Solving Kakuro puzzle – comparison of deterministic approaches

Stojanche Panov[1] and Saso Koceski[2]

Abstract – **This paper presents comparison of existing deterministic approaches to solving Kakuro puzzles and provides a novel deterministic approach to obtaining an optimal solution for such puzzles, i.e. the Reducing Domain Values algorithm. This algorithm has been compared to other published deterministic approaches and proved to obtain better results in terms of time execution for obtaining an optimal solution. This provides a great foundation for development of effectual deterministic algorithms which would have great impact on solving coding theory problems.**

Keywords – **Backtracking algorithms, Coding theory, Deterministic algorithms, Kakuro puzzles.**

I. INTRODUCTION

Kakuro puzzles are considered to be a mathematical transliteration to crossword puzzles. They are consisted of an *n* x *m* grid of black and white cells, where black cells can be either empty, having one number or two numbers, indicating the row or column sum they indicate, whereas every white cell can be typically filled with numbers in range [1; 9]. There are also variations of this puzzle that use greater or smaller ranges. Hence, the typical maximal sum that can be obtained with these puzzles is 45, which gives enough information for constructing domain sets of candidate values for the white cells to be filled with. Every constructed sum of elements in a row or column must satisfy the constraint of unique numbers, i.e. none of the numbers must appear more than once in a concrete combination sum. Example of Kakuro puzzles are shown in Fig. 1 and Fig. 2.

Kakuro puzzles have been considered as a great logical challenge, not only for manual solving, but also for developing algorithms that can solve these puzzles as effectively as possible in a real-time acceptable manner. There have been many approaches that solved these puzzles, both with deterministic and metaheuristic methods.

The main goal of this research study is to present a novel method of solving a Kakuro puzzle, namely the Reducing Domain Values (RDV) Algorithm. This new algorithm is then compared to other deterministic approaches for solving and gives detailed and elaborated results that show the relevance of the discovery of this novel method.

[1]Stojanche Panov is with the Faculty of Computer Science at 'Goce Delchev' University - Stip, blvd. Krste Misirkov bb., 2000 Stip, R. Macedonia, E-mail: stojance.panov@ugd.edu.mk.

[2]Saso Koceski is with the Faculty of Computer Science at 'Goce Delchev' University - Stip, blvd. Krste Misirkov bb., 2000 Stip, R. Macedonia, E-mail: saso.koceski@ugd.edu.mk.

II. RELATED WORK

A Kakuro puzzle consists of several constraints that ought to be respected in order to get a unique solution, hence one can treat such puzzle as a constraint satisfaction problem, which has been recently published [1]. The NP-completeness of the Kakuro solving problem has already been proven and documented [2], but there have been also some other proofs of this NP-completeness that included using SAT solvers for the purposes of the research studies [3]. A relatively new study presented an approach that significantly reduced the execution time of the algorithm by using generalized arc consistent (GAC) version of all-different sums constraint, and these performances have been compared to MIP [4] and SAT techniques [5].

Existing algorithms published in the past several years solved this problem and presented many deterministic and metaheuristic approaches to solving a Kakuro puzzle. One such type of a deterministic method with using backtracking and simple heuristics and pruning has been shown to be of an eminent matter to coding theory problems as well [6]. A research study presented several types of algorithms, both deterministic and metaheuristic approaches, including stack-based backtracking solvers, genetic algorithms and tabu search [7]. There has also been another metaheuristic approach, known as nested Monte-Carlo of level 2 method which proved to be effective and performed with accelerations in execution [8].

