Applied Artificial Intelligence

Eduardo Eloy

Universidade de Évora, Portugal

Abstract. This paper provides an implementation of a constraint programming solution to the Costas Array Problem that through a Difference Triangle built with the Costas Array ensures that no two lines in the array have the same length and slope. Constraint programming being a powerful tool for finding and optimising solutions to these kinds of combinatorial problems makes it, alongside MiniZinc, a good choice to solve this problem widely used in military applications.

Keywords: Constraint · Costas · Difference.

1 The Problem

Out of the 4 available options I chose the Costas Array problem[5]. These types of arrays are named after John P. Costas[7], who first wrote about them in 1965 and have since been used in military applications of radar and sonar systems to, for example, detect submarines. The problem can be described as a N*N matrix of Unmarked (U) and Marked (M) elements where each row and column may only have one M element and no two lines formed between pairs of M elements may have the same length and the same slope, meaning every pair of of M elements must form a unique line segment. Line segments may have the same length but different slopes or the same slopes but different lengths.

2 State of The Art

The Costas array problem has been solved and as of 06/01/2021 the On-Line Encyclopedia of Integer Sequences has gathered solutions to the problem up to problem size 29[4] although there are known solutions with a larger size. There are some popular constructions of Costas arrays like the Welch–Costas construction, the Lempel construction and the Golomb construction, however I won't be using any of these constructions to solve or model the problem. Parallel computing has also been used in solutions of this problem

In January of 2012 an article, by Salvador Abreu Et Al[2], was written proposing the solving of Costas arrays problem through a type of Constraint based Local search method called Adaptive search[1], in doing so, it proposes a model of the Costas array as a permutation of N, similarly to the N-queens[6] model we are used to solving. Because this gives e a familiar setup to solve the problem I will orient myself by the guidance of this article and attempt to implement this type of construction.

3 The Tool

The Minizinc[3] constraint modelling language is a powerful modelling tool that allows the use of several pre-defined constraints. The model and constraints are compiled into FlatZinc, a solver input language that is understood by a wide range of solvers like Gecode, Chuffed and Coin-BC. It as extensive documentation and can be installed along with a specialized IDE

4 The Model

As mentioned earlier, the matrix is represented through an integer array of N values, this array being a permutation of N and having N! possible permutations. The value at each index of the array refers to the position of the M element, this makes it so we apply an implicit constraint so no two M elements can be at the same row (or column depending on how you see it). Due to being a permutation, each value in the array must be different. Now the interesting part about assuring that the permutation we reach is a Costas array is that it relies on a modification of a structure called a "Difference Triangle". Normally with a regular difference triangle we start with an initial array of N integers and we calculate the difference between the values at position x and position x+1 until we reach the end of the array. This leaves us with a new array of N-1 integers and we repeat this cycle until we get to a single integer. For our purpose though, the algorithm that builds our triangle is a bit different. For each array position and for each array position higher than that first position, we subtract the value in the higher position of the array by the value in the position obtained by subtracting the higher position and the first position. By doing that we obtain N-1 arrays of 1..N-1 values, if for each array all values are different, then the initial array is a Costas array.

Putting this in more mathematic terms we have the following
Problem size is N
Array α has N values
Values must go from 1 to N and must be different:

$$\forall v \; in \; \alpha, v \in 1..N$$

$$\forall i, j \in \alpha, i \neq j$$

To build the difference triangle T, which is 2 dimensional array:

$$\forall i, j \in \alpha \; where \; i < j, T[i, j] = \alpha[j] - \alpha[j - i]$$

Every array in T must have all different values:

$$\forall i \in 1..N, \forall j, k \in 1..N, T[i, j] \neq T[i, k]$$

5 The Results

5.1 Specifications

These results were obtained by compiling and running the programs on my PC which has the following specifications:
Memory: 7,6 GiB
Processor: Intel® Celeron(R) N4100 CPU @ 1.10GHz × 4
Graphics: Mesa Intel® UHD Graphics 600 (GLK 2)
OS: Ubuntu 20.04.1 LTS 64-bit

5.2 NaiveMethod

Table 1. Time to find the first solution, Naive method

Size	Real	User
1	0m0,295s	0m0,110s
2	0m0,286s	0m0,155s
3	0m0,260s	0m0,138s
4	0m0,260s	0m0,143s
5	0m0,244s	0m0,122s
6	0m0,294s	0m0,143s
7	0m0,375s	0m0,169s
8	0m0,267s	0m0,168s
9	0m0,261s	0m0,164s
10	0m0,257s	0m0,162s
11	0m0,289s	0m0,173s
12	0m0,342s	0m0,181s
13	0m0,410s	0m0,275s
14	0m2,083s	0m1,878s
15	0m15,855s	0m14,954s
16	0m58,398s	0m55,523s
17	7m49,817s	7m44,406s
18	0m48,189s	0m47,642s
19	345m49,232s	341m49,306s

Table 2. Time to find all solutions, Naive method

Size	Real	User
1	0m0,169s	0m0,113s
2	0m0,173s	0m0,091s
3	0m0,171s	0m0,104s
4	0m0,287s	0m0,155s
5	0m0,166s	0m0,100s
6	0m0,235s	0m0,156s
7	0m0,328s	0m0,197s
8	0m0,437s	0m0,352s
9	0m1,216s	0m1,116s
10	0m5,196s	0m4,854s
11	0m24,013s	0m23,543s
12	2m18,867s	2m9,116s
13	11m2,120s	10m47,396s
14	53m58,181s	53m34,183s
15	315m10,565s	312m22,656s

6 Improved method

6.1 The Improved Model

The improved model is based on adding 3 redundant constraints to the naive model and modifying the search strategy. A redundant constraint is a constraint that is already logically implied by the the existing model but adding them may improve the search time Minizinc has a "redundant constraint" function to specify the nature of these constraints.

