Optimized SAT Encoding For Sudoku Puzzles

Will Klieber and Gi-Hwon Kwon

Sept 27, 2007

rev. 2

Changes from v1:

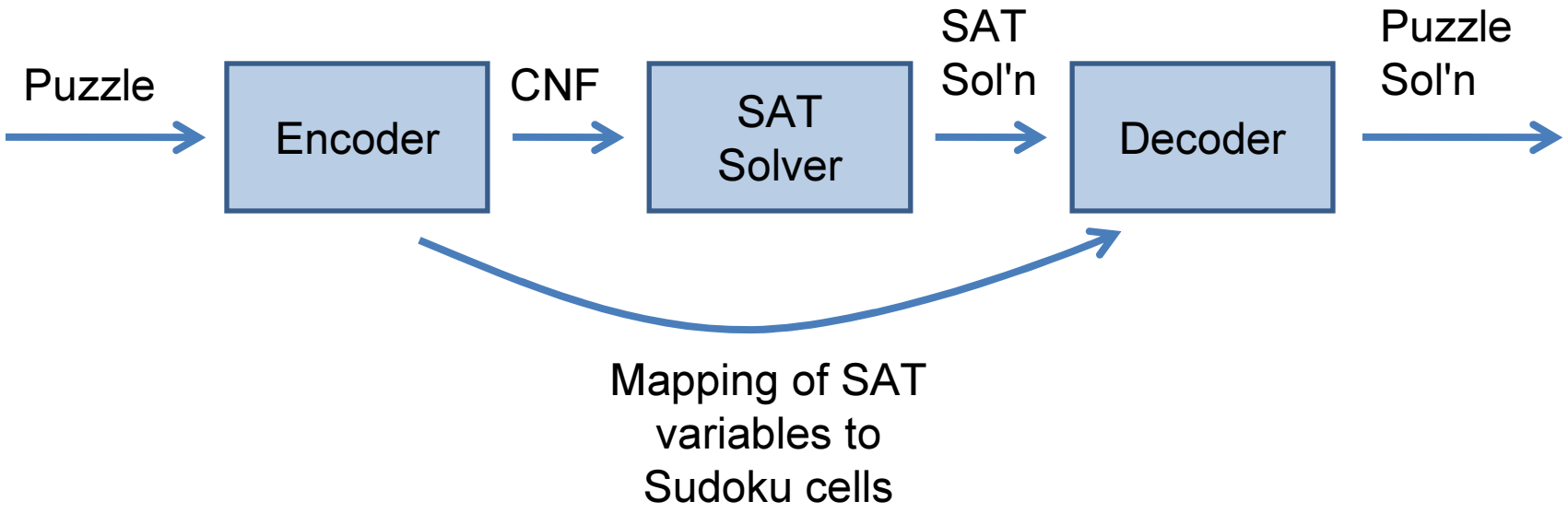
- Added a new slide after “A Better Encoding (1)” to explain how we deal with clauses that would contain skipped variables.
- In the “Implementation” slides, use different codes (`±0xFFFFFF` instead of -1 and -2) to reduce confusion that resulted from another possible interpretation of the old codes.

