An Interactive Application to Solve Sudoku Variant Puzzles

Helmut Simonis
Insight Research Ireland Centre for Data Analytics, School of Computer Science and Information Technology, University College Cork, Cork, Ireland

Luis Quesada
Insight Research Ireland Centre for Data Analytics, School of Computer Science and Information Technology, University College Cork, Cork, Ireland

Abstract

Logic puzzles like Sudoku have been very popular for several decades, some videos showing the solution steps for a complex Sudoku variant instance have over 10 million views. This makes this problem domain an ideal area to demonstrate the advantages of using Constraint Programming to a wider audience. We present an interactive application for solving Sudoku variants, extending the classical Sudoku grid with a variety of additional constraints. The aim is not just to search for solutions of a given puzzle, but to help a user to find a solution interactively. We use an off-the-shelf constraint solver, Choco-solver, to model and propagate just over 200 common Sudoku variant constraints. This uses a combination of specific global constraints and table constraints to describe feasible or infeasible tuples. In other cases, we use logical expressions over basic constraints to model complex side constraints. The user can add constraints interactively if the solver does not find the solution by propagation alone, or rely on an agent to add deduced constraints dynamically. We evaluate the solver on different data sets, and show that over 97% of the problems in the 2025 Sudoku Grand Prix can be solved by the tool, and that many popular, complex puzzle instances can be solved by a handful of manual constraints and propagation alone.

2012 ACM Subject Classification Applied computing Computer games

Keywords and phrases Sudoku,Interactive Solving,Computer Games,Constraint Programming

Funding This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 12/RC/2289-P2 at Insight, the SFI Research Centre for Data Analytics at UCC, which is co-funded under the European Regional Development Fund.

1 Introduction

Sudoku and its variants have been around since the 1980s, they are still a very popular hobby today, with players at very different skill levels either solving puzzles themselves, or watching videos of experts solving puzzles, while explaining the reasoning process. Popular sites, like "Cracking the Cryptic", provide daily videos, often more than an hour long. A single video, about the Magic Square Sudoku designed by Aad van de Wetering (see Section 6.4), has more than 10 million views on YouTube.

We do not aim to develop a solver that can solve Sudoku variant puzzles automatically by search. Instead we want to have a solver that supports a human user in solving the puzzles, taking care of tedious detail, and providing all information that can be useful to the human puzzle solver. It therefore relies heavily on propagation, deducing that certain values in a cell cannot be part of the (unique) solution, instead of automatically finding values for all cells in the puzzle by search. We have coded a total of just over 200 Sudoku variant constraints in our solver, using the Choco-solver [17] back-end.

The paper is structured as follows: In Section 2 we introduce a running example which shows some of the most common variant constraints, and Section 3 gives an overview of the overall application architecture. This is followed in Section 4 by an overview of the existing

literature on constraint-based Sudoku solvers. We then present a morphology of the different variant constraints used in the system in Section 5, and describe how selected structural and manual constraints are modelled in our system. We then (Section 6) present an evaluation of the system on four sets of problem instances, before concluding with a description of future work in Section 7.

2 A Running Example

To show some of the possible Sudoku variant constraints, and give an idea of the user interface of the application we use the puzzle "Second Breakfast" given by designer "99%Sneaky", for which a manual solution video was presented by BremSter. Figure 1 shows the initial state of the puzzle. We see the Sudoku grid on the left, the rules used by the puzzle on the right, and buttons for possible actions at the bottom. Cell numbers from 1 to 81 are shown at the bottom right of each cell, we refer to cells by that number in our explanations.

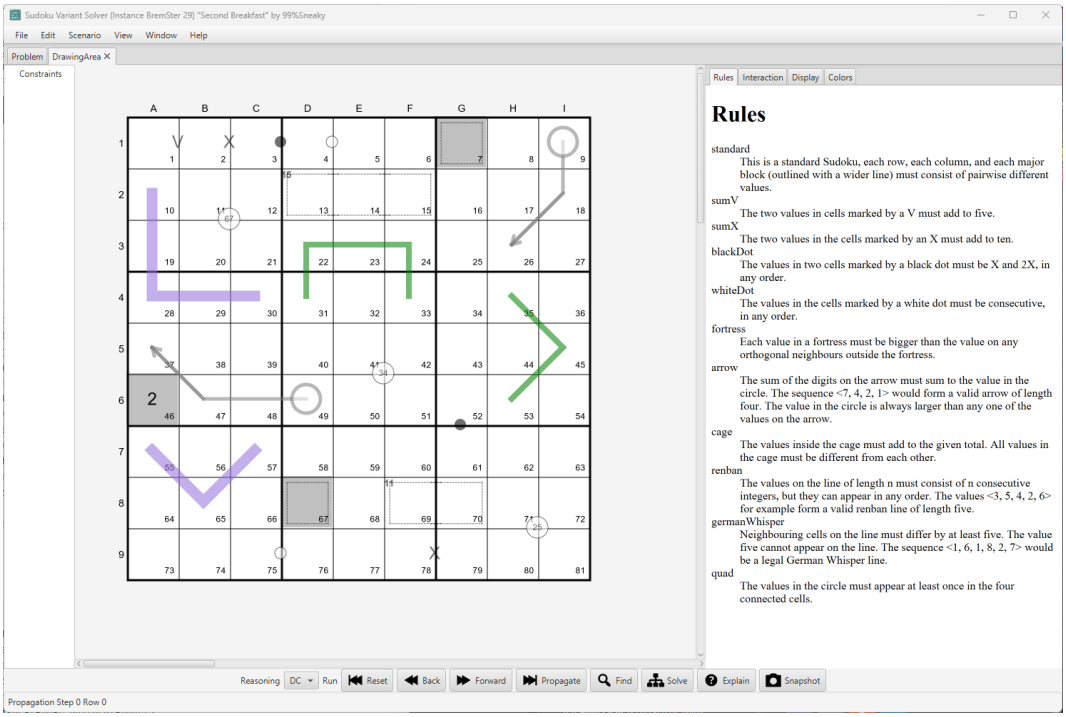

Figure 1 Running Example, Initial State and Rule Set Used

The constraints used in this instance are[1]:

standard the standard Sudoku rules apply, the values in each row, each column, and each outlined 3x3 block must be pairwise different, and range over values from 1 to 9.

preset A large number written in a gray cell gives the preset value for the cell.

fortress Nodes marked with gray must be greater than all orthogonally connected neighbours that are not marked in gray.

sumV Node pairs marked by a V must sum to 5.

