SUDOKUSAT—A Tool for Analyzing Difficult Sudoku Puzzles

Martin Henz and Hoang-Minh Truong
National University of Singapore
School of Computing
Computing 1, Law Link, Singapore 117590, Singapore
henz@comp.nus.edu.sg, g0605790@nus.edu.sg

Abstract

Sudoku puzzles enjoy world-wide popularity, and a large community of puzzlers is hoping for ever more difficult puzzles. A crucial step for generating difficult Sudoku puzzles is the fast assessment of the difficulty of a puzzle. In a study in 2006, it has been shown that SAT solving provides a way to efficiently differentiate between Sudoku puzzles according to their difficulty, by analyzing which resolution technique solves a given puzzle. This paper shows that one of these techniques—unit resolution with failed literal propagation—does not solve a recently published Sudoku puzzle called AI Escargot, claimed to be the world's most difficult. The technique is also unable to solve any of a list of difficult puzzles published after AI Escargot, whereas it solves all previously studied Sudoku puzzles. We show that the technique can serve as an efficient and reliable computational method for distinguishing the most difficult Sudoku puzzles. As a proof-of-concept for an efficient difficulty checker, we present the tool SUDOKUSAT *that categorizes Sudoku puzzles with respect to the resolution technique required for solving them.*

1. Sudoku

Sudoku puzzles have fascinated puzzle solvers since their invention in 1979. A Sudoku puzzle is a 9×9 grid of cells, which is composed of nine 3×3 non-overlapping boxes of cells. The objective is to fill the grid with digits from 1 to 9 so that each row, column and box contains any digit at most once. Some cells are already filled with digits; these cells are called *hints*. To be a proper Sudoku puzzle, the grid must admit a unique way to fill the remaining cells to meet the objective (Uniqueness Property). Figure 1 shows a Sudoku puzzle that can be solved within a few minutes by an experienced puzzler.

Whereas precursors of Sudoku—based on magic squares and Latin squares—were known since the late 19th century, modern Sudoku puzzles were first published anonymously by Dell Magazines in 1979, and are widely attributed to Howard Garns. The puzzles became popular in Japan starting in the mid 1980s and world-wide in 2005, giving rise to a large international community of puzzlers who enjoy solving puzzles of various levels of difficulty.

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

Figure 1. A Typical Sudoku Puzzle

The Sudoku community spends much effort to generate new and ever more difficult puzzles. A milestone in this quest was reached by Arto Inkala in 2006.

2. AI Escargot

In November 2006, Arto Inkala published a Sudoku puzzle that he called "AI Escargot"[1], shown in Figure 2, which he

[1] The letters "A" and "I" represent Dr. Inkala's initials.

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

Figure 2. AI Escargot

claimed to be "the most difficult Sudoku puzzle known so far". In his recently published book [7], Inkala notes:

> In addition to my own evaluation method, I gave AI Escargot to others for evaluation in October 2006. It was soon found to be the most difficult Sudoku puzzle known according to at least three established evaluation methods. In the case of two methods, in fact, AI was the first puzzle that they were unable to solve. These methods were RMS (Ravel Minimal Step) and Sudoku Explainer 1.1. The third and fourth method was Gfsr and Suexrat9 method. The fifth method I used to evaluate AI Escargot also found it to be the most difficult, but that method is my own.
>
> What happened next was what I had anticipated: AI Escargot motivated other makers of difficult Sudoku puzzles to continue the work. It continues to be the most difficult Sudoku puzzle, when the results of all evaluation methods are combined, but some puzzles can now be rated more difficult by a particular method.

The quest for difficult Sudoku puzzles continued since publication of AI Escargot, and a web site lists the most difficult puzzles discovered since the publication of AI Escargot [12]. A common approach for generating difficult Sudoku puzzles is generate-and-test: generate large numbers of promising candidate puzzles[2], and test each of these candidates for the Uniqueness Property and difficulty. In order to *efficiently* and *effectively* search for difficult puzzles, the rating has to be both fast and correspond to the human intuition of Sudoku difficulty. In the quote above, Inkala mentions five techniques of rating:

Ravel Minimal Step (RMS). The minimal number of "difficult steps" needed to solve the puzzle. RMS is not able to rate AI Escargot. Even if appropriate "difficult steps" would be added to the rating scheme, a time-consuming search-based technique is required for RMS, since the choice of steps and their order greatly affects the number of steps needed.

