UNIVERSITY OF AMSTERDAM

A SAT Attack on Killer Sudoku Problems*

Shuai Wang Aashish Venkatesh

September 22, 2016

Killer sudoku is a special class of sudoku where the sum of some adjacent cells is specified. This case study presents the first attempt using a SAT solver for killer sudoku problems. More specifically, we introduce an encoding from arithmetic constraints to propositional constraints. A qualitative evaluation is presented regarding different cage sizes. In addition, the first opensource killer sudoku database is developed.

1. SUDOKU PROBLEMS AS SAT PROBLEMS

Sudoku is a Japanese game since 1986. In Japanese, ”Sudoku” means ”single number” [3]. A Sudoku Problem is a problem to fill in a n*n boards with numbers from 1-9. The filled board should have the first three of following constrains satisfied [2].

1. There is exactly one number in each cell.
2. Each number must appear exactly once in each row / column / 3*3 block (a.k.a. nonet).
3. Some numbers are pre-filled and we need to consider about them
4. **The sum of pairwise distinct numbers in cells of the same cage equals to that as labelled on the cage (i.e. the *label* of the cage)**

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

Figure 0.1: An example sudoku puzzle

A killer sudoku is also called Sumdoku, It differs from classical sudokus since it doesn't have pre-filled cells. Instead, its constraints are about the sum of numbers in cells (constraint 1-2 and 4). A set of such cells are named *cage*. We define the maximum allowed cage size as K. Sudoku puzzles are hard problems. There are more than 6×10^{21} possible sudoku grids. Due to its large searching space, a Naive backtracking algorithm is infeasible. A SAT approach of sudoku problem is to translated into equi-satisfiable propositional formula. A general puzzle solbing process is as illustrated in Fig. 1.2. Based on experience solving killer sudokus by hand, our hypothesis is as follows:

The bigger K is, the harder it is to solve.

*Project homepage: `https://uva-kr16.github.io/KilerSudoku/`.

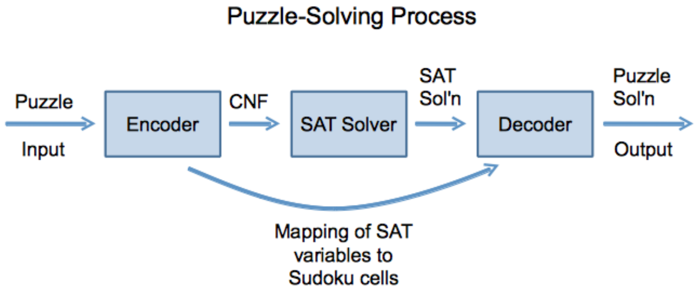

Figure 1.2: General puzzle solving process using SAT solvers

.

2. Experimental Setup

2.1. Generation of Killer Sudokus

This project is about evaluating the hardness of killer sudokus regarding the K value. We measure it in two ways: the amount of clauses generated and the time taken for solving. This experiment was based on evaluation of 1000 sudokus for each K value. The database of killer sudoku was generated using a free (solved) sudoku database[1]. The database had to be generated as there was no existing database grouped by the K value. Hence, to test the hypothesis, killer sudokus with different K values were generated. During the generation, K varied from 2 to 9 and at least three such cages were created in each killer sudoku. And the remaining cages were randomly generated of size 2 to K.

2.2. Metrics

Metrics used to analyse the experiment are average number of clauses (the total amount and only those corresponding to arithematic constraints) for each K and average time taken to solve the sudoku by the SAT solver as well as the total time. These entries were used to infer about the hardness of the sudoku.

