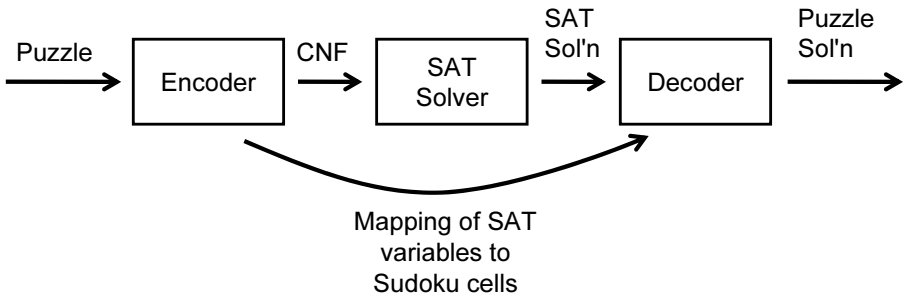
What is Sudoku?

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

- 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.

Puzzle-Solving Process



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

3	2	1	4
4	1	2	3
1	4	3	2
2	3	4	1

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

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

All others are false.

row, column, digit

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 ($x1$, $x2$, $x3$).
- "At least one is true":

$$x1 \vee x2 \vee x3.$$

- "At most one is true":

$$(\sim x1 \vee \sim x2) \;\&\; (\sim x1 \vee \sim x3) \;\&\; (\sim x2 \vee \sim x3).$$

- "Exactly one is true":

$$(x_1 \vee x_2 \vee x_3) \;\&\; (\sim x_1 \vee \sim x_2) \;\&\; (\sim x_1 \vee \sim x_3) \;\&\; (\sim x_2 \vee \sim x_3).$$

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

$$(a \vee d) \,\&\, (a \vee b \vee c) \,\&\, (c \vee \sim b \vee e).$$

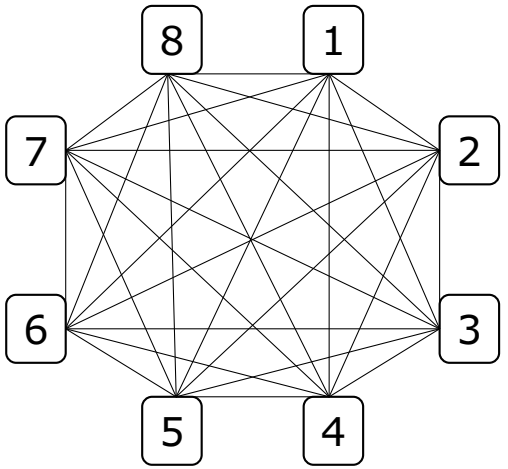
- Suppose the variable *b* is preset to **`true`**.
- Then the clause $(a \vee b \vee c)$ is automatically true, so we skip the clause.
- Also, the literal $\sim b$ is false, so we leave it out from the 3rd clause.
- Final result: $(a \vee d) \,\&\, (c \vee e)$.

New Encoding: Implementation

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

Room for Another Improvement

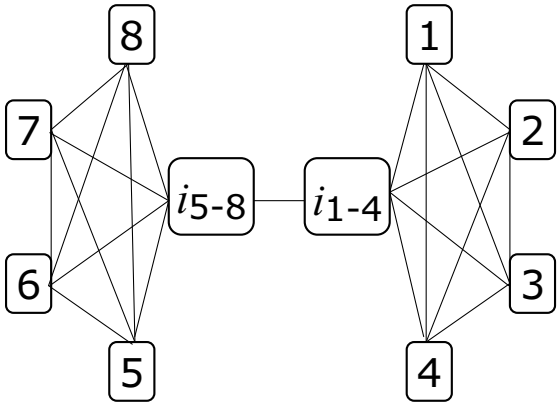
- It still takes $O(n^2)$ clauses to encode an "**at most one**" constraint.
- When one of these vars becomes **`true`**, the SAT solver examines clauses that contain the newly true var.
- This allows the SAT solver to quickly realize that none of the other vars in the "at most" set can be true.
- But requires (n)(n-1)/2 clauses.
- Improvement: Use 'intermediary' nodes (next slide).



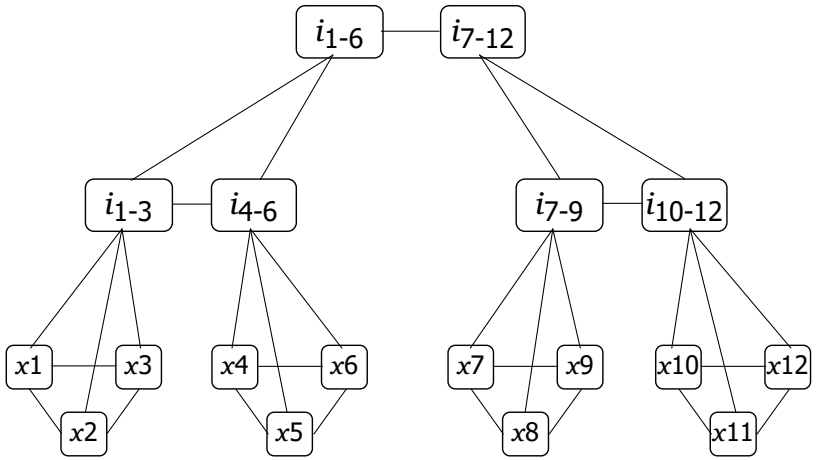
Intermediary Variables

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 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 varibles, 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.

PUZZLE 100x100	NumVars	NumClauses	Sat Time
Var Elim Only	36,415	712,117	1.04 sec
Elim & Intermediary	61,793	428,231	0.76 sec

PUZZLE 144x144	NumVars	NumClauses	Sat Time
Var Elim Only	38,521	596,940	0.91 sec
Elim & Intermediary	58,843	405,487	0.76 sec