Sudoku Explainer. This Java software by Nicolas Juillerat uses a complex algorithm to assess the difficulty of a puzzle. Apparently, the more recent version Sudoku Explainer 1.2 is able to solve AI Escargot, but assessing its difficulty reportedly takes several minutes.

Gfsr. At the time of submission, the authors were not able to find details on this evaluation method.

Suexrat9. Suexrat9 returns the average number of depth-first search steps, where only simple reasoning steps are carried out at each node. This method is slowed down by the need for multiple runs of a depth-first solver, and the result does not correspond well to the experience of human solvers, who typically apply techniques radidally different from depth-first search.

Inkala's method. At the time of submission, the authors were not able to obtain details on this method.

It is not clear whether any of these techniques are able to serve as an effective testing step in generate-and-test. In this paper, we propose to use a SAT-based technique called failed literal propagation (FLP) as effective test. In order to argue the case, we briefly discuss the computational techniques that have been applied to Sudoku puzzles in the next section. Section 4 zooms into SAT-based solving techniques. Section 5 discusses how a SAT-based solver can be augmented with a Sudoku-specific reasoning method to optimize the test for the Uniqueness Property. Section 6 presents SUDOKUSAT, a SAT-based tool for analyzing the difficulty of Sudoku puzzles. We have used SUDOKUSAT for assessing the difficulty of AI Escargot and other difficult puzzles, and Section 7 establishes the first known Sudoku puzzles that cannot be solved with FLP, which gives rise to that hope—expressed in Section 8—that SAT-solving may prove to be a valuable tool in the quest for the most difficult Sudoku puzzles.

[2] Enumerating possible Sudoku grids is interesting task in itself; for details, see [4].

3. Sudoku in AI

The problem of finding solutions of $n^2 \times n^2$ Sudoku puzzles is NP-hard [17]. All known 9×9 puzzles can be quickly solved using a variety of techniques, ranging from general search algorithms such as Dancing Links [9], over constraint programming techniques [14, 6] and SAT-based techniques (see next section), to a host of Sudoku-specific algorithms such as Sudoku Explainer [8] and Sudoku Assistant/Solver [5].

Researchers in Artificial Intelligence have used constraint programming and SAT solving to assess the difficulty of Sudoku puzzles. Simonis shows that finite-domain constraint programming (CP) can be employed to assess the difficulty of Sudoku puzzles [14]. CP models use finite domain variables for each cell representing the digit placed in the cell, as well as finite domain variables that indicate in which cell of a given row/column/box a particular digit is placed. Both kinds of variables are connected with all-different constraints. Channeling constraints connect variables of different kinds. Simonis presents several constraints that exploit the interaction between rows, columns and boxes. Of particular value for solving Sudoku puzzles is a technique called *shaving* [15], which speculatively assigns a value to a variable. When propagation detects an inconsistency as a result of the assignment, the value is removed from the variable domain. Simonis proceeds to compare the published difficulty of a range of puzzles with the kinds of propagation techniques required to solve them without search. Since he did not have access to the difficult instances published after 2006, the effectiveness of CP to assess the difficulty of Sudoku puzzles remains a subject of future work.

Weber presented the first SAT-based tool for solving Sudoku [16], by translating a puzzle using the theorem prover Isabelle/HOL into clause form, solving the resulting SAT problem using the SAT solver zChaff [11], and reporting that the encoding and solving is done "within milliseconds". Lynce and Ouaknine apply various SAT-based resolution techniques to Sudoku [10]. They confirm that the SAT-equivalent of shaving called *failed literal propagation* provides a very efficient and effective method for solving the puzzles. Similar to Simonis, Lynce and Ouaknine did not have access to the difficult puzzles published after 2006, and thus, the significance of the different resolution techniques for assessing Sudoku difficulty remains unclear. Comparing constraint programming and SAT solving on Sudoku, we conclude that SAT solving provides simpler and equally powerful solving techniques, and therefore focus on SAT solving for this work.

