Kakuro Solver

Introduction

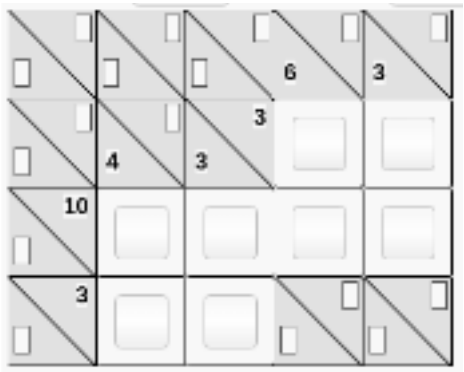
Kakuro is a puzzle game largely popularized in the 1960s by Canadian resident Jacob E Funk. It is an easier and special version of the Survo puzzle. Kakuro puzzles vary in their sizes and difficulties. In a standard Kakuro puzzle game, the board is set with several blank squares with a number written either on top or on the left of a column and row respectively. Our job is to fill the blank squares with numbers ranging from 1 to 9 such that a number is not repeated in a row and a column and the sum written either on top or left is formed.
For this project, we assume solving a kakuro puzzle as a Constraint Satisfaction Problem (CSP).
A constraint satisfaction problem is defined by a set of variables (X_1, X_2, X_3,..., X_n) belonging to A domain D and a set of constraints C_1, C_2, C_3,..., C_k, where a variable or a set of variables is under a constraint Ci and is allowable only if all the constraints on that set of variables is satisfied. An Assignment of the variables that satisfies all the constraints associated with it is known as a consistent or a legal assignment. A complete assignment to a CSP is one in which all the variables and their associated constraints are mentioned. A solution is a complete assignment in which all the assignments are satisfied. Some examples of constraint satisfaction problems are the Map Coloring problem. Sudoku, N-queens problem. Some real-world CSP s are assigning timetables, scheduling problems, Circuit layout Etc. This is a very broad topic and many important real-life applications exist under this set of problems.
Kakuro puzzle games are generally easier than a survo puzzle where we have to solve to fill all the boxes in an AxB box with only outer constraints so many possible games are possible. However, kakuro removes those extra game sets by adding additional constraints in the middle of the game board so that a unique solution can be formed. However, despite this as soon as the sum number goes high or the number of variables increases the search tree becomes vast. This makes the game quite challenging.

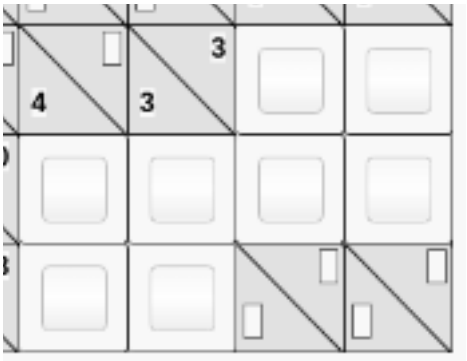
Problem Definition

In this problem, the job is to find a solution that satisfies the Constraints laid on each of the kakuro puzzle boxes. For each kakuro puzzle, we first define the dimension of the puzzle and then set up the variable and constraints. The variables in this problem are the unfilled boxes which are denoted here with a lighter shade. The darker squares are used to store the constraint information. The constraint For each variable is the following:

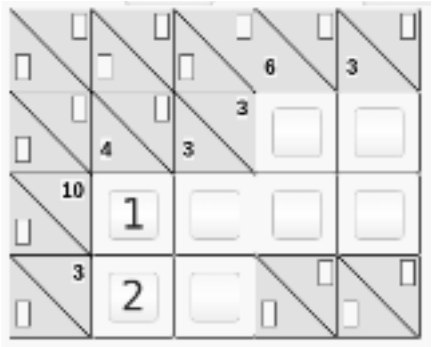
1. It must be less than the sum at the beginning of the row or column
2. It must be between 1 and 9.

3. The number of variables that form the sum must be equal to the number of unfilled boxes associated with the sum.
4. For this problem, we will not count the first row and column in the dimensions as they are never unfilled.
5. The variables are named starting from zero after eliminating the first row and column

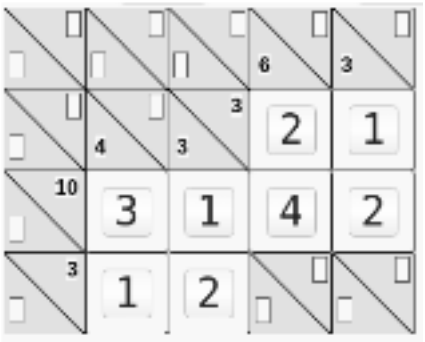




6. The squares are now named starting from zero at the top left corner up to the bottom right corner.
7. Each state in the problem is defined by the variable filled with a value



8. The start state defined by all the unfilled boxes empty and all the dark blanks set to their position
9. The goal state is defined by all the unfilled boxes filled and all the constraints (Sum) satisfied by the variables.



For our solution, we denote the unfilled squares with an '_' and a '.' to denote the darker blank boxes.

So the start state for this kakuro puzzle will be :

. . _ _
_ _ _ _
_ _ . .

