Implementation of a solver for Multi-balance Puzzle with Prolog constraint programming in finite domains

André Mamprin Mori [up201700493], Muriel de Araújo Pinho [up201700132]

FEUP-PLOG, Class 3MIEIC05, Group Multi-Balance_5

Abstract: This paper was written for Logic Programming classes as part of the second practical project. The goal was to apply the theoretical knowledge on Prolog's constraint programming interface to solve a constraint satisfaction problem, a puzzle called Multi-balance. We then developed a program using SICStus Prolog's Constraint Logic Programming Finite Domain library. The program uses a database of predefined puzzles, unfortunately we were not able to implement a puzzle generator, as the puzzle needs to meet really specific criteria to be solvable. we developed our own predicates to solve 123 puzzles and arrived at somewhat satisfying execution times. It became clear to us that constraint programming can be a very powerful tool to solve constraint driven problems then again, discovering an effective and clean solution, can prove itself quite challenging for most problems.

Keywords: Multi-balance Puzzle, Constraint Programming, Constraint Satisfaction Problem, Finite Domain, Prolog, SICStus.

1. Introduction

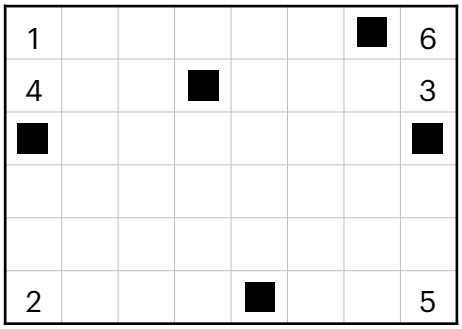
This paper is part of the second practical project of Logic Programming class from the Faculty of Engineering of the University of Porto (FEUP). The goal of the project was to solve a constraint driven problem using Constraint Logic Programming (CLP) class of programming languages. This class is mainly declarative and quite efficient on solving constraints. We also aim to introduce CLP in a practical way focused on a concrete example, so the reader can better understand the theory behind this kind of problems.

The problem we chose to explore was the Multi-Balance Puzzle from Erich Friedman. The puzzle's rules are explained in the next chapter. One can quickly understand that a puzzle is nothing more than a set of rules that lead us to arrive at a particular goal/solution. Rules are in fact just constraints of the given problem and, as such, puzzles are a perfect example of a Constraint Satisfaction Problem (CSP) – problem where the satisfaction of all (or a set of) constraints produces valid solutions.

2. Problem Description

The Multi-Balance puzzle was created by Erich Friedman. It is a two-dimensional board game where the player must place sequential digits from 1 to N (determined at the beginning of the game) on the board, these digits must be on the same column or row of a fulcrum, no row or column can contain exactly one digit, every row or column that contains 2 or more digits must contain exactly one fulcrum, the fulcrum is used only vertically or horizontally, the digits must be placed in a way that the torques on each side of the fulcrum are balanced, the torque of a side is calculated by adding the torque of the digit(s) on a side of the fulcrum, the torque is calculated by multiplying the digit by the distance of the digit to the fulcrum.

Here's a puzzle with a valid solution for a puzzle:



Solved puzzle

To calculate the torque for a row or column, the formula applied is:

*Torque = Digit * Distance_to_fulcrum*

In the solved puzzle above, the calculation for the resulting torque of the first row is:

*LeftTorque = 1 * 6 = 6,*

*RightTorque = 6 * 1 = 6*

The resulting torque of the first column is:

*AboveTorque = 1 * 2 + 4 * 1 = 6*

*BelowTorque = 2 * 3 = 6*

3. Approach

SICStus Prolog with CLPFD library was used to develop our program. The library provides functions for applying domains and generate solutions, while the constraints were developed by the group to adapt to the puzzle.

The three essential steps (in the correct order) to solve using constraints are:

- Declare the decision variables and assign them the problem domain
- Declare the constraints that will be applied to the variables
- Begin searching for solutions

3.1. Decision Variables

There is 1 decision variable for each digit to be inserted in the board, these variables are inside a list containing the identification to the cell inside the board of each digit, this identification's possible values are restricted by the domain.

The domain has possible values from 0 to the number of rows multiplied by the number of columns, these values are added to the domain if the identification of the cell equals a cell where a digit could be inserted. The evaluation of these values use the predicates *calcRowDomain* and *calcColDomain,* which iterate through the rows and columns respectively, evaluating if the cells inside that row or column could contain digits or not, only possible cells are added to the domain.

By only adding possible values we remove a great portion of values from the domain, increasing the efficiency for solving the problem as values that couldn't contain digits aren't evaluated as possible values for the decision variables. After deciding the domain the variables have constraints applied to them.

3.2. Constraints

In order to solve this problem respecting its rules, two constraint predicates were developed: *constrainPlacing* and *torqueConstraint*.

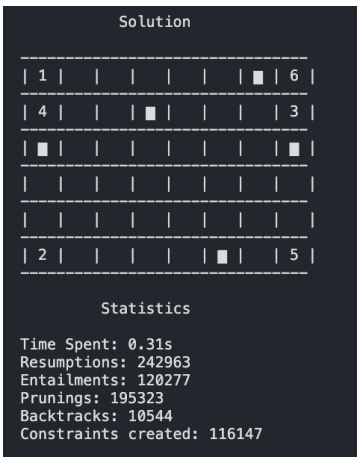
Predicate *constrainPlacing* constrains the decision variables by not allowing it to place a digit on: a cell that is not empty; a row/column without exactly 1 fulcrum; row/column where the fulcrum is placed on the edge (index 0 or size of row/column).

