Solver of Tricky Triple Puzzles Based on Constraint Programming

up201303828@fe.up.pt Ângelo Daniel Pereira Mendes Moura
up201806528@fe.up.pt Clara Alves Martins

Faculty of Engineering of the University of Porto
https://www.fe.up.pt

Logic Programming
3MIEIC06 - Tricky Triple 2

January 4, 2021

Abstract. In this article, we show how to use constraint programming to solve Tricky Triple Puzzles. We present a consistent method to analyze these puzzles and to obtain a solution to them.

Keywords: tricky-triple · prolog · clpfd · constraint-programming

1 Introduction

This project consists of building a program, in Logic Programming with Restrictions, for solving a combinatorial decision. The problems studied are Tricky Triple Puzzles, which are grid puzzles. To do so, we will analyze these problems and proceed to implement our solver using SICStus Prolog. Afterward, we will be discussing the performance results obtained. This article has the following structure:

1. **Problem Description**: tricky triple puzzle detailed description
2. **Approach**: problem modulation
   (a) **Decision Variables**: decision variables' meaning and domain
   (b) **Constraints**: problem constraints' description
3. **Solution Presentation**: explanation for the predicates that provide a user interface
4. **Experiments and Results**: results from all the tests
   (a) **Search Strategies**: result's analysis from different search strategies
   (b) **Dimensional Analysis**: result's analysis from different puzzle's dimensions
5. **Conclusions and Future Work**: conclusions withdraw and limitations of this project
6. **References**: books, web pages, and articles used
7. **Annex**: result tables and other extras

2 Problem Description

The Tricky Triple puzzles are a type of grid puzzle. The goal of the puzzle is to fill each of the grid's white cells with one of 3 symbols, a square, a circle, or a triangle. The only rule is that each group of 3 adjacent white cells (horizontally, vertically, or diagonally) must contain exactly 2 of one of the symbols. So, each group of 3 white cells will have 2 of 3 symbols. Each puzzle given has a unique solution.

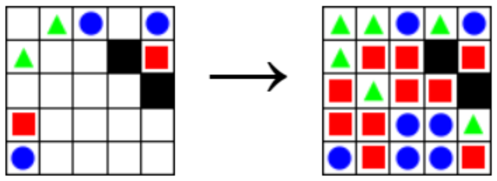


Fig. 1. Example of Tricky Triple Puzzles, before and after solving it

3 Approach

Throughout the development of this project, we used a Constraint Logic Programming approach. Our grid is a list of lists, where each element is a number indicating the symbol placed on that cell. Each one of these numbers represents a different symbol. The number 0 represents a black cell, which cannot be generated by the solver. The number 1 represents a green triangle, while the number 2 represents a red square and the number 3 a blue circle.

3.1 Decision Variables

All tricky triple puzzles take the form of an NxN square grid divided by cells. The grid is represented internally by a list of lists forming a square matrix.

The grid cells can either be black, a cell that the program can't fill, or, more commonly, white.

All the white cells need to be either a triangle (represented by 1), a square (represented by 2), or a circle (represented by 3) to reach the puzzle solution.

The Decision Variables are all the elements of the list of lists, meaning all the puzzle cells. Their domain is 0, 1, 2, 3. However, being the representative of a black cell, the zero is not considered a valid value to fill an empty cell, since that would transform a white cell into a black cell.

3.2 Constraints

The restrictions implemented in our solver are faithful to the ones from the puzzle rules.

Each cell must contain a symbol
In the puzzle solution, all the cells have to be assigned.

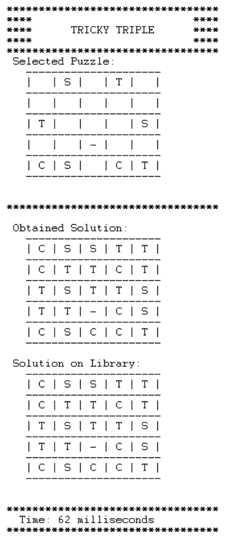
No black cells can be assigned
In every white cell, we need to put a square, a triangle, or a circle. Meaning we can't put a black cell on a white cell.

Each group of 3 adjacent white cells (horizontally, vertically, or diagonally) must contain exactly 2 of one of the symbols
Consider a group of three adjacent, horizontally, vertically, or diagonally, white cells. In this group, two of the cells have the same symbol, and the last cell must have a different one.

4 Solution Presentation

To present the solution, we use two predicates from the file *display.pl*.

