and E is the edges set, an edge connect to node if it exist possibility to pass from one to an other by a knight's move.

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

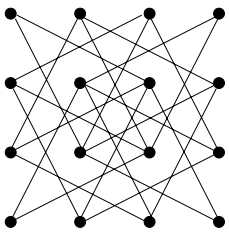
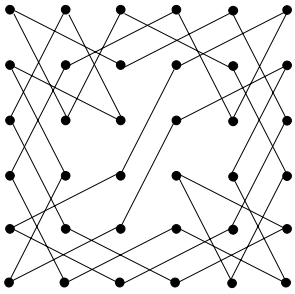


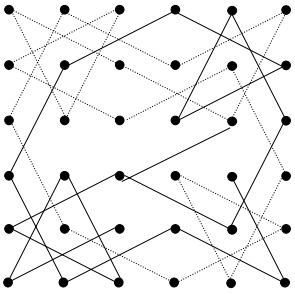
Figure 1. Associated graph of a4x4 chessboard

If we have several knights, we look for a graph's partition between distinct cycles as every node must be includes in one cycle. Then we try to balance length of cycles. This length is the number of visited cases divided by the number of knights. If this result isn't an integer, we authorise a length between floor and cell. In fact the length of a knight's tour must be an even number. Knight move swap from a white case to a black case, then to do a tour we will back on the same colour, then we have an even number of moves. We will restrict the length between the two even integers around the result. Then with C knights covering a $N \times M$ chessboard, number of moves must be between $2 \times \left\lfloor \frac{\left\lfloor \frac{(N \times M)}{C} \right\rfloor}{2} \right\rfloor$ and $2 \times \left\lceil \frac{\left\lceil \frac{(N \times M)}{C} \right\rceil}{2} \right\rceil$.

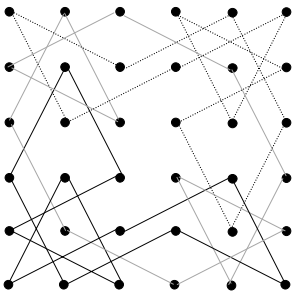
Let's follow with an example of a 6x6 chessboard. We have in figure 2, solutions example for number of knights between 1 to 5. We give in parenthesis for each solution, length of different cycles for each knight. It is interesting to notice than for a 6x6 chessboard, we haven't any solution well balanced where each knight do exactly 6 moves.



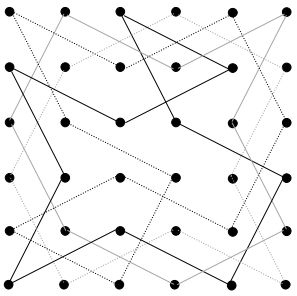
1 knight (36)



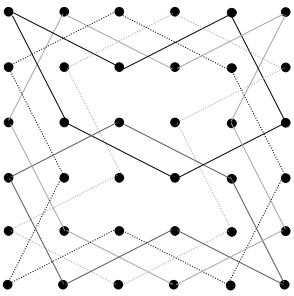
2 knights (18,18)



3 knights (12,12,12)



4 knights (8,8,10,10)



5 knights (6,6,8,8,8)

Figure 2. Examples of solutions on a 6x6 chessboard

Resolution

In the CHIP constraint programming language [Din88], it exist a global constraint call *cycle* [Bel94] which want to satisfy on an oriented graph G a perfect covering by C circuits. Our problem can be exactly express with this constraint when we give the number of knight C and the move's graph associated to the knight's moves on a $N \times M$ chessboard. We can also constraint the maximum and minimum number of visited cases for all knights.

The following figure shows a typical use of a *cycle* constraint. After the post, deductions already exist and some inconsistent edges are already remove. The goal is to keep consistent values for variables attach to the constraint. When all the variables are assigned, the constraint is satisfied.