[1] The constraint names and visualisations follow the common use in the Sudoku community. A definition of all constraints defined in our system is available as the Sudoku Variant Catalog.

sumX Node pairs marked with X must sum to 10.

blackDot Node pairs marked by a black dot must be in a 1:2 ratio, i.e. one cell is double the value of the other cell.

whiteDot Node pairs marked with a white dot must be consecutive.

quad The numbers occurring in a white circle at the intersection of four cells must occur in those four cells in any order.

arrow The values on the shaft of the gray arrows must sum to the value given in the circle at its base.

renban The values on the purple Renban line must be a set of distinct, consecutive integers, they can occur in any order on the line.

germanWhisper Each pair of consecutive cells on the green German Whisper line must differ by at least 5.

cage The values in a cage, each outlined by a dashed line, must be different from each other, and sum to the value given in the top-left corner.

The buttons at the bottom of the user interface allow the user to find a solution (Solve), propagate all constraints without search (Propagate), and to single step through the propagation (Reset, Back, Forward). More complex interactions are possible through the tabs on the right hand side.

3 Overall Architecture

The application was developed as a desktop application written in Java and JavaFX, using the Choco-solver [17] constraint engine. Figure 2 shows the overall system architecture. Puzzles are imported into the system through a JSON description format, and are stored inside the object model representing the grid, cells and their assigned values, and constraints. The user interacts with the application through a graphical user interface (GUI), which at each point shows the state of the solution process. The user can interact with the model by creating/deleting/modifying additional constraints, and by directing the solver to perform an operation.

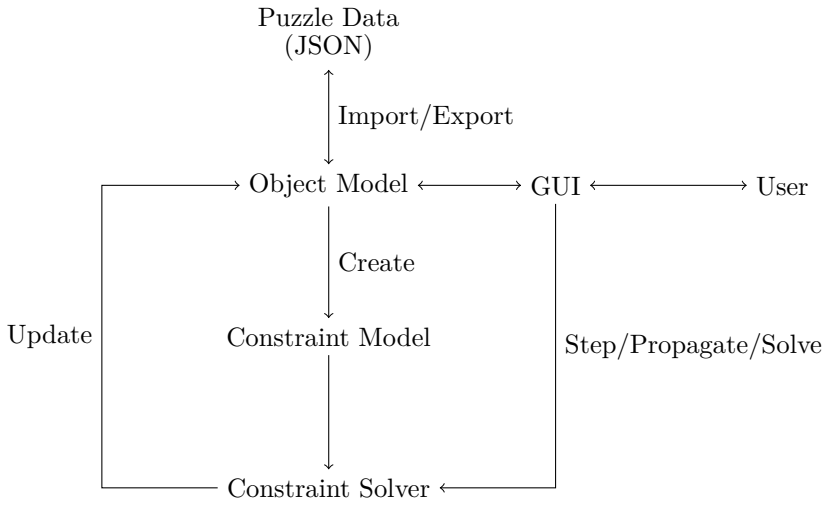

Figure 2 Application Architecture

The object model contains a full description of the puzzle to be solved, from it we create the constraint model, which is then passed to the back-end solver to solve or propagate. Information about the current state of the solution process, basically variable assignments and domain restrictions, are used to update the object model, and from there to update the user interface. This is different from most other constraint applications, where only complete solutions are extracted from a solver.

Note that this design requires that the back-end solver allows to deal with partial assignments, propagation only reasoning, and extracting domain information in a partially solved problem. Many constraint solvers do not allow this granularity of interaction, we selected Choco-solver because these APIs are available.

4 Related Work

Sudoku has been a very popular example for Constraint Programming for a long time. Most constraint solvers include a Sudoku model as one of the basic examples, but they nearly always rely on search to find solutions for more complex instances. The ModRef 2005 paper [20] probably was the first to evaluate standard Sudoku instances in more detail, and to study the interaction of multiple `alldifferent` constraints to improve propagation. Other solver techniques have been proposed to solve the puzzle, for example SAT in [12].

Using visualisation to explain the solving process has been discussed in a number of papers [18], a specialised tool to understand the reasoning and impact of different consistency techniques was presented in [11]. We largely follow concepts from CP-Viz [22] to visualise changes in the grid assignment. An interactive tutoring system for standard Sudoku was presented in [7], it aims to present hints to a user when the system detects that the user might be stuck. This is a much more sophisticated approach compared to our technique of displaying hints only on demand. The question of which consistency techniques should be used in interactive model solving is addressed in [5]. A more recent work [8] considers Sudoku as an example for an auditable CP Solver, where each reasoning step is recorded to prove correctness and completeness of the solution process.

An interesting approach to combine machine learning and CP when solving hand-drawn Sudoku puzzles is discussed in [1, 9]. The constraint model and image recognition model work together to improve the accuracy of the image analysis. This idea is extended in [14].

Nearly all previous work has focused on the standard Sudoku problem, and variant constraints have been rarely discussed. An exception is [15], which uses Killer Sudoku as an example. 16x16 random instances are used to find some of the interactions between the `cage` and `alldifferent` constraints.

Sudoku puzzles have also been long used as examples for Constraint Acquisition, see for example [23]. The aim is to build a model of the problem by either asking questions to a user, or by providing one or more example solutions. The ModelSeeker [4] uses a number of Sudoku variants as examples. An alternative way to learn Sudoku problems is described in [6], which uses Cost Function Networks to learn from a large set of Sudoku instances.

A much more general, but very relevant problem is considered in [10]. How do humans approach modelling a problem, and how does that relate to constraint models? Sudoku is one of the problems studied in a set of user experiments.

5 Modelling

In this section we describe how to classify variant constraints, and how to implement some of them in a constraint system.

5.1 A Sudoku Variant Morphology

A huge number of Sudoku variant constraints have been proposed over the last 30 years. Some have become popular, while other may occur only in a single instance. As far as we know, no systematic classification of these variants has been proposed so far. Figure 3 makes an attempt at a structural morphology of variant constraints, the number in parenthesis give the number of such constraints in the Sudoku Variant Catalog. The structure of the constraint is largely determined by the number and relation of cells involved. Some basic *node* constraints only involve a single node, like `preset` or `odd`. An extension considers a node and its neighbours, either orthogonally of diagonally connected. There are also constraints which consider a node and a ray (or multiple rays) of cells originating from the node. *Pair* constraints involve two, typically orthogonally connected, nodes. *Triple* constraints involving three cells are rare, but some do exist. *Quad* constraints, typically involving four cells in a 2x2 block, are much more common. These constraints can either deal with the four cells as an unordered collection, or respect an ordering imposed by the layout.

