Modeling Str8ts Puzzles as Propositional Formulas

Lina Mehrle

Bachelor's Thesis
under supervision of Claudia Grunke
examined by Prof. M. Helmert

Department of Mathematics and Computer Science
University of Basel
January 31, 2026

Acknowledgements

First and foremost, thank you to Malte Helmert for offering me the opportunity to write this thesis, and thank you to Claudia Grunke for being an amazing supervisor. Thank you to Flo, for once again proofreading my thesis. You are a great editor and an even greater friend. Thank you to Nico, for being my rubber duck and for joining me in my current Str8ts obsession. And last but not least, thank you to Jeff Widderich and Andrew Stuart for kindly providing me the test data and for inventing Str8ts in the first place.

Abstract

Propositional logic can be used to formalize and solve a multitude of problems, including many number puzzles. This thesis models the puzzle Str8ts as three different propositional formulas and compares them in terms of simplicity and performance.

The first encoding is generated with full knowledge of the layout of the puzzle. This entails all information on where the black and white cells are and which numbers are predefined. The second and third encodings are generated without this knowledge and model the general rules of the puzzle. Version 1 of the general encodings focuses on knowing which cells are in the same straight while version 2 uses the concept of border cells.

All three versions are tested using a set of 1000 Str8ts puzzles and 100 mini Str8ts puzzles and the SAT solver CaDiCal. They are compared in terms of the number of clauses and variables in the formula, the generation time of the encoding, and the time and memory needed by CaDiCal to find the solution. The first encoding, using the layout of the puzzle, shows the best performance in each metric. From the two general encodings, version 2 shows a superior performance compared to version 1.

Contents

Chapter 1

Introduction

Inspired by the bachelor's thesis of Sebastian Schlachter [5], where he encoded different Sudoku variants as SAT problems, this thesis aims to do the same for the puzzle Str8ts. Similarly to Sudoku, Str8ts is a number puzzle on a nine by nine grid, where the goal is to fill the empty cells with numbers between one and nine in a way that each number appears at most once per row and column. Unlike Sudoku, there are no smaller three by three grids but instead some cells are blacked out and do not need to be filled in. Each row and column is divided into straights by the black cells, and each straight needs to contain consecutive numbers, not necessarily in the correct order.

The goal of this thesis is to find multiple ways to formalize a given Str8ts puzzle as a formula in propositional logic and then compare the different encodings in terms of structural simplicity and computational resources (time and memory) required for generation and solving by a SAT solver. The encodings adopt ideas introduced in [4], where the rules of Str8ts and Sudoku overlap.

Chapter 2 gives a short introduction to propositional logic and then further explains the puzzle Str8ts.

In chapter 3, the three encodings are introduced. The first encoding centers on an already given puzzle and will use all given information on the positions of the white and black cells as well as the prefilled numbers in the puzzle grid. The other two encodings are independent of a specific puzzle and encode the general rules of Str8ts. To encode a given puzzle, it is only necessary to add clauses denoting the layout to the general formula. Version 1 of the general encoding uses the notion of two cells being in the same straight, while version 2 introduces the concept of border cells.

Chapter 4 analyses the three encodings theoretically in terms of used clauses and variables, which influence the expected performance.

And lastly, chapter 5 details the practical experiment in which a set of puzzles was encoded and then solved by a SAT solver. During the experiment, the number of clauses and variables in the encodings was tracked, as well as the time it took to generate the encodings and the time and memory needed by the SAT solver to solve the formula.

Chapter 2

Background

The goal of this chapter is to introduce the background information needed later. The first part briefly defines the core concepts of propositional logic, the second part introduces the rules of Str8ts. The theory on propositional logic presented mostly corresponds to [8].

2.1 Propositional logic

The goal of propositional logic is to formally model statements that can be either true or false, for example statements of the form "if A and B are true, then C must follow". The result is called a propositional formula.

Definition 2.1 The set of propositional formulas (or simply formulas) over a set of variables V can be defined recursively as follows.

- Every variable in V is a propositional formula.
- For a propositional formula φ, its negation $\neg\varphi$ is also a propositional formula.
- For two propositional formulas φ and ψ, their conjunction $\varphi \wedge \psi$ is also a propositional formula.

Example 2.2 Take for example the variables x_1 and x_2, then the expression

$$\neg x_1 \wedge x_2$$

is a formula.

Remark 2.3 The following two abbreviations will be used for simpler notation and understanding.

- For two formulas φ and ψ, define their disjunction

$$\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi).$$

- For two formulas φ and ψ, define the implication

$$\varphi \rightarrow \psi := \neg(\varphi \wedge \neg\psi).$$

A priori, a formula has no meaning. For this, an interpretation (or truth assignment) needs to be fixed first. It denotes whether each variable is true of false from which it follows if the whole formula is true or false.

Definition 2.4 Take two formulas φ and ψ over the set of Variables V. Then an interpretation of V is a map

$$\mathcal{I} : V \to \{true, false\}.$$

Further, a formula holds under an interpretation $\mathcal{I}$ according to the following recursive rules.

- A variable v holds under $\mathcal{I}$ if and only if $\mathcal{I}(v) = true$.
- The formula $\neg\varphi$ holds under $\mathcal{I}$ if and only if φ does not hold under $\mathcal{I}$.
- The formula $\varphi \wedge \psi$ holds under $\mathcal{I}$ if and only if both φ and ψ hold under $\mathcal{I}$.

The notation $\mathcal{I} \models \varphi$ will be used to denote that φ holds under $\mathcal{I}$ and $\mathcal{I}$ will be called a model of φ.

In this regard, the conjunction $\wedge$ can be seen as an "and" and the disjunction $\vee$ as a (not exclusive) "or".

Example 2.5 Regard the formula

$$(x_1 \wedge x_2) \to (x_1 \vee \neg x_3).$$

It can be interpreted as "if x_1 and x_2 are true, then x_1 is true or x_3 is false". Now consider the interpretation

$$\mathcal{I} = \{x_1 \mapsto true, x_2 \mapsto true, x_3 \mapsto true\}.$$

It is a model for this formula, however it is not the only one.

Definition 2.6 A formula is called satisfiable, if it has a model.

The goal of this thesis will be to find multiple propositional formulas which are satisfiable and whose solutions correspond to the solution of a given Str8ts puzzle. Due to the nature of a SAT solver, the program which will later be used to find the model of a given formula, it is necessary for the formulas to be in a fixed normal form, the conjunctive normal form.

Definition 2.7 A clause is the disjunction of variables or their negations.

Definition 2.8 A formula is in conjunctive normal form (CNF) if it is of the form

$$\bigwedge_{i=1}^{n} c_i$$

for clauses c_i.

It is always possible to rewrite a formula in CNF without changing its meaning.

Definition 2.9 Two propositional formulas φ and ψ over the same set of variables V are equivalent, if every model of φ is also a model of ψ and vice versa, that is if

$$\mathcal{I} \models \varphi \Leftrightarrow \mathcal{I} \models \psi$$

for every interpretation $\mathcal{I}$.

Theorem 2.10 Every formula has an equivalent formula in CNF.

A proof of this can be found in [2, Chapter 3.9]. While the transformation is always possible, it is important to note that the resulting formula can be significantly longer than the original.

2.2 Str8ts

Str8ts is a logic puzzle invented in 2008 by Jeff Widderich and Andrew Stuart [9]. It consists of a nine by nine grid of white and black cells where some cells are filled with a number between one and nine. An example can be seen in figure 2.1.

