What is Sudoku?

- Played on a $n \times n$ board.
- A single number from 1 to $n$ must be put in each cell; some cells are pre-filled.
- Board is subdivided into $\sqrt{n} \times \sqrt{n}$ blocks.
- Each number must appear exactly once in each row, column, and block.
- Designed to have a unique solution.

Naive Encoding (1)

- Use $n^3$ variables, labelled “$x_{0,0,0}$” to “$x_{n,n,n}$”.
- Variable $x_{r,c,d}$ represents whether the number $d$ is in the cell at row $r$, column $c$.
- “Number $d$ must occur exactly once in column $c$” $\Rightarrow$ “Exactly one of $\{x_{1,c,d}, x_{2,c,d}, \ldots, x_{n,c,d}\}$ is true”.
- How do we encode the constraint that exactly one variable in a set is true?

Example of Variable Encoding

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Variable $x_{r,c,d}$ represents whether the digit $d$ is in the cell at row $r$, column $c$.

$x_{1,1,3} = \text{true}, x_{1,2,2} = \text{true}, x_{1,3,1} = \text{true}, x_{1,4,4} = \text{true}$
$x_{2,1,4} = \text{true}, x_{2,2,1} = \text{true}, x_{2,3,2} = \text{true}, x_{2,4,3} = \text{true}$
$x_{3,1,1} = \text{true}, x_{3,2,4} = \text{true}, x_{3,3,3} = \text{true}, x_{3,4,2} = \text{true}$
$x_{4,1,2} = \text{true}, x_{4,2,3} = \text{true}, x_{4,3,4} = \text{true}, x_{4,4,1} = \text{true}$

All others are false.
Naive Encoding (2)

- How do we encode the constraint that **exactly one** variable in a set is true?
- We can encode “**exactly one**” as the conjunction of “**at least one**” and “**at most one**”.
- Encoding “**at least one**” is easy: simply take the logical OR of all the propositional variables.
- Encoding “**at most one**” is harder in CNF. Standard method: “no two variables are both true”.
- i.e., enumerate every possible pair of variables and require that one variable in the pair is false. This takes $O(n^2)$ clauses.
- [ Example on next slide ]

Naive Encoding (3)

- Example for 3 variables ($x_1$, $x_2$, $x_3$).
- “At least one is true”:
  $$x_1 \lor x_2 \lor x_3.$$  
- “At most one is true”:
  $$(\neg x_1 \lor \neg x_2) & (\neg x_1 \lor \neg x_3) & (\neg x_2 \lor \neg x_3).$$
- “Exactly one is true”:
  $$(x_1 \lor x_2 \lor x_3) & (\neg x_1 \lor \neg x_2) & (\neg x_1 \lor \neg x_3) & (\neg x_2 \lor \neg x_3).$$

Naive Encoding (4)

The following constraints are encoded:
- Exactly one digit appears in each cell.
- Each digit appears exactly once in each row.
- Each digit appears exactly once in each column.
- Each digit appears exactly once in each block.

Each application of the above constraints has the form: “exactly one of a set variables is true”.

All of the above constraints are independent of the prefilled cells.

Problem with Naive Encoding

- We need $n^3$ total variables. ($n$ rows, $n$ cols, $n$ digits)
- And $O(n^4)$ total clauses.
  - To require that the digit “1” appear exactly once in the first row, we need $O(n^2)$ clauses.
  - Repeat for each digit and each row.
- For your project, the naive encoding is OK.
- For large $n$, it is a problem.
A Better Encoding (1a)

- Simple idea: Don’t emit variables that are made true or false by prefilled cells.
  - Larger grids have larger percentage prefilled.
- Also, don’t emit clauses that are made true by the prefilled cells.
- This makes encoding and decoding more complicated.

A Better Encoding (1b)

Example: Consider the CNF formula 
\[(a \lor d) \land (a \lor b \lor c) \land (c \lor \neg b \lor e)\].

- Suppose the variable \(b\) is preset to true.
- Then the clause \((a \lor b \lor c)\) is automatically true, so we skip the clause.
- Also, the literal \(\neg b\) is false, so we leave it out from the 3rd clause.
- Final result: \((a \lor d) \land (c \lor e)\).

New Encoding: Implementation

- Keep a 3D array \(\text{VarNums}[r][c][d]\).
  - Map possible variables to an actual variable number if used.
  - Assign \(0x\text{FFFFFFF}\) to true variables.
  - Assign \(-0x\text{FFFFFFF}\) to false variables.
- Initialize \(\text{VarNums}[r][c][d]\) to zeros.
- For each prefilled cell, store the appropriate code (\(\pm 0x\text{FFFFFFF}\)) into the array elems for the cell.
- For every cell in the same row, column, or block:
  - Assign \(-0x\text{FFFFFFF}\) (preset_false) to the var that would put the same digit in this cell.

Room for Another Improvement

- It still takes \(O(n^2)\) clauses to encode an \textbf{at most one} constraint.
- When one of these vars becomes true, the SAT solver examines clauses that contain the newly true var.
- This allows the SAT solver to quickly realize that none of the other vars in the “at most” set can be true.
- But requires \((n)(n-1)/2\) clauses.
- Improvement: Use ‘intermediary’ nodes (next slide).
Intermediary Variables

Idea:
- Divide the $n$ variables into groups containing only a handful of vars.
- Add an intermediary variable to each group of vars.
- An intermediary variable is to be true if one of the (original) vars in its group is \textit{true}.
- Add a constraint to ensure that at most one intermediary variable is \textit{true}.
- If there are too many intermediary variables, then they themselves may be grouped, forming a hierarchy.

Hierarchical Grouping

Results

- Number of clauses for an “at most one” clause reduced from $O(n^2)$ to $O(n \log n)$.
- But in the larger puzzles, most of the cells are prefilled, so this only offered a 10%-20% performance benefit.

<table>
<thead>
<tr>
<th>PUZZLE 100x100</th>
<th>NumVars</th>
<th>NumClauses</th>
<th>Sat Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var Elim Only</td>
<td>36,415</td>
<td>712,117</td>
<td>1.04 sec</td>
</tr>
<tr>
<td>Elim &amp; Intermediary</td>
<td>61,793</td>
<td>428,231</td>
<td>0.76 sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PUZZLE 144x144</th>
<th>NumVars</th>
<th>NumClauses</th>
<th>Sat Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var Elim Only</td>
<td>38,521</td>
<td>596,940</td>
<td>0.91 sec</td>
</tr>
<tr>
<td>Elim &amp; Intermediary</td>
<td>58,843</td>
<td>405,487</td>
<td>0.76 sec</td>
</tr>
</tbody>
</table>