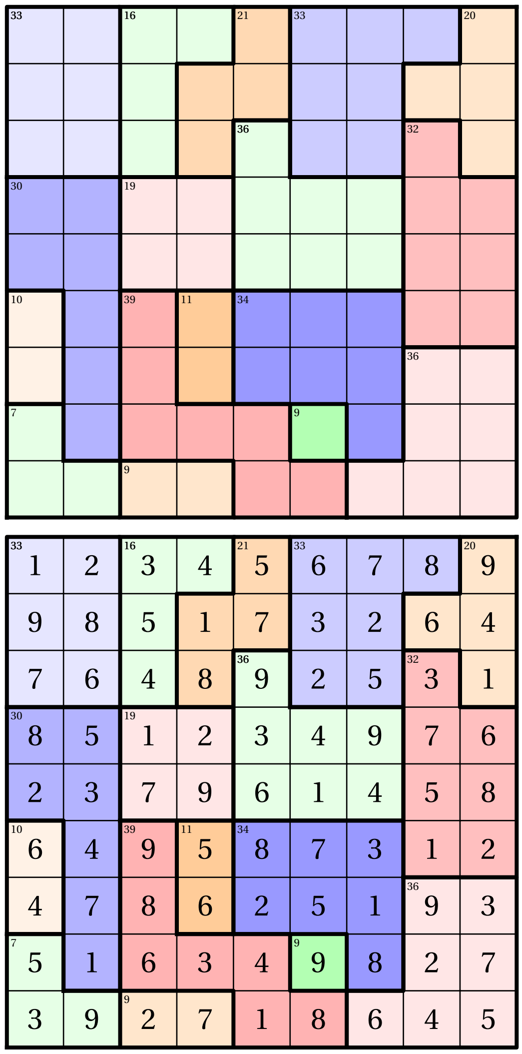

Figure 1.1: A Killer Sudoku puzzle and its solution

2.3. SAT Solver

The metric used in evaluating the experiment is based on a large sample of sudokus. Thus, there is a need for a robust SAT solver. PicoSAT is a state of the art SAT solver [1], written in C, is both deterministic and efficient. Pycosat is a package that provides user-friendly Python bindings. And therefore PicoSAT was employed as the SAT solver to solve killer sudokus using Pycosat.

[1] The database made use of a free sudoku database at `http://www.printable-sudoku-puzzles.com/wfiles/`.

2.4. Encoding of Constraints

A SAT approach of a sudoku problem is to translate constraints into equi-satisfiable propositional formulas in Conjunctive Normal Form (CNF) and obtain the result by interacting with a *SAT solver*, a program to solve a satisfiability problem in propositional logic [2]. By introducing for each cell on column i and row j and a possible number k a proposition $p^k_{i,j}$, we can encode the constraints as follows respectively:

1. $\bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{k_1,k_2=1}^{9} (\neg p^{k_1}_{i,j} \vee \neg p^{k_2}_{i,j})$, $k_1 \neq k_2$.
2. $\bigwedge (\neg p^{k_1}_{i_1,j_1} \vee \neg p^{k_2}_{i_2,j_2})$, for two different cells at (i_1, j_1) and (i_2, j_2) in the same rows, or same columns, or any 3*3 blocks (i.e. nonets), $1 \leq k_1, k_2 \leq 9$.
3. set $p^k_{i,j}$ to *True* if a cell at (i, j) is labelled with k.
4. For each possible combination $pos = \{b_0, b_1, \ldots, b_n\} \in POS$ regarding a cage C labelled s (with its cells named $c_0, c_1, \ldots, c_n \in C$), where $sum(b_i) = s$, we introduce a new proposition x_i for each pos and y_i for each number b_i.
   a) y_i iff i was labelled on one of the cells in the cage.
   b) x iff every $y_i \in b_i$ if true.
   c) $\bigvee x$ for each x corresponding to a possible combination $pos \in POS$