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		

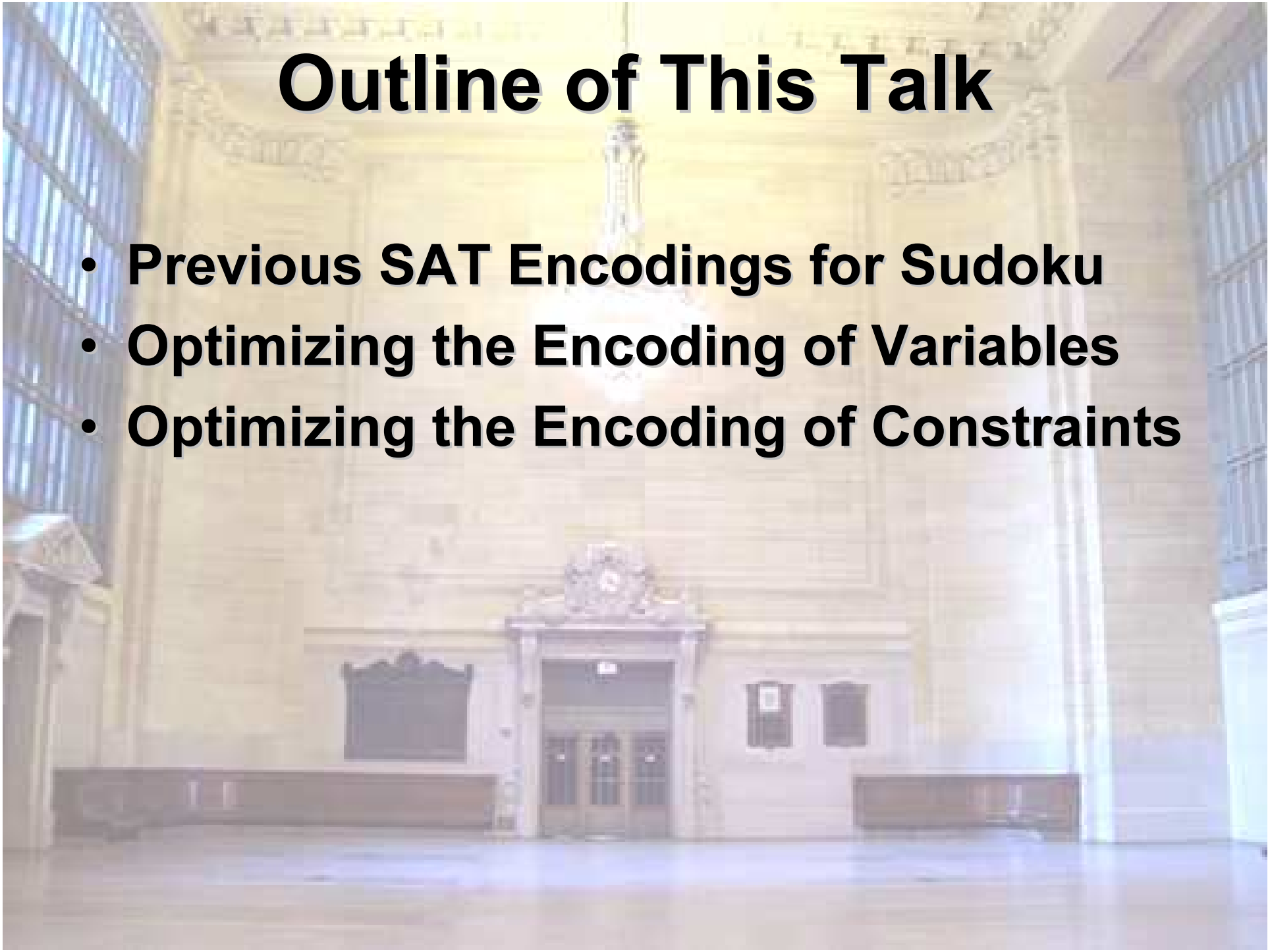
- Played on a $n \times n$ board.
- A single number from 1 to n must be put in each cell; some cells are pre-filled.
- Board is subdivided into $\sqrt{n} \times \sqrt{n}$ blocks.
- Each number must appear exactly once in each row, column, and block.

Puzzle-Solving Process



Outline of This Talk

- Previous SAT Encodings for Sudoku
- Optimizing the Encoding of Variables
- Optimizing the Encoding of Constraints



Previous Encodings (1)

- Lynce & Ouaknine (2006) and Weber (2005) proposed various SAT encodings for Sudoku.
- 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 ***exactly once*** in each row/col/block?

Example of Variable Encoding

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

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

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

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

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

$$\text{All others are false.}$$

- 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 ($x1$, $x2$, $x3$).
- "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":

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

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 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 large n, this is a problem.

