University of Victoria

CSC 320
Final Project
Sudoku to SAT

Dryden Linden-Bremner V00849440

Marina Dunn V00844643

Divyapreet Chawla V00862263

Whitney Dluhosh V00839944

Introduction

This report is designed to summarize and show the results of our implementation of SudToSAT as well as the performance that results from the MiniSAT Solver that comes from our CNF encoding. Further this report will summarize and discuss the results of the hard inputs located at *magictour.free.fr/top95*. The use of some other sat solvers other than mini-SAT will also be discussed.

Outline Of Program:

A minimal encoding version can be seen in sud2sat729 and sat2sud729 versus our extended coded version which is sat2sud and sud2sat.

The program sudoku2SAT takes a sudoku puzzle represented by either a 9x9 grid or by a single line of length 81, that contains wildcards such as 0 or * and numbers 1-9. The program takes the input contained in a file and converts it to CNF.

The program SAT2Sudoku takes the text file outputted by the minisat/sudoku2SAT and converts it to a sudoku puzzle of the same form discussed above.

Solutions:

Each line is a single successful output

```
492571386865423719371896245938157624247369851156248937684912573529734168713685492
218954376463718925597632184925147638876293541134586792352469817781325469649871253
473162985698754231251398647847935126329816754165427893914273568582641379736589412
194685237326714598785392614238561749519478326647239185853927461461853972972146853
576241398289356417341978256854129763197463825632587941718692534423815679965734182
158964273729531684643782519986317425215496837374825961562143798837659142491278356
162934875478651293395278164947362518823517649651489732236195487589746321714823956
439852176861497235725631489143578692597126843682943517978215364356784921214369758
867921354153476982492853176918237645725649831634518297541382769376194528289765413
794382615623591847158746923976435182241678539835129476419263758562817394387954261
914285736752361894683479251148793562237658149569124378491837625325916487876542913
915346827647289315283175469392658741476913258158427693761894532524731986839562174
362817459941265738758394216625973841873541962194628375589736124217459683436182597
648572319372691584915438267256847931137259846894316752421783695563924178789165423
643529178921387456785164923164235897237498561859671234316942785498756312572813649
354612798689375421172849635798261354263584917541937286926458173415793862837126549
367851942541729368982643157835976421614238795729415683158394276293567814476182539
725134986813629547694578123982465731467813259531297864348951672156742398279386415
751968342482375916396412875539247681148596723267183594923754168814639257675821439
746251398835964271291387654689435712123796485457128936314872569572619843968543127
439265871267198453851473926592841637346957182178632594714529368983716245625384719
```

```
413867952276951834589423761735186249862549173194732685951378426328694517647215398
352618497476925183819734526291386745648572931735149862963451278527893614184267359
183426795269753184475891236947382651831567942652914873796235418318649527524178369
375846129682915437194327856538169742267584913419732685743651298826493571951278364
536927184824531796197684235413795628752368941968142573281459367645273819379816452
345176829861259374792384516274591683583462791619837452438925167126748935957613248
328146759476958312591723864864375291752419638139682475215837946683594127947261583
248759361316428957597613428735184296624397815189562734452976183871235649963841572
261854379953276184874139256132687495798425613546913827615742938427398561389561742
785316492634729158129485736576294381492138675318567924261953847843672519957841263
179528634356479128284361975627854319418932756935716842761285493842193567593647281
631598724957342618842167539165834972724915386389726145513489267476251893298673451
814269735596743812273851649789615423142378956635492187961534278458927361327186594
461732598527918436398645271274591863159863742836274915745386129983127654612459387
587631924143925678692874153214398765876152349935746281329587416451263897768419532
973215468482637519516984723627198354835746291149523687791352846268471935354869172
319856274627134958584972631978421365451367892263589417845613729796245183132798546
126387945379524168485916723512849376863172459794635812238451697951768234647293581
147562893968371245325849617639214758751938426284657139496723581872195364513486972
867921354153476982492853176918237645725649831634518297541382769376194528289765413
317589246825641397469372158548736921632914785791825463973268514284157639156493872
475931628613284597928756341142675983569318472837492165386547219754129836291863754
394526817185937426672481593536249178427318659918675234269853741743162985851794362
574126938829374156631598724148965273265713489793482615982651347456237891317849562
968715243513624879472389156257893614149576328386142795695231487824967531731458962
123695487574218369968473512756921843481736295392854176847362951635189724219547638
854932716693157842172684935346291587581763429729548163237819654968425371415376298
278146953916573824453298617847935162625481379391762485564829731139657248782314596
347296815598731264126584937219458673853627491674913528465172389931865742782349156
```