- The predicate `display_grid/2` displays the grid in a human-friendly way so that the user can identify the cells and the grid's symbols.
- The predicate `get_readable_symbol/2` translates the grid elements' internal representation into more readable symbols for those to be displayed to the user.



```
********************************
****                        ****
****      TRICKY TRIPLE     ****
****                        ****
********************************
 Selected Puzzle:
   ---------------------
   |   | S |   | T |   |
   ---------------------
   |   |   |   |   |   |
   ---------------------
   | T |   |   |   | S |
   ---------------------
   |   |   | - |   |   |
   ---------------------
   | C | S |   | C | T |
   ---------------------


********************************

 Obtained Solution:
   ---------------------
   | C | S | S | T | T |
   ---------------------
   | C | T | T | C | T |
   ---------------------
   | T | S | T | T | S |
   ---------------------
   | T | T | - | C | S |
   ---------------------
   | C | S | C | C | T |
   ---------------------

 Solution on Library:
   ---------------------
   | C | S | S | T | T |
   ---------------------
   | C | T | T | C | T |
   ---------------------
   | T | S | T | T | S |
   ---------------------
   | T | T | - | C | S |
   ---------------------
   | C | S | C | C | T |
   ---------------------


********************************
  Time: 62 milliseconds
********************************
```

Fig. 2. Output Example for a Puzzle Solution

5 Experiments and Results

The tests performed on this solver aren't as exhaustive as we could have hoped since we could only use the pre-generated puzzles, and there weren't that many of them. However, we can still draw some conclusions from the times measured when solving these puzzles.

5.1 Dimensional Analysis

To compensate for our lack of different puzzles, we did extensive tests with the predefined ones. After that, we grouped the tests by the puzzle's dimension and the labeling options used in that solution. The graph below illustrates the obtained results. Table 1 contains the average values for each dimension and labeling option.

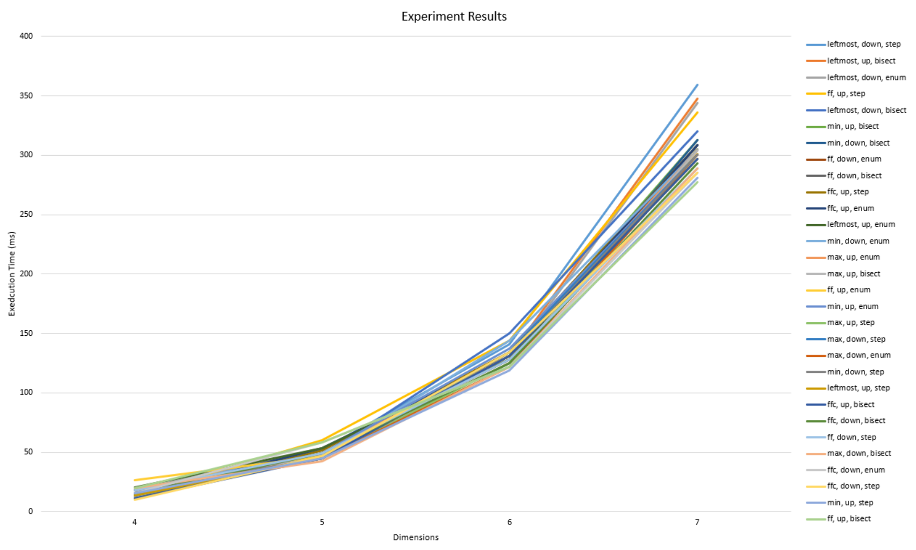


Fig. 3. Experiment Results

Analyzing Figure 3, we can conclude that the fastest labeling option, for grids up to 7x7, is "ff, up, bisect".

From all the collected information, we can withdraw some conclusions:

- A small board with bad labeling options can take more time to solve than a bigger one with good labeling options.
- When introducing some partially solved boards, the solver would be consistently faster.
- The solver's execution time increases with the increase in the grid's dimensions.

5.2 Search Strategies

However, using the same data, we can try to predict the behavior of the labeling options when increasing the grid's dimensions by making a trendline. Using a degree 3 polynomial trendline, we obtained a prediction for execution times when solving bigger grids. The equations for these trendlines are in Table 2. Analyzing Figure 4, we can conclude that perhaps the best options would be ”min, down, enum” or ”leftmost, down, bisect”.

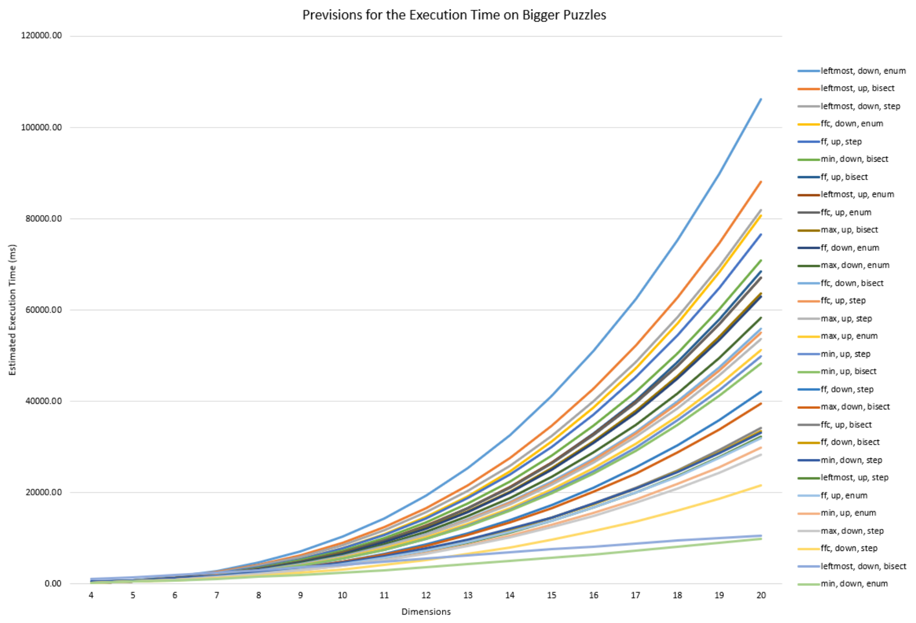


Fig. 4. Execution Times Predictions

Table 1. Average Times Obtained (ms)