The naive approach used n^3 total variables and in total $O(n^4)$ total clauses (approx. 13k clauses). By introducing new propositional variables, we can better encode the constraint of **at most one** literal is satisfied among a list of literals. The idea is, for a list of literals, if at most one of them is satisfied, it is either the first, or one of the rest. While one of the rest could be defined similarly in a recursive fashion. Here we introduce a new proposition representing the 'rest'. This way, we reduce the total clauses to $O(n)$ by introducing $O(n)$ new propositions. This reduce the total amount of clauses to around 8k, including around 1500 clauses corresponding to the arithmetic clauses. These arithmetic clauses were generated using an approach inspired by Tseytin transformation [2]. For a cage, its K value and the sum uniquely determines a set of possible combinations. For the numbers appear in every possible combinations, we take the disjunction of the corresponding literals. For every number from 2 to 9, we introduce a proposition to represent the present of the number in the cage, say y_2 for the number 2. For each possible combination, a new proposition is introduced as x_i for the ith combination. Thus, x_i implies $y_r \wedge y_s \wedge \ldots$. To make sure at least one combination is realised, we introduce the last clause $x_1, x_2, \ldots$. All clauses can then be passed on to the solver to generate solutions.

3. Evaluation

3.1. Experimental Results

This project employed PicoSAT as the SAT solver [1]. By interacting with its Python API, it took less than one second to obtain a solution for any killer sudoku problem. In addition, the total time and the solving time (of PicoSAT) were recorded and the average time for each sudoku corresponding to a specified cage size is presented as in Table 3.1. We obtained good efficiency as proved by the evaluation results [3]. It is clear that the more possibilities regarding a cage and its label, the harder it is to solve. We completed 5 runs of evaluation and took the average. The table consisting of the raw data are attached in the Appendix A.

3.2. Interpretation

The experimental results is against our hypothesis. The main reason that the solving time decreases after 6 is that, when the K value is greater than 5, the amount of possible combinations decreases, making the solving time decrease as a result. This contradiction indicates that there is a difference between human reasoning

[2] `https://en.wikipedia.org/wiki/Tseytin_transformation`
[3] PC specification: Intel Core i-3-3120M CPU × 4, 6 GB memory

Table 3.1: Evaluation Results using PicoSAT

Max. Cage Size (K-value)	2	3	4	5	6	7	8	9
Possible Labels	15	19	21	21	19	15	9	1
Possible Combinations	36	84	127	127	87	54	9	1
Avg. No. Clauses(A.C.[4])	1475	1555	1638	1630	1532	1406	1342	1317
Avg. Total No. Clauses	8907	9007	9090	9080	8984	8858	8794	8769
Avg Total Time (s)	0.2027	0.2049	0.2102	0.2140	0.2182	0.2158	0.2138	0.2142
Avg Solving Time (s)	0.0253	0.0264	0.0296	0.0335	0.0378	0.0364	0.0356	0.0359

and machine reasoning. One reason is that we human are not good at calculating big numbers. For example, we may find it hard to find possible values for cells of a cage of size 7 and labelled 35 but easier to find the values for a cage of size 3 and labelled 14. In fact, the first case has only 4 possible combinations numbers while the latter has 8. Another possible reason is that we lack of enough experience solving killer sudokus by hand before making the hypothesis. In the result, it was noticed that although few possible combinations were possible for k value of 8 and 9, the total time is not as little. This is possibly due to the presents of cages of other size, for example, cages of size 4, 5 or 6.