4. SAT-based Sudoku Solving

Satisfiability is the problem of determining if the variables in a given Boolean formula can be assigned to truth values such that the formula evaluates to TRUE. In SAT problems, the formula is a conjunction of disjunctions of literals, where a literal is either a variable or the negation of a variable. The disjunctions are called *clauses*, and the formula a *clause set*. If the clause set is satisfiable, complete SAT solvers provide an assignment of the variables which satisfies it. When we encode a given Sudoku puzzle as a SAT problem, and give it to a complete SAT solver, the solver returns the solution to the puzzle, encoded in a variable assignment.

We use the following *extended encoding* of Sudoku, given by Lynce and Ouaknine [10]. With each cell of the grid, this encoding associates 9 variables, one for each possible digit. The variable s_{xyz} is assigned to TRUE if and only if the cell in row x and column y contains the digit z. The following conjunctions of clauses combined make up the extended encoding (for a detailed explanation, see [10]):

- $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$
- $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$
- $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$
- $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$
- $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$
- $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigwedge_{z=1}^{8} \bigwedge_{i=z+1}^{9} (\neg s_{xyz} \vee \neg s_{xyi})$
- $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{x=1}^{9} s_{xyz}$
- $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{y=1}^{9} s_{xyz}$
- $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}$

Complete solving techniques for SAT problems are based on simplification and search. A basic simplification technique is the *unit clause rule* [3]. The rule eliminates a unit clause of the form $\omega_i = (l_j)$, where l_j is a literal, by removing all clauses in which l_j appears, and removing $\neg l_j$ from all clauses containing it. *Unit resolution* applies the unit clause rule exhaustively until the clause set contains no unit clause. The most common complete solving technique for SAT problems is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [2], which combines tree search, where each branching assigns a selected literal to a truth value,

with unit resolution. Solving Sudoku puzzles is done by humans using complex kinds of reasoning such as the "Swordfish" pattern [5], as opposed to tree search. Thus, in order to use SAT solving to assess the difficulty of a puzzle, Lynce and Ouaknine examine—in addition to unit resolution—the following three propagation-based techniques. .

Failed literal propagation (FLP). The so-called *failed literal rule* is applied to a clause set f and a literal l, by creating a new clause set $f' = f \wedge (l)$. If unit resolution applied to f' yields a contradiction (an empty clause), f is simplified by adding $\neg l$ to it and performing unit resolution. *Failed literal propagation* repeatedly applies this failed literal rule to all literals, until no more simplification is achieved.

Binary failed literal propagation (BFLP). This technique extends failed literal propagation by considering pairs of literals l_1 and l_2. If unit resolution after adding the clauses (l_1) and (l_2) to the clause set leads to a contradiction, the clause $(\neg l_1 \vee \neg l_2)$ can be added to the clause set.

Hyper-binary resolution (HBR). This technique infers from clause subsets of the form

$$(\neg l_1 \vee x_i) \wedge (\neg l_2 \vee x_i) \wedge \cdots \wedge (\neg l_k \vee x_i) \wedge (l_1 \vee l_2 \vee \cdots \vee l_k \vee x_j)$$

the clause $(x_i \vee x_j)$. Similar to the previous two techniques, unit resolution is peformed after each application of the rule, and the rule is applied exhaustively.

5. Grid Analysis

During the search for Sudoku puzzles using generate-and-test, millions of puzzles need to be assessed, whether they admit a unique solution. The only known technique for this problem is to find the set of all solutions, and check whether that set is a singleton. The DPLL algorithm family is an efficient technique for finding all solutions of Sudoku puzzles.

We are proposing to combine DPLL with the following Sudoku-specific propagation rules, which are given by a technique called *Grid Analysis* (GA), employed by human puzzlers and Sudoku software such as Sudoku Assistant/Solver.

- When a candidate digit is possible in a certain set of cells that form the intersection of a set of n rows and n columns, but are not possible elsewhere in that same set of rows, then they are also not possible elsewhere in that same set of columns.
- Analogously, when a candidate digit is possible in a certain set of cells that form the intersection of a set of n rows and n columns, but are not possible elsewhere in that same set of columns, then they are also not possible elsewhere in that same set of rows.

