Optimized CNF Encoding for Sudoku Puzzles *

Gihwon Kwon[1], Himanshu Jain[2]

[1] Department of Computer Science, Kyonggi University
khkwon@kyonggi.ac.kr
[2] Department of Computer Science, Carnegie Mellon University
hjain@cs.cmu.edu

Abstract. A Sudoku puzzle can be regarded as a propositional SAT problem. Various encodings are known for encoding Sudoku as a Conjunctive Normal Form(CNF) formula. Using these encodings for large Sudoku puzzles, however, generates too many clauses, which impede the performance of state-of-the-art SAT solvers. This paper presents an optimized CNF encoding in order to deal with large instances of Sudoku puzzles. We use fixed cells in Sudoku to remove obvious redundancies during the encoding phase. This results in reducing the number of clauses in the CNF encoding and a significant speedup in the SAT solving time.

1 Introduction

Sudoku puzzle can be regarded as a propositional satisfiability (SAT) problem. A given puzzle can be formulated as a SAT formula which is satisfiable if and only if the puzzle has a solution. Subsequently, the formula can be checked for satisfiability using state-of-the-art SAT solvers. A formula in CNF is represented by a set of clauses. The standard input format for most SAT solvers is CNF.

Recently, various encodings - the minimal one [1], the efficient one [2], and the extended one [1] - were proposed to formulate Sudoku into a set of clauses. The number of clauses generated from these encodings for $n \times n$ Sudoku puzzle is $O(n^4)$. These encodings allow us to solve small instances of Sudoku puzzle such as 9×9 efficiently. However, these encodings generates too many clauses for Sudoku puzzles of large size such as 81×81. This in turn makes the satisfiability checking of the generated encodings difficult. This paper presents a new Sudoku encoding that reduces the number of clauses in order to deal with large puzzles. One way of reducing the size of the formula is to work on sophisticated encoding which removes redundant clauses and eases the burden of SAT solving. A fixed cell in Sudoku has a pre-assigned number which is not changed. Using the knowledge

* This research was sponsored by the Gigascale Systems Research Center (GSRC), the Semiconductor Research Corporation (SRC), the Office of Naval Research (ONR), the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796, the Army Research Office (ARO) under contract no. DAAD19-01-1-0485, and the General Motors Collaborative Research Lab at CMU. And this work also was supported by grant No. R01-2005-000-11120-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

of fixed cells during the encoding of a puzzle can lead to the removal of obvious redundancies. One advantage of removing such redundancy is that the size of the CNF file produced can be significantly smaller. On some benchmarks we observed a reduction of 79X in the size of the CNF encoding. This in turn leads to dramatic improvements in the SAT solving time, which we suspect is due to reduced cache and memory overheads. With the proposed encoding, all sizes of the puzzles from 9×9 to 81×81 can be solved with less effort as compared to previous encodings.

2 Previous Encodings

Sudoku is a number placement puzzle where one assigns a number to each cell. Consider the 9×9 puzzle shown in Fig. 1. The aim is to assign a number from 1 to 9 in each cell so that each row, column, and block contains only one instance of each number. A Sudoku puzzle can be regarded as a SAT problem. A

		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		

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

Fig. 1. Sudoku puzzles and its solution

SAT problem is represented using propositional variables, which can be assigned truth values 1(true) or 0(false). In an $n \times n$ Sudoku puzzle each cell can contain a number from 1 to n. Thus, each cell is associated with n propositional variables. Let a 3-tuple (r, c, v) denote a variable which is true if and only if the cell in row r and column c is assigned a number v; i.e. $[r, c] = v$. The resulting set of propositional variables is $V = \{(r, c, v) | 1 \leq r, c, v \leq n\}$. Sudoku rules are represented as a set of clauses to guarantee that each row, column and block contains only one instance of each number from 1 to n. The following set of clauses refers to each cell, each row, each column and each block having *at least* one number from 1 to n (i.e. definedness) and *at most* one number from 1 to n (i.e. uniqueness). We use a subscript d to denote definedness constraints and subscript u to denote the uniqueness constraints. We denote the modulo operation by mod below.

