# Step-wise Explanations of Sudokus using IDP

William Dumez[1], Simon Vandevelde[2,3], and Joost Vennekens[2,3]

[1] william.dumez@skynet.be
[2] KU Leuven, De Nayer Campus, Dept. of Computer Science, Belgium
[3] Leuven.AI – KU Leuven Institute for AI, B-3000 Leuven, Belgium
{s.vandevelde, joost.vennekens@kuleuven.be}

Most Sudoku solvers are typically written in imperative programming languages, and follow one of two main approaches. The first approach is backtracking, which can solve any Sudoku (or variant), but not in a way that humans can easily understand. The second approach is to apply hand-crafted patterns, called *solving strategies* [3]. With this approach, the steps taken to solve the Sudoku also form a step-wise explanation. However, this approach only works if the set of solving strategies that is used actually suffices for the given Sudoku. Moreover, while many strategies are well-documented for regular Sudoku [3], they are not well-known for variants such as Killer Sudoku or Thermo Sudoku. The goal of this thesis was to develop a one-size-fits-all, universal tool to solve and explain Sudokus and Sudoku variants that requires only knowledge of the Sudoku (variant) rules.

ZebraTutor [1] is an application developed with the aim of solving logic grid puzzles, and explaining them in a step-wise manner. It uses a general method for explaining constraint satisfaction problems, based on *minimal unsatisfiable subsets* (MUS) [1]. The application employs a greedy algorithm, that builds an explanation sequence by finding the "easiest next step" by calculating a MUS for each possible step, and applying a cost function. For the MUS generation, ZebraTutor uses the IDP System [2], a knowledge-based reasoning engine for first order logic.

We applied this method to Sudoku and found that, in general, it works well for generating understandable explanations. However, the algorithm accepts a cost function which is specific to the constraint satisfaction problem. We chose to use a cost function that is a linear combination of the number of cells used in the MUS, as well as the number of constraints. Intuitively, the more cells and constraints are required to explain a step, the harder it is to understand. This cost function was chosen mainly for its modularity, since it allows for easily adding new constraints, which is necessary when combining Sudoku variants. Fig. 1 shows two visualisations of a preferred solving step, as generated by the algorithm.

The main downside of this algorithm is the large quantity of MUSs that must be calculated; a single MUS can take a couple of minutes, so generating the explanations for an entire Sudoku can take a full week. Another problem lies in IDP's MUS generation. It generates MUSs that are subset-minimal, but do not necessarily contain a minimal number of atoms. As such, sometimes explanations are more complex than they could be.

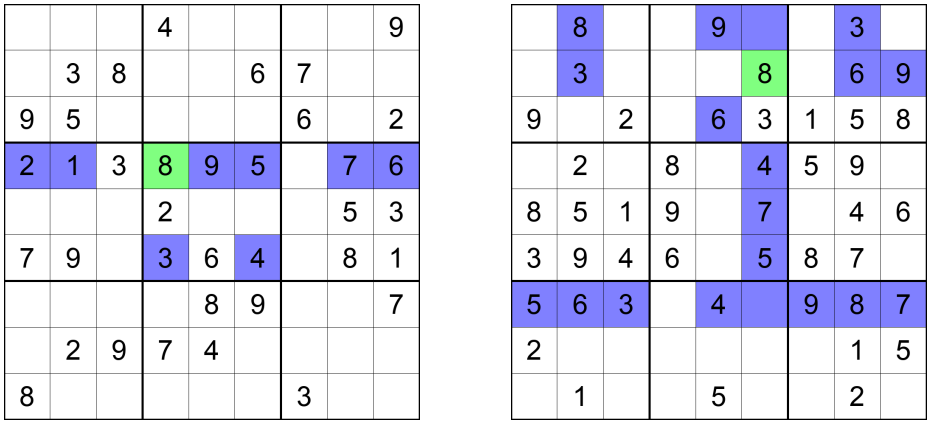


**Fig. 1.** Explanation steps generated by the Sudoku explanation tool for different puzzles. The cell in green is filled in with an "8". All blue cells are necessary to understand why this is the case. The image on the left corresponds to an easy explanation, and the image on the right corresponds to a harder explanation.

In order to provide more detail for harder-to-understand explanations, we tried a variant of the ZebraTutor algorithm which produces eight sub-explanations per solution step. This is done by explaining why a cell cannot be each of the eight other values, instead of explaining why it must be a specific value. We expected that this would result in more fine-grained explanations that show exactly which cells are involved in ruling out each value. However, instead we found that previously easy explanations now had redundant sub-explanations, while previously hard explanations remained hard to understand as one of the sub-explanations was virtually identical to the full explanation generated by the original ZebraTutor method.

The running time can be improved by using multithreading and calculating fewer unsat cores. We did this by first searching for cells which could be explained by the easiest solving strategies, and then skipping the unsat core calculations for any cells that could not be solved through those strategies. The reasoning here is that any cells that are skipped are guaranteed to have harder explanations anyway, and are therefore not worth calculating unsat cores for. This method can provide a significant speedup, but is ultimately reliant on the complexity of the Sudoku puzzle. The main problem with this method is that searching for solving strategies requires extra IDP code that is specific to Sudoku variants, which means this code must be updated to allow solving for new Sudoku variants.

In summary, a universal Sudoku explanation tool was created that uses the ZebraTutor algorithm and requires no knowledge of existing solving strategies. This algorithm works well for easy steps, but harder steps could use more detailed explanations. The solver is slow because of the many required MUS calculations, and IDP's MUS generation sometimes results in explanations that are more complicated than would intuitively be deemed necessary.

## References

1. Bogaerts, B., Gamba, E., Guns, T.: A framework for step-wise explaining how to solve constraint satisfaction problems. Artificial intelligence **300**, 103550 (2021)
2. De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., Denecker, M.: Predicate logic as a modelling language: The idp system (2014)
3. Stuart, A.: Sudokuwiki.org - strategies for popular number puzzles (2019), https://www.sudokuwiki.org/Main_Page