At a particular node in the DPLL search tree, the first rule can be implemented by iterating through all possible digits k and numbers n, where $1 \leq 2 \leq 8$. For each value of k and n, we check whether there are n rows, in which there arel less than n candidate cells for k, whether all candidate cells belong to n columns, and whether each of these columns contains at least one candidate cell. If these conditions are met, we delete k from all cells in the n columns that do not belong to any of the n rows, by assigning the corresponding variable s_{ijk} to TRUE. The second rule can be implemented similarly. The technique can be further improved by switching the roles of value and row/column index, as explained in [14].

6. SUDOKUSAT

In order to experiment with FLP, BFLP, HBR and GA for solving Sudoku puzzles, we implemented a SAT solver, in which DPLL, FLP, BFLP and HBR can be used as alternative solving techniques. For each technique, the user can choose to add GA. In the case of FLP, BFLP and HBR, GA is performed after propagation. Whenever GA obtains a simplification, propagation is repeated, leading to a fixpoint with respect to a combination of FLP/BFLP/HBR and GA. Similarly, during DPLL, GA is combined with unit resolution to achieve a combined fixpoint. Figure 3 shows the user interface of SUDOKUSAT after entering the hints of a problem and setting the solver options.

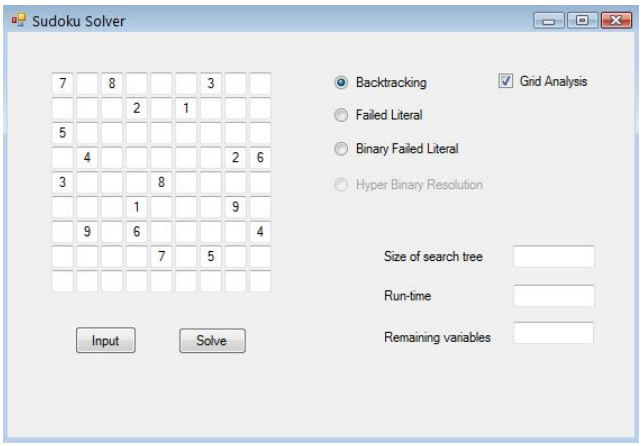

Figure 3. Interface of SUDOKUSAT

Figure 4 shows the user interface of SUDOKUSAT after solving the puzzle, displaying statistics on the size of the search tree (in case DPLL was chosen), the overall run-

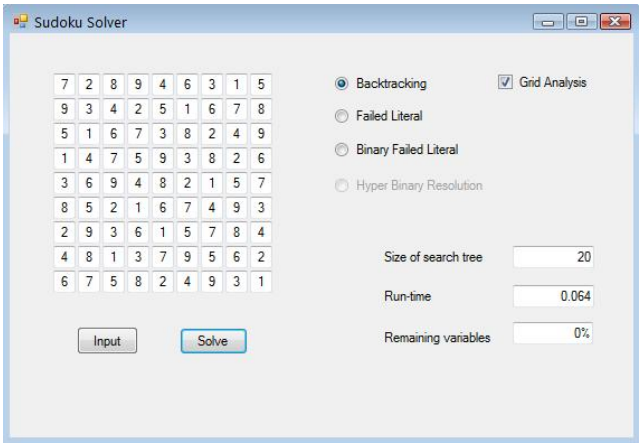

Figure 4. Display of solved puzzle and statistics on the search tree size, runtime and remaining propositional variables