$$Cell_d = \bigwedge_{r=1}^{n} \bigwedge_{c=1}^{n} \bigvee_{v=1}^{n} (r, c, v)$$

$$Cell_u = \bigwedge_{r1}^{n} \bigwedge_{c=1}^{n} \bigwedge_{v_i=1}^{n-1} \bigvee_{v_j=v_i+1}^{n} \neg(r, c, v_i) \vee \neg(r, c, v_j)$$

$$Row_d = \bigwedge_{r=1}^{n} \bigwedge_{v=1}^{n} \bigvee_{c=1}^{n} (r, c, v)$$

$$Row_u = \bigwedge_{r=1}^{n} \bigwedge_{v=1}^{n} \bigwedge_{c_i=1}^{n-1} \bigwedge_{c_j=c_i+1}^{n} \neg(r, c_i, v) \vee \neg(r, c_j, v)$$

$$Col_d = \bigwedge_{c=1}^{n} \bigwedge_{v=1}^{n} \bigvee_{r=1}^{n} (r, c, v)$$

$$Col_u = \bigwedge_{c=1}^{n} \bigwedge_{v=1}^{n} \bigwedge_{r_i=1}^{n-1} \bigwedge_{r_j=r_i+1}^{n} \neg(r_j, c, v) \vee \neg(r_j, c, v)$$

$$Block_d = \bigwedge_{r_{offs}=1}^{\sqrt{n}} \bigwedge_{c_{offs}=1}^{\sqrt{n}} \bigwedge_{v=1}^{n} \bigvee_{r=1}^{\sqrt{n}} \bigvee_{c=1}^{\sqrt{n}} (r_{offs} * \sqrt{n} + r, c_{offs} * \sqrt{n} + c, v)$$

$$Block_u = \bigwedge_{r_{offs}=1}^{\sqrt{n}} \bigwedge_{c_{offs}=1}^{\sqrt{n}} \bigwedge_{v=1}^{n} \bigwedge_{r=1}^{n} \bigwedge_{c=r+1}^{n} \neg(r_{offs} * \sqrt{n} + (r \bmod \sqrt{n}), c_{offs} * \sqrt{n} + (r \bmod \sqrt{n}), v) \vee \neg(r_{offs}\sqrt{n} + (r \bmod \sqrt{n}), c_{offs} * \sqrt{n}(c \bmod \sqrt{n}), v)$$

In addition to these sets, one more set of clauses is needed to represent fixed cells. A fixed cell contains a pre-assigned number; for example, if [1,3] = 6 in the input puzzle, then the cell [1,3] is a fixed cell containing number 6. Let $V^+ = \{(r, c, v) \in V | [r, c]$ is a fixed cell and has a pre-assigned value $v\}$ be a set of variables representing fixed cells and k be the number of such fixed cells. The fixed cells are naturally represented as unit clauses in the encoding of the puzzle $Assigned = \bigwedge_{i=1}^{k}(r, c, v)$, where $(r, c, v) \in V^+$

A puzzle is encoded as a set of clauses which is satisfiable if and only if the puzzle has a solution. Recently, three encodings were proposed to formulate Sudoku into a set of clauses:

$$\phi_{minimal} = Cell_d \wedge Row_u \wedge Col_u \wedge Block_u \wedge Assigned$$
$$\phi_{efficient} = Cell_d \wedge Cell_u \wedge Row_u \wedge Col_u \wedge Block_u \wedge Assigned$$
$$\phi_{extended} = Cell_d \wedge Cell_u \wedge Row_d \wedge Row_u \wedge Col_d \wedge Col_u \wedge Block_d \wedge Block_u \wedge Assigned$$

We implemented these three encodings in Java and applied them to Sudoku puzzles of different size. Eleven puzzles were randomly selected from an on-line database [3], which has a huge set of Sudoku puzzles with diverse difficulties (from easy to hard) and with different sizes (from 9×9 to 81×81). Table 1 shows our experimental results using zChaff version 2004.5.13 [4]. The experiments were performed on a 1.10GHz Intel Pentium M with 1.25 GB of memory running Windows XP. It is easy to see that the extended encoding which adds redundant clauses to the minimal encoding shows the best performance among the three encodings. Using these encodings for large Sudoku such as 81×81, however, generates huge CNF files which lead to stack overflow error during SAT solving.

