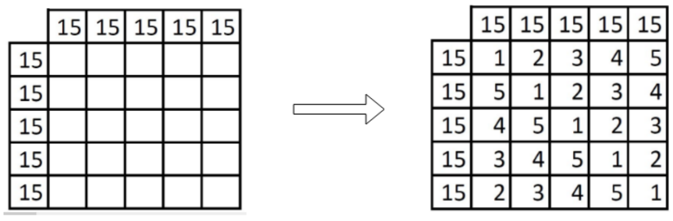
C++ Project - Solving Kakuro

Elie ABI HANNA DAHER and Bilal EL CHAMI
GitHub repository : github.com/elieahd/kakuro

***Abstract*—This document illustrate the work done for an academic project that consists of solving a Kakuro Grid as a contraint satisfaction problem (CSP) in C++.**

I. Introduction

Kakuro consists in filling a grid with numbers that sum up to a certain values for each column and row. Each cell needs to be filled with a value between 1 and 9.
The following example represents a 5x5 grid where the sum of each row has to be 15 and the sum of each column has to be 15.



	15	15	15	15	15
15					
15					
15					
15					
15					

	15	15	15	15	15
15	1	2	3	4	5
15	5	1	2	3	4
15	4	5	1	2	3
15	3	4	5	1	2
15	2	3	4	5	1

Fig. 1. Example of solving a 5x5 kakuro gird

II. Implementation

In our project we created 4 classes: $Position$, $Cell$, $Grid$ and $Kakuro$. The $Cell$ class is represented by a value and a $Position$. The $Grid$ contain a list of $Cell$. Finally the $Kakuro$ contains the $Grid$ with all the possible value that a $Cell$ can have and the target sum of rows and columns. $Kakuro$ class contains the implemented algorithm: Forward Checking and Monte Carlo.

A. Grid

The initial grid of the kakuro will be present in a file, which will contain the possible values of a cell and the target sum of each column and row. We have tested our project in 5 different grids with different sizes between 4x4 and 6x6 grid. You can find the different girds under the folder *grids* in the GitHub repository.

B. Choosing variable

While solving the Kakuro grid, we have decided to choose the variable (Cell) that has the smallest domain size. In both algorithm, forward checking and monte carlo, we have implemented the same logic regarding the choice of variable.

III. Search Algorithms

We implemented two algorithms: Forward Checking and Monte Carlo.

A. Forward Checking

This algorithm consist in reducing the domain of each variable not assigned. Each time a value is assigned to a cell, the value is removed from the domain of the cell in the same row and column. To check if we need to backtrack, two conditions are evaluated, the first one is making sure that all the unassigned Cell don't have an empty domain and the other condition is making sure that the grid is consistent which will evalute the row or column where all the cell are assigned and make sure that they match the target sum.

B. Monte Carlo

This algorithm consist in assiging a random value for a Cell, and in case of inconsistency in the grid or in case of an empty domain for an unassigned cell we will reinitialize the grid and restart the algorithm.

IV. Experimentals - Simulation

In order to test the perfomance of our algorithm, we runned several grids 100 times and calculated the estimated time to solve the kakuro grid using both algorithm.

TABLE I
SIMULATION

Grids	Forward Checking	Monte Carlo
Grid 1	0.02 s	0.019 s
Grid 2	0.021 s	0.02 s
Grid 3	2.804 s	0 s
Grid 4	5.02 s	0 s
Grid 5	$10,152.1$ s	0 s

The time of execution of both algorithm are both very dependant from several criteria. The first one is the size of the grid, the second criteria is the target sum the final criteria is possible values. The forward checking algorithm is more optimal and more efficient than the monte carlo algorithm.
We could have tested both algorithm together by modifing the forward checking algorithm to take a random value instead of having to iterate through the values.

References

[1] T. Cazenave. *Monte-Carlo Kakuro*
[2] Wikipedia page on *Kakuro*. [Online].
Available: https://en.wikipedia.org/wiki/Kakuro
[3] Wikipedia page on *CSP* . [Online].
Available: https://en.wikipedia.org/wiki/Constraint_satisfaction_problem
[4] Wikipedia page on *Backtracking algorithm*. [Online].
Available: https://en.wikipedia.org/wiki/Look-ahead_(backtracking)