The following constraint types all work on collections of cells. *Lines* are formed by linking cells that are orthogonally or diagonally connected. The constraint on the line may be about the unordered collection of cells (like the `renban` constraint), it may be partially ordered (in the `arrow` constraint the first element is different from the others), or it may be on an ordered collection. In the `palindrome` constraint for example, cells at the same distance from the centre must have the same value. More specialised variants of ordered lines may be decomposed into constraints on pairs of directly connected cells, like the `germanWhisper` constraint, or may be about (possibly overlapping) subsequences. These may be regular, for example involving every subsequence of three consecutive cells, or structural, where the subsequences are defined by the layout of the puzzle. An example is the `regionSumLine` constraint, where the line is cut into subsequences whenever the line crosses a block border. Finally, the subsequence may be defined by the values on the line itself, the subsequences are therefore only known when the cells are assigned to a value. The `tenLine` constraint (see below) is an example. A *multi-line* constraint links multiple lines together. A further generalisation considers arbitrary sets of cells which define an *area*. A typical example is the `cage` constraint. Another type of constraints are *outside* constraints, where hints are given outside the grid itself. This allows to express constraints about specific rows, columns and/or diagonals in the grid, by placing hints on the top, the left or right, or the bottom of the grid.

The final constraint type are *global* conditions. The first sub-type imposes a structural background constraint on the grid, very often by stating a disequality or distance constraint on pairs of cells arranged in a regular pattern. The `antiKnight` constraint is such a pattern, it states that all pairs of cells that are a Knight's move apart must have different values. The second involves the negation of one or multiple basic constraint types. It states that if a constraint (say on a pair of connected nodes) is not explicitly stated, its negation must hold. The last type aggregates variables of all constraints of a certain type. An example would be a constraint that states the all `renban` constraints in the puzzle must consist of different value sets.

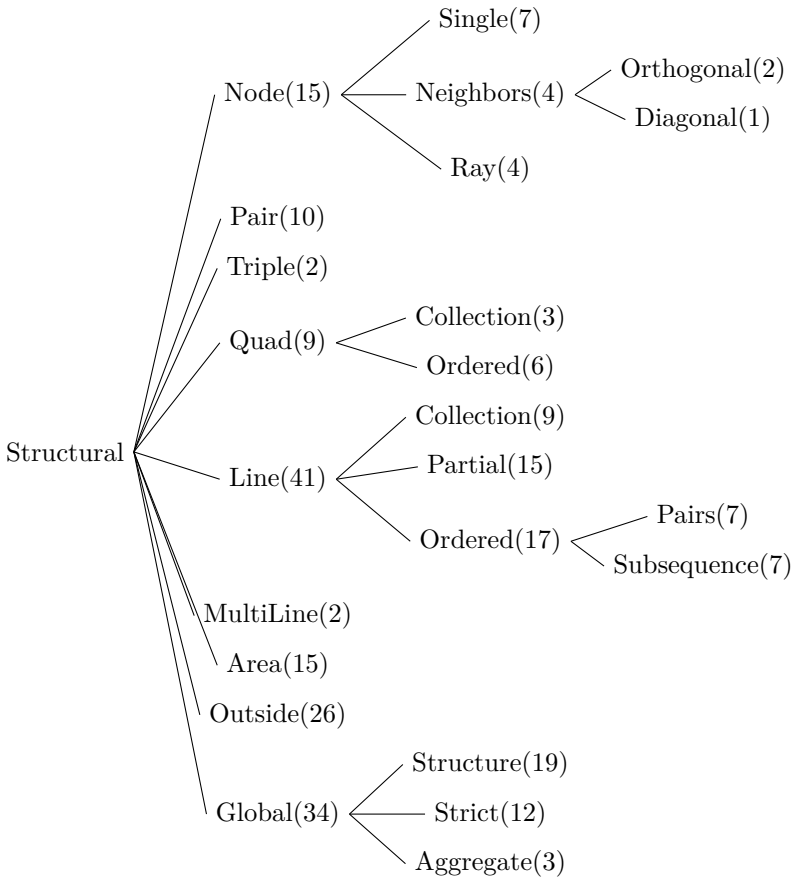

Figure 3 Variant Morphology

5.2 Implementing Specific Constraints

A typical constraint model consists of three elements, the variables, the constraints, and an objective. In Sudoku variant puzzles the variables are typically defined as a grid, with all variables ranging over the same domain. There is no objective, we are interested in finding the unique solution of the puzzle. The constraints can be implemented in many different ways, we now present some representative examples.

5.2.1 Alldifferent

The `alldifferent` constraint [19] is the cornerstone of the model. We not only use it to model the standard constraints for rows, columns and 3x3 blocks, it is also used in many variations, as additional constraints on diagonals, irregular sets of cells, or additional blocks in the layout. In some cases, other constraints are equivalent to an `alldifferent` constraint when expressed over a set of nine variables, for example a nine-cell `renban` line must consist of nine, alldifferent values.

5.2.2 Pair Constraints

Many of the constraints on orthogonally connected node pairs (say, X and Y) can be expressed in symbolic form, for example `sumV` as $X + Y = 5$, `sumX` as $X + Y = 10$, `blackDot`

as $X = 2 * Y \vee Y = 2 * X$, and `whiteDot` as $|X - Y| = 1$. But the constraint reasoning for these constraints can be weak, so that we instead enumerate all possible tuples and then use a `table` constraint to achieve domain consistency. For some constraints, we can use the disequality constraint imposed by the row or column containing the node pair to further reduce the number of value pairs that need to be considered.

5.2.3 Renban

The `renban` constraint enforces that the values on a line form a set of distinct consecutive integers. There is a specific global constraint for this, called `alldifferent_consecutive_values` [3], but this constraint is not available in Choco-solver. A naive decomposition of the constraint is

$$\text{renban}(X) \equiv \min(X, \text{Min}) \wedge \max(X, \text{Max}) \wedge \text{Max} - \text{Min} = |X| - 1 \tag{1}$$

This has limited constraint propagation, instead we use this formulation to create all possible renban tuples of size 2 to 8, and then impose a `table` constraint for each line occurring in a puzzle instance.

