Sudoku as a Constraint Problem

Helmut Simonis

IC-Parc
Imperial College London
hs@icparc.ic.ac.uk

Abstract. Constraint programming has finally reached the masses, thousands of newspaper readers (especially in the UK) are solving their daily constraint problem. They apply complex propagation schemes with names like "X-Wing" and "Swordfish" to find solutions of a rather simple looking puzzle called Sudoku. Unfortunately, they are not aware that this is constraint programming. In this paper we try to understand the puzzle from a constraint point of view, show models to solve and generate puzzles and give an objective measure of the difficulty of a puzzle instance. This measure seems to correlate well with grades (e.g. easy to hard) that are assigned to problem instances for the general public. We also show how the model can be strengthened with redundant constraints and how these can be implemented using bipartite matching and flow algorithms.

1 Introduction

Sudoku [1] is a puzzle played on a partially filled 9x9 grid. The task is to complete the assignment using numbers from 1 to 9 such that the entries in each row, each column and each major 3x3 block are pairwise different. Like for many logical puzzles the challenge in Sudoku does not just lie in finding a solution. Well posed puzzles have a unique solution and the task is to find it without guessing, i.e. without search. In this paper we compare a number of propagation schemes on how many problem instances they can solve by constraint propagation alone.

The basic Sudoku problem can be modelled with constraint programming [2] by a combination of alldifferent constraints[3]. Using different consistency techniques for these constraints we derive a number of propagation schemes with differing strength. We can extend the model either by shaving[4], testing each value in each domain with a one-step lookahead technique or by adding redundant constraints. We use simplified versions of the colored matrix constraint[5, 6] and the same with cardinality constraint [7] and propose additional, new constraints which are also based on matching and flow algorithms.

We have evaluated our methods on different sets of example problems and can see how the grades assigned by the puzzle designers often, but not always match the propagation schemes described here. This evaluation also gives a fair comparison of the difficulty of the different puzzle sources.

The paper is structured as follows: We first discuss some related work in section 2 and then give a formal problem definition in section 3. This is followed in section 4 by a description of the constraint model used to express the problem. Adding redundant constraints can improve the reasoning power of a constraint model, we present different alternatives in section 5. We continue in section 6 by listing the different combinations of propagation schemes used for the evaluation in section 7. Finally, we discuss methods to generate puzzles in section 8, before ending with some conclusions.

2 Related work

There is a whole sub area in constraint programming concerned with backtrack-free search[2]. But its focus is on identifying problem classes where all instances can be solved without backtracking. For the Sudoku puzzle we want to know which problem instances we can solve without search.

Solving puzzles has always been a favorite activity for the constraint programming community. Starting with Lauriere[8], puzzles were used to come up with new constraint mechanisms and to compare different solution methods. Examples are n-queens[9], the five houses puzzle[2], the progressive party problem[10] and more recently the social golfer problem[11]. Some well known games like Peg Solitaire[12] and minesweeper[13] were also approached with constraint programming. The Sudoku problem can be easly modelled using the alldifferent[3] constraint. There are other problems where this constraint plays a central role, in particular quasi-group completion[6][14][15][16], but also industrial problems like aircraft stand allocation[17] [18].

An alternative view of the Sudoku problem is provided by [19], this could be the basis for additional, redundant constraints.

3 Problem Definition

Definition 1. *A Sudoku square of order n consists of n^4 variables formed into a $n^2 \times n^2$ grid with values from 1 to n^2 such that the entries in each row, each column and in each of the n^2 major $n \times n$ blocks are alldifferent.*

An example of a Sudoku square is shown in table 1. The major blocks are outlined by a thicker framing.

Currently, only Sudoku problems of order 3 (9×9 grid) are widely used. It was claimed in [1] that there are 6,670,903,752,021,072,936,960 valid Sudoku squares of order 3.

Definition 2. *A Sudoku problem (SP) consists of a partial assignment of the variables in a Sudoku square. The objective is to find a completion of the assignment which extends the partial assignment and satisfies the constraints.*

Table 2 shows a SP in the form normally presented. Its solution is the square shown in table 1.