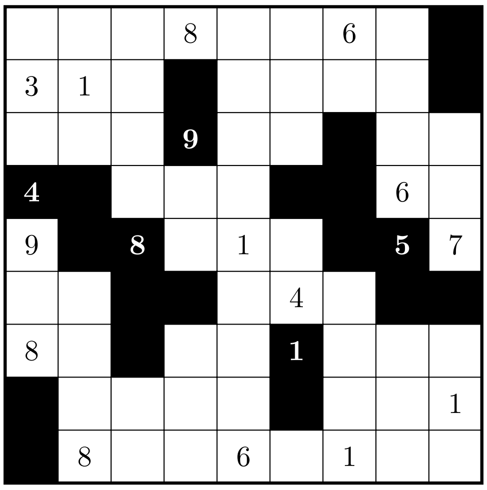


Figure 2.1: Example Str8ts

The goal of the puzzle is to fill all empty white cells with a number between one and nine according to the following rules.

- White cells may not be colored black, and black cells may not be colored white. Prefilled numbers may not be erased.
- Each white cell needs to be filled with exactly one number between one and nine. The black cells may not be filled further.
- Each number between one and nine can appear at most once in each row and each column. This includes numbers that are written in black cells.
- Connected white cells, horizontally or vertically but not diagonally, must contain consecutive numbers. However, the order of the numbers in the cells does not matter. The connected white cells will be called "straights" and be frequently identified with the set of consecutive numbers written in them.

A solution of a given Str8ts puzzle is a grid with the same layout and predefined numbers where all empty white cells have been filled in accordance with the rules above. Note that a puzzle is only valid if the solution is unique.

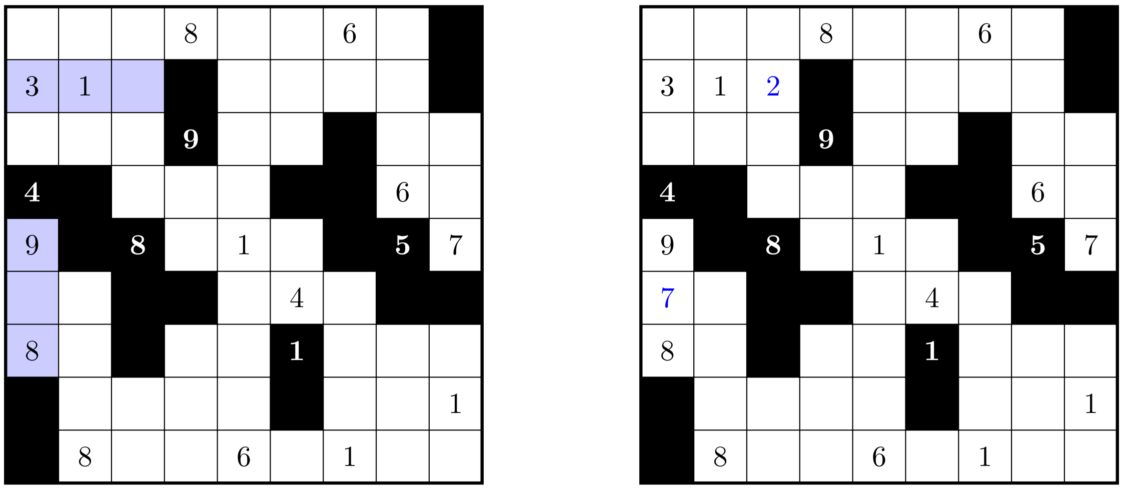


Figure 2.2: Step 1 of solving the example Str8ts

To start solving the example puzzle, there are two cells that can be filled in directly, simply by completing the straight. The vertical straight highlighted in figure 2.2 has only the number two missing to complete the numbers from one to three, so the empty white cell needs to be filled with a two. Likewise, in the vertical straight highlighted, the seven is missing since the numbers that may be written in the cell only go up to nine.

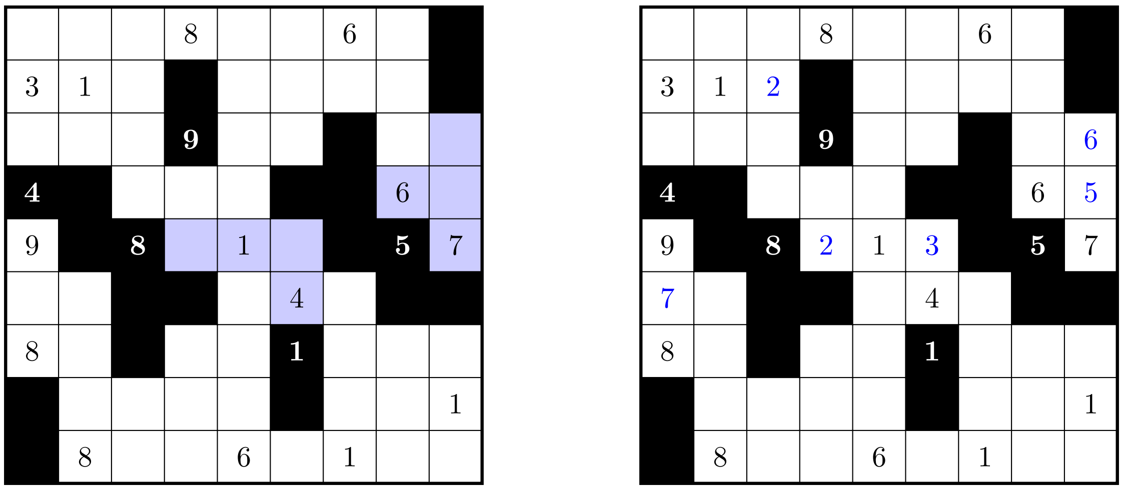
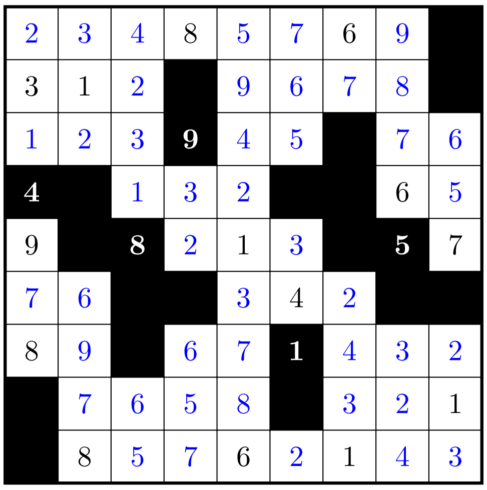


Figure 2.3: Step 2 of solving the example Str8ts

Consider now the middle cells highlighted in 2.3. The cell above the four can only contain a three or a five, else the straight of length two would not contain consecutive numbers. But the five is already blocked in this row and so it needs to be a three. With this information, the cell next to the one can also be filled with a two, analogously to before. The highlighted cells on the right can be filled similarly. Next to the six, only the five is possible, since the seven is already blocked by the cell below. And then the other empty cell needs to be filled with a six to complete the straight.

Figure 2.4 below shows the completed solution. For an in depth walkthrough of the solution process, see [9].



2	3	4	8	5	7	6	9	■
3	1	2	■	9	6	7	8	■
1	2	3	**9**	4	5	■	7	6
4	■	1	3	2	■	■	6	5
9	■	**8**	2	1	3	■	**5**	7
7	6	■	■	3	4	2	■	■
8	9	■	6	7	**1**	4	3	2
■	7	6	5	8	■	3	2	1
■	8	5	7	6	2	1	4	3

Figure 2.4: Solved example Str8ts

One alternative version of the classical Str8ts puzzle also considered here is the so called mini Str8ts. It follows the same rules as a regular Str8ts but on a six by six grid and thus only using numbers between one and six. The formulas to encode mini Str8ts will not be explicitly given in the following, but they can be constructed completely analogously.