Experimental Results

		minimal encoding			efficient encoding			extended encoding		
size	level	vars	clauses	time	vars	clauses	time	vars	clauses	time
9x9	easy	729	8854	0.00	729	11770	0.00	729	12013	0.00
9x9	hard	729	8859	0.00	729	11775	0.00	729	12018	0.00
16x16	easy	4096	92520	0.10	4096	123240	0.09	4096	124008	0.01
16x16	hard	4096	92514	0.46	4096	123234	0.21	4096	124002	0.01
25x25	easy	15625	563417	9.07	15625	750917	17.48	15625	752792	0.07
25x25	hard	15625	563403	time	15625	750903	time	15625	752778	0.21
36x36	easy	46656	2451380	time	46656	3267860	time	46656	3271748	0.50
36x36	hard	46656	2451400	time	46656	3267880	time	46656	3271768	0.67
49x49	easy	117649	8474410	time	117649	11297986	time	117649	11305189	1.47
64x64	easy	262144	24779088	stack	262144	33036624	stack	262144	33048912	stack
81x81	easy	531441	63783464	stack	531441	85041104	stack	531441	85060787	stack

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

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

- 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

- Most SAT solvers use an input format wherein vars are identified by number.
- Keep a 3D array **`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 **`0xFFFFFF`** to true variables.
  - Assign **`-0xFFFFFF`** to false variables.
  - (This assumes less than 16 million vars.)

New Encoding: Implementation (2)

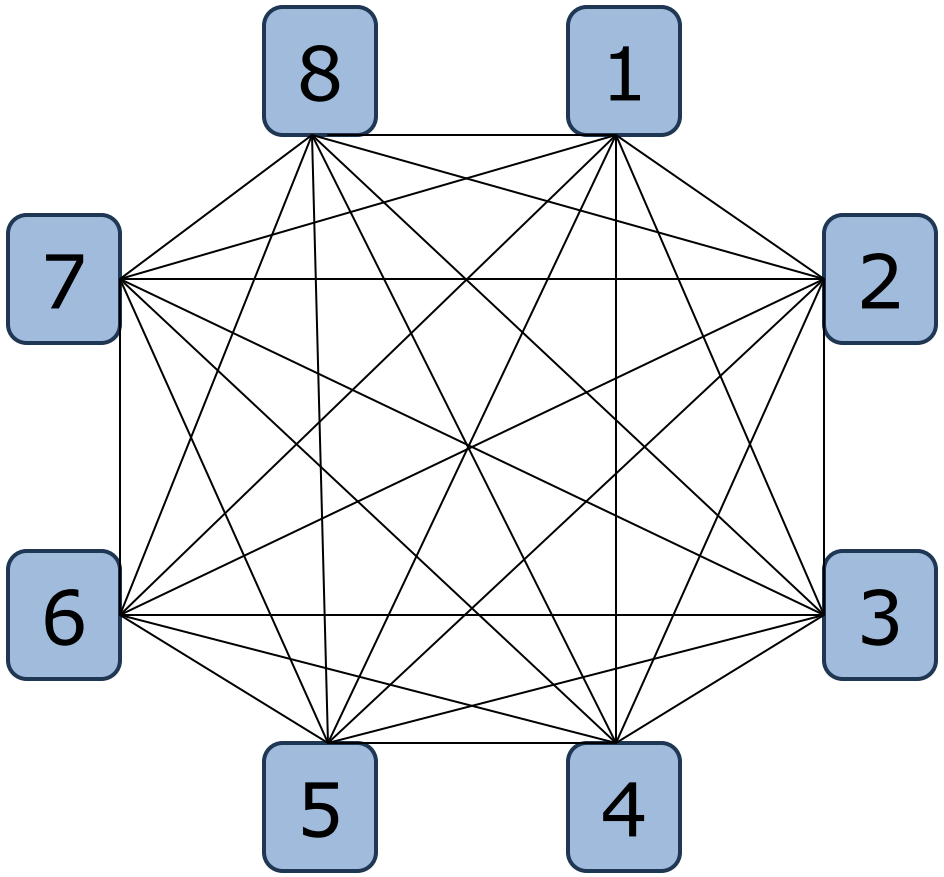
- Initialize **`VarNums[r][c][d]`** to zeros.
- For each prefilled cell, store the appropriate code (**`±0xFFFFFF`**) into the array elems for the cell.
- Also, for every other cell in the same row, column, or block:
  - Assign **`-0xFFFFFF`** (preset_false) to the var that would put the same digit in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