5.2.4 Arrow

The `arrow` constraint enforces a `sum` constraint on a partially ordered list, the first element is equal to the sum of the remaining elements

$$\text{arrow}([X_1, X_2, ...X_n]) \equiv X_1 = X_2 + X_3 + ... + X_n \tag{2}$$

The bounds reasoning on the `sum` constraint will not remove all infeasible values, this also ignores the fact that some, or even all variables must be different from each other. There have been proposals to adapt the reasoning on `sum` constraints in the presence of `alldifferent` constraints [2], we found it easier to generate `table` constraints for all sums up to nine variables. This was already proposed in [21] for another puzzle type, Kakuro, where the only constraint is a combination of `sum` and `alldifferent`.

The same type of reasoning can be used for the `cage` constraint, which states that the sum of all cells in an area must be equal to the given hint.

5.2.5 GermanWhisper

The `germanWhisper` constraint (also called Whisper-5) states that any two consecutive cells on a line differ by at least five. This can be expressed by a constraint on pairs of variables

$$\text{germanWhisper}([X_1, X_2, ..., X_n]) \equiv \forall_{i \in 1..n-1} \quad |X_{i+1} - X_i| \geq 5 \tag{3}$$

We again use a `table` constraint on pairs to achieve domain consistent reasoning. More propagation is possible if we know that certain values on the line must be different, or must be equal.

5.2.6 TenLine

The `tenLine` constraint states that a line can be partitioned into non-overlapping subsequences which must each add up to ten. This can be expressed very concisely with a regular constraint [16] which implements a finite automaton where each node represents a partial sum, and edges describe increasing/resetting a sum by some increment.

5.2.7 GreatestLine

Another interesting constraint is the `greatestLine` constraint, which states that the largest value on the line is greater than the sum of any other two cells on the line. This can be expressed with the `sort` constraint [13] and an inequality:

$$\text{greatestLine}([X_1, X_2, ..., X_n]) \equiv sort([X_1, X_2, ..., X_n], [Y_1, Y_2, ..., Y_n]) \wedge Y_n \geq Y_{n-1} + Y_{n-2} \quad (4)$$

The second argument of the `sort` constraint is sorted in increasing order, therefore Y_n is the largest element. By stating that Y_n is greater than the sum of the second and third largest elements $Y_{n-1} + Y_{n-2}$ we impose the original constraint.

5.2.8 AntiKnight

An example of a structural global constraint is the `antiKnight` constraint, which states that any two cells that are a Knight's move apart cannot take the same value. This can be modelled with a large conjunction of disequality constraints

$$\forall_{\text{KM}(Y,Y)} : \quad X \neq Y \quad (5)$$

5.2.9 StrictKropki

The final example constraint is the `strictKropki` constraint, which states that any orthogonally connected pair of cells which does not have a `whiteDot` or `blackDot` constraint cannot be consecutive or in a 1:2 ratio. This results in the constraint

$$\forall_{\text{OC}(X,Y) s.t. \neg\text{whiteDot}(X,Y) \wedge \neg\text{blackDot}(X,Y)} : \quad X \neq 2{*}Y \wedge Y \neq 2{*}X \wedge X \neq Y{+}1 \wedge Y \neq X{+}1 \quad (6)$$

5.3 Table Sizes

The models in the previous section rely heavily on table constraints. How many tuples are required to express the conditions? Table 1 shows some examples. For the combination of `alldifferent` and `sum` over a domain from 1 to 9 we need 72 tuples of size 2, but 362k tuples for size 8. Recent progress in the handling of the `table` constraints mean that even these large tables can be handled very efficiently. Note that for nine `alldifferent` variables we know that the sum must be equal to 45, we do not need a table for that case. The number of tuples increase more rapidly for a single `sum` constraint without the `alldifferent`, we only use tables for up to five variables, and rely on the standard bounds-consistent `sum` constraint for larger sums. For the `renban` constraint, the largest table size is 80k tuples for size 8, for size 9 an `alldifferent` constraint is used instead. The largest table is needed for the `between` constraint $(\min(X_1, X_n) < X_{2 \leq i \leq n-1} < \max(X_1, X_n))$, which requires over 3.3 million tuples for size 9. On the other hand, some quite complex constraints can be expressed very concisely with a table. A `magicSquare` constraint on a 3x3 block only has 41 possible assignments, including all possible symmetries, which allows for much stronger propagation of the table than the defining set of equations on rows, columns and diagonals.

5.4 Manual Constraints

The typical manual solution process consists of performing some mental reasoning that allows us to fix some cell to one value, or to annotate a cell with some restriction. In our application,

Table 1 Table Sizes

	Number of Variables							
Table	2	3	4	5	6	7	8	9
Sum+AllDifferent	72	504	3,024	15,120	60,480	181,440	362,880	*
Sum	81	729	6,561	59,049	-	-	-	-
Renban	16	42	144	600	2,880	15,120	80,640	*
Between	-	168	672	3,192	16,800	94,488	556,512	3,390,072
Magic Square	-	-	-	-	-	-	-	41

we can set and remove values from cells with the `setValue` or `removeValue` primitives. We always see the most up-to-date information about values left in the domain, so there is no need to further annotate the grid manually. But we may find it useful to state some more complex, structural constraints. At the moment we allow:

in$(< X_1, X_2, ..., X_n >, < V_1, V_2, ..., V_k >)$ We restrict the domain of all X_i variables to the set V.

mustContain$(< X_1, X_2, ..., X_n >, < V_1, V_2, ..., V_k >)$ For each value V_j, one of the X_i variables must take that value. This leads to a disjunction, but more reasoning can be performed (see below).

mustNotContain$(< X_1, X_2, ..., X_n >, < V_1, V_2, ..., V_k >)$ None of the values V_j are possible for any of the X_i variables. We can remove these values from their domains.

sameValues$(< X_1, X_2, ..., X_n >, < Y_1, Y_2, ..., Y_n >)$ The values in vectors X and Y must be the same. This often arises when considering the interaction of two `alldifferent` constraints.

samePairs$(< X_1, X_2, ..., X_n >, < Y_1, Y_2, ..., Y_n >, C)$ The values must match pairwise, i.e. $X_i = Y_i + c$. In most scenarios offset c is zero.

allEqual$(< X_1, X_2, ..., X_n >)$ All variables X_i must take the same values. This is a constraint often needed when a lot of initial reasoning is performed to find equivalence classes of cells in the grid.

alldifferent$(< X_1, X_2, ..., X_n >)$ This allows to add additional, redundant `alldifferent` constraints. This can dramatically improve the propagation performed.