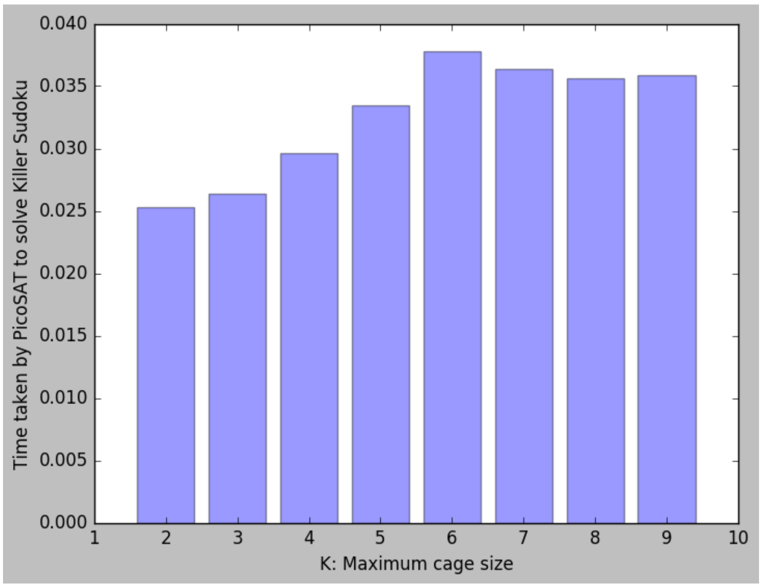

Figure 3.1: Bar chart of Avg Solving time taken by the PicoSAT for different K

4. Conclusion and Acknowledgement

This report presents a case study using a SAT solver to solve killer sudoku problems by encoding arithmetic and logical constrains to propositional constraints. The database and sourcecode of this project are available on the project page[5]. The authors would like to thank Prof. Frank van Harmelen for his guidance and inspiration. This work is going to be presented in the Colloquium on Combinatorics to take place at Paderborn University, Germany [6].

References

[1] Armin Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[5] `https://uva-kr16.github.io/KilerSudoku/`
[6] `http://www.kolkom.de/`

[2] Inês Lynce and Joël Ouaknine. Sudoku as a SAT problem. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006.

[3] Tjark Weber. A SAT-based sudoku solver. In *12 th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005*, pages 11–15, 2005.

A. PicoSAT Observation Data

The table is the observational data from PicoSAT solver after running the 1000 killer sudokus for each K for five rounds. In the table, K = maximum size of the cage, T.T = Total time taken by the system including I/O in seconds, S.T = Total time taken by the system to solve the sudoku in seconds.

Table A.1: PicoSAT observation over five rounds

	Round 1		Round 2		Round 3		Round 4		Round 5	
K	T.T(s)	S.T(s)	T.T (s)	S.T(s)	T.T (s)	S.T(s)	T.T (s)	S.T(s)	T.T (s)	S.T(s)
2	0.2047	0.0258	0.2016	0.0253	0.2029	0.0252	0.2011	0.0252	0.2033	0.0252
3	0.2071	0.0270	0.2054	0.0262	0.2051	0.0263	0.2040	0.0263	0.2031	0.0261
4	0.2102	0.0297	0.2107	0.0296	0.2110	0.0297	0.2100	0.296	0.2089	0.0293
5	0.2136	0.0336	0.2143	0.0334	0.2137	0.0334	0.2157	0.0335	0.2128	0.0335
6	0.2184	0.0378	0.2181	0.0377	0.2187	0.0378	0.2187	0.0377	0.2172	0.380
7	0.2157	0.0364	0.2165	0.0363	0.2167	0.0362	0.2155	0.0364	0.2145	0.0365
8	0.2138	0.0358	0.2138	0.0356	0.2133	0.0358	0.2146	0.0355	0.2133	0.0355
9	0.2146	0.0358	0.2152	0.0358	0.2149	0.0359	0.2121	0.0360	0.2141	0.0358

B. Possible Combinations of Numbers for Different Cage Size and Sums

Table B.1 to Table B.7 shows all the possible combinations of cages of size 2 to 8. For cages of size 9, all numbers from 1 to 9 must appear. Thus, there is only one possible case.

Table B.1: Possible combinations for cages of size 2

3	12
4	13
5	14 23
6	15 24
7	16 25 34
8	17 26 35
9	18 27 36 45
10	19 28 37 46
11	29 38 47 56
12	39 48 57
13	49 58 67
14	59 68
15	69 78
16	79
17	89

Table B.2: Possible combinations for cages of size 3

6	123
7	124
8	125 134
9	126 135 234
10	127 136 145 235
11	128 137 146 236 245
12	129 138 147 156 237 246 345
13	139 148 157 238 247 256 346
14	149 158 167 239 248 257 347 356
15	159 168 249 258 267 348 357 456
16	169 178 259 268 349 358 367 457
17	179 269 278 359 368 458 467
18	189 279 369 378 459 468 567
19	289 379 469 478 568
20	389 479 569 578
21	489 579 678
22	589 679
23	689
24	789