		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

Table 1. Test results of three encodings
(time means timeout; and stack denotes a stack overflow error)

3 Proposed Encoding

The runtime of a SAT solver is highly related to the input formula size, especially, when the input formula size is in gigabytes. An optimized encoding removes the obvious redundancies when encoding a given problem in CNF. Using an optimized CNF encoding can speed up SAT solving significantly. A fixed cell in

Sudoku has a pre-assigned number which is not changed. It is easy to see that a fixed cell implies obvious redundancies in the encodings presented above. For instance, many facts can be derived from the single fixed cell [1,3] = 6 in Fig. 1:

- Variable(1,3,6) representing the call is true;
- Other variables {(1,3,1),(1,3,2),(1,3,3),(1,3,4),(1,3,5),(1,3,7),(1,3,8),(1,3,9)} representing the cell are false;
- Clause {(1,3,1),(1,3,2),(1,3,3),(1,3,4),(1,3,5),(1,3,6),(1,3,7),(1,3,8),(1,3,9)} representing the cell definedness is true;
- Clauses {{¬(1,3,1),¬(1,3,2)},{¬(1,3,1),¬(1,3,3)},{¬(1,3,1),¬(1,3,4)},...} representing the cell uniqueness are true; etc

These variables and clauses can be derived from a Sudoku compilation phase and can be eliminated in advance before executing a SAT solver. This reduction can be formalized as follows. The set V of variables can be partitioned into three subsets

$$V = V^+ \cup V^- \cup V^{'}$$

As defined in the previous section V^+ is the set of true variables since each variable in this set represents a fixed cell. V^- is the set of false variables and defined using V^+ as follows:

$$V^- = \{(r,c,v) \in V \mid \exists_{(r^{'},c^{'},v^{'}) \in V^+} \cdot sameCell(r,c,v,r^{'},c^{'},v^{'}) \vee sameRow(r,c,v,r^{'},c^{'},v^{'}) \vee sameRow(r,c,v,r^{'},c^{'},v^{'}) \vee sameBlock(r,c,v,r^{'},c^{'},v^{'})\}$$

Since an assigned number at a fixed cell is never changed, all variables representing other numbers associated with a fixed cell can be removed. In addition, variables whose value is duplicated in the same row or column or block are eliminated because duplicated numbers are not allowed within the same region in Sudoku. Related predicates are defined as follows

$$sameCell(r,c,v,r^{'},c^{'},v^{'}) = (r = r^{'}) \wedge (c = c^{'}) \wedge (v \neq v^{'})$$
$$sameRow(r,c,v,r^{'},c^{'},v^{'}) = (r = r^{'}) \wedge (c \neq c^{'}) \wedge (v = v^{'})$$
$$sameCol(r,c,v,r^{'},c^{'},v^{'}) = (r \neq r^{'}) \wedge (c = c^{'}) \wedge (v = v^{'})$$
$$sameBlock(r,c,v,r^{'},c^{'},v^{'}) = (r \neq r^{'}) \wedge (c \neq c^{'}) \wedge (v = v^{'})$$
$$\exists_{lucRow,lucCol} \cdot (lucRow \leq r, r^{'} \leq lucRow + \sqrt{n}) \wedge (lucCol \leq c, c^{'} \leq lucCol + \sqrt{n})$$

, where a cell $[lucRow, lucCol]$ is the starting cell in a block, located at the left-upper corner, for a given cell [r,c]. And two variables are defined in the Java-like syntax

$$lucRow = (r \leq \sqrt{n})?1 : ((r-1)/\sqrt{n}) * \sqrt{n} + 1$$
$$lucCol = (c \leq \sqrt{n})?1 : ((c-1)/\sqrt{n}) * \sqrt{n} + 1$$