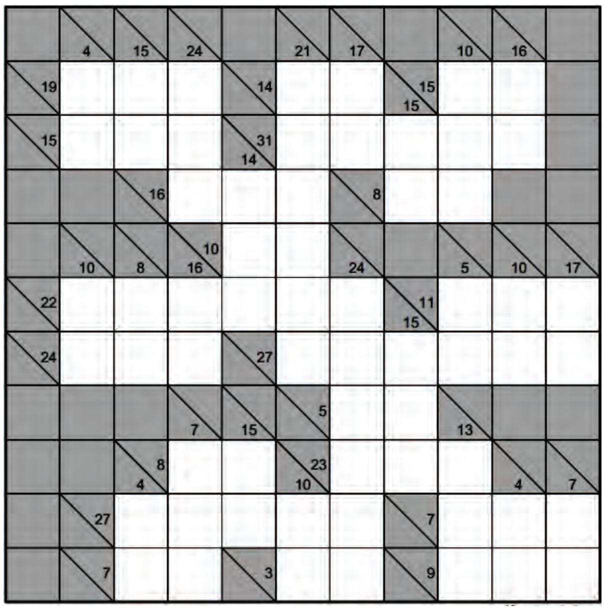

Fig. 1. Example Kakuro puzzle

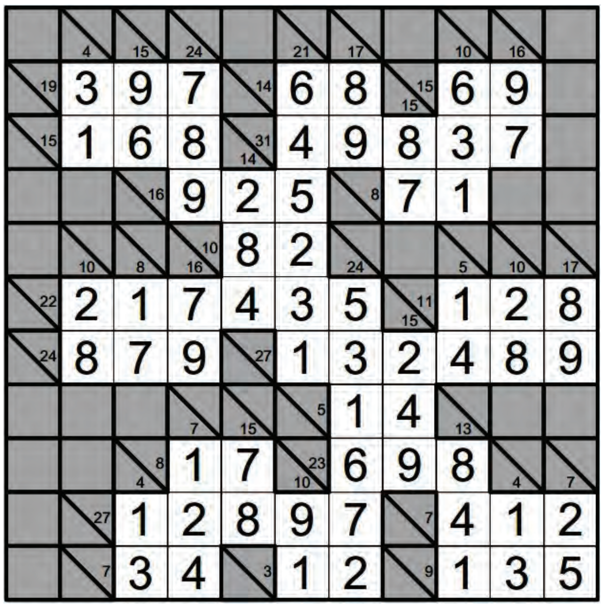

Fig. 2. Solution to Kakuro puzzle in Fig. 1.

III. Stack-Based Backtracking Solver

The Stack-Based Backtracking Solver [8] implements a simple backtracking technique starting with an empty Kakuro grid with domain sets of values available for each of the white cells in the grid. It presents a depth first search technique that starts with smaller number values and makes assignments of these numbers to the white cells, whilst having affinity to assign the smallest numbers first, i.e. in the earlier stages of the algorithm. It's based on keeping the grid states on a stack, thus repeatedly checking the validity of the grid. When assigning the values to the grid, it performs checking of the constraint satisfaction for the stored states. If the next value needed to be assigned to a cell violates such constraint, then it's excluded for searching, otherwise it's kept on the stack of grid states. Continuing in this fashion, this algorithm checks all of the possible states until it finds a certain state that satisfies all of the constraint having filled all of the white cells with the proper numbers.

This algorithm can be described in the next detailed steps:

1. Initialize the stack of states and other iteration variables.
2. Initialize the start state as current state and start cell as current cell.
3. Check if there are empty cells in the grid. Check for validity of state. If state is valid, continue to step 4. If this is false, then backtrack and try other assignments, continuing to step 3.a. Otherwise, continue to step 3.a.
    a. Assign a value to next free cell.
    b. Check for validity of assignment of the value. If the check is valid, then push this state on the stack and repeat from step 3.a. for another free cell. Otherwise, continue with assigning another value for the cell.
4. Print optimal solution.

There are also several variations to this well-known simple algorithm. These are the Run-Based Ordering, Value Ordering, Decisive Value Ordering and Project Run Pruning. These algorithms are detailed in the next following sections.

IV. Run-Based Ordering

This type of backtracking approach uses a simple elimination of not needed values that are meant to be assigned to the white cells. This means that a kind of heuristics needs to be implemented for this to be fulfilled. This heuristics utilizes the values that are candidate members in the row and column sums. This would mean that for a given white cell, an intersection of domain set values is computed and only the numbers that are valid candidates for that cell remain in the backtracking process. For instance, if there's a column sum of 6 containing 3 elements and a row sum of 4 containing 2 elements, the possible column sums are contained from the numbers {1, 2, 3}, whereas the row sum can be obtained by the numbers {1, 3} (since 2+2 is not acceptable according to the puzzle constraints), so the intersection of the two domain sets is {1, 3} and these would be the candidate values for that white cell.