Results of Basic Puzzles:

Below are the results from running our program on the basic inputs from *https://projecteuler.net/project/resources/p096_sudoku.txt*

[Table 1]

Solved Problems	Clauses	Decisions	CPU Time
1	4477	1	0.007417
2	4826	17	0.006714
3	5706	26	0.007079
4	4621	3	0.00731
5	3340	1	0.005411

6	5956	36	0.007601
7	5388	14	0.006925
8	3019	1	0.00524
9	5895	22	0.007306
10	5330	14	0.00746
11	5269	14	0.006888
12	4266	1	0.006134
13	6207	3	0.006985
14	6418	8	0.007472
15	5328	1	0.006528
16	2814	1	0.005149
17	3292	1	0.005435
18	5095	19	0.006669
19	4062	1	0.005801
20	4703	1	0.006013
21	4986	9	0.00676
22	5425	9	0.006766
23	4988	4	0.006892
24	5608	9	0.006783
25	5191	20	0.007196
26	6081	72	0.007646
27	5347	3	0.006742
28	5592	10	0.006927
29	5755	21	0.007478
30	5691	14	0.007474
31	6136	45	0.008316
32	6149	19	0.007162
33	4217	2	0.006134
34	3984	1	0.005659
35	4750	5	0.006416
36	3949	1	0.005892
37	4334	4	0.007009
38	4302	1	0.006315
39	4968	1	0.007521
40	4375	1	0.005841

41	5895	22	0.007385
42	5489	11	0.006944
43	5658	25	0.007279
44	5990	52	0.007677
45	5829	20	0.007842
46	6228	10	0.007569
47	6466	162	0.008251
48	6142	56	0.007609
49	6905	140	0.008654
50	5824	18	0.007322
			AVERAGE
			0.006889408163

Solutions To Hard Problems:

Below are solutions to the first 10 hard problems due to space, the rest of the outputs from the program for these are in hardproblems.txt

```
469873251135294876728516934317459628652381497984762315893147562241635789576928143
583164972291785463764293158359426817148579236627318549475832691836951724912647385
625931874143875296789246315471528639532697481968413752894352167216789543357164928
451792836892316574736548192365827419189463725247951683623185947978634251514279368
915863274637452891248719536352981647189674352476235918824597163561348729793126485
497528631183796452625134798841273965362945187759861324534612879276489513918357246
693482571857319624214765893178594236569231748432876159381927465746153982925648317
689432157537619824241758639463875912872196345195243768918564273356927481724381596
693482571857319624214765893178593246569241738342876159481927365736154982925638417
756198342948523617231746895374612589582439761169875234615284973423967158897351426
```

Results Of Harder Puzzles:

Below are the results obtained by running our converter on the 95 'hard' problems from *magictour.free.fr/top95* and the respective averages of the results

[Table 2]

Hard Problems	Clauses	Decisions	CPU Time
1	7828	188	0.010355
2	7545	276	0.00926
3	7416	522	0.01604
4	7143	255	0.009247
5	7986	375	0.010084

6	7409	446	0.009975
7	7315	1942	0.016917
8	7356	168	0.009495
9	7317	1208	0.014665
10	7213	161	0.009176
11	7312	664	0.070984
12	5743	242	0.008467
13	5918	162	0.008777
14	7241	761	0.01258
15	7414	182	0.009425
16	7306	1116	0.013466
17	7410	50	0.008399
18	7504	166	0.009062
19	5524	460	0.009498
20	6097	234	0.008276
21	7437	313	0.009621
22	6587	96	0.008478
23	7846	276	0.009697
24	6736	55	0.007914
25	6637	103	0.008562
26	7657	415	0.010141
27	7617	57	0.009052
28	7357	1444	0.015741
29	6563	29	0.007969
30	7986	215	0.009554
31	6991	40	0.009362
32	6693	104	0.008413
33	6432	53	0.008216
34	7631	251	0.009912
35	6577	270	0.008893
36	7960	227	0.009832
37	7411	154	0.009257
38	6663	147	0.009818
39	6287	90	0.008838
40	6683	154	0.008904