Chapter 3

Encoding

To find an encoding for the puzzle Str8ts, the starting point is the encoding for Sudoku proposed in [4]. The rules that were considered and also apply to Str8ts are that

1. there is at least one number in each entry (in each white cell),
2. there is at most one number in each entry and
3. each number appears at most once in each row and column.

The main differences that still need to be formalized are the existence of black cells and the rules concerning the straights, where in Sudoku the three by three sub-grids exist. Note that, while rule 2 is not necessary for the Sudoku encoding and only appears in the proposed extended encoding in [4], it is necessary in Str8ts. Since not all cells are filled in each row and column, the fact that each number may appear at most once in each row and column does not imply that there can only be one number per cell.

Also like in [4], the coordinates shown in figure 3.1 will be used to identify single cells in the puzzle grid.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)	(1,9)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)	(2,9)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)	(3,9)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	(4,8)	(4,9)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	(5,8)	(5,9)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	(6,8)	(6,9)
(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)	(7,8)	(7,9)
(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)	(8,8)	(8,9)
(9,1)	(9,2)	(9,3)	(9,4)	(9,5)	(9,6)	(9,7)	(9,8)	(9,9)

Figure 3.1: Coordinates

Since all encodings below are given in CNF, all clauses making up the three formulas will be stated separately.

3.1 Encoding of an already given puzzle

This first encoding will be of an already given puzzle and is generated using all information about the layout and already filled cells. The general idea is to only add clauses concerning the empty white cells. The information on the black cells and prefilled numbers will only be used implicitly to reduce the set of possible entries for the empty white cells. To exemplify the procedure, the puzzle 2.1 will be used again.

Two sets of variables are needed for this encoding.

- $X_{i,j,k}$, taken as *true* if the cell with coordinates (i, j) contains the number k and
- $S_{i,s}$, taken as *true* if the straight i starts with the number s.

Logically, the specific ordering of the straights is not important. Here, the order is from top to bottom, left to right and from horizontal to vertical. In the example puzzle, the straight with number 1 is the horizontal straight between the cells $(1, 1)$ and $(1, 8)$ and the last straight, the straight with number 34, is the vertical straight between the cells $(7, 9)$ and $(9, 9)$. Figure 3.2 shows both straights highlighted.

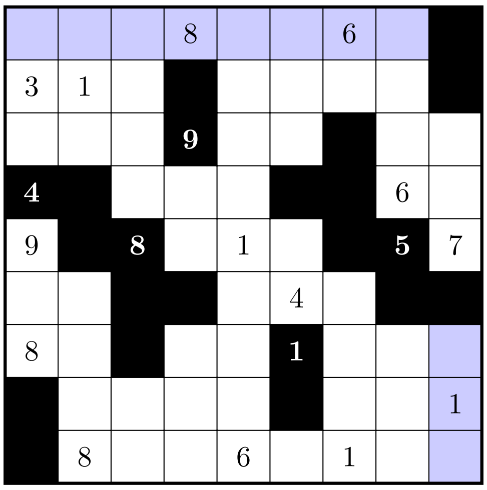


Figure 3.2: Straight 1 and 34

Saying that a given straight starts with a given number means that this number is the smallest one appearing in that straight. Since in the solution the highlighted straight 1 starts with the number 2 (it contains all numbers from 2 to 9), the model of the formula representing this puzzle needs to map the variable $S_{1,2}$ to *true* and the alternative start $S_{1,1}$ to *false*. Analogously, since the cell $(1, 1)$ needs to contain the number 2, the variable $X_{1,1,2}$ needs to be mapped to *true* and the variables

$$X_{1,1,1},\ X_{1,1,3},\ X_{1,1,4},\ X_{1,1,5},\ X_{1,1,6},\ X_{1,1,7},\ X_{1,1,8} \text{ and } X_{1,1,9}$$

to *false* (if they appear in the formula).

3.1.1 Individual cells

Referencing back to the rules 1 and 2 of Sudoku that also apply to Str8ts, the first task is to assure that there is at least one number in each empty white cell and that there is also at most one number in each empty white cell.

The existence of a number will follow from the clauses concerning the straights introduced in chapter 3.1.3 later. To ensure that that each cell is only filled with one digit, like in [4], the clause

$$\neg X_{i,j,k_1} \vee \neg X_{i,j,k_2} \tag{3.1}$$

is added to the formula for each empty white cell (i, j) and for every possible two digits k_1 and k_2 between one and nine where $k_1 \neq k_2$. Note that numbers k_1 and k_2 are only considered "possible", if the number does not already occur in the same row or column as the the cell (i, j), forbidding that number from being written in the cell. In the example, for cell $(1, 1)$, the possible values are $\{1, 2, 5, 7\}$ and thus the the needed clauses are

$$\begin{gathered}
\neg X_{1,1,1} \vee \neg X_{1,1,2} \\
\neg X_{1,1,1} \vee \neg X_{1,1,5} \\
\neg X_{1,1,1} \vee \neg X_{1,1,7} \\
\neg X_{1,1,2} \vee \neg X_{1,1,5} \\
\neg X_{1,1,2} \vee \neg X_{1,1,7} \\
\neg X_{1,1,5} \vee \neg X_{1,1,7}.
\end{gathered}$$

3.1.2 Columns and rows

Next, for rule 3 above, in [4] the clauses

$$\neg X_{i,j_1,k} \vee \neg X_{i,j_2,k} \text{ and } \neg X_{i_1,j,k} \vee \neg X_{i_2,j,k} \tag{3.2}$$

were introduced for rows i, columns j and values k. But again, for this encoding, only empty white cells (i, j_1) and (i, j_2) with $j_1 \neq j_2$ or (i_1, j) and (i_2, j) with $i_1 \neq i_2$ respectively and values for k that are possible for both cells need to be considered. So for the cells $(1, 1)$ and $(1, 2)$ in the example, the only possible values for k are $\{2, 5, 7\}$, giving the clauses

$$\begin{gathered}
\neg X_{1,1,2} \vee \neg X_{1,2,2} \\
\neg X_{1,1,5} \vee \neg X_{1,2,5} \\
\neg X_{1,1,7} \vee \neg X_{1,2,7}.
\end{gathered}$$

3.1.3 Straights

The most complicated part, and the part that differs from Sudoku, is to make sure that all straights contain consecutive numbers. As an example, consider the 7th straight, marked in figure 3.3 below. It has length three and no fixed entries that initially restrict which straights might be possible. But, there are already a four and a six in the same row, so it can be deduced that the straight can only start with a one or a seven. This corresponds to the clause

$$S_{7,1} \vee S_{7,7}.$$

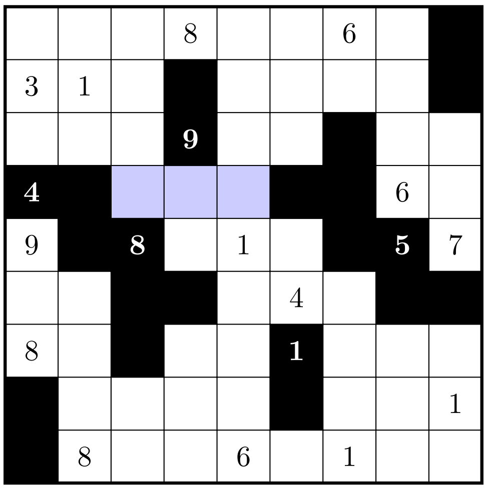


Figure 3.3: Straight 7

Now, if the straight started from one, all cells would have to contain a number between one and three and analogously if it started fom seven, all cells would have to contain a number between seven and nine. But the one can be ruled out for cell $(4,5)$ because there is a one in the cell below it. Further, the eight can be ruled out for cells $(4,3)$ and $(4,4)$ and the nine for cell $(4,4)$. This gives the formulas

$$S_{7,1} \to (X_{4,3,1} \lor X_{4,3,2} \lor X_{4,3,3}) \land (X_{4,4,1} \lor X_{4,4,2} \lor X_{4,4,3}) \land (X_{4,5,2} \lor X_{4,5,3})$$

and

$$S_{7,7} \to (X_{4,3,7} \lor X_{4,3,9}) \land X_{4,4,7} \land (X_{4,5,7} \lor X_{4,5,8} \lor X_{4,5,9}).$$