Labeling Options			Grids			
			4x4	5x5	6x6	7x7
leftmost	up	step	13.43	44.57	131.4	296.75
leftmost	up	enum	20.14	53.57	128	304.75
leftmost	up	bisect	13.43	46.86	131.2	347.75
leftmost	down	step	17.71	51.43	140.6	359.25
leftmost	down	enum	17.86	53.57	128.2	343.75
leftmost	down	bisect	17.71	47	150	320.25
ff	up	step	13.42	60.14	143.8	336
ff	up	enum	26.71	46.86	131.2	301
ff	up	bisect	20	58	122	277.25
ff	down	step	18	49.14	128	289
ff	down	enum	15.71	49.14	128	308.5
ff	down	bisect	15.71	44.57	134.4	308.5
ffc	up	step	11.14	51.43	134.4	308.5
ffc	up	enum	13.29	53.57	131.4	308.5
ffc	up	bisect	11.14	44.57	131.4	296.75
ffc	down	step	9.71	46.86	134.4	285
ffc	down	enum	15.57	58.14	121.8	285.25
ffc	down	bisect	15.57	49	125	293
min	up	step	15.71	44.58	118.8	281.25
min	up	enum	15.57	49.14	137.4	300.75
min	up	bisect	13.43	44.71	131.2	312.5
min	down	step	17.86	44.57	131.4	300.5
min	down	enum	15.71	46.86	143.8	304.5
min	down	bisect	13.43	46.86	125	312.5
max	up	step	13.43	44.58	125	300.75
max	up	enum	13.43	49	131.4	304.5
max	up	bisect	13.29	47	125	304.5
max	down	step	13.43	44.71	134.4	300.75
max	down	enum	17.86	44.58	122	300.75
max	down	bisect	20	42.43	122	289

Table 2. Trendlines

Labeling Options			Equations
leftmost	up	step	$y = ax^3 + bx^2 + cx + d = 0$
leftmost	up	enum	$y = ax^3 + bx^2 + cx + d = 0$
leftmost	up	bisect	$y = ax^3 + bx^2 + cx + d = 0$
leftmost	down	step	$y = ax^3 + bx^2 + cx + d = 0$
leftmost	down	enum	$y = ax^3 + bx^2 + cx + d = 0$
leftmost	down	bisect	$y = ax^3 + bx^2 + cx + d = 0$
ff	up	step	$y = ax^3 + bx^2 + cx + d = 0$
ff	up	enum	$y = ax^3 + bx^2 + cx + d = 0$
ff	up	bisect	$y = ax^3 + bx^2 + cx + d = 0$
ff	down	step	$y = ax^3 + bx^2 + cx + d = 0$
ff	down	enum	$y = ax^3 + bx^2 + cx + d = 0$
ff	down	bisect	$y = ax^3 + bx^2 + cx + d = 0$
ffc	up	step	$y = ax^3 + bx^2 + cx + d = 0$
ffc	up	enum	$y = ax^3 + bx^2 + cx + d = 0$
ffc	up	bisect	$y = ax^3 + bx^2 + cx + d = 0$
ffc	down	step	$y = ax^3 + bx^2 + cx + d = 0$
ffc	down	enum	$y = ax^3 + bx^2 + cx + d = 0$
ffc	down	bisect	$y = ax^3 + bx^2 + cx + d = 0$
min	up	step	$y = ax^3 + bx^2 + cx + d = 0$
min	up	enum	$y = ax^3 + bx^2 + cx + d = 0$
min	up	bisect	$y = ax^3 + bx^2 + cx + d = 0$
min	down	step	$y = ax^3 + bx^2 + cx + d = 0$
min	down	enum	$y = ax^3 + bx^2 + cx + d = 0$
min	down	bisect	$y = ax^3 + bx^2 + cx + d = 0$
max	up	step	$y = ax^3 + bx^2 + cx + d = 0$
max	up	enum	$y = ax^3 + bx^2 + cx + d = 0$
max	up	bisect	$y = ax^3 + bx^2 + cx + d = 0$
max	down	step	$y = ax^3 + bx^2 + cx + d = 0$
max	down	enum	$y = ax^3 + bx^2 + cx + d = 0$
max	down	bisect	$y = ax^3 + bx^2 + cx + d = 0$

6 Conclusions and Future Work

7 Bibliography

8 Annex

8.1 Average Execution Times Table

8.2 Trendlines using to Estimate Execution Time for Bigger Grids

9 Section Sample

9.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 3. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 1).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Fig. 5. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, `http://www.springer.com/lncs`. Last accessed 4 Oct 2017