Figure 4 shows the solution state of our running example after the initial propagation. We can see that some cells in the top row have been assigned (the cells contain a single, large number), and many domains have been reduced (cells coloured in green), the small numbers in the cells indicate which values are still allowed. At this point we can manually enter some additional constraints to add more information. In this example we can deduce that the arrow starting in cell C49 must contain a value 3, since otherwise the arrow would be at least 1+4+5=10. The value 3 must be in either cell C47 or C48, as cell C37 does no longer allow that value. We can select these cells in the grid (outlined in green), and use the dialog box on the right to add the `mustContain` constraint. This will enter a disjunction $C47 = 3 \vee C48 = 3$, but more importantly, our system can consider the interaction of this constraint with the other, existing constraints, say the row 6 and block 4 `alldifferent` constraints. If C47 or C48 must contain a 3, then none of the other cells in their row or block can contain that value. All of these values can be removed from the respective domains in one step, showing the advantage of this more abstract constraint over multiple `removeValue` constraints.

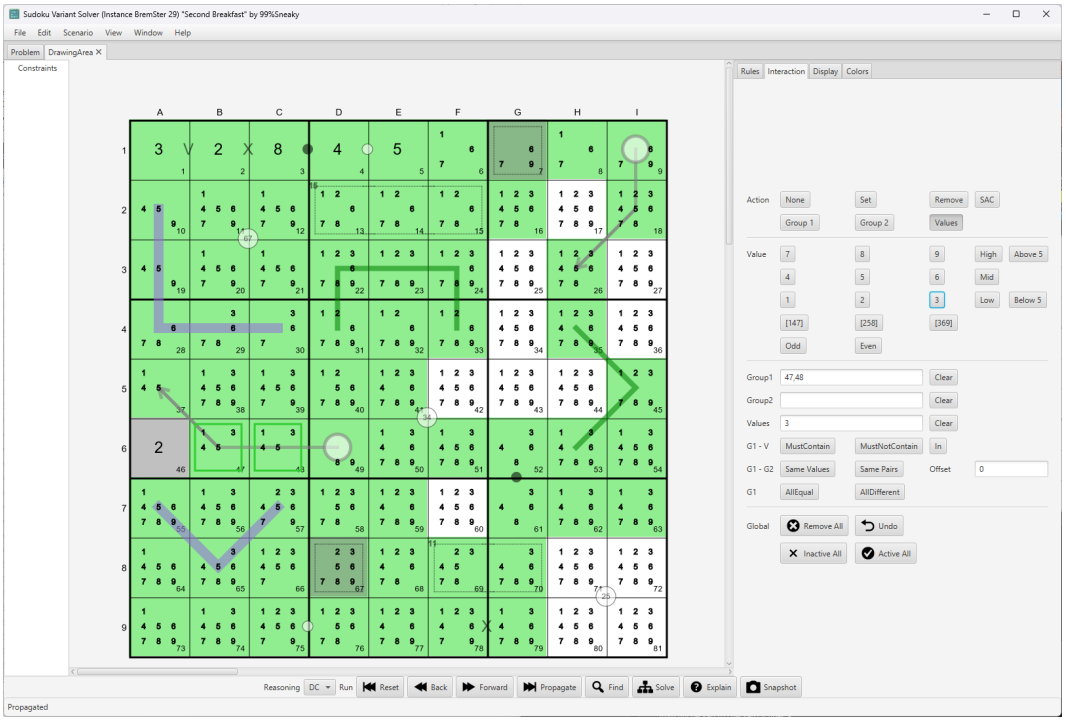

Figure 4 Running Example: Entering "Must Contain 3" Constraint for Cells C47 and C48, after initial Propagation of Constraint Set

Figure 5 shows the solved grid of our running example. Each cell is now filled with a value, all constraints are satisfied, and there is a single solution. Two manual constraints were needed, the `mustContain` for value 3 for cells C47 and C48 (outlined in green), and a `removeValue` constraint which removes value 9 from the domain of cell C33 (the red circle marks a `removeValue` constraint). Cells that were assigned in the last propagation step are shown in green, the reader may find it beneficial to check all constraints stated against the solution.

5.5 Dynamic Constraints

It has been long known that reasoning on single `alldifferent` constraints is not able to find solutions of all Sudoku instances by propagation alone [20]. We can either extend the reasoning by considering the interaction of multiple `alldifferent` constraints, or apply stronger consistency techniques like *singleton arc consistency* or *constructive disjunction* to find more infeasible values. A third method is used in most existing, non constraint-based Sudoku solvers. They define rules that specify configurations in a puzzle where we can remove some value. In the most advanced solvers, like sudokuwiki, over 40 such rules are defined for standard Sudoku problems alone. Many of these rules are subsumed by more global reasoning in a constraint solver, but we found it useful to allow such methods in our system as well. This can be easily implemented as a post-processing step at each propagation step, when we can check the current assignment to see if some rules apply. If we find such a rule instantiation, we add a dynamic constraint that removes the detected infeasible values, and which records the reasoning that led to this removal. At the moment, we have prototypical implementations of `pointingTuple`, `xWing` and `yWing` constraints, see the excellent description in sudokuwiki