Transforming those formulas into CNF produces the list of clauses

$$\begin{aligned}
&\neg S_{7,1} \lor X_{4,3,1} \lor X_{4,3,2} \lor X_{4,3,3}\\
&\neg S_{7,1} \lor X_{4,4,1} \lor X_{4,4,2} \lor X_{4,4,3}\\
&\neg S_{7,1} \lor X_{4,5,2} \lor X_{4,5,3}\\
&\neg S_{7,7} \lor X_{4,3,7} \lor X_{4,3,9}\\
&\neg S_{7,7} \lor X_{4,4,7}\\
&\neg S_{7,7} \lor X_{4,5,7} \lor X_{4,5,8} \lor X_{4,5,9}.
\end{aligned}$$

Lastly, each straight can only have one beginning. So for each two possible beginnings, at least one has to be false. In the example case, this can be formalized using the clause

$$\neg S_{7,1} \lor \neg S_{7,7}.$$

The same procedure leads to the clauses for all other straights. First, check which beginnings are possible considering the already filled in cells (if there is only one possibility, the clause stating all possible beginnings can be left out). Second, formalize all possible digits the cells can have in accordance with the given beginnings and the already filled in cells. And lastly state that only one beginning can be true (this, also, can be left out if there is only one option).

Formally, for each straight i, let l be its length, let $\mathcal{K}_1$ be the set of numbers already written in any cell which is part of the straight and let $\mathcal{K}_2$ be the set of numbers that are possible for at least one cell which is part of the straight. The possible starting values for the straight are

$$\mathcal{S} := \{s \in \{1, \dots, 9\} \mid s \leq k < s + l \, \forall k \in \mathcal{K}_1, \, \{s, \dots, s + l - 1\} \subseteq \mathcal{K}_2\}.$$

If $\mathcal{S}$ has cardinality one, no clause is needed. Else, the clause

$$\bigvee_{s \in \mathcal{S}} S_{i,s} \tag{3.3}$$

is added to the formula. Note that $\mathcal{S}$ can not be empty, since each puzzle has a solution.

Next, for each possible starting value s in $\mathcal{S}$ and for each cell (i_1, j_1) that is part of the straight, the clause

$$\neg S_{i,s} \vee \bigvee_{k} X_{i_1,i_2,k} \tag{3.4}$$

is added, where k runs over all values that are possible for the cell (i_1, j_1) such that

$$s \leq k < s + l.$$

Lastly, again if $\mathcal{S}$ has cardinality grater than one, for each two values s_1 and s_2 in $\mathcal{S}$ where $s_1 \neq s_2$, the clause

$$\neg S_{i,s_1} \vee \neg S_{i,s_2} \tag{3.5}$$

is added.

3.2 General encoding

The main issue with the previous encoding is the fact that one needs to know the layout of the puzzle for every step. The goal now is to find an encoding of the general rules of Str8ts without having to know the puzzle. Later, only the information about the white and black cells and already filled in numbers needs to be added.

Where previously only clauses concerning empty white cells were part of the formula, now a way to denote that a cell is black is needed. This will be done by giving it the numerical value 0. So if the variable $X_{i,j,0}$ is *true*, this means that the cell with coordinates (i, j) is black.

3.2.1 Individual cells

Like in chapter 3.1.1, the clauses (3.1) are needed to ensure that there is at most one number per cell. This time though, the clause is needed for each cell (i, j) and every two numbers k_1 and k_2 between one and nine where $k_1 \neq k_2$. As the specific layout of the puzzle is not known, no cases can be excluded here.

Also, like before, rule 1 does not need to be encoded separately, as this follows again from the clauses concerning the straights in chapters 3.2.3 and 3.2.4, respectively.

3.2.2 Columns and rows

Like in chapter 3.1.2, the clauses (3.2) are needed to ensure that every number appears at most once per row and column. Again, this needs to be done for each two cells (i, j_1) and (i, j_2) with $j_1 \neq j_2$ or (i_1, j) and (i_2, j) with $i_1 \neq i_2$ respectively and each k between one and nine.

3.2.3 Straights - Version 1

For this first version of the general encoding, the idea is to determine for each cell whether it belongs to a horizontal or vertical straight of a given length and with a given starting value. Additionally, the encoding must decide whether any two given cells are members of the same straight. To encode this as a formula, four new sets of variables are needed:

- the variables $HS_{i,j,l,s}$, to mean that the cell with coordinates (i, j) is part of a horizontal straight with given length l and start s,
- the variables $VS_{i,j,l,s}$, to mean that the cell with coordinates (i, j) is part of a vertical straight with given length l and start s,
- the variables HSS_{i,j_1,j_2}, to mean that the cells with coordinates (i, j_1) and (i, j_2) are part of the same horizontal straight and
- the variables $VSS_{i_1,i_2,j}$, to mean that the cells with coordinates (i_1, j) and (i_2, j) are part of the same vertical straight.

Note that not every combination of length and starting value is possible for a straight. For example a straight with length nine can only start with a one while a straight with length one can start with every value (it is the only value). So by "possible straights", lengths and starting values

$$\mathcal{S} := \{(l, s) \in \{1, \dots, 9\}^2 \mid s \leq 9 - l + 1\}$$

are meant.

Now onto the concrete encoding. First, every white cell needs to be part of a vertical and horizontal straight. For every cell (i, j) this amounts to the formulas

$$\neg X_{i,j,0} \rightarrow \bigvee_{(l,s)\in\mathcal{S}} HS_{i,j,l,s} \text{ and } \neg X_{i,j,0} \rightarrow \bigvee_{(l,s)\in\mathcal{S}} VS_{i,j,l,s},$$

stating that if a cell is not black, it is part of at least one of the possible straights. The equivalent clauses in CNF are

$$X_{i,j,0} \vee \bigvee_{(l,s)\in\mathcal{S}} HS_{i,j,l,s} \text{ and } X_{i,j,0} \vee \bigvee_{(l,s)\in\mathcal{S}} VS_{i,j,l,s}. \tag{3.6}$$

Next, the possible lengths of the straights can be determined by method of exclusion. An upper limit can be determined by finding the black cells in each row and column. Surely, all white cells to the right/left of or above/below a black cell can not be in a straight that is longer than the distance between the black cell and the right/left or upper/lower edge of the puzzle field. Also, if a white cell is between two black cells, it can not be part of a straight that is longer than the their distance. Formally, for each cell (i,j), each cell (i,j_1) with $j_1 < j$ and for each straight with length $l_1 \geq j$ and possible start s_1, this means that

$$X_{i,j,0} \rightarrow \neg HS_{i,j_1,l_1,s_1}$$

or equivalently in CNF

$$\neg X_{i,j,0} \vee \neg HS_{i,j_1,l_1,s_1}. \tag{3.7}$$

Analogously for each cell (i,j), each cell (i,j_2) with $j_2 > j$ and for each straight with length $l_2 > 9-j$ and possible starts s_2 it follows

$$X_{i,j,0} \rightarrow \neg HS_{i,j_2,l_2,s_2},$$

or equivalently in CNF the clause

$$\neg X_{i,j,0} \vee \neg HS_{i,j_2,l_2,s_2} \tag{3.8}$$

is added. Lastly, for each three cells (i,j), (i,j_1) and (i,j_2) with $j_1 < i < j_2$ and each straight with length $l > j_2 - j_1 - 1$ and start s, the formula

$$X_{i,j_1,0} \wedge X_{i,j_2,0} \rightarrow \neg HS_{i,j,l,s},$$

which in CNF is equivalent to the clauses

$$\neg X_{i,j_1,0} \vee \neg X_{i,j_2,0} \vee \neg HS_{i,j,l,s}, \tag{3.9}$$

is needed. Of course, the clauses (3.7), (3.8) and (3.9) are also added in the vertical case, amounting to

$$\neg X_{i,j,0} \vee \neg VS_{i_1,j,l_1,s_1} \tag{3.10}$$

and

$$\neg X_{i,j,0} \vee \neg VS_{i_2,j,l_2,s_2} \tag{3.11}$$

as well as

$$\neg X_{i_1,j,0} \vee \neg X_{i_2,j,0} \vee \neg VS_{i,j,l,s}, \tag{3.12}$$

under the analogous conditions as in the horizontal case.