Table B.3: Possible combinations for cages of size 4

10	1234
11	1235
12	1236 1245
13	1237 1246 1345
14	1238 1247 1256 1346 2345
15	1239 1248 1257 1347 1356 2346
16	1249 1258 1267 1348 1357 1456 2347 2356
17	1259 1268 1349 1358 1367 1457 2348 2357 2456
18	1269 1278 1359 1368 1458 1467 2349 2358 2367 2457 3456
19	1279 1369 1378 1459 1468 1567 2359 2368 2458 2467 3457
20	1289 1379 1469 1478 1568 2369 2378 2459 2468 2567 3458 3467
21	1389 1479 1569 1578 2379 2469 2478 2568 3459 3468 3567
22	1489 1579 1678 2389 2479 2569 2578 3469 3478 3568 4567
23	1589 1679 2489 2579 2678 3479 3569 3578 4568
24	1689 2589 2679 3489 3579 3678 4569 4578
25	1789 2689 3589 3679 4579 4678
26	2789 3689 4589 4679 5678
27	3789 4689 5679
28	4789 5689
29	5789
30	6789

Table B.4: Possible combinations for cages of size 5

15	12345
16	12346
17	12347 12356
18	12348 12357 12456
19	12349 12358 12367 12457 13456
20	12359 12368 12458 12467 13457 23456
21	12369 12378 12459 12468 12567 13458 13467 23457
22	12379 12469 12478 12568 13459 13468 13567 23458 23467
23	12389 12479 12569 12578 13469 13478 13568 14567 23459 23468 23567
24	12489 12579 12678 13479 13569 13578 14568 23469 23478 23568 24567
25	12589 12679 13489 13579 13678 14569 14578 23479 23569 23578 24568 34567
26	12689 13589 13679 14579 14678 23489 23579 23678 24569 24578 34568
27	12789 13689 14589 14679 15678 23589 23679 24579 24678 34569 34578
28	13789 14689 15679 23689 24589 24679 25678 34579 34678
29	14789 15689 23789 24689 25679 34589 34679 35678
30	15789 24789 25689 34689 35679 45678
31	16789 25789 34789 35689 45679
32	26789 35789 45689
33	36789 45789
34	46789
35	56789

Table B.5: Possible combinations for cages of size 6

21	123456
22	123457
23	123458 123467
24	123459 123468 123567
25	123469 123478 123568 124567
26	123479 123569 123578 124568 134567
27	123489 123579 123678 124569 124578 134568 234567
28	123589 123679 124579 124678 134569 134578 234568
29	123689 124589 124679 125678 134579 134678 234569 234578
30	123789 124689 125679 134589 134679 135678 234579 234678
31	124789 125689 134689 135679 145678 234589 234679 235678
32	125789 134789 135689 145679 234689 235679 245678
33	126789 135789 145689 234789 235689 245679 345678
34	136789 145789 235789 245689 345679
35	146789 236789 245789 345689
36	156789 246789 345789
37	256789 346789
38	356789
39	456789

Table B.6: Possible combinations for cages of size 7

28	1234567
29	1234568
30	1234569 1234578
31	1234579 1234678
32	1234589 1234679 1235678
33	1234689 1235679 1245678
34	1234789 1235689 1245679 1345678
35	1235789 1245689 1345679 2345678
36	1236789 1245789 1345689 2345679
37	1246789 1345789 2345689
38	1256789 1346789 2345789
39	1356789 2346789
40	1456789 2356789
41	2456789
42	3456789

Table B.7: Possible combinations for cages of size 8

36	12345678
37	12345679
38	12345689
39	12345789
40	12346789
41	12356789
42	12456789
43	13456789
44	23456789