

Application of Integer Linear Programming to Kakuro Puzzles

Hwanseok Yi¹ *

¹Portola High School, Irvine, CA, USA

*Corresponding Author: ryanhyi2008@gmail.com

Advisor: Benjamin Cates, catesb@uci.edu

Received December 4, 2025; Revised February 1, 2026; Accepted February 13, 2026

Abstract

Kakuro puzzles are numerical logic puzzles that combine uniqueness constraints with sum constraints, making them structurally different from other combinatorial puzzles and posing distinct challenges for constraint-based solution techniques. While Integer Linear Programming has been applied to related logic puzzles, there is limited prior work that directly compares alternative ILP variable encodings for Kakuro or evaluates their practical performance trade-offs. This paper investigated two Integer Linear Programming encoding strategies for Kakuro: either (1) binary one-hot vectors for each cell or (2) integer value variables. A dataset of 1,500 Kakuro puzzles, spanning five difficulty levels, was solved using both encodings in the PuLP solver. Runtime and solver iteration counts were collected across ten repeated trials per puzzle. The results showed that the integer encoding consistently outperformed the one-hot binary encoding by a factor of 1.5-2x across all difficulty levels, suggesting that the reduced constraint count and simplified sum requirements provided a practical computational advantage. These findings highlight that variable encoding choices impact performance when solving puzzles with additive and uniqueness constraints and may inform modeling decisions in real-world optimization tasks that share similar structural properties.

Keywords: Constraints, Mathematics, Linear programming, Puzzles

1. Introduction

Kakuro puzzles, also known as Cross-Sum puzzles, are a rectangular grid of black and white cells, which the solver must fill with digits. The empty white cells are organized into overlapping continuous runs that are either horizontal or vertical. A run total is given in the adjacent black clue cell. The puzzle is solved by entering values 1 through 9 into the white cell such that each run sums to the correct run total and no digit is duplicated in the same run (Miyahara, 2019). Since no digit can be duplicated, a run can be at most 9 digits long, and the sum must be in the range 1 through 45. An example of a kakuro puzzle is reproduced in Figure 1.

Sudoku puzzles have been associated with several significant real-world applications, such as timetabling (Gomes, 2002), conflict-free wavelength routing (Duto, 2003), and photovoltaic arrangement (Ye, 2023). In contrast, relatively little research has focused specifically on Kakuro. Nevertheless, solving Kakuro has been proven to be NP-complete (Ruepp, 2010) by establishing its relationship to the Hamiltonian Path Problem and to 3SAT, a restricted form of the Boolean satisfiability problem.

Because of the similarity to Sudoku puzzles, Kakuro puzzles present a unique problem because they may have applications in applied mathematics. However, Kakuro is significantly different from Sudoku

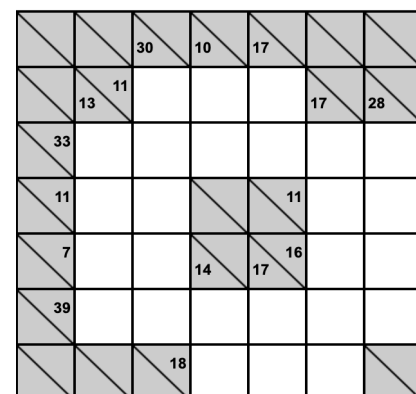


Figure 1: An example 6x6 unsolved Kakuro puzzle contained in a 7x7 grid.

from a constraint-satisfaction angle. Sudoku only considers groups under the constraint that every member must be different, but Kakuro has groups where every member must be different, and every member must sum to a certain value. This difference increases the number of constraints, which reduces the possible search space, but also requires more computational effort to hold the many constraints.

Integer Linear Programming (ILP) is a popular way to encode and solve combinatorial puzzles as constraint satisfaction problems. However, encoding puzzles like these requires a strategy to keep the number of variables and constraints limited (Chen, 2011). For Kakuro puzzles, there are two ways to do it. First, each cell could be encoded as nine binary variables, where a one-hot encoding of that vector would describe the digit in the cell. This requires the sum of each cell-vector to be 1, and the difference constraint requires all of the variables corresponding to the same digit in the same group to sum to 1. Then, the run sum constraint requires many variables in one expression because each boolean in that group must be multiplied by its corresponding value and summed together, meaning the constraint has 9 boolean variables per cell.

The other method is to encode each cell as an integer variable. This removes the need for the constraint on each cell to only represent one number. Still, now each adjacency requires a new constraint because the constraint requires two numbers to be different. This requires a third variable and two constraints for each difference requirement.

For Sudoku puzzles, the first encoding requires 243 boolean variables with 972 sum constraints, and the second encoding requires 81 integer variables, 405 binary variables, and 810 difference constraints. In Kakuro, the same two strategies can be applied; however, no academic research has studied the comparison between these two encoding methods.