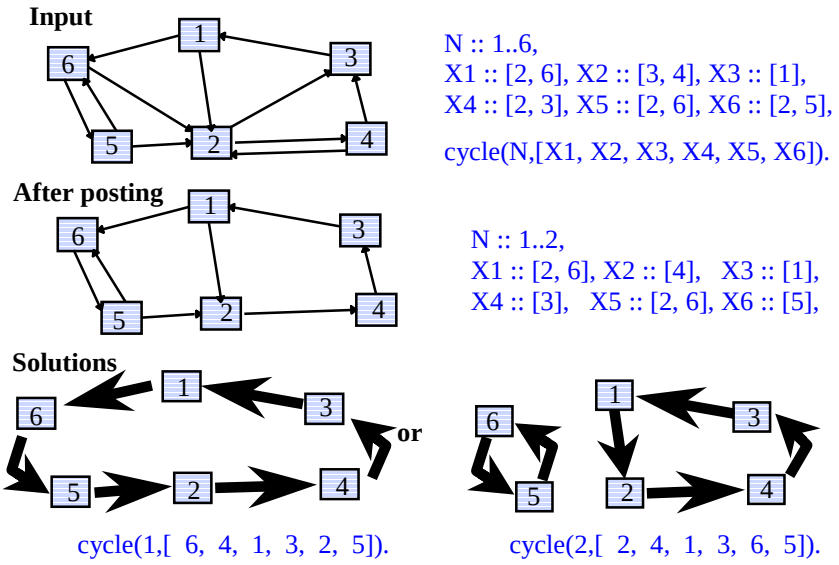


Figure 3: example of the *cycle* constraint

The move's graph proposes to the *cycle* constraint is an oriented graph. Each possible move between a case a to a case b is traduce by an arc from a to b and an arc from b to a. Chessboard cases are numbered in increasing way from value 1 (bottom left) to the right direction. Each node of the graph (corresponding to a case) have a domain variable where initials values are the indexes of cases where move for a knight are authorised.

For the 4x4 chessboard with 4 knights we have the following CHIP program:

```
top(List) :-
  List = [C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16],
  C1  :: [7,10],    C2  :: [8, 9,11],    C3  :: [5,10,12],    C4  :: [6,11],
  C5  :: [3,11,14], C6  :: [4,12,13,15], C7  :: [1, 9,14,16], C8  :: [2,10,15],
  C9  :: [2, 7,15], C10 :: [1, 3, 8,16], C11 :: [2, 4, 5,13], C12 :: [3, 6,14],
  C13 :: [6,11],    C14 :: [5, 7,12],    C15 :: [6, 8, 9],    C16 :: [7,10],
  length(Move,16),  Move :: 1..1,
  cycle(4,Liste,Move,4,4),
  labeling(Liste,0,first_fail,indomain).
```

For the enumeration processing, we use two approaches. First we use the predefined CHIP predicate, *labeling*/4. The predicate *labeling* want two strategies. First, one to choose which variable will be instantiated between all remaining free variables. Second, we describe the order to fix the different possible values to this variable. This assignment is done in a non-deterministic way, then we have a automatic depth-first search describe by the *labeling* predicate.

We define the strategy by choosing variable with the smallest number of possible value and we assign them in an increasing way. This approach is only based on the constraint paradigm, and didn't take in account the problem models with constraints. Only propagation and automatic domain reduction will reduce the combinatorial aspect.

The request `'top(List)'` return the result:
List = [**7**, 8, 12, 11, 3, 4, **16**, 15, 2, **1**, 13, 14, 6, 5, 9, **10**] as first solution.

This mean that first variable take value 7, second variable take value 8 and so one. Then if we want to reconstruct the travel of a knight starting in 1 we have: start from case 1 go to value 7, on case 7 go to case 16, on case 16 go to case 10 and in case 10 go to case 1.

This simple method gives already good results. In fact the strategy choose variables in very independent way compare to the physical representation. Some pieces of path will be construct and joined them to finally produce cycles. This technique isn't efficient when number of circuit increase because it is only near the end that the total number of circuits to build looks likes important to satisfy. Then we decide to have another strategy where cycles are constructively build.