To find a lower limit on the possible lengths of the straights a cell can be in, the notion of two cells being in the same straight can be used. If two cells are indeed in the same straight, then the straight can not be shorter than the distance between those two cells. So for each two cells (i,j_1) and (i,j_2) with $j_1 < j_2$ and for each straight with start s and a length l which is not possible, i.e. if $l < j_2 - j_1$, the clause

$$\neg HSS_{i,j_1,j_2} \vee \neg HS_{i,j_1,l,s} \tag{3.13}$$

is added. Note that it is enough to make sure that the cell (i,j_1) is not part of a straight that is too short, because later a rule will be added which enforced that (i,j_1) and (i,j_2) have to be part of the same straight. So the fact that (i,j_2) is also not part of the too short straight will follow. The same thing is then also added for vertical straights, which is the clause

$$\neg VSS_{i_1,i_2,j} \vee \neg VS_{i_1,j,l,s}. \tag{3.14}$$

After having figured out in which straight a certain cell is, this then in turn implies which numbers can be written in that cell. If it is part of a straight with length three and start two for example, clearly only numbers between two an four can be filled in. The first rules that thus need to be added are, for each filed (i, j) and each possible straight with length l and start s,

$$HS_{i,j,l,s} \rightarrow \bigvee_{d=s}^{s+l-1} X_{i,j,d} \text{ and } VS_{i,j,l,s} \rightarrow \bigvee_{d=s}^{s+l-1} X_{i,j,d}.$$

In CNF this is equivalent to the clauses

$$\neg HS_{i,j,l,s} \vee \bigvee_{d=s}^{s+l-1} X_{i,j,d} \text{ and } \neg VS_{i,j,l,s} \vee \bigvee_{d=s}^{s+l-1} X_{i,j,d}. \tag{3.15}$$

Next, two cells that are in the same straight can not be part of different kind of straights, meaning different starting values or lengths. So for each two cells (i, j_1) and (i, j_2) with $j_1 < j_2$ and for each two possible straights with starts s_1 and s_2 and lengths l_1 and l_2 where $(l_1, s_1) \neq (l_2, s_2)$ the formula

$$HSS_{i,j_1,j_2} \rightarrow \neg(HS_{i,j_1,l_1,s_1} \wedge HS_{i,j_2,l_2,s_2}),$$

or equivalently in CNF

$$\neg HSS_{i,j_1,j_2} \vee \neg HS_{i,j_1,l_1,s_1} \vee \neg HS_{i,j_2,l_2,s_2}, \tag{3.16}$$

and analogously for the vertical straights,

$$\neg VSS_{i_1,i_2,j} \vee \neg VS_{i_1,j,l_1,s_1} \vee \neg VS_{i_2,j,l_2,s_2}. \tag{3.17}$$

is added.

Lastly, it has to be determined what it means for two cells to be in the same straight. This can be done by checking whether there is a black cell between the two. If this is not the case, they are in the same straight. For two cells (i, j_1) and (i, j_2) with $j_1 < j_2$ this is formalized through

$$\bigwedge_{j=j_1}^{j_2} \neg X_{i,j,0} \rightarrow HSS_{i,j_1,j_2},$$

or in CNF

$$\bigvee_{j=j_1}^{j_2} X_{i,j,0} \vee HSS_{i,j_1,j_2}. \tag{3.18}$$

And analogously in the vertical case this means

$$\bigvee_{i=i_1}^{i_2} X_{i,j,0} \vee VSS_{i_1,i_2,j}. \tag{3.19}$$

3.2.4 Straights - Version 2

In this second attempt, the two sets of variables $HS_{i,j,l,s}$ and $VS_{i,j,l,s}$ will be kept to again denote that the cell with coordinates (i, j) is part of a horizontal or vertical straight with length l and start s. But instead of directly encoding which cells are in the same straight, this time the idea is to find out which cells are the border cells of a straight. Cells are counted as border cells if (and only if) they are white and at the edge of the grid or right next to a black cell. All cells that are in between two border cells and have no black cell between them are then part of the same straight. For this the two new sets of variables needed are

- $HB_{i,j}$, to mean that the cell with coordinates (i, j) is a border cell of a horizontal straight and
- $VB_{i,j}$, to mean that the cell with coordinates (i, j) is a border cell of a vertical straight.

Figure 3.4 shows the example Str8ts puzzle with all border cells of the horizontal straights highlighted.

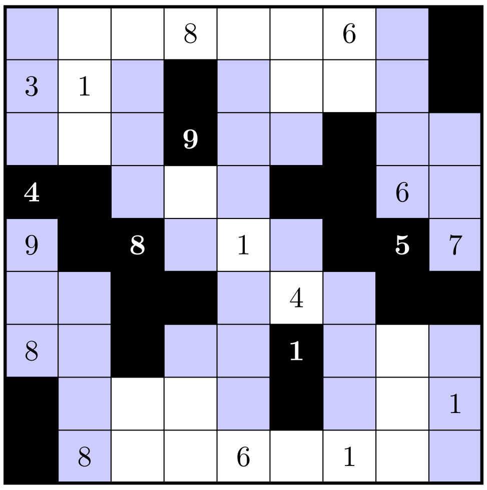


Figure 3.4: Horizontal border cells

The rules that can be kept from the first version are (3.6) and (3.15), denoting that every white cell is part of at least one straight and which possible numbers follow for a cell if it is part of a certain straight.

Next, a formal definition of border cells is needed. As mentioned before, every cell at the edge of the grid that is not black is a border cell, and every cell that is not black and right next to a black cell is a border cell. So for each row i this means

$$\neg X_{i,1,0} \rightarrow HB_{i,1} \text{ and } \neg X_{i,9,0} \rightarrow HB_{i,9}$$

and further for each $1 < j < 9$

$$\neg X_{i,j,0} \wedge X_{i,j-1,0} \rightarrow HB_{i,j} \text{ and } \neg X_{i,j,0} \wedge X_{i,j+1,0} \rightarrow HB_{i,j}.$$

In CNF this is equivalent to the clauses

$$X_{i,1,0} \vee HB_{i,1} \text{ and } X_{i,9,0} \vee HB_{i,9} \tag{3.20}$$

and

$$X_{i,j,0} \vee \neg X_{i,j-1,0} \vee HB_{i,j} \text{ and } X_{i,j,0} \vee \neg X_{i,j+1,0} \vee HB_{i,j}. \tag{3.21}$$

Analogously for each column j it also follows that

$$X_{1,j,0} \vee VB_{1,j} \text{ and } X_{9,j,0} \vee VB_{9,j} \tag{3.22}$$

and

$$X_{i,j,0} \vee \neg X_{i-1,j,0} \vee VB_{i,j} \text{ and } X_{i,j,0} \vee \neg X_{i+1,j,0} \vee VB_{i,j}. \tag{3.23}$$