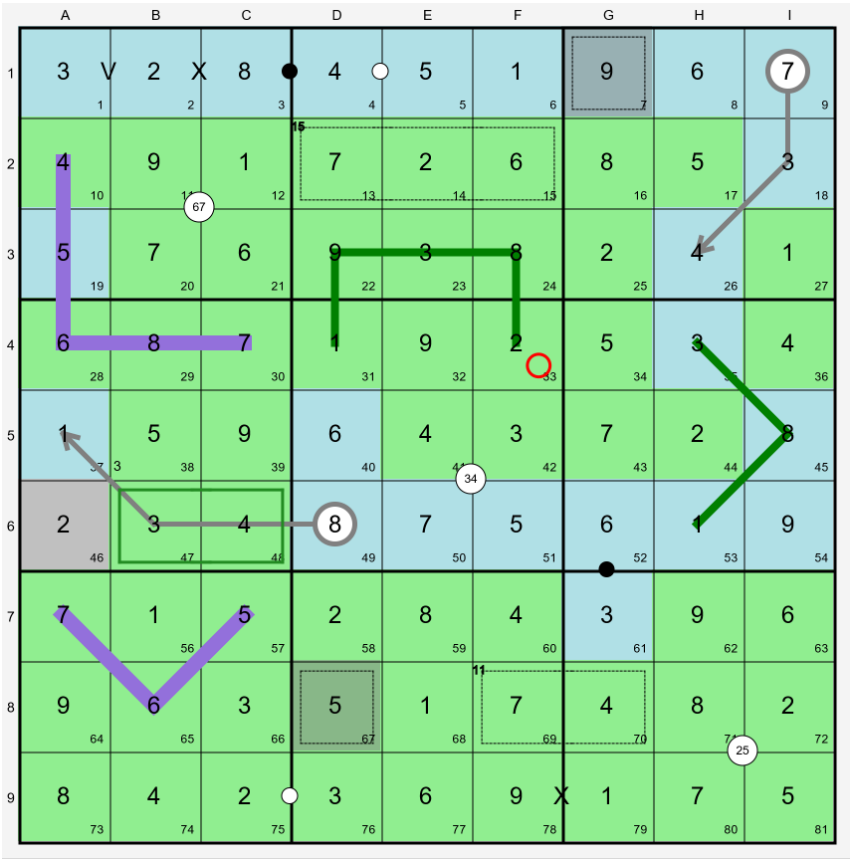

Figure 5 Running Example: Problem Solved after Adding Two Constraints Manually ("Must Contain 3" for cell C47, C48, and "Remove Value 9" for Cell C33)

for detailed descriptions of these rules.

6 Evaluation

We test our application on four different benchmark sets, each intended to answer one specific question.

6.1 Nikoli Puzzles

To test our system with standard Sudoku puzzles, we use some classical collections of problems from the Japanese puzzle publisher Nikoli [24, 25, 26]. Each book consists of around 100 puzzles, arranged by increasing difficulty. The puzzles are graded (easy-hard, 1-10), but none of the puzzles would today be considered as a hard problem instance.

We see that while for the simpler Nikoli1 set we find all solutions by propagation alone, we need the more advanced dynamic constraints for some of the more complex instances in sets gekikara1 and gekikara2. This shows that our interactive design adding dynamic constraints after propagation works for standard Sudoku instances.

Table 2 Results for Nikoli Standard Sudoku Puzzles

		Propagation Only		Propagation + Dynamic	
Book	Instances	Nr Solved	Percent	Nr Solved	Percent
nikoli1	99	99	100.00	99	100.00
gekikara1	105	90	85.71	105	100.00
gekikars2	105	85	80.95	105	100.00

6.2 Genuinely Approachable Sudoku (GAS) Puzzles

The Genuinely Approachable Sudoku project (also `http://https://www.gmpuzzles.com/blog/tag/gas/`) presents a daily puzzle on their website, and offer solution videos on their YouTube channel. Other puzzle solvers also provide solution videos on their own channels. The puzzles are intended as an introduction to variant constraints, typically focussing on a single constraint type in addition to the standard Sudoku structure. We found these puzzles very valuable not only to understand the intended meaning of a constraint, but also to check that the propagation provided by our implementation is powerful enough to solve typical instances. We tested our system on 77 instances, including all puzzles from January and February 2026. As new constraints are frequently introduced in this series, it serves as a good test case to see whether our constraint morphology from Section 5.1 is comprehensive enough, and new constraints can be easily added to our application. Figure 3 summarises the results. Of the 77 instances included, 69 (or 89%) are solved by propagation, all other instances are solved by adding no more than four manual constraints.

Table 3 Results for GAS Sudoku Variant Puzzles from 2025-2026, including all of January/February 2026

		Solved by Propagation		Adding Manual Constraints	
Book	Instances	Nr Solved	Percent	Nr Solved	Percent
GAS	77	69	89.61	77	100.0

6.3 2025 World Puzzle Federation Sudoku Grand Prix

The World Puzzle Federation organises a yearly Sudoku Grand Prix since 2014. This is an on-line event in multiple (currently 8) stages, where participants in each stage work on a set of around 15 Sudoku variant puzzles in 90 minute sessions. No electronic tools are allowed, this is intended strictly as a pen-and-paper event. We have extracted the puzzles for the 2025 competition in our data format, all but three can be modelled and solved. The three missing cases either use an irregular grid structure, or ask the user to identify irregular boxes in the grid.

In the competition, each solved puzzle gains a number of points which varies with the perceived difficulty. In addition, participants can earn time points for finishing the complete set of all puzzles in a round before the time limit, 1/6th of a point is awarded for every second early. Table 4 shows the results for our tool, which achieves 5,400 out of 5,610 possible regular points, and earns 4,495 time points from the rounds where it solved all instances. This gives a total of 9,895 points for our tool. The best human participant achieved 6,160.2 points (including time points), our result without time points would place the tool fifth

compared to all human participants, considering their combined regular and time points. The organizers do not report regular points only for participants.

Table 4 Results for Rounds of World Puzzle Federation Sudoku Grand Prix 2025

Round	Puzzles	Points Available	Solved	Points Achieved	Time Points
R1	13	660	13	660	899
R2	13	720	13	720	899
R3	18	600	17	555	0
R4	12	780	11	674	0
R5	15	750	15	750	899
R6	13	750	13	750	899
R7	14	660	14	660	899
R8	14	690	13	631	0
Total	112	5,610	109	5,400	4,495

Examples of the types of constraints used in the competition are available for training purposes before the competition, making it possible to add any missing constraints to the tool before the competition if required. Also note that in this speed-puzzling competition search is readily accepted, you only have to show the result, not the solution path used.

6.4 Cracking the Cryptic (CTC)

The YouTube channel "Cracking the Cryptic" is probably the most popular Sudoku solving channel world-wide. The presenters Simon Anthony and Mark Goodliffe typically add solution videos for two puzzles per day. The emphasis is on the interest and novelty of the puzzles, and the elegance of the solution process. The solution videos often exceed one hour duration, the presentation aims to explain the reasoning at each step in sufficient detail for the audience to follow the process. Most of the puzzles presented were created by other designers, very often the puzzles are originally posted on the website Logic Masters Deutschland.

Figure 6 shows the puzzle for the most popular videos on the CTC site; It has achieved, at time of writing, over 10 million views on YouTube (Magic Square Sudoku). It is quite typical for the puzzles shown on CTC, very few (if any) hints are given, and the unique solution is imposed by a combination of multiple variant constraints. In this case the constraints are a standard Sudoku grid, additional `alldifferent` constraints for the two main diagonals (indicated by the blue lines), an `antiKnight` global constraint, and a 3x3 `magicSquare` constraint for the centre box. The `antiKnight` constraint states that two cells which are a Knight's move apart cannot take the same value. The magic square requires that in a 3x3 box all values are different, and the sum of rows, columns, and diagonals add to the same number.