41	7945	232	0.009919
42	7999	289	0.00983
43	5785	23	0.007454
44	6173	44	0.007858
45	6065	147	0.008242
46	7422	1242	0.016222
47	7493	460	0.010174
48	6266	198	0.008863
49	7317	530	0.010376
50	7354	1066	0.013021
51	6474	9	0.007765
52	7613	217	0.009523
53	5358	84	0.007522
54	5164	51	0.007371
55	6744	40	0.00814
56	5944	42	0.007753
57	6868	343	0.00941
58	7502	126	0.008796
59	6416	63	0.0081
60	6938	163	0.009036
61	7168	79	0.008839
62	7434	173	0.00887
63	6402	157	0.008935
64	7112	46	0.009904
65	6941	50	0.008955
66	6973	101	0.009186
67	7949	105	0.009105
68	7409	216	0.009062
69	7214	79	0.008412
70	6981	110	0.008352
71	7460	75	0.009518
72	6430	28	0.007772
73	7982	452	0.010129
74	7495	272	0.009134
75	6038	80	0.007945

76	7635	408	0.010418
77	6163	190	0.008065
78	5760	106	0.008041
79	8002	218	0.010262
80	6449	194	0.00854
81	6454	76	0.008271
82	6370	78	0.007943
83	7707	94	0.009001
84	7429	325	0.009675
85	7014	144	0.008488
86	5791	10	0.007278
87	6281	123	0.007853
88	5960	30	0.007554
89	7344	36	0.008175
90	6652	273	0.008733
91	6646	103	0.008652
92	6773	56	0.008773
93	6928	296	0.009605
94	7027	90	0.008856
95	7935	268	0.0101
			AVERAGE
			0.01006109574

Alternative to Minimal Encoding:

We had an extended encoding which had the form of IJK for the values of the variable yielding 999 vars, this was primarily designed to be more understandable in the code rather than going for the smaller minimal encoding.

Comparison to Specialized Sudoku Solvers:

Backtracking:

A backtracking algorithm as used for a Sudoku Solver is a type of brute force measure used to find the correct solution. This method requires the algorithm to essentially “try” different solutions to the Sudoku puzzle. The benefits of applying this method to a Sudoku puzzle include that a solution is always guaranteed, despite the solution possibly taking more time. Additionally, the solving time itself it mostly unrelated to the level of difficulty of the puzzle, this is due to the fact that the placement of numbers is sequential and the only real factor is the amount of missing numbers that need to be found. Finally, the backtracking algorithm has relatively simple code. It does not require an extensive program because the idea behind the algorithm is not complex either.

Dancing Links:

Dancing links algorithm is modified version of backtracking. In this approach, when reaching a invalid state which cannot lead to a solution,abandon the branch of computation. Also, instead of checking if a board is valid, dancing links algorithm checks a valid state by placing a number n at position (i,j). This approach, unlike backtracking does not spend time in cells that cannot contain a solution.

Stochastic Search:

The stochastic search algorithm is an algorithm that randomly places numbers into their slots, it then proceeds to calculate the number of errors, then places the numbers back into their slots in different orders until the number of errors reaches 0. The benefit of this algorithm is that these are typically fast algorithms, however they are not necessarily as fast as smarter programs who use logic to determine where not to place the numbers in the next iteration.

Crook’s algorithm:

Crook’s “paper and pencil” algorithm is based on solving sudoku by first finding all forced numbers. Then use the appropriate crossout actions iteratively to find unsolved numbers using rows, columns, and 3X3 boxes. If a solution is not found, a random choice has to be made in order to continue to solve the puzzle. This method, in hard puzzles, may take up time because of the choice of random number. However this algorithm could be efficient where all forced numbers are found.

Discussion:

Both sets of puzzles were solvable given our cnf converter however as shown the harder puzzles took on average .01006109574 vs the .0068894081 of the easy puzzles this shows that clearly the harder puzzles took longer for the sat solver to solve and thus must have taken more iterations to pass through to solve. The results we have obtained are better than some of the brute force methods but do not have the same theoretical efficiency as some of the better solutions for the problem.

Conclusions:

We have shown that both easy and hard Sudoku problems are solvable with CNF and running it with a SAT solver such as MiniSAT. We have also shown that the harder problems do indeed take longer to solve. In addition, the applications and benefits of special-purpose algorithms other than SAT solvers for sudoku puzzles are explored in Backtracking, Stochastic, Dancing Links, and Crook’s.