First & last name	Cyril Bousmar
NOMA UCLouvain	63401800

LINFO2266: Advanced Algorithms for Optimization

Project 5: Constraint Programming

Exercise 1 Modeling the Magic Square Problem

The Magic Square Problem is a famous combinatorial optimization problem where a square of $n \times n$ cells has to be filled such that all rows, columns and both diagonals sum to the same value and such that each number $i \in 1..n^2$ appears exactly once within the square.

This problem can be modeled in constraint programming by applying 2 types of constraints on the variables representing the problem: `NotEqual` and `Sum` constraints.

1. What is the minimum number of variables needed to represent the problem?

   The problem requires n^2 variables that all need to be tested with `NotEqual`. We could need much less if not for that fact.

2. What is the minimum number of `NotEqual` and `Sum` constraints that needs to be applied on the problem to have a valid model? Give the number of times each type of constraint appears.

   `NotEqual` is used once for each pair of numbers, which amount to: $\frac{n^2(n^2-1)}{2}$.

   `Sum` is used to check sums of: rows, columns, and both diagonals. In other words, respectively: n, n, and 2 times. This amounting to: $2(n+1)$.

3. The sum of each row, column and of both diagonal must be the same. What is the value of this sum for a $n \times n$ magic square? How can you derive it?

   The sum of each variable can be expressed as: $\frac{n^2(n^2+1)}{2}$.
   With n rows in the Magic Square, it means each rows' sum should be: $\frac{\frac{n^2(n^2+1)}{2}}{n}$.
   In other words, the *Magic Constant* should be: $\frac{n(n^2+1)}{2}$.

In the Magic Square Problem, we are interested in counting the number of solutions that exist. However, some symmetrical solutions can be easily found once you have one. For instance, given the following magic square solution (left), you can derive the other one (right) by applying some kind of transformation.

2	7	6
9	5	1
4	3	8

Initial Solution

4	9	2
3	5	7
8	1	6

Symmetrical Solution

4. Given the following magic square solution, how many symmetrical solutions can you derive from it? What are the operations that you can apply to find them?

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

Initial Solution

Useful transformations are *Rotations* and *Orthogonal Projections (OP)*. Respectively, 90, 180, 270 degrees rotations, and horizontal, vertical, first diagonal, and second diagonal orthogonal projections. So height transformations in total (initial + seven).

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

initial

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

90° rotation

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

180° rotation

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

270° rotation

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

horizontal OP

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

vertical OP

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

diag 1 OP

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

diag 2 OP

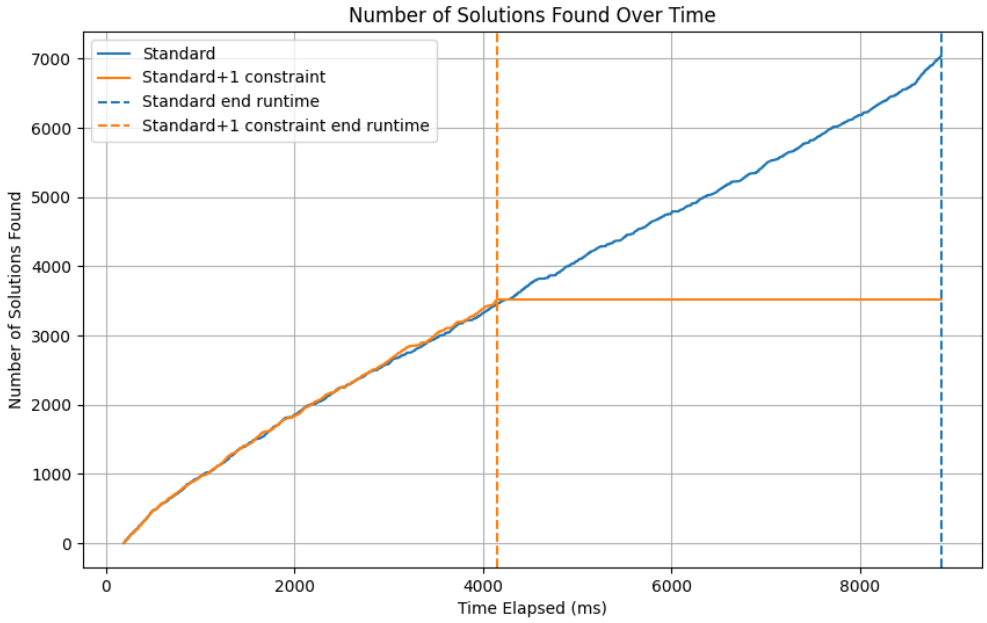
5. Use your solver to solve a magic square of size 4×4 cells, without any cell with an initial value (i.e. all values are unknown). Report the number of solutions found, the number of failures encountered and the number of recursive calls to the method `dfs` when looking for all solutions to the instance.

Solutions found	7 040
Failures encountered	664 239
`dfs` Recursive Calls	678 318

6. Experiment with symmetry breaking. For the same instance as in the previous question, experiment with the two following situations:
   1. find all solutions to the instance ;
   2. add first a constraint $cell[0][0] \leq cell[0][1]$ to break some symmetrical solutions. Then, find all solutions to the instance.

   Plot the number of solutions found over time in the two scenarios. The x-axis must be the time elapsed, in seconds. The y-axis must be the total number of solutions encountered so far.

As expected, with more constraints, the search algorithm will find less solutions and will cover the search domain faster.



Exercise 2 Knight Tour

In the Knight Tour problem, we have chosen a model where the $x[i]$ is the position visited in step i. An alternative *successor* model is to use $x[i]$ as a variable representing the next position after having visited the position i.

1. Discuss the advantages and inconvenients of each modeling approach with respect to the number of symmetrical solutions that are generated?

   In one hand, the first model is straightforward when it comes to enforce the uniqueness constraint. But it can generate a lot of symmetric solutions as a lot of valid knight moves have symmetrical properties of rotation or projection (8 in total).

   On the other hand, the *successor* model represent the search as a graph, an Hamiltonian cycle in this case. Due to its inherent nature, rotation symmetries are prevented, the number of successor is reduced, and the solution is represented as a cycle and not a sequence of moves. Successors are only valid moves which restraints the space search and reduce the number of constraints.

2. How would you enforce the valid knight moves using the successor model?

   Pre-compute the valid knight moves from position to have a list of successor nodes. We can then restrain the domain of each variables.

   Most importantly, we should have two global constraints:

   - `AllDiff`, which enforces that the successor do not exists already.
   - `NotEqual`, which enforces that the successor is not the starting point, except for the last position.