From this it can be derived which cells are part of a certain straight. Clearly, if there are two border cells right next to each other (vertically or horizontally), then they both belong to the same straight, a straight of length two. Likewise, if there is only one cell between two border cells and it is white, then all three belong to the same straight of length three and so forth. In general, for each two cells (i, j_1) and (i, j_2) with $j_1 < j_2$, this can be formalized as

$$HB_{i,j_1} \wedge HB_{i,j_2} \bigwedge_{j=j_1+1}^{j_2-1} \neg X_{i,j,0} \to \bigvee_{s} HS_{i,j_1,j_2-j_1+1,s}$$

where s runs over all possible starting values for a straight with length $j_2 - j_1 + 1$. Equivalently in CNF this is represented by the clause

$$\neg HB_{i,j_1} \vee \neg HB_{i,j_2} \bigvee_{j=j_1+1}^{j_2-1} X_{i,j,0} \vee \bigvee_{s} HS_{i,j_1,j_2-j_1+1,s} \tag{3.24}$$

and analogously, in the vertical case, for two cells (i_1, j) and (i_2, j) where $i_1 < i_2$, by the clauses

$$\neg VB_{i_1,j} \vee \neg VB_{i_2,j} \bigvee_{i=i_1+1}^{i_2-1} X_{i,j,0} \vee \bigvee_{s} VS_{i_1,j,i_2-i_1+1,s}. \tag{3.25}$$

Note that the clauses above only imply that the left most or upper most cell has to be in the given straight, not all the cells between (and including) the border cells. This can be solved by demanding that every white cell that is to the right or below a cell that is in a certain straight has to be in the same straight. Then, knowing which straight the left most or upper most cell is in will propagate to the end of the straight. This can be done by demanding for each cell (i, j) where $j \leq 8$ and for each possible straight with length l and start s that

$$HS_{i,j,l,s} \wedge \neg X_{i,j+1,0} \to HS_{i,j+1,l,s},$$

or formulated as a clause

$$\neg HS_{i,j,l,s} \vee X_{i,j+1,0} \vee HS_{i,j+1,l,s} \tag{3.26}$$

and analogously for each cell (i, j) where $i \leq 8$ that

$$\neg VS_{i,j,l,s} \vee X_{i+1,j,0} \vee VS_{i+1,j,l,s}. \tag{3.27}$$

Lastly, each cell can only be part of one straight. With the clauses as they are at the moment, it would still be possible for a cell to be in multiple straights with different starting values (but not with different lengths). So for each cell (i,j) and each two straights with lengths l and starts s_1 and s_2 the clauses

$$\neg HS_{i,j,l,s_1} \vee \neg HS_{i,j,l,s_2} \text{ and } \neg VS_{i,j,l,s_1} \vee \neg VS_{i,j,l,s_2} \tag{3.28}$$

are needed.

3.2.5 Specific layout

After formalizing the general rules, only the information about the layout of the specific puzzle that is encoded needs to be added. This means that for every cell (i,j) that is black, the clause

$$X_{i,j,0} \tag{3.29}$$

is added, for each cell (i,j) that is white, the clause

$$\neg X_{i,j,0} \tag{3.30}$$

is added, for each cell (i,j) (black or white) that is already filled with a number k, the clause

$$X_{i,j,k} \tag{3.31}$$

is added and lastly, for each cell (i,j) that is black and does not contain a number, the clauses

$$\neg X_{i,j,1},\ \neg X_{i,j,2},\ \neg X_{i,j,3},\ \neg X_{i,j,4},\ \neg X_{i,j,5},\ \neg X_{i,j,6},\ \neg X_{i,j,7},\ \neg X_{i,j,8} \text{ and } \neg X_{i,j,9} \tag{3.32}$$

are added.

The first set of clauses encodes the puzzle layout by specifying the locations of all black cells. All cells that are not black are automatically white. The second set of clauses ensures that the white cells stay white as the rules of the game do not allow for the layout of the puzzle to be changed. The third set of clauses simply encodes all already filled in digits and the last set of clauses ensures that empty black cells can not be filled with a number.

Chapter 4

Theoretical analysis

The goal of this chapter is to theoretically analyze the three encodings introduced in the previous chapter in terms of clauses and variables needed. The larger the number of clauses and variables, the longer the expected times to generate the formula for a given puzzle and to then solve it.

4.1 Number of clauses

The specific encoding formalized in chapter 3.1 is expected to generate the least amount of clauses overall. This is because the given information is already considered for all clauses, so many cases can be eliminated that still need to be considered in the general encodings. Also, in the specific case it is already known where the straights are and how many cells they contain. Formalizing this makes up the largest amount of clauses in both versions of the general encoding.

The theoretical minimum of clauses for the specific encoding is zero, for an already fully filled out puzzle. The maximum is harder to determine, as this encoding highly depends on the layout of the puzzle. The largest part of clauses in the specific encoding results from ensuring that every empty white cell is only filled with one number and that every number occurs only once per column and row, clauses (3.1) and (3.2). So the worst case puzzle layout has the most amount of empty white fields possible. A completely empty six by six grid would generate

$$6^2 \cdot \binom{6}{2} = 540$$

clauses for ensuring that every cell contains only one number and

$$2 \cdot 6^2 \cdot \binom{6}{2} = 1080$$

clauses for ensuring that every number only occurs once per column and row. Analogously, for an empty nine by nine grid, the numbers would be

$$9^2 \cdot \binom{9}{2} = 2916$$

and

$$2 \cdot 9^2 \cdot \binom{9}{2} = 5831$$

clauses.

The second largest impact on the number of clauses has the number of straights, in particular straights with many possible staring values. Figure 4.1 shows the proposed puzzle grid with the most amount of straights possible.

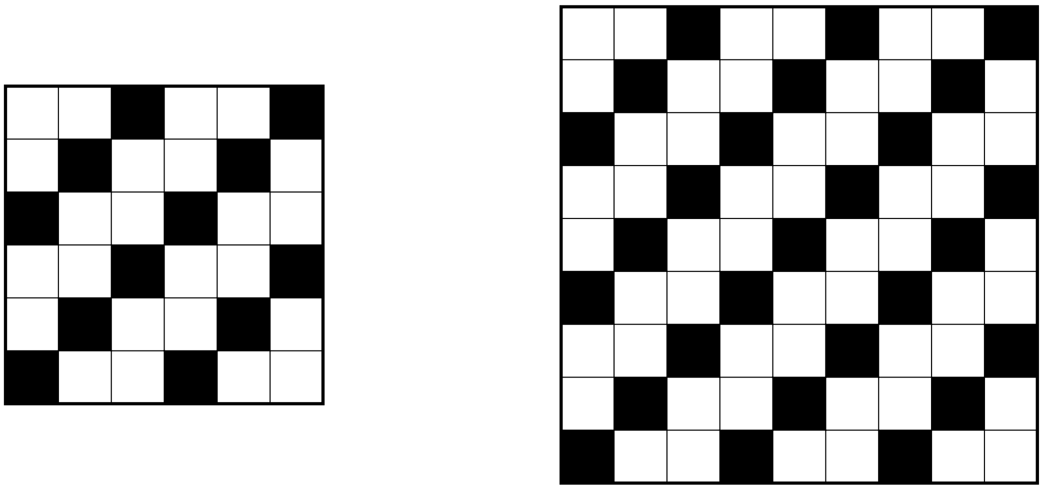


Figure 4.1: Most amount of straights

The puzzles contain 20 or 48 straights respectively, each with length two and each with five or eight possible starting values. In total, without additional numbers in the grid, this would generate

$$20 \cdot \left(1 + 2 \cdot 5 + \binom{5}{2}\right) = 420$$

clauses in the six by six and

$$48 \cdot \left(1 + 2 \cdot 8 + \binom{8}{2}\right) = 2160$$

clauses in the nine by nine case to encode the straights.