Experimental Results

		extended encoding			proposed encoding		
size	level	vars	clauses	time	vars	clauses	time
9x9	easy	729	12013	0.00	220	1761	0.00
9x9	hard	729	12018	0.00	164	1070	0.00
16x16	easy	4096	124008	0.01	648	5598	0.00
16x16	hard	4096	124002	0.01	797	8552	0.00
25x25	easy	15625	752792	0.07	1762	19657	0.04
25x25	hard	15625	752778	0.21	1990	24137	0.05
36x36	easy	46656	3271748	0.50	4186	57595	0.06
36x36	hard	46656	3271768	0.67	3673	45383	0.08
49x49	easy	117649	11305189	1.47	7642	112444	0.13
64x64	easy	262144	33048912	stack	11440	169772	0.04
81x81	easy	531441	85060787	stack	17793	266025	0.06

Room for Another Improvement

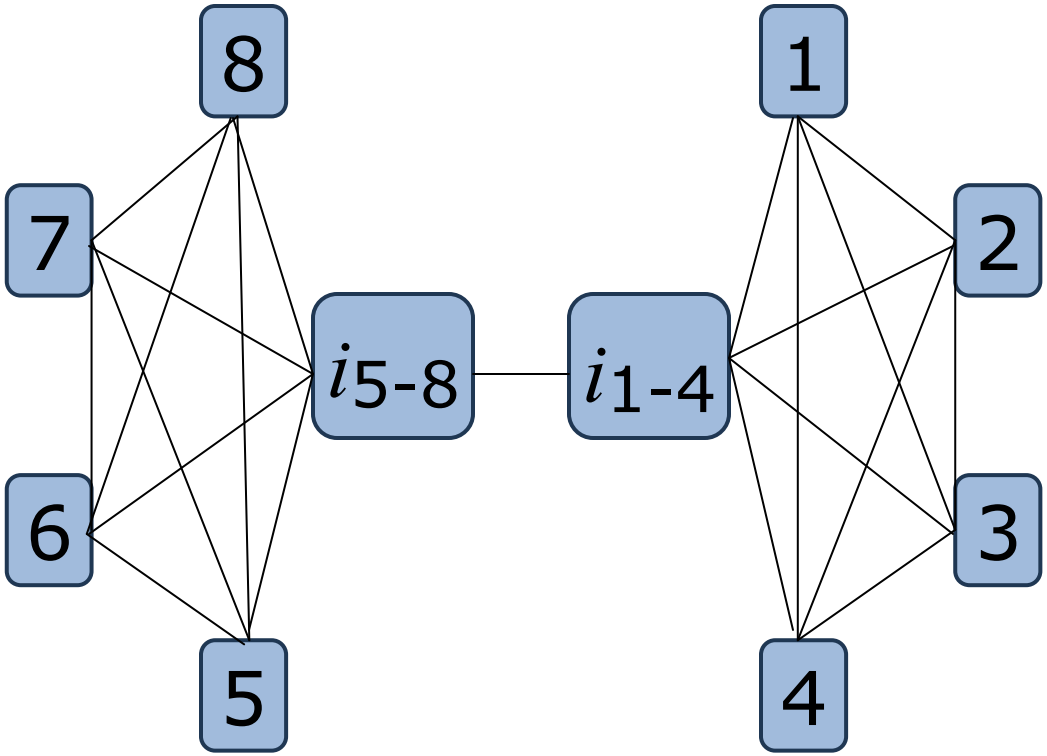
- It still takes $O(n^2)$ clauses to encode an "**at most one**" constraint.
- When one of these vars becomes **`true`**, the SAT solver examines clauses that contains the newly true var.
- This allows the SAT solver to quickly realize that none of the other vars in the "at most" set can be true.
- But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