2. Methods

This paper evaluates the computational performance of two Integer Linear Programming formulations to solve Kakuro puzzles: (1) a binary one-hot encoding and (2) a single integer variable encoding. The purpose of this comparison is not only to determine which method is faster in practice, but also to explore how the structure of Kakuro itself interacts with different modeling strategies. All experiments were conducted on the same hardware (M2 MacBook Air with 8 GB RAM) to maintain consistency in runtime and solver iteration behavior.

For both models, a puzzle was represented as a collection of horizontal and vertical runs. A run was defined as a sequence of between two and nine contiguous white cells paired with a clue specifying the sum of its digits. The one-hot encoding treated each cell as a vector of nine binary decision variables $x_{i,j,k}$, where ij refers to the grid location and k refers to the candidate 1-9. The model enforced single decisions by adding the constraint $x_{i,j,1} + x_{i,j,2} + x_{i,j,3} + \dots + x_{i,j,9} = 1$ for each white cell ij , which is a common technique when solving combinatorial puzzles with ILP (Bartlett, 2008). Then, the region sum constraint is enforced by summing by multiplying each binary variable by its corresponding digit and summing across the run. An example constraint for a run of three white cells (A,B,C) summing to 11 would be $1 * (x_{A,1} + x_{B,1} + x_{C,1}) + 2 * (x_{A,2} + x_{B,2} + x_{C,2}) + \dots + 9 * (x_{A,9} + x_{B,9} + x_{C,9}) = 11$. Finally, to ensure each run has distinct numbers, the variables for a specific value across a whole run must sum to less than or equal to one. An example for a run of three white cells (A,B,C) checking for exclusivity of 5 would be $x_{A,5} + x_{B,5} + x_{C,5} \leq 1$. These three kinds of constraints convert puzzle information into an integer linear programming problem.

For integer encoding, each cell is represented as a single integer variable $y_{i,j}$ where i,j is the position of the cell. There is no need to ensure each cell chooses one value because each cell is only one variable. The sum constraint is encoded as the sum of variables in a run must equal the run sum. For example, a run of three white cells (A, B, and C) summing to 11 would be $y_A + y_B + y_C = 11$. The difficulty of integer encoding is ensuring there are no duplicates in the same run. Since integer linear programming does not have a “not equals” condition, the constraint representing their difference has to use inequalities. For each combination of two variables that share the same run, a third binary “delta” variable is created to encode which variable is bigger. Then, the constraints $y_A < y_B + 20*\delta$ and $y_A > y_B - 20*(1-\delta)$. Twenty is chosen because it is more than twice the maximum difference between variables. If y_A is greater than y_B , then delta will be 1, and both conditions will be true. If y_A is less than y_B , then delta will be 0, and both conditions will be true. If y_A equals y_B , then there is no valid value of delta that satisfies both equations, meaning

y_A cannot equal y_B . These three kinds of constraints encode all of the puzzle information into an integer linear programming problem.

2.1 Dataset

The dataset was taken from randomly generated puzzles on the website Kakuro Conquest (<https://kakuroconquest.com>). Three hundred puzzles sized 9 by 17 were collected for each level of difficulty listed on the website (easy, intermediate, hard, challenging, expert), then converted into a standardized format that was a list of clues and empty cells.

2.2 Code

A Python program was written using the PuLP 3.2.1 library to solve the Kakuro puzzles in two different ways. To reduce the risk of solver-specific biases, the built-in default solver CBC was used without enabling optional cuts or heuristics.

Each puzzle was encoded in two different methods, and each was solved using PuLP. The time to solve each puzzle with each method was averaged across 10 runs to account for variability in computer speed. Additionally, the number of iterations was stored to compare the amount of effort across difficulties without the uncertainty of computer runtime.

To verify correctness, solver output was parsed and checked after every solution to ensure (1) no duplicates exist within a run, (2) the assigned digits satisfy the run sum, and (3) values remained within the domain [1–9]. All data were aggregated and analyzed using Python’s pandas library.

3. Results

The results show that encoding the puzzle cells as single integer variables was 1.5-2x faster across all difficulties. The runtime is graphed in Figure 2. This demonstrates that the variable reduction and simplicity of the sum constraints outweighed the additional complexity of larger variable values.

Across all five difficulty categories, the integer encoding not only reduced mean runtime but also demonstrated significantly lower variance between trials, indicating greater prediction stability and less sensitivity to puzzle structure.

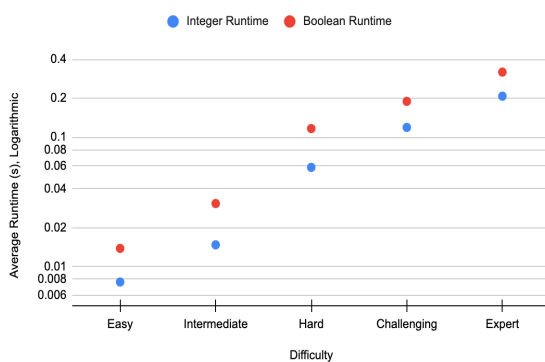


Figure 2. Integer encoding runtime vs Boolean encoding runtime as a function of difficulty. The y-axis is logarithmic.