So 2'040 clauses for the mini Str8ts and 10'907 clauses for the regular Str8ts can be taken as a rough upper limit for the number of clauses in the specific encoding.

For the fist version of the general encoding, the general rules in the six by six case contain 84'780 clauses and 1'352'376 clauses in the nine by nine case. Here, the minimum amount of total clauses would be generated by an empty grid, which would add only 6^2 or 9^2 new clauses respectively. The maximum amount of total clauses would be generated by a completely black grid without any numbers, adding 6^3 or 9^3 clauses. In both cases, the number of specific clauses is negligible. For the second version of the general encoding, the general rules contain 62'496 and 9'804 clauses and like in the first version, the clauses added by the specific layout can be disregarded. Between the two general encodings, the second version is by far favorable in terms of generated clauses.

The biggest impact on the number of clauses in the first version have (3.16) and (3.17). Since the locations and lengths of the straights is determined by knowing which cells are in the same straight, a lot of cases have to be handled to make sure that cells that are in the same straight are actually in the same type of straights, i.e a straight with the same length and starting value. This can be avoided in the second version, since the information about the locations and lengths of the straights directly follows from knowing the border cells, see clauses (3.24) and (3.25). And then the start propagates from the beginning of the straight to the end in (3.26) and (3.27), making sure that each cell is in the same type of straight and using a significantly reduced number of clauses.

4.2 Number of variables

Like with the number of clauses, it is expected that the specific encoding uses the least amount of variables in the encoding. It needs one variable per possible number in each empty white cell and one variable per straight and possible starting value for this straight. This makes at most

$$6^3 + 20 \cdot 5 = 316$$

variables in the six by six case and

$$9^3 + 48 \cdot 8 = 1113$$

variables in the nine by nine case. The estimates again use an empty grid for the number of variables per cell and the grid shown in figure 4.1 for the number of variables per straight.

Both general encodings always use the same amount of variables, since the general rules do not change between puzzles and the encoding of the specific layout does not add new variables. For the first version this makes

$$6^2 \cdot 7 + 2 \cdot 6^2 \cdot 21 + 2 \cdot 6 \cdot \binom{6}{2} = 1944$$

variable in the six by six case and

$$9^2 \cdot 10 + 2 \cdot 9^2 \cdot 45 + 2 \cdot 9 \cdot \binom{9}{2} = 8748$$

variables in the nine by nine case. For the second version

$$6^2 \cdot 7 + 2 \cdot 6^2 \cdot 21 + 2 \cdot 6^2 = 1836$$

variables and

$$9^2 \cdot 10 + 2 \cdot 9^2 \cdot 45 + 2 \cdot 9^2 = 8262$$

variables are used respectively. So also in terms of variables, the second version of the general encoding is favorable over the first version, though not by a margin as large as in the number of clauses.

Chapter 5

Experiments

This chapter details the experiments performed to verify and further analyze the three encodings.

5.1 Setup

To test the different encodings, a so called SAT solver is used, a solver for propositional satisfiability problems. That is, for a given propositional formula in CNF, it is able to determine whether it is satisfiable, and if so, which model satisfies it. For the experiments, the SAT-solver CaDiCal [1] is used.

The input for a SAT solver is a file in DIMACS CNF file format [3]. It first contains a "problem" line, beginning with "p cnf" followed by a space, the number of variables in the formula, another space and the number of clauses in the formula. Below the problem line, each clause of the formula is written line by line, each line ending with a "0". Each clause is a sequence of integers, separated by a space. Each positive integer corresponds to a variable, if the integer is negative it corresponds to the negation of the variable. The formula

$$(X_1 \vee \neg X_3) \wedge (X_2 \vee X_3 \vee \neg X_1)$$

corresponds to the following three lines.

```
p cnf 3 2
1 -3 0
2 3 -1 0
```

Note that CaDiCal finds a model including all variables up to the number stated in the header, even if the integer does not appear in the file. In those cases, it returns the value *true* for those variables by default. In the implementation of the experiment, all variables were encoded as integers in the following way.

- The variables $X_{i,j,k}$ correspond to the integers $i \cdot 10^2 + j \cdot 10 + k$.
- The variables $S_{i,s}$ correspond to the integers $10^3 + i \cdot 10 + s$.
- The variables $HS_{i,j,l,s}$ correspond to the integers $10^4 + i \cdot 10^3 + j \cdot 10^2 + l \cdot 10 + s$.
- The variables $VS_{i,j,l,s}$ correspond to the integers $2 \cdot 10^4 + i \cdot 10^3 + j \cdot 10^2 + l \cdot 10 + s$.
- The variables HSS_{i,j_1,j_2} correspond to the integers $1 \cdot 10^3 + i \cdot 10^2 + j_1 \cdot 10 + j_2$.

- The variables $VSS_{i_1,i_2,j}$ correspond to the integers $2 \cdot 10^3 + i_1 \cdot 10^2 + i_2 \cdot 10 + j$.
- The variables $HB_{i,j}$ correspond to the integers $1 \cdot 10^3 + i \cdot 10 + j$.
- The variables $VB_{i,j}$ correspond to the integers $2 \cdot 10^3 + i \cdot 10 + j$.

This of course leads to a larger variable range than is strictly necessary which may have influenced the time and memory used by CaDiCal. However, it is not clear if this influence is significant for the result.

To verify the result of the solver, the model given by CaDiCal was saved. In the specific encoding, a list of used variables was generated together with the formula. The variables marked as "true" by the solver that were also in the initial formula were then compared with the expected variables resulting from the solution of the puzzle. For the general encodings, all variables of the form $i \cdot 10^2 + j \cdot 10 + k$ for i, j and k between one and nine that were marked as "true" by CaDiCal were extracted and compared with the solution.

To test the encodings, a set of 1'000 regular Str8ts, split equally into four difficulties (easy, medium, hard and extra-hard) and 100 mini Str8ts split equally into three difficulties (easy, medium and hard) was used. The test data, including the grading, was kindly provided by Andrew Stuart and will be kept private. The grading algorithm used by Andrew Stuart is not published, but for his grading schema for Sudokus see [6].

For all 1'100 puzzles and all three encodings the DIMACS file was generated using Java. For the two general encodings, the general rules were saved as a text file, then copied into the DIMACS file and completed with the specific information. The calculations were performed at sciCORE [7] using a memory limit of 8'192 MB, a time limit of 1'800 seconds and one node per task.

5.2 Results

All puzzles in the test data set were solvable by CaDiCal and produced the correct result for all three encodings. In the following, the experiment will be analyzed in terms of number of variables and clauses, generation time of the DIMACS file and time and memory used by CaDiCal during solving.

In the following, the "Specific version" corresponds to the encoding introduced in chapter 3.1, the "General version 1" to the encoding introduced in chapter 3.2.3 and the "General version 2" to the encoding introduced in chapter 3.2.4.

5.2.1 Number of straights

To contextualize the number of variables and clauses used by the specific encoding, the number of straights per size and difficulty was tracked, the average number is shown in table 5.1. It does not seem to correlate with the difficulty and lies not too far below the determined maximum.

	mini Str8ts	Str8ts
Easy	17.58	37.09
Medium	17.03	37.28
Hard	17.29	37.00
Extra-Hard		36.55

Table 5.1: Average number of straights

5.2.2 Number of clauses

Table 5.2 below shows the average number of clauses generated for the mini Str8ts puzzle ordered by difficulty and encoding, table 5.3 shows the same for the regular Str8ts puzzles.

