Optimized SAT Encoding
For Sudoku Puzzles

Will Klieber and Gi-Hwon Kwon
Sept 27, 2007

rev. 2

Changes from v1:
• Added a new slide after “A Better Encoding (1)” to explain how we deal with clauses that would contain skipped variables.
• In the “Implementation” slides, use different codes ($\pm 0\times FFFFFF$ instead of -1 and -2) to reduce confusion that resulted from another possible interpretation of the old codes.
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.
Puzzle-Solving Process

Puzzle \rightarrow \text{Encoder} \rightarrow \text{CNF} \rightarrow \text{SAT Solver} \rightarrow \text{SAT Sol'n} \rightarrow \text{Decoder} \rightarrow \text{Puzzle Sol'n}

\text{Mapping of SAT variables to Sudoku cells}
Outline of This Talk

• Previous SAT Encodings for Sudoku
• Optimizing the Encoding of Variables
• Optimizing the Encoding of Constraints
Previous Encodings (1)

• All use $n$ variables per cell: one for each possible number.
• Variables are labelled “$x_{r,c,d}$”, where $r$ is the row, $c$ is the column, and $d$ is the digit. The variable is true iff the digit occurs in the cell.
• How do we encode the constraint that each digit occurs \textbf{exactly once} in each row/col/block?
Example of Variable Encoding

\[
\begin{array}{cccc}
3 & 2 & 1 & 4 \\
4 & 1 & 2 & 3 \\
1 & 4 & 3 & 2 \\
2 & 3 & 4 & 1 \\
\end{array}
\]

\begin{align*}
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} \\
\end{align*}

All others are false.

- Variables are labelled “\(x_{r,c,d}\)”, where \(r\) is the row, \(c\) is the column, and \(d\) is the digit.
Previous Encodings (2)

• How do we encode (in CNF) that each digit occurs exactly once in each row/col/block?
• 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. Std 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.
Previous Encodings (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”:
    \[(\sim x_1 \lor \sim x_2) \& (\sim x_1 \lor \sim x_3) \& (\sim x_2 \lor \sim x_3).\]
  • “Exactly one is true”:
    \[(x_1 \lor x_2 \lor x_3) \& (\sim x_1 \lor \sim x_2) \& (\sim x_1 \lor \sim x_3) \& (\sim x_2 \lor \sim x_3).\]
Previous Encodings (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.
- Prefilled cells.
Problem with Previous Encodings

• We need $O(n^3)$ total variables.
  $(n \text{ rows, } n \text{ cols, } n \text{ digits})$

• 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 large $n$, this is a problem.
# Experimental Results

<table>
<thead>
<tr>
<th>size</th>
<th>level</th>
<th>minimal encoding</th>
<th>efficient encoding</th>
<th>extended encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>vars</td>
<td>clauses</td>
<td>time</td>
</tr>
<tr>
<td>9x9</td>
<td>easy</td>
<td>729</td>
<td>8854</td>
<td>0.00</td>
</tr>
<tr>
<td>9x9</td>
<td>hard</td>
<td>729</td>
<td>8859</td>
<td>0.00</td>
</tr>
<tr>
<td>16x16</td>
<td>easy</td>
<td>4096</td>
<td>92520</td>
<td>0.10</td>
</tr>
<tr>
<td>16x16</td>
<td>hard</td>
<td>4096</td>
<td>92514</td>
<td>0.46</td>
</tr>
<tr>
<td>25x25</td>
<td>easy</td>
<td>15625</td>
<td>563417</td>
<td>9.07</td>
</tr>
<tr>
<td>25x25</td>
<td>hard</td>
<td>15625</td>
<td>563403</td>
<td>time</td>
</tr>
<tr>
<td>36x36</td>
<td>easy</td>
<td>46656</td>
<td>2451380</td>
<td>time</td>
</tr>
<tr>
<td>36x36</td>
<td>hard</td>
<td>46656</td>
<td>2451400</td>
<td>time</td>
</tr>
<tr>
<td>49x49</td>
<td>easy</td>
<td>117649</td>
<td>8474410</td>
<td>time</td>
</tr>
<tr>
<td>64x64</td>
<td>easy</td>
<td>262144</td>
<td>24779088</td>
<td>stack</td>
</tr>
<tr>
<td>81x81</td>
<td>easy</td>
<td>531441</td>
<td>63783464</td>
<td>stack</td>
</tr>
</tbody>
</table>
A Better Encoding (1a)

• Simple idea: Don’t emit variables for prefilled cells.
  – Larger grids have larger percentage prefilled.
• Also, if we know that a given variable must be false (e.g., to avoid the same digit appearing twice in a row), don’t emit it.
• 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 \text{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

• Most SAT solvers use an input format wherein vars are identified by number.
• Keep a 3D array \texttt{VarNums[r][c][d]}.
  – Map each possible SAT variable to an actual variable number.
• But don’t give a variable number if the value is known in advance.
  – Assign \texttt{0xFFFFFFFF} to true variables.
  – Assign \texttt{-0xFFFFFFFF} to false variables.
  – (This assumes less than 16 million vars.)
New Encoding: Implementation (2)

- Initialize `VarNums[r][c][d]` to zeros.
- For each prefilled cell, store the appropriate code (`±0xFFFFFFFF`) into the array elems for the cell.
- Also, for every other cell in the same row, column, or block:
  - Assign `-0xFFFFFFFF` (`preset_false`) to the var that would put the same digit in this cell.
- Finally, assign a real var number to each array element that is still zero.
## Experimental Results

<table>
<thead>
<tr>
<th>size</th>
<th>level</th>
<th>extended encoding</th>
<th>proposed encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>vars</td>
<td>clauses</td>
</tr>
<tr>
<td>9x9</td>
<td>easy</td>
<td>729</td>
<td>12013</td>
</tr>
<tr>
<td>9x9</td>
<td>hard</td>
<td>729</td>
<td>12018</td>
</tr>
<tr>
<td>16x16</td>
<td>easy</td>
<td>4096</td>
<td>124008</td>
</tr>
<tr>
<td>16x16</td>
<td>hard</td>
<td>4096</td>
<td>124002</td>
</tr>
<tr>
<td>25x25</td>
<td>easy</td>
<td>15625</td>
<td>752792</td>
</tr>
<tr>
<td>25x25</td>
<td>hard</td>
<td>15625</td>
<td>752778</td>
</tr>
<tr>
<td>36x36</td>
<td>easy</td>
<td>46656</td>
<td>3271748</td>
</tr>
<tr>
<td>36x36</td>
<td>hard</td>
<td>46656</td>
<td>3271768</td>
</tr>
<tr>
<td>49x49</td>
<td>easy</td>
<td>117649</td>
<td>11305189</td>
</tr>
<tr>
<td>64x64</td>
<td>easy</td>
<td>262144</td>
<td>33048912</td>
</tr>
<tr>
<td>81x81</td>
<td>easy</td>
<td>531441</td>
<td>85060787</td>
</tr>
</tbody>
</table>
Room for Another Improvement

• It still takes $O(n^2)$ clauses to encode an "at most one" constraint.
• When one of these vars becomes true, the SAT solver examines clauses that contains 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 iff one of the (original) vars in its group is true.
- Add a constraint to ensure that at most one intermediary variable is 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>