000001020300040500000600007002000001080090030400000800500002000090030400006700000	1.062
003000009400000020080600100200004000090800007005030000000900800000005030070010006	0.843
100300000020090400005007000800000100040000020007060003000400800000020090006005007	0.578
100007000020030500004900000008006001090000020700000300000001008030050060000400700	0.937
003400080000009200000060001007010000060002000500800040010000900800000030004500007	0.766
000001020300040500000600007002000006050030080400000909000020000800504000017000000	0.313
003900000040070001600002000800000002070050030009000400200001008000040050000600900	1.594
100007000020080000004300500005400001080000020900000300000005007000020060003100900	1.39
002900000030070000500004100008000002090000030600500400000003008000060070100200500	0.344
000001020300040500000600007001000006040080090500000300800002000050090400006700000	0.391
900004000050080070001200000002600009030000040700000500000009002080050030000700600	1.218
003400000050009200700060000200000700090000010008005004000300008010002900000070060	0.281
000001020300040500000600001001000007050030040800000900400002000090050800006700000	0.422
400009000030010020006700000001000004050200070800000600000004008070030010000500900	0.625
100050080000009003000200400004000900030000007800600050002800060500010000070004000	1.078
100050000006009000802000040400300080070000609000001000308000020000040500000107000	0.656
003200000400900006000080102000000030100060400070005000000010020900400600005007000	0.796
020400000006080100700003000000060300002000050900070403000008000010000900405000002	0.594
002600000030080000500009100006000002080000030700001400000004005010020080000700900	1.188
500000009020100070008000300040702000000050000000006010003000800060004020900000005	0.813

Figure 5. The 20 most difficult Sudoku puzzles [12] and the runtimes in seconds required for (unsuccessfully) attempting to solve them using FLP. The puzzles are given by concatenating the rows of cells containing their hints, where a cell with no hint is represented by 0.

time, and the percentage of remaining variables (in case FLP, BFLP or HBR was chosen).

Note that Holst reports on a tool for solving Sudoku puzzles, based on constraint logic programming [6].

7. Results

The experiments reported by Lynce and Ouaknine [10] use a set of 24,260 very difficult Sudoku puzzles collected by Gordon Royle [13]. These puzzles are *minimal* in the sense that they have the smallest known number of hints, while still admitting a unique solution. The puzzles contain 17 hints, as there are no known Sudoku puzzles with only 16 hints. A common assumption at the time of publication of previous studies on using AI techniques for Sudoku [10, 16, 14, 6] was that the most difficult Sudoku puzzles would be minimal. Lynce and Ouaknine show that unit resolution alone can only solve about half of their studied puzzles, whereas the remaining three techniques solve them all.

Inkala argues that the number of hints is not indicative of the difficulty of the puzzle [7], and presents AI Escargot, a puzzle with 23 hints, which was acknowledged as the most difficult Sudoku puzzle in November 2006 [1]. The 20 most difficult Sudoku puzzles known at the time of this writing all have 21 hints [12].

We have applied the three techniques given in Section 4 to these 20 instances and AI Escargot and obtained the following results:

- Not surprisingly, UR fails to solve any of the puzzles.
- None of the puzzles can be solved using FLP.
- All puzzles are solved using BFLP, HBR, and Grid Analysis.

In our view, the most significant of these results is the second. Lynce and Ouaknine [10] report that FLP solves all puzzles listed by Royle [13], a fact that we verified with SUDOKUSAT. However, FLP is not able to solve any of new difficult puzzles. The runtime for attempting to solve them varies between 0.28 and 1.59 seconds, with an average of 0.79, using a PC with a Pentium 4 processor running at 3GHz using Windows Vista. Figure 5 shows the detailed runtimes for the 20 most difficult puzzles.

For AI Escargot, we obtained the runtime results given in Table 6. Note that the most efficient method for solving AI Escargot appears to be DPLL, combined with the Grid Analysis technique presented in Section 5.

The runtime results indicate that our implementation of FLP in SUDOKUSAT provides an efficient and reliable test for identifying the most difficult Sudoku puzzles, and that DPLL together with Grid Analysis is a promising technique for quickly checking that a candidate puzzle has a unique solution.

Technique	Runtime (s)	Size of search tree
DPLL	0.13	50 nodes
DPLL + GA	0.076	20 nodes
BFLP	5.200	1 node (no search needed)
BFLP + GA	4.672	1 node (no search needed)
HBR	48.000	1 node (no search needed)

Figure 6. Runtimes for solving AI Escargot using SUDOKUSAT using successful techniques (Pentium 4, 3GHz, Windows Vista). Neither FLP nor FLP + GA are able to solve AI Escargot, and in both cases, 76% of the propositional variables remain.