The picture in Figure 6 shows the state of the puzzle after the initial propagation. Cells in blue have been deduced, while other cells show which values are still in the domain. The four cells marked in gray are the initial givens, which are needed both to force the magic square in block 5 into a unique orientation, and to remove enough values from other cells to make the resulting solution unique. To complete the solution, the user has to enter some additional constraints. Hints marked with an exclamation point show where additional reasoning is possible, these hints at the moment are defined manually for each instance. Selecting a hint shows an explanation and highlights the cells involved. Here the hint states that the value

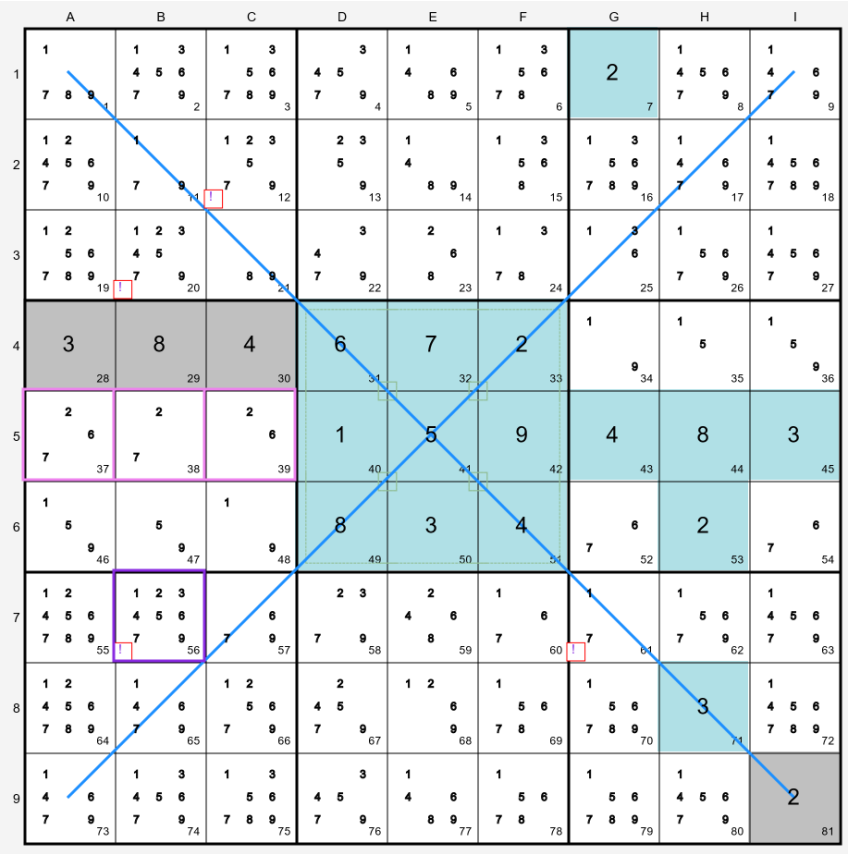

Figure 6 Magic Square Sudoku after Initial Propagation Showing Hints

2 in cell C56 can be removed. If C56 is 2, then it attacks cells C37, C38, and C39, which are the only possible cells for value 2 in box 4. In a similar way, some other values can be removed from cells C61, C20, and C12. Imposing these constrains, either manually, or by accepting the hints given, will solve the remaining assignments of the puzzle.

Table 5 shows results for other, selected puzzles of the CTC channel. A link to the YouTube solution video is provided as a hyperlink. The puzzles at the top of the list each have more than a million views. We show the number of givens for each puzzle, as well as the number of cells filled in by propagation (Column Prop.). In some cases, no cells at all are filled after the initial propagation. We also show the number of manual constraints that were used to solve the puzzle without choices or search. These manual constraints were obtained by solving the puzzle by hand, and recording the solution process as a set of hints. Other constraint sets may be even shorter.

The results seem to indicate that the combination of automated reasoning and manually added constraints is able to solve complex puzzles, while relieving the user from the tedium of much of the basic constraint reasoning.

7 Summary and Future Work

In this application paper we have described an interactive tool to solve Sudoku variant puzzles. It is based on Constraint Programming, and at the moment handles just over 200 variant

Table 5 Manual Constraints Used for Selected CTC Instances

Nr	Instance	Cells Filled		Manual Constraints Needed								
		Givens	Prop.	=	≠	In	MC	MNC	SV	SP	AE	AD
52	Magic Square Sudoku	4	15	-	4	-	-	-	-	-	-	-
94	Non Consecutive Killer	1	0	4	1	-	-	-	2	-	3	-
30	The Miracle	2	12	-	15	-	-	-	-	-	-	-
33	Antiknight Sudoku	13	5	-	3	-	1	-	-	-	-	-
51	10 Million Views	4	13	-	-	-	-	3	-	-	-	-
56	Centipede	1	0	3	-	-	3	-	-	-	-	-
60	Spring Has Sprung	0	11	1	1	-	-	-	-	-	-	-
62	Game Of Arrows	3	2	2	1	-	1	-	-	-	-	-
68	The Climb	0	0	2	-	-	4	1	-	-	-	-
69	Simply Beautiful Sudoku	29	0	-	-	-	-	1	-	-	-	-
92	<= 5 Sudoku	11	7	-	2	-	-	-	-	-	-	-
93	Superking	6	3	1	-	-	-	-	-	-	-	-

constraint types. An interactive design allows the user to work together with the constraint solver, stating additional constraints when the propagation alone is not sufficient to determine the solution. Alternatively, search can be used to find solutions quickly. Rule based agents can be used to capture some of the deductions performed by conventional Sudoku solvers after each propagation step. We have evaluated the system against four main benchmark types. The first shows that the interaction of propagation and dynamic constraints can be used to find solutions for some well-known standard Sudoku instances. The second set shows that the constraints implemented are enough to solve most "approachable" instances for many different variant constraint types by propagation alone. The third benchmark set is from the 2025 Sudoku Grand Prix, our results would place our system in the top five overall, even if we ignore the extra time points that our solver could gain by solving the puzzles much more rapidly than a human. The final benchmark set, from the CTC YouTube channel, shows that very advanced, popular Sudoku variants can be solved by propagation alone, when adding often only a handful of manual constraints.