V. Value Ordering

The Value Ordering variation of the Stack-Based Backtracking algorithm consists of having all of the numbers in the range [1, 9] in the domain sets for the white cells, but with additional heuristics of certain ordering (sorting) of the numbers. One such example would be if the domain set candidate variables are sorted in descending order, which is certainly a poor heuristic, but it is a heuristic that would help if the values in the first few cells have greater solution values for those cells. This means that this heuristic still stays efficiently applicable only for smaller grid sizes and concrete types of solutions, which is the same case with the Stack-Based Backtracking algorithm.

VI. Decisive Value Ordering

As an addition to the Value Ordering heuristics, the heuristic of the Decisive Value Ordering can be defined based on research on what solution values are statistically more present in the first few white cells. This would mean computing some kind of average values for numbers appearing in the white cells, and then using this information to construct a type of sorting of the domain set elements that works best for all of the puzzles that will be processed as input to the algorithm. For instance, if the average of values is smaller than 5, then the ordering of the values is in increasing order. Otherwise, if the average value is greater than 5, then a decreasing ordering of values is used.

VII. Projected Run Pruning

This approach, previously mentioned as a modification to the Stack-Based Backtracking method, differentiates from previously described techniques in the way of reducing the domain sets of values for the white cells. Namely, this method introduces a type of pruning that excludes all of the values in domain sets that give sums smaller than the required column sum or row sum, i.e. if a value of 7 needs to be obtained from three numbers, than we know that when getting to the value assignment of the triple {1, 2, 3} it doesn't add up to 7, then this combination of values is discarded from search and only sums that add up to 7 and greater are included in the search space. This proved to be of an eminent improvement of the algorithm and gave great accelerations in time execution of the algorithm [8].

VIII. Reducing Domain Values Algorithm

This novel deterministic algorithm, called the Reducing Domain Values algorithm, is an approach that uses a very sophisticated heuristics that reduces the domain sets of values for the white cells, thus converging to the optimal solution in a shorter period of time than other naive backtracking approaches. This reduction is done in a continuous manner that reduces number of the sets to the point where there's no more numbers to be removed and only one value remains in each of the domain sets, i.e. the solution value for the cell. This algorithm proved to perform better than other deterministic techniques, which is described later in this research.

The Reducing Value algorithm consists of the next simple steps:

1. For each white cell, compute the intersections of members in the column and row sums and set these newly generated sets to be the new domain sets for the cells.
2. Perform a check if there's a white cell that has a domain set with more than one candidate value. If this is false and all of the domain sets have one value, then proceed to step 5. Otherwise, proceed to step 3.
3. Check for unique numbers in the cells, i.e. numbers that are not present in each of the other cells in the same row and column.
4. If there are unique numbers, these values indicate the solution values for these cells. Return to step 2.
5. Print found optimal solution.

IX. Results

All of the above mentioned algorithms, including the novel Reducing Domain Values (RDV) algorithm, have been compared in terms of average time execution for a puzzle. The results for the Stack-Based Backtracking, Run-Based Ordering, Value Ordering, Decisive Value Ordering and Projected Run Pruning have been utilized from the research presented in [8], and then these results have been compared with the results that were obtained from examining the Reducing Domain Values. The RDV algorithm was written in Java programming language and tested on a machine having Intel® Core i5 processor with frequency of 2.53 GHz, 4GB of RAM, and 64bit Windows 8 operating system. Grid sizes of 2x2, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9 and 10x10 have been tested, each of the grids having 0-40% black cells coverage in the Kakuro grid.

The results from the examination of the algorithms and their comparison are detailed in diagrams in Fig. 3, Fig. 4 and Fig. 5. As results have shown, the RDV algorithm performed better than all of the other available deterministic algorithms. This is due to the fact that RDV continuously reduces the domain sets, thus reducing the search space and leaving the solution number as the only label for the white cells of interest.