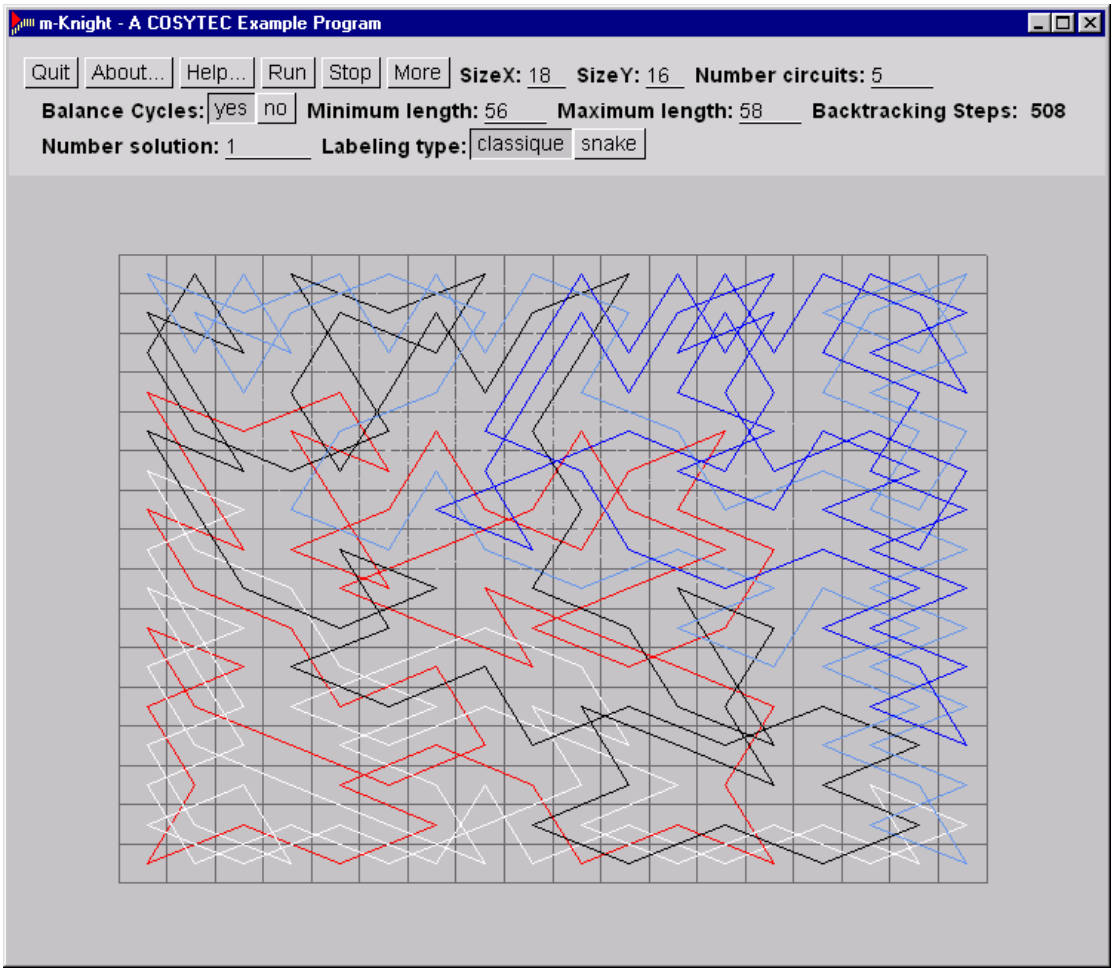
We program a new predicate for enumeration most *cycle*'s specific and graph covering specific. This is a constructive method where each circuit is build one by one by extension. We force first an extension of the path before deciding to close it. It is of course a depth-first search enumeration. We can notice than on an 8x8 chessboard with 1 knight this strategy is to build step by step the solution. It is the well-known 'planchette de Vandermonde [Van1771]' (Vandermonde board).

With this kind of strategy called *snake*, if we look for a solution with 3 knights on a 9x6 chessboard, the request `top(L)` return the following result:

L = [12, 13, 14, 15, 22, 17, 18, 25, 26, 3, 4, 19, 6, 7, 8, 9, 24, 35, 30, 1, 2, 11, 40, 5, 44, 45, 16, 39, 10, 37, 20, 49, 50, 27, 54, 53, 48, 21, 46, 47, 34, 23, 36, 51, 52, 29, 28, 41, 38, 31, 32, 33, 42, 43] as first solution without any choice point in less than one second.

If we look for all solution in a 6x6 chessboard with only one knight, the request findall(X,top(X),L) return a list with the well-known 9862 solutions [Shu94].

We developed a little application to be able to look for solutions. The next picture show a solution for the 18x16 chessboard and 5 knight with 56 to 58 moves each.



Results

We have generated systematically all rectangular $N \times M$ chessboard with values for N and M between 3 to 10 and 4 to 10 and for each chessboard, we have try C knight between 1 and $\left\lfloor \frac{N \times M}{4} \right\rfloor$ note Max(C). The value greater than Max(C) haven't solution because the smallest cycle size for a knight is 4. We haven't express odd chessboard because there is no solution and the *cycle* constraint discover if and fail directly without any enumeration.

We have 289 instances and we look for finding the first solution or proof of non-existence of solutions. Then we have 119 instances without any solution. We describe it in the next table. We have in the first colon the different size of the chessboards, in the second colon the C value express the fact that we have not well balanced solution. For example on the third line, we have no solution for a 4x2 chessboard with 1, 2 and 3 knight but it exist solutions with 4. The last colon give Max(C), the maximums value authorised to find a solution. All value greater than Max(C) haven't solution.

This result illustrates already discovered properties. For example there is no solution for a 4x4 chessboard with 1 knight [Sch91].