After *constrainPlacing* has finished, it calls the predicate *addSolution*, which places the digits in the specified positions on the board, which will be used by *torqueConstraint*.

The predicate *torqueConstraint* will calculate the torque from each row and each column from the board with the applied decision variables and constrain that it should be balanced.

4. Solution Presentation

The solution is presented as the resulting matrix of the decision variables placed on the initial board along with statistics for the solution. The decision values array for the solution has the one value for each digit position where each value indicates the cell where that digits position was inserted, this solution's array would be [0,40,15,8,47,7] where 1 is at cell 0 and 2 is at cell 40.

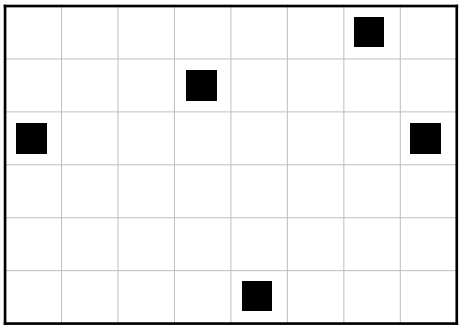


5. Experiments and Results

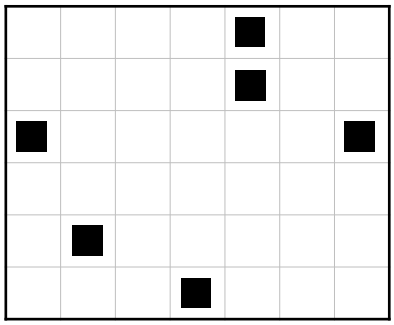
The following experiments may have different time results depending on the machine and the current load on its processor. All results are from the same machine and had similar loads on the processor.

5.1. Dimensional Analysis

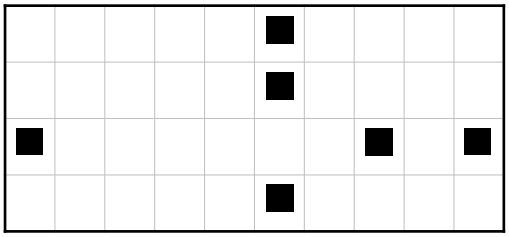
All 4 puzzles in our database were analysed using the time measuring predicate created by the teachers and the *fd_statistics* predicate from the CLPFD library.



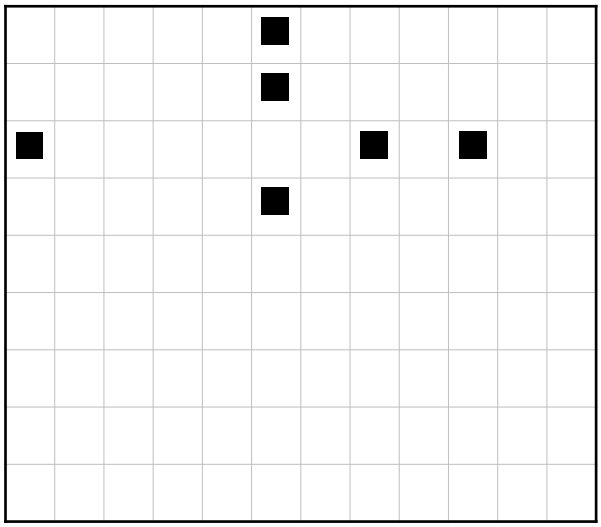
Puzzle 1



Puzzle 2



Puzzle 3



Puzzle 4

These puzzles have similar dimensions, fulcrums placed, number of rows and number of columns but they vary in digits to be inserted.

Table 1: Puzzle Solving Results

	Puzzle 1	Puzzle 2	Puzzle 3	Puzzle 4
Time spent	0.31s	391.24s	20.34s	87.59s
Resumptions	242 963	271 849 725	18 979 172	78 790 885
Entailments	120 277	140 460 697	9 363 288	37 247 869
Prunings	195 323	224 030 177	15 675 906	58 098 515
Backtracks	10 544	12 027 966	722 357	1 932 539
Constraints Created	116 147	135 759 025	9 126 703	36 412 366

Based on the results we can analyse that even with the similar puzzles, the amount of digits is the characteristic that had the greatest impact as the execution statistics values and time of execution for the puzzle, exponentially grew when the digits were increased. But the other characteristics also had impact, Even though puzzle 1 and 4 had the same amount of digits to be inserted with different fulcrum, column and row amounts, puzzle 4 took 280% more time and consequently had bigger executions statistics values.

5.2. Search Strategies

To decide which was the best combination of options for labelling we tested all 30 possible combinations of strategies for labelling for each puzzle, no significant time diference was observed when using a combination as opposed to the default options for labelling. These are the average of the time spent finding the solution with the combination of the next variable selection options with the other options for labelling:

	Leftmost	Min	Max	FF	FFC
Time Spent	19.8s	19.8s	21.24s	23.27s	20.47s

6. Conclusions and Future Work

In conclusion, working on this project showed how using Constraint Logical Programming can be a powerful tool for these types of problems, that can reach results far better than the standard generate & test approach. However, it was a good example of how even simple constraints can be difficult to implement. With more time, the next step would be to try and improve the efficiency of our solver, and implement a puzzle generator that generated valid puzzles.

7. References

1. Multi-Balance Puzzle Page, https://erich-friedman.github.io/puzzle/2Dweight/
2. SICStus Constraint Logic Programming over Finite Domains, https://sicstus.sics.se/sicstus/docs/4.3.0/html/sicstus/lib_002dclpfd.html
3. Torque Formula, https://www.toppr.com/guides/physics-formulas/torque-formula/