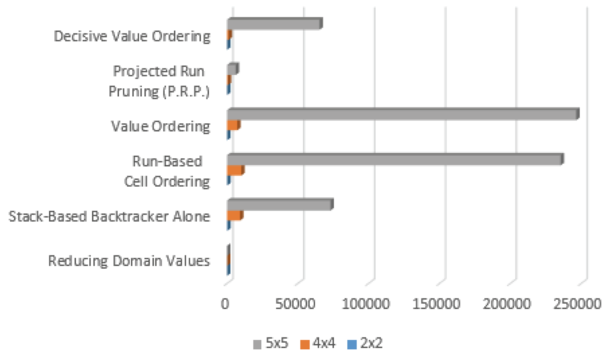

Fig. 3. Comparing algorithms for 2x2, 4x4 and 5x5 grids in dependence of time execution.

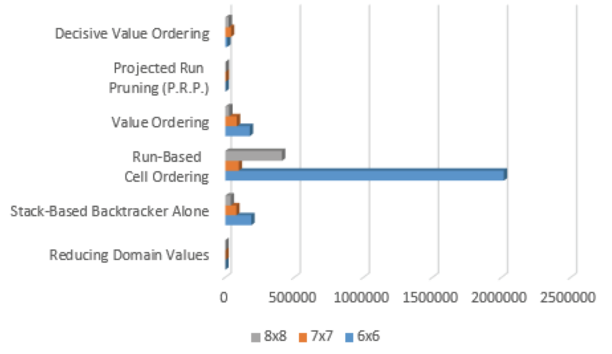

Fig. 4. Comparing algorithms for 6x6, 7x7 and 8x8 grids in dependence of time execution.

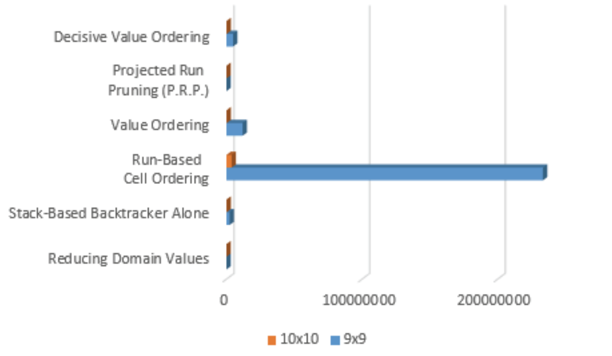

Fig. 5. Comparing algorithms for 9x9 and 10x10 grids in dependence of time execution.

X. Conclusion

This research paper introduced a novel Reducing Domain Values algorithm, i.e. a deterministic algorithm that is intended to provide an efficient solution and accelerate the process of obtaining an optimal solution, thus converging in a small amount of time. This Algorithm has been compared to other available deterministic approaches and has proved to perform better in terms of time execution. This algorithm is a great foundation for developing efficient deterministic algorithms for game theory, thus applying their principles in other fields, such as coding theory.

REFERENCES

[1] Simonis, Helmut. "Kakuro as a constraint problem." Proc. seventh Int. Works. on Constraint Modelling and Reformulation (2008).

[2] Seta, T. A. K. A. H. I. R. O. "The complexity of CROSS SUM." IPSJ SIG Notes, AL-84 (2002): 51-58.

[3] Ruepp, Oliver, and Markus Holzer. "The computational complexity of the KAKURO puzzle, revisited." Fun with Algorithms. Springer Berlin/Heidelberg, 2010.

[4] Achterberg, Tobias, Thorsten Koch, and Alexander Martin. "MIPLIB 2003."Operations Research Letters 34.4 (2006): 361-372.

[5] Eén, Niklas, and Niklas Sörensson. "An extensible SAT-solver." Theory and Applications of Satisfiability Testing. Springer Berlin/Heidelberg, 2004.

[6] Davies, R. P., P. A. Roach, and S. Perkins. "Automation of the Solution of Kakuro Puzzles." Research and Development in Intelligent Systems XXV(2009): 219-232.

[7] Davies, Ryan P. "An investigation into the solution to, and evaluation of, Kakuro puzzles." Unpublished M Phil thesis. University of Glamorgan (2009).

[8] Cazenave, Tristan. "Monte-Carlo Kakuro." Advances in Computer Games(2010): 45-54.