$N \times M$	Max(C)	C										
4x3	3	1		3								
6x3	4	1	2	3	4							
8x3	6	1	2	3		5	6					
10x3	7		2	3	4	5	6	7				
4x4	4	1	2	3								
5x4	5	1		3	4	5						
6x4	6	1		3		5	6					
7x4	7	1		3	4	5		7				
8x4	8	1		3		5	6	7				
9x4	9	1		3		5		7	8	9		
10x4	10	1		3		5	6	7		9	10	
6x5	7	2	3		5	6	7					
8x5	10				5	6	7	8	9	10		
10x5	12							8	9	10	11	12
6x6	9	6	7	8	9							
7x6	10		7	8	9	10						
8x6	12	6			9	10	11	12				
9x6	13				9	10	11	12	13			
10x6	15					10	11	12	13	14	15	
8x7	14	9	10	11		13	14					
10x7	17				12		14	15	16	17		
8x8	16	11		13		15						
9x8	18					15		17	18			
10x8	20							17	18	19	20	
10x9	22	15					20	21	22			
10x10	25	22	23	24	25							

Table 1. Chessboard and knight number for which we no solution

A typical lecture of this table is for example: 'an 8x8 chessboard (N x M) haven't any well-balanced solutions if we want to cover it by 11, 13 or 15 knights. Exceeding 16, there is too many knights. Else for the remaining possible values (1 to 10, 12 and 14) we can build easily a solution'.

Allan J. Schwen proof [Sch91] for one knight all value where it exist a solution:

> An m x n chessboard with m ≤ n has a knight's tour unless one or more of these three conditions holds:
>
> (a) m and n are both odd
>
> (b) m=1, 2, or 4
>
> (c) m=3 and n=4, 6, or 8

Like it was already done with one knight [Con94], we hope with this table, give a start for a complete resolution of the generalised Euler's knights!

Conclusion

We generalised the well-known Euler's knight problem by considering the problem with several knights. We increase the difficulty by imposing a well-balanced solution. Then we discover some combination of chessboard size and number of knight without any solutions. These unsolved problems didn't look like have common property. Then we generate instances for all the rectangular chessboards with size less than 10 and all-reasonable values for knight's number. We solve these problems with the *cycle* constraint without difficulties and we hope this work will be the start for the complete solving of the generalised Euler's knights.

This example illustrate that the *cycle* constraint must not only be considered like a logistics constraint build to models routing problems [Sim96] [Sim98], but more generally as a modelling tool for graph covering by cycles or circuits.

More specifically for well balanced Euler's knight, the results for non-existence of solutions are encouragingly. To the opposite of constructive method or local search, constraint programming has possibility to find possible solutions but also ability to make proof on non-existence.

Finally, the constraint approach offers more flexibility. We were able without many efforts to models a variation of the original problem where the knight must travel of the facets of a cube [Dud70]. In this case the cube corners haven't no more only 2 neighbours but 6. This is more difficult to start propagation.

We hope these preliminary results about well-balanced covering of a chessboard by knight will conduct to solve the general problem where we want to consider all sizes. This work was already done for one knight [Con94], next step is the generalisation…

Bibliography

[Ber73] C. Berge, "Graphes", *Gauthiers-villars*. page 181 (1973).

[Bel94] N. Beldiceanu and E. Contejean, Introducing Global Constraints in CHIP, *Mathematical Computation Modelling*. 20 (12), pages 97-123 (1994).

[Con94] A. Conrad, T. Hindricks, H. Morsy, I. Wegener, "Solution of the knight's hamiltonien path problem on chessboards", *Discrete Applied Mathematics*. 50, pp. 125-134, (1994).

[Din88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier. The Constraint Logic Programming Language CHIP. *In proceeding of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, Tokyo, pages 693-702, (1988).

[Dud70] H. E. Dudeney, "Amusements in mathematics", *Dover publications inc, NY*, p103 , (1970).

[Lau76] J.-L. Laurière, "Un langage et un programme pour énoncer et résoudre des problèmes combinatoires", *Thèse d'état PARIS VI*, (1976).

[Sch91] A. J. Schwenk, "Which rectangular chessboards have a knight's tour?", *Mathematical Magazine.* 64 (5) pp. 325-333, (1991).

[Shu94] J. A. Shufet, H. J. Berliner, "Generating hamiltonien circuits without backtracking from errors", *Theoretical Computer Sciences*. 132, pages 347-375, (1994).

[Sim96] Simonis, H. "A Problem Classification Scheme for Finite Domain Constraint Solving", Proceeding of workshop on constraint applications, CP96,Boston, August (1996).

[Sim98] Simonis, H.. "Standard Models 2 for Finite Domain Constraint Solving", PAPPACT98, London.

[Van1771] A. Vandermonde, "Remarques sur les problèmes de situation", *Mémoires de l'Académie des Sciences de Paris*. (1771).