One of these redundant constraints ensure the already implied constraint that the values in the Costas Array higher than 0 but lower than or equal to N.

The other 2 enforce the same constraint that all values in the Costas Array are different (which is already implied through the global "all different" constraint), one of them does this by iterating through the Costas Array and saying that the value in each index must be different than the values in the indices after it. The other one enforces an all different constraint on the Costas Array but through the Difference Triangle, we iterate through the already built 2 dimensional Difference Triangle, and anytime i is lower than j (i being the first iteration, through the rows, and j being the second iteration, through the columns) the value at Difference Triangle[i,j] must be different than 0, because if it is 0 that would mean two values in the Costas Array were equal since these values are obtained by computing CostasArray[j] - CostasArray[j-i].

The search strategy was changed from the default strategy to an "Input Order Indomain Min" strategy, meaning that when the solver is choosing which variable's value it is going to try and find and from its domain which value it is

going to try (to see it doesn't break any constraints) it is going to do this from the order of the array, starting with the first index and going forward, and from the values in the domain it's going to choose the lowest value.

I arrived at this strategy by trying some of the other options, like "First Fail", where the solver goes from the variable with the smallest domain size to the next, and "Indomain Median", where the median value in the domain is chosen, and comparing the results to find the combination that performed better.

Another constraint I added was a "symmetry breaking constraint" so that the solver wouldn't propose solutions that are just "inversions" of other solutions. This is done by constraining a variable at a specific index to be lower or greater than the variable at its "inverse" index, for example the inverse of the first index is the last one. So by declaring the CostasArray[1] must be lower than CostasArray[N] we are get solutions like [1, 2, 4, 3] but not its inverse [3,4,2,1].

6.2 Results

Table 3. Time to find the first solution, Improved method

Size	Real	User
1	0m0,213s	0m0,107s
2	0m0,348s	0m0,134s
3	0m0,218s	0m0,115s
4	0m0,227s	0m0,148s
5	0m0,299s	0m0,128s
6	0m0,286s	0m0,169s
7	0m0,246s	0m0,118s
8	0m0,240s	0m0,153s
9	0m0,245s	0m0,149s
10	0m0,256s	0m0,150s
11	0m0,333s	0m0,184s
12	0m0,249s	0m0,164s
13	0m0,423s	0m0,223s
14	0m1,095s	0m0,972s
15	0m9,112s	0m8,922s
16	0m34,591s	0m33,417s
17	7m5,511s	6m55,474s
18	0m33,119s	0m29,748s
19	180m48,177s	180m32,328s
20	172m22,697s	171m15,115s

Table 4. Time to find all solutions, Improved method

Size	Real	User
1	0m0,250s	0m0,137s
2	0m0,374s	0m0,163s
3	0m0,514s	0m0,158s
4	0m0,350s	0m0,133s
5	0m0,409s	0m0,147s
6	0m0,311s	0m0,162s
7	0m0,524s	0m0,219s
8	0m0,722s	0m0,324s
9	0m1,131s	0m0,600s
10	0m3,852s	0m2,742s
11	0m17,382s	0m13,716s
12	1m19,371s	1m12,336s
13	7m15,399s	6m38,845s
14	33m26,841s	32m55,638s
15	140m56,205s	139m33,558s

7 Overall Assessment

In the end both methods perform similarly, the only outlier being finding the first solution with the naive method with a size of 17, the improved method does find solutions faster and in the long run that results in also finding all solutions faster but finding solutions when the size is 19/20 begins to take tens of minutes or even hours, and finding all solutions when the size is 15 starts to take hours.

Although the improved method doesn't perform significantly better than the naive method it still is a light improvement and overall I'm satisfied with the result.

References

[1] P. Codognet C. Truchet D. Diaz. "The Adaptive Search Method for Constraint Solving and its application to musical CSPs". In: (2002). URL: https://www.researchgate.net/publication/228870263_The_Adaptive_Search_Method_for_Constraint_Solving_and_its_application_to_musical_CSPs.

[2] S. Abreu D. Diaz F. Richoux P. Codognet Y. Caniou. "Constraint-Based Local Searchfor the Costas Array Problem". In: (2012). URL: https://dspace.uevora.pt/rdpc/bitstream/10174/6283/1/lion6-published.pdf.

[3] *MiniZinc*. URL: https://www.minizinc.org/.

[4] *On-Line Encyclopedia of Integer Sequences*. URL: https://oeis.org/A008404.

[5] Wikipedia. “Costas Array. (English)”. In: (). URL: https://en.wikipedia.org/wiki/Costas_array.
[6] Wikipedia. “Eight queens puzzle. (English)”. In: (). URL: https://en.wikipedia.org/wiki/Eight_queens_puzzle.
[7] Wikipedia. “John P. Costas. (English)”. In: (). URL: https://en.wikipedia.org/wiki/John_P._Costas_(engineer).