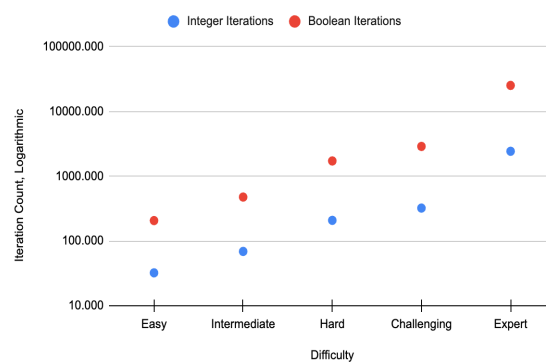


Figure 3: Average iteration counts for boolean and integer puzzle encodings. The vertical axis is logarithmic.

The iteration count follows a similar pattern to the runtime, but cannot be compared across constraint encoding because the increased number of binary variables requires more iterations that will be processed faster. The average iteration counts for each difficulty are reproduced in Figure 3.

4. Discussion

The results indicate a consistent performance advantage for the integer-variable encoding over the binary one-hot encoding across all tested difficulty levels. This finding contrasts with the common assumption in the constraint-programming literature that increasing the granularity of decision variables (e.g., using binary expansion) improves solver performance by giving the linear relaxation more structure to exploit. In the case of Kakuro, however, the additional constraints introduced by the one-hot model appear to dominate any potential benefit from a tighter LP relaxation.

These results underscore a key difference between Sudoku and Kakuro when encoded as ILPs. Sudoku's constraint structure is dominated by all-different constraints, for which binary encodings naturally align with OR-tools such as clique inequalities and set-packing formulations. Kakuro, in contrast, introduces sum constraints that restrict which sets of digits can appear together in a run. Many of these allowable sets are small and combinatorially rigid; the solver can immediately prune large portions of the search space without relying on difficult-to-solve binary encodings.

However, limitations should be noted. First, the puzzles all share the same grid size (9×17); larger instances may shift the balance between encodings. Second, all of the puzzles come from the website Kakuro Conquest, which follows symmetrical structures that may not be present in an application of a Kakuro solver. Future work could evaluate both encodings on a broader range of grid sizes and irregular puzzle layouts to assess scalability and generalizability. Additionally, experiments using synthetically generated Kakuro instances or puzzles from multiple sources could help isolate how structural properties influence solver performance.

5. Conclusion

This work shows that using a single integer variable for each Kakuro cell offers a clear performance advantage over the more traditional binary one-hot encoding. While the binary approach is often favored in constraint programming because of its structure, Kakuro's unique combination of sum and uniqueness requirements benefits from a simpler model that avoids a large buildup of constraints. The results suggest that the compactness of the integer formulation reduces solver workload enough to outweigh the benefits of finer-grained binary decision variables, offering a practical and efficient strategy for this type of puzzle.

Although this study focused on recreational puzzles, the findings point toward broader applications. Kakuro mirrors Sudoku puzzles, whose solutions are found in real scheduling, packing, and resource management problems. Understanding which encoding strategy performs better can help inform how similar real-world optimization problems should be modeled, especially in fields like logistics, education scheduling, or constrained resource distribution. In this way, Kakuro is a valuable test case for exploring how best to frame and solve additive, constraint-driven problems.

References

- Bartlett, A., et al. (2008). An integer programming model for the Sudoku problem. *Journal of Online Mathematics and its Applications*, 8(1), 1798.
- Chen, D. S., Batson, R. G., & Dang, Y. (2011). *Applied integer programming: modeling and solution*. John Wiley & Sons.
- Davies, R. P., Roach, P. A., & Perkins, S. (2010). The Use of Problem Domain Information in the Automated Solution of Kakuro Puzzles. *IAENG International Journal of Computer Science*, 37(2).
- Dotú, I., Del Val, A., & Cebrián, M. (2003, September). Redundant modeling for the quasigroup completion problem. In *International Conference on Principles and Practice of Constraint Programming* (pp. 288-302). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-45193-8_20
- Gomes, C. P., & Shmoys, D. B. (2002, March). The promise of LP to boost CSP techniques for combinatorial problems. In *Proc., Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, Le Croisic, France (pp. 25-27).

Miyahara, D., et al. (2019). Card-based physical zero-knowledge proof for Kakuro. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 102(9), 1072-1078.

Ruepp, O., & Holzer, M. (2010, June). The computational complexity of the Kakuro puzzle, revisited. In International Conference on Fun with Algorithms (pp. 319-330). Berlin, Heidelberg: *Springer Berlin Heidelberg*. https://doi.org/10.1007/978-3-642-13122-6_31

Ye, C. E., Tai, C. C., & Huang, Y. P. (2023). Disperse partial shading effect of photovoltaic array by means of the modified complementary SuDoKu puzzle topology. *Energies*, 16(13), 4910. <https://doi.org/10.3390/en16134910>