	Specific version	General version 1	General version 2
Easy	485.52	84863.97	9887.97
Medium	492.79	84862.67	9886.67
Hard	564.50	84859.97	9883.97

Table 5.2: Average number of clauses for the mini Str8ts puzzles

	Specific version	General version 1	General version 2
Easy	1334.78	1352646.72	62766.72
Medium	1975.82	1352622.76	62742.76
Hard	3151.15	1352612.08	62732.08
Extra-Hard	3611.85	1352600.97	62720.97

Table 5.3: Average number of clauses for the Str8ts puzzles

The number of clauses for the specific version gets larger with increasing difficulty, as there is less information given initially. In each case it lays far below the estimated maximum as the number of empty white cells, the number of possible entries per cell and the number of straights and their possible starting values was greatly overestimated.

Interestingly, the number of clauses gets smaller with increasing difficulty in both general encodings. This is because, besides the fixed rules, the clauses stem from the encoding of the specific layout. And here, if less information is given, as is expected in a more difficult puzzle, there are also less clauses.

As expected, the specific version has the smallest amount of clauses overall. Between the two general encodings, the second version is preferable.

5.2.3 Number of variables

As discussed in chapter 4.2, the number of variables only changes in the specific encoding. The average number of used variables per difficulty is shown in table 5.4.

	mini Str8ts	Str8ts
Easy	119.36	278.78
Medium	119.21	355.23
Hard	131.88	476.15
Extra-Hard		518.63

Table 5.4: Average number of variables

Clearly, the number of variables gets larger with increasing difficulty. Since the average number of straights does not correlate with the difficulty, this results only from having less information initially. Still, not even half as many variables are used as the estimated maximum amount.

5.2.4 Time

During the experiments, both the real time to generate the DIMACS files and the time to solve the formulas (CPU time and real time) were tracked. Table 5.5 shows the average time it took to generate the files, grouped again by difficulty and encoding, for the mini Str8ts, table 5.6 shows the same for the Str8ts puzzles. Tables 5.7 and 5.8 show the average real time to solve the formulas as reported by CaDiCal and tables 5.9 and 5.10 show the CPU times. All times data is given in seconds.

	Specific version	General version 1	General version 2
Easy	0.22	0.28	0.19
Medium	0.17	0.24	0.19
Hard	0.20	0.31	0.25

Table 5.5: Average time to generate the DIMACS file for the mini Str8ts puzzles

	Specific version	General version 1	General version 2
Easy	0.23	1.31	0.26
Medium	0.25	1.28	0.27
Hard	0.25	1.28	0.27
Extra-Hard	0.30	1.27	0.26

Table 5.6: Average time to generate the DIMACS file for the Str8ts puzzles

The time to generate the DIMACS file does not seem to greatly correlate with the difficulty in both general encodings. But the generation time for the general encodings seems to stem predominantly from having to copy the general clauses into the DIMACS file. If this can be optimized, the general encodings can be expected to have a significantly shorter generation time that the specific version, since the generation does not contain much further calculation and only iterates over the puzzle once.

In the specific case, a slight increase of time with increased difficulty can be observed as there are more cases to consider and more clauses to write. However, the time difference is marginal.

	Specific version	General version 1	General version 2
Easy	0.01	0.13	0.07
Medium	0.01	0.12	0.06
Hard	0.01	0.13	0.06

Table 5.7: Average real time to solve the formula by CaDiCal, mini Str8ts

	Specific version	General version 1	General version 2
Easy	0.01	1.06	0.09
Medium	0.01	0.91	0.14
Hard	0.03	1.05	0.26
Extra-Hard	0.04	0.90	0.24

Table 5.8: Average real time to solve the formula by CaDiCal, Str8ts

	Specific version	General version 1	General version 2
Easy	0.01	0.09	0.03
Medium	0.01	0.09	0.03
Hard	0.01	0.10	0.03

Table 5.9: Average CPU time to solve the formula by CaDiCal, mini Str8ts

	Specific version	General version 1	General version 2
Easy	0.01	1.01	0.06
Medium	0.01	0.87	0.10
Hard	0.03	1.01	0.23
Extra-Hard	0.03	0.85	0.20

Table 5.10: Average CPU time to solve the formula by CaDiCal, Str8ts

Looking at the CPU time needed to solve the formulas, the specific encoding has by far the best performance. This was to be expected considering that the specific version contains by far the least amount of clauses and also the least amount of variables. Even for the regular Str8ts, the solve time here is almost negligible. Comparing the general encodings, the second version is faster by a factor of about three for the mini Str8ts and of about five to ten for the regular Str8ts. In both cases, the performance is significantly slower in the nine by nine case than in the six by six case, which is not the case for the specific encoding. Interestingly, the difficulty seems not to be a factor for the first version of the general encoding.

5.2.5 Memory

Lastly, CaDiCal reports on the used memory (in MB) during solving. Table 5.11 shows the average memory reported for the mini Str8ts, grouped again by difficulty and encoding. Table 5.12 sows the same for the Str8ts puzzles.

	Specific version	General version 1	General version 2
Easy	11.58	22.76	11.60
Medium	11.60	23.15	11.98
Hard	11.57	23.38	12.70

Table 5.11: Average memory used to solve the formula by CaDiCal, mini Str8ts

	Specific version	General version 1	General version 2
Easy	11.57	145.38	16.81
Medium	11.59	146.77	18.96
Hard	11.58	147.79	22.82
Extra-Hard	11.56	148.43	23.01

Table 5.12: Average memory used to solve the formula by CaDiCal, Str8ts

All three encodings are surprisingly consistent when it comes to memory usage. The differences can again be easily explained by the numbers of clauses and variables in each encoding.

5.3 Conclusion

All three encodings produce the correct output and use reasonably little time and memory. The specific encoding has the best performance overall but needs some amount of logic and knowledge of the puzzle layout to create the formula. The general encodings can be generated with a very limited amount of calculations and without initial knowledge of the puzzle, making them more versatile and simpler to implement. Between the two general encodings, the second version is preferable in both simplicity and performance.

Bibliography

[1] Armin Biere et al. "CaDiCaL 2.0". In: *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*. Ed. by Arie Gurfinkel and Vijay Ganesh. Vol. 14681. Lecture Notes in Computer Science. Springer, 2024, pp. 133–152. DOI: 10.1007/978-3-031-65627-9_7.

[2] Colin Howson. *Logic with trees: an introduction to symbolic logic*. Routledge, 2005.

[3] David S. Johnson and Michael A. Trick, eds. *Cliques, coloring, and satisfiability. Second DIMACS implementation challenge. Proceedings of a workshop held at DIMACS, October 11–13, 1993*. English. Vol. 26. DIMACS, Ser. Discrete Math. Theor. Comput. Sci. Providence, RI: American Mathematical Society, 1996. ISBN: 0-8218-6609-5.

[4] Ines Lynce and Joel 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.

[5] Sebastian Schlachter. "Encoding Diverse Sudoku Variants as SAT Problems". Bachelor's Thesis. University of Basel, 2022.

[6] Andrew C. Stuart. "Sudoku creation and grading". In: *Mathematica* 39.6 (2007), pp. 126–142.

[7] Center for Scientific Computing University of Basel. URL: https://scicore.unibas.ch/ (visited on 01/07/2026).

[8] Department of Mathematics University of Basel and Computer Science. 2025. URL: https://dmi.unibas.ch/en/studies/computer-science/course-offer-fall-semester-25/lecture-discrete-mathematics-in-computer-science/ (visited on 12/31/2025).

[9] Jeff Widderich and Andrew Stuart as Syndicated Puzzles. 2008. URL: https://www.str8ts.com/ (visited on 12/29/2025).