And the goal state will be:

. . 2 1
3 1 4 2
1 2 . .

Background survey

Since kakuro is a very popular game, many people have tried to come up with a solution that saves them from the hassle of spending time on the puzzle.

According to my survey, many approaches have been taken to solve the Kakuro puzzle including Backtracking Forward checking and maintaining arc consistency. However, these problems required a lot of work and were not easily implemented. The solution for sudoku and kakuro has got the same roots as both of them are similar, hence this project is also inspired by the sudoku problem. For sudoku too a similar approach was taken; backtracking forward checking and maintaining arc consistency.

Discussion

My project is the only stand-alone solution provides which can work without any pre installation required. One can simply run the executable and enter the details to get the solution. Due to the uniqueness of my solution, I believe this to be an engineering solution.

Kakuro puzzles can range from small 3x3 to gigantic 19x19 puzzles. My solution will answer all the range of puzzles. If we make some sacrifices with the time then this solution is one of the easiest and most feasible solutions available. It must be noted that this solution is only

considered feasible for up to medium sie puzzles as the large ones require a large amount of computation

Algorithm

The most efficient algorithm that is guaranteed to find an optimal solution is using the least Constraining Value Ordering algorithm. For this, we will choose the value that gives us the most options to select from, for the numbers to be put in the unfilled squares.
The solution that will be practical for large problems but incomplete will be a greedy algorithm
Since in case of large problems of kakuro it will almost certainly be guaranteed that the numbers will be large so we can use a greedy algorithm to fill the values using the largest possible numbers rather than all the numbers. This will also apply to nonstandard kakuro puzzles where the numbers that can be put in the unfilled squares increase to 12 or 20.
So we can use a greedy algorithm to fill the largest number first. This will not form the optimal solution but will give suboptimal solutions fast.

Algorithm Complexity

Let the kakuro grid be called **K** with **n**x**m** dimentionsand each cell is named $\mathbf{k_{i,j}}$ for the i^{th} row and j^{th} column. Each cell can be an unfilled square or a blank. Each constraint will be either horizontal or vertical. Let the number of unfilled boxes in a sum be equal to **p**. Therefore each horizontal constraint will be: $\mathbf{r_1 = (k_{i,j} , ki_{+1,j} \dots k_{ie,j})}$ where i is the starting position and e is the end
The value of a number should not be repeated in a row or a column.
$\mathbf{K_{i,ju} != k_{i,jv}}$
And each constraints values must sum up to the provided constraint sum. For $\mathbf{r_i}$ summation of Summation ($\mathbf{k_{i,j}}$ belonging to constraint list) $\mathbf{k_{i,j} = t_i}$
Upper bound to the number of possible states of a kakuro puzzle is 9^x where x is the number of unfilled squares. For the puzzle above it is 9^8 which is 43 million. This limit is greatly reduced by using the limit to what the numbers can be placed.

A set of values, , that may be assigned to cells in a run is constructed, such that:

$$P_l = \{1, \dots, 9\} \qquad \text{if } t_l > 9$$

$$P_l = \{1, \dots, a\text{-}1\} \qquad \text{if } t_l = a,\ a \leq 9$$

The improved upper bound would then be:

$$\prod_{i=1}^{w} \min\{|P_l| \mid c_i \in r_l\}$$

The positioning of runs, and the selections of run totals of Kakuro puzzles can vary greatly. This makes the task of devising a general formula for the exact number of possible Kakuro grid arrangements of a given size difficult, if not impossible. We have used backtracking algorithm for our solution.

For a MxN kakuro the recurrence relation is T(NxM) = 9*T(NxM - 1) + O(1)
The time complexity is **O(9^{NxM})**
Space complexity will be **O(NxM)**

Worked example

For the above problem the working will be :
For all the values which are '_' we will try to fill with a number from 1 to 9.
For the first row the sum is 3 in two combination 1 2 and 2 1 bot will be put and checked for all the other possible solutions derived from these sattes if at a point all the squares are filled with satisfaction of all the associated constraints we return this solution

```
. . _ _                          . . 2 1
_ _ _ _                          _ _ _ _
_ _ . .                          _ _ . .


. . 1 2
_ _ _ _          .....           . . 2 1
_ _ . .                          3 1 4 2
                                 1 2 . .
```

Summary and Conclusions

This project to solve a kakuro puzzle is the only solver which does not need any background installation and can run out of the box. It does not require any excessive data entry so even though it's a bit slow, it can give a solution to 12x12 puzzles in reasonable time. Given a hard time constraint of an hour this puzzle will always find the solution to a reasonable sized kakuro puzzle.
In order to improve this project we can include another algorithm such as forward checking to solve faster. This will be the future scope of this project

References

1. http://techieme.in/solving-sudoku-using-backtracking/#:~:text=Talking%20about%20the%20space%20complexity,would%20be%20O(M)
2. http://www.cs.toronto.edu/~fbacchus/Papers/liu.pdf
3. https://pure.southwales.ac.uk/files/119984/Roach_automation_2008.pdf
4. http://ai.berkeley.edu/home.html