And $V^{'} = V - (V^+ \cup V^-)$ is a set of unknown variables; i.e. its value is not determined at the encoding phase. A SAT problem is to find a satisfying assignment for a set of variables occurring in a formula. Having a partial satisfying assignment for a set V^+ of true variables and a set V^- of false ones, variables in these sets and related clauses and literals can be reduced without affecting a SAT problem. Reduction operators are defined

$$\phi \Downarrow V^+ = \{C \in \phi \mid \neg\exists_{L \in C} \cdot (L = x \wedge x \in V^+)\}$$

$$\phi \Downarrow V^- = \{C \in \phi \mid \neg\exists_{L \in C} \cdot (L = \neg x \wedge x \in V^-)\}$$
$$C \downarrow V^+ = \{L \in C \mid \neg(L = \neg x \Rightarrow x \in V^+\}$$
$$C \downarrow V^- = \{L \in C \mid \neg(L = x \Rightarrow x \in V^-\}$$

The operator $\Downarrow$ deletes clauses if they have a true value and the operator $\downarrow$ eliminates literals if they have a false value. In order to apply the operator $\downarrow$ to a formula, it can be overloaded such as $\phi \downarrow V^- = \{C \downarrow V^-\}$ for all $C \in \phi$. Let $\phi^{'}$ be a reduced formula from $\phi_{extended}$ which is the CNF formula for the extended encoding in the previous section

$$\phi^{'} = Assigned \Downarrow V^+ \cup (Cell_u \cup Row_u \cup Col_u \cup Block_u) \Downarrow V^- \cup (Cell_d \cup Row_d \cup Col_d \cup Block_d) \downarrow V^-$$

Then a satisfiability equivalence relation holds between $\phi_{extended}$ and $\phi^{'}$ with respect to a satisfying assignment α; i.e, satisfies $\phi_{extended}$ if and only if α satisfies $\phi^{'}$. Thus we can solve the Sudoku puzzle using the reduced formula $\phi^{'}$. And the variable set to be considered restricts to $V^{'}$ since $\phi^{'}$ is represented with variables in $V^{'}$. The same puzzles as used in previous section were used to evaluate the new technique. Compared to the extended encoding, the number of variables and clauses is reduced on average by 12 times and 79 times. For instance, the proposed encoding generates 17,793 variables and 266,025 clauses for the largest puzzle (81×81) in the database; however, the extended one generates 531,441 variables and 85,060,787 clauses. Since the number of variables and clauses are reduced, zChaff solves the puzzle in 0.06 second. In our encoding, the number of variables and clauses are strongly dependent on the number k of the fixed cells shown in Table. 2. The larger the k is, the smaller is the number of variables and clauses present in the CNF generated by our encoding.

		extended encoding			proposed encoding			analysis of pre-assigned cells			
size	level	vars	clauses	time	vars	clauses	time	k	ratio	vars↓	claus↓
9x9	easy	729	12013	0.00	220	1761	0.00	26	32	331	682
9x9	hard	729	12018	0.00	164	1070	0.00	30	37	445	1123
16x16	easy	4096	124008	0.01	648	5598	0.00	104	41	632	2215
16x16	hard	4096	124002	0.01	797	8552	0.00	98	38	514	1450
25x25	easy	15625	752792	0.07	1762	19657	0.04	292	47	887	3830
25x25	hard	15625	752778	0.21	1990	24137	0.05	278	45	785	3119
36x36	easy	46656	3271748	0.50	4186	57595	0.06	644	50	1115	5681
36x36	hard	46656	3271768	0.67	3673	45383	0.08	664	51	1270	7209
49x49	easy	117649	11305189	1.47	7642	112444	0.13	1282	53	1540	10054
64x64	easy	262144	33048912	stack	11440	169772	0.04	2384	58	2291	19467
81x81	easy	531441	85060787	stack	17793	266025	0.06	3983	61	2987	31957

Table 2. Result of proposed encoding and effect analysis of pre-assigned cells

Able to solve 81x81 puzzles in 2sec at a second while others overflow on memory.

References

1. I. Lynce and J. Ouaknine, Sudoku as a SAT problem, in The Proceedings of AIMATH'06. 2006.
2. T. Webber, A SAT-based Sudoku Solver, in The Proceedings of LPAR'05, 2005.
3. http://www.menneske.no/sudoku
4. M.W.Moskewicz, et.al., Chaff: Engineering an efficient SAT solver, in the Proceedings of DAC'01, 2001.