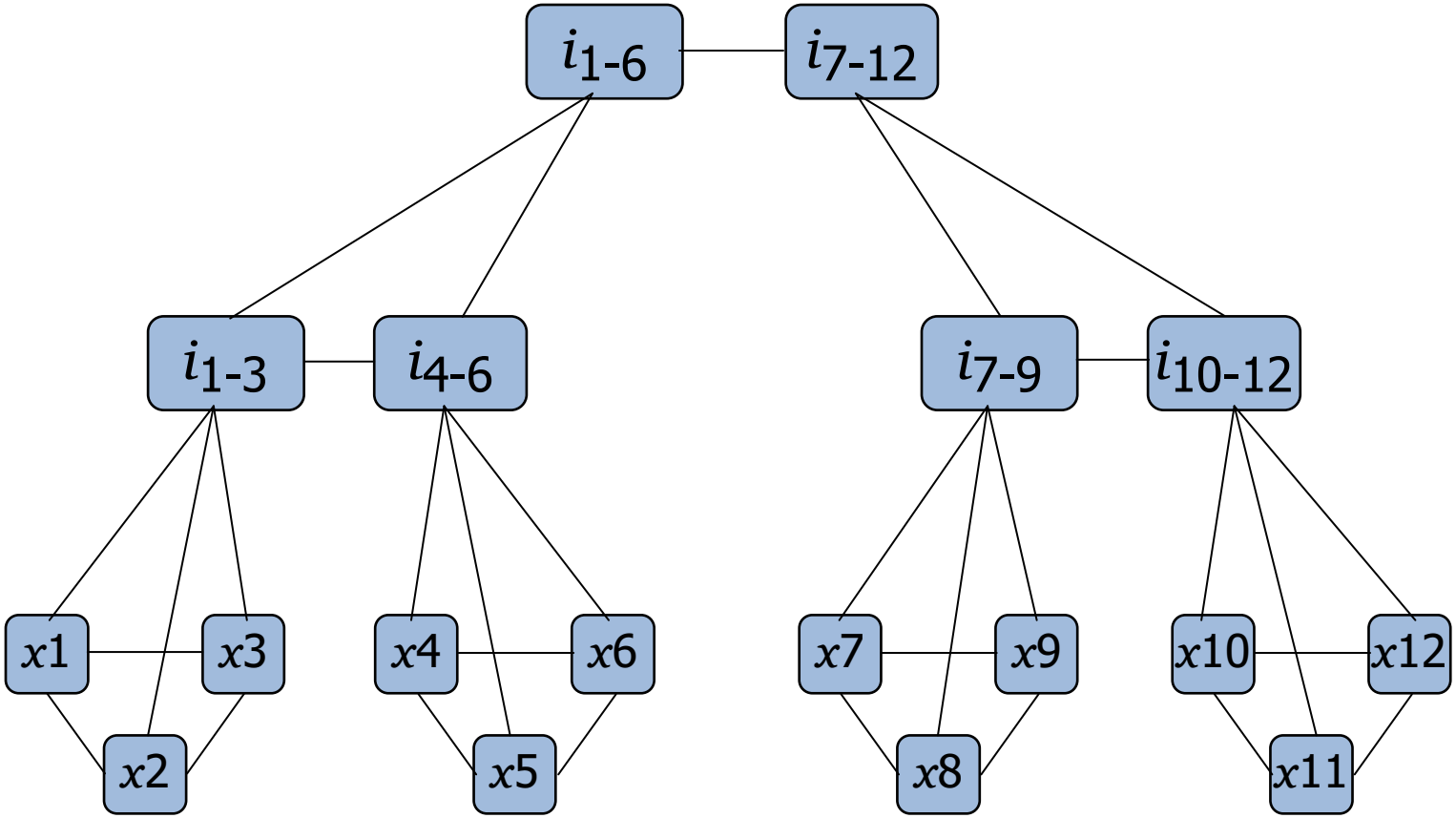
Intermediary Variables

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 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