There are still many puzzles that our solver currently does not solve. This can be due to a non-rectangular grid layout, for example triangular, hexagonal, or 3D layout pattern. These would be easy to add on the solver side, but would require additional effort in describing and visualising the puzzles. Other restrictions are more fundamental: A number of variant constraint are based on graph concepts, requiring to identify paths, or connected components, or identifying the irregular block structure in addition to find a valid grid assignment. This might require implementing specific graph global constraints, or using a constraint solver that provides more concepts of graph variables and constraints.

There are also many less common constraints still to be added. It would be helpful if we could automate the process of adding and testing a new variant constraint as much as possible. There currently seems to be no text-based interchange format for generic Sudoku variant puzzles, typically instances are presented as images or pdf documents only. An extension that could use computer vision to automatically extract the JSON, logical description of a puzzle from an image would be very helpful.

Even with these limitations, the current tool is an interesting first step to solving generic Sudoku variants with Constraint Programming. While this might be an interesting commercial application in itself, it also seems clear that this application domain can be used for outreach at different levels, and perhaps interest the millions of puzzle enthusiasts in Constraint Programming as well.

References

1 Yiwei Bai, Di Chen, and Carla P. Gomes. Clr-drnets: Curriculum learning with restarts to solve visual combinatorial games. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.CP.2021.17, doi:10.4230/LIPICS.CP.2021.17.

2 Milan Bankovic. Solving finite-domain linear constraints in presence of the `alldifferent`. *Log. Methods Comput. Sci.*, 12(3), 2016. doi:10.2168/LMCS-12(3:5)2016.

3 Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global Constraint Catalog, 2005. Research Report SICS T2005-08. URL: https://hal.science/hal-00485396.

4 Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2012. doi:10.1007/978-3-642-33558-7_13.

5 Christian Bessiere, Hélène Fargier, and Christophe Lecoutre. Global inverse consistency for interactive constraint satisfaction. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2013. doi:10.1007/978-3-642-40627-0_15.

6 Céline Brouard, Simon de Givry, and Thomas Schiex. Pushing data into CP models using graphical model learning and solving. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 811–827. Springer, 2020. doi:10.1007/978-3-030-58475-7_47.

7 Allan Caine and Robin Cohen. MITS: A mixed-initiative intelligent tutoring system for sudoku. In Luc Lamontagne and Mario Marchand, editors, *Advances in Artificial Intelligence, 19th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2006, Québec City, Québec, Canada, June 7-9, 2006, Proceedings*, volume 4013 of *Lecture Notes in Computer Science*, pages 550–561. Springer, 2006. doi:10.1007/11766247_47.

8 Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, Haifa, Israel, July 31 - August 8, 2022*, volume 235 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.CP.2022.25, doi:10.4230/LIPICS.CP.2022.25.

9 Tias Guns, Emilio Gamba, Maxime Mulamba, Ignace Bleukx, Senne Berden, and Milan Pesa. Sudoku assistant - an ai-powered app to help solve pen-and-paper sudokus. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 16440–16442. AAAI Press, 2023. URL: https://doi.org/10.1609/aaai.v37i13.27072, doi:10.1609/AAAI.V37I13.27072.

10 Ruth Hoffmann, Xu Zhu, Özgür Akgün, and Miguel A. Nacenta. Understanding how people approach constraint modelling and solving. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, Haifa, Israel, July 31 - August 8, 2022*, volume 235 of *LIPIcs*, pages 28:1–28:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.CP.2022.28, doi:10.4230/LIPICS.CP.2022.28.

11 Ian Howell, Robert J. Woodward, Berthe Y. Choueiry, and Christian Bessiere. Solving sudoku with consistency: A visual and interactive approach. In Jérôme Lang, editor, *Proceedings*

of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pages 5829–5831. ijcai.org, 2018. URL: https://doi.org/10.24963/ijcai.2018/852, doi:10.24963/IJCAI.2018/852.

12 Inês Lynce and Joël Ouaknine. Sudoku as a SAT problem. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006. URL: http://anytime.cs.umass.edu/aimath06/proceedings/P34.pdf.

13 Kurt Mehlhorn and Sven Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In Rina Dechter, editor, *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 of *Lecture Notes in Computer Science*, pages 306–319. Springer, 2000. doi:10.1007/3-540-45349-0_23.

14 Maxime Mulamba, Jayanta Mandi, Ali Irfan Mahmutogullari, and Tias Guns. Perception-based constraint solving for sudoku images. *Constraints An Int. J.*, 29(1-2):112–151, 2024. URL: https://doi.org/10.1007/s10601-024-09372-9, doi:10.1007/S10601-024-09372-9.

15 Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, and Ian Miguel. Automatically improving constraint models in savile row through associative-commutative common subexpression elimination. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 590–605. Springer, 2014. doi:10.1007/978-3-319-10428-7_43.

16 Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004. doi:10.1007/978-3-540-30201-8_36.

17 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. doi:10.21105/joss.04708.

18 Christopher G. Reeson, Kai-Chen Huang, Kenneth M. Bayer, and Berthe Y. Choueiry. An interactive constraint-based approach to sudoku. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1976–1977. AAAI Press, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-362.php.

19 Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pages 362–367. AAAI Press / The MIT Press, 1994. URL: http://www.aaai.org/Library/AAAI/1994/aaai94-055.php.

20 Helmut Simonis. Sudoku as a constraint problem. In *Modelling and Reformulating Constraint Satisfaction Problems (ModRef Workshop 2005)*, 2005. URL: https://www.researchgate.net/publication/228803795_Sudoku_as_a_constraint_problem.

21 Helmut Simonis. Kakuro as a constraint problem, 2008. URL: https://www.researchgate.net/publication/228524341_Kakuro_as_a_Constraint_Problem.

22 Helmut Simonis, Paul Davern, Jacob Feldman, Deepak Mehta, Luis Quesada, and Mats Carlsson. A generic visualization platform for CP. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 460–474. Springer, 2010. doi:10.1007/978-3-642-15396-9_37.

23 Dimosthenis C. Tsouros, Kostas Stergiou, and Christian Bessiere. Structure-driven multiple constraint acquisition. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 of *Lecture Notes in Computer Science*, pages 709–725. Springer, 2019. doi:10.1007/978-3-030-30048-7_41.

24 various. *Sudoku 1*. Nikoli, 1988. In Japanese.

25 various. *Gekikara Sudoku 1*. Nikoli, 2004. In Japanese.

26 various. *Gekikara Sudoku 2*. Nikoli, 2004. In Japanese.