8. Conclusions

We have argued in this paper that SAT solvers provide effective measures of the difficulty of Sudoku puzzles by demonstrating that the recently discovered puzzle AI Escargot—claimed to be the most difficult Sudoku puzzle in the world—is the first known Sudoku puzzle that cannot be solved by unit resolution combined with failed literal propagation (FLP). We also show that 20 puzzles discovered after AI Escargot cannot be solved using FLP, whereas the technique is able to solve all previously studied Sudoku puzzles. These results lead us to suggest the following method for generating difficult puzzles:

1. Generate a candidate puzzle p using known enumeration techniques (e.g. [4]).
2. Test if p has a solution and satisfies the Uniqueness Property. For this step, the runtime results obtained for AI Escargot suggest a combination of DPLL with grid analysis. If the test fails, go to 1.
3. Test if FLP solves p. If not, save p as a difficult puzzle.
4. Go to 1.

As a prototype implementation of Steps 2 and 3, we presented SUDOKUSAT, a Sudoku solver that allows a puzzle designer to assess the difficulty of candidate Sudoku puzzles by applying different resolution techniques in the solving process. We hope that this tool, or an adaptation of it, allows Sudoku enthusiasts to generate further difficult Sudoku instances for the benefit of Sudoku puzzlers around the world.

References

[1] AFP. Mathematician claims to have penned hardest Sudoku. *USA Today*, Nov. 2006.

[2] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the Association for Computing Machinery*, 5(7):394–397, 1962.

[3] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, July 1960.

[4] B. Felgenhauer and F. Jarvis. Enumerating possible Sudoku grids. Online article, 2005. `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf`.

[5] B. Hanson. Sudoku Assistant/Solver. Online software written in JavaScript, 2007. `http://www.stolaf.edu/people/hansonr/sudoku/`.

[6] C. K. Holst. Sudoku—an excercise in constraint programming in Visual Prolog 7. In *1st Visual Prolog Applications and Language Conference, VIP-ALC 2006*, Faro, Portugal, Apr. 2006.

[7] A. Inkala. *AI Escargot—The Most Difficult Sudoku Puzzle*. Lulu, Finland, 2007. ISBN 978-1-84753-451-4.

[8] N. Juillerat. Sudoku Explainer. Version 1.2, Dec. 2006. `http://diuf.unifr.ch/people/juillera/Sudoku/Sudoku.html`.

[9] D. E. Knuth. Dancing links. In J. Davies, B. Roscoe, and J. Woodcock, editors, *Millenial Perspectives in Computer Science*, pages 187–214. Palgrave, Houndmills, Basingstoke, Hampshire, 2000.

[10] I. Lynce and J. Ouaknine. Sudoku as a SAT problem. In S. Zilberstein, editor, *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006*, Fort Lauderdale, Florida, USA, Jan. 2006.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, Las Vegas, USA, June 2001.

[12] ravel. The hardest sudokus. Sudoku Players' Forums, Jan 25, 2007. `http://www.sudoku.com/forums/viewtopic.php?t=4212&start=587`.

[13] G. Royle. Minimum sudoku. Online article, 2007. `http://people.csse.uwa.edu.au/gordon/sudokumin.php`.

[14] H. Simonis. Sudoku as a constraint problem. In *Proceedings of the CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems*, pages 13–27, Sitges, Spain, Oct. 2005.

[15] P. Torres and P. Lopez. Overview and possible extensions of shaving techniques for job-shop problems. In *2nd International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CP-AI-OR 2000*, pages 181–186, IC-PARC, UK, Mar. 2000.

[16] T. Weber. A SAT-based Sudoku solver. In G. Sutclife and A. Voronkov, editors, *12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005*, pages 11–15, Montego Bay, Jamaica, Dec. 2005. Short Paper Proceedings.

[17] T. Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, Univ. of Tokyo, Dept. of Information Science, Faculty of Science, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, JAPAN, Jan

2003. http://www-imai.is.s.u-tokyo.ac.jp/
~yato/data2/MasterThesis.ps.