We are only interested in puzzles which have a unique solution. If a partial assignment allows more than one solution, we need to add hints until it is well posed.

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

Table 1. Example Sudoku Square

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

Table 2. Example Sudoku Problem

Definition 3. *A SP is well posed if it admits exactly one solution.*

Ideally, the hints of a puzzle should not contain any redundant information, e.g. there should be no hint that is itself a logical consequence of the other hints.

Definition 4. *A well posed SP is locally minimal if no restriction of the assignment is a well posed problem.*

Whether a puzzle can be solved without guessing depends on the deduction mechanism used. We only use a informal notion here, a more formal treatment can be found in [2].

Definition 5. *A SP is search free wrt. a propagation method if it is solved without search, just by applying the constraint reasoning.*

4 Constraint Model

We now discuss the basic model for the Sudoku puzzle. The programs are written in ECLiPSe 5.8[20, 21]. The program uses the IC library (which defines a forward checking[3, 22] version of alldifferent) and its extension ic_global, which provides a bound-consistent[23, 24] alldifferent. A propagator for a hyper arc-consistent alldifferent[25] was added for the study.

The program uses a $n^2 * n^2$ matrix of finite domain variables with domain 1 to n^2. We set up $3 * n^2$ alldifferent constraints for each row, each column and each major $n * n$ block.

4.1 Channeling

In our problem, each alldifferent constraint sets up a bijection between n^2 variables and n^2 values. In this bijection the roles of variables and values are interchangeable, but the constraint handling of the forward checking alldifferent is oriented. For example, it will assign a variable which has only one value left in its domain, but it will not assign a value which occurs only once in the domains of all variables. A method of overcoming this limitation is the use of channeling[26]. For each alldifferent constraint we introduce a dual set of variables, which are linked to the original variables via an inverse constraint [5], and which themselves must be alldifferent. This improves the reasoning obtained from the forward checking and the bound-consistent versions of the alldifferent, but obviously is useless for the hyper arc-consistent alldifferent.

5 Redundant Constraints

Even if we use hyper arc-consistency as the method for each of the alldifferent constraints we will not always achieve global consistency. The constraints interact in multiple ways and the local reasoning on each constraint alone can not exploit these connections. A common technique to help with this problem is the use of redundant constraints which can strengthen the deduction by combining several of the original constraints. We show four such combinations here. The first is a simplified version of the colored matrix [5]. This was already used in [6] to improve a method for quasi-group completion. The others are new, but use bi-partite matching and flow techniques inspired by [27].

5.1 Row/Column interaction

This constraint handles the interaction of rows and columns in the matrix. The colored matrix constraint[5], also called cardinality matrix in [6], expresses constraints on the number of occurrences of the values in rows and columns of a matrix. This reduces to a set of simple matching problems for a permutation matrix. Each value must occur exactly once in each row and column, this corresponds to a matching between row and columns (the two sets of nodes), and edges which link a row and a column if the given value is in the domain of the corresponding matrix element. By finding a maximal matching and then identifying strongly connected components in a re-oriented graph we can eliminate those values from all domains which do not belong to any maximal matching. Figure 1 shows the form of the graph considered.

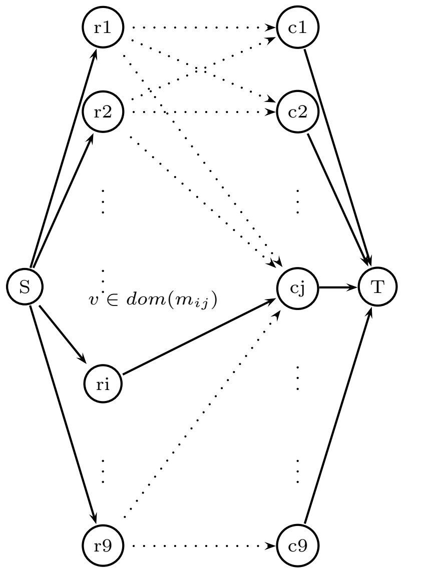

Fig. 1. Row/Column Matching

We have to solve one matching problem per value, this makes 9 constraints.

5.2 Row/Block interaction

We now look at the interaction of a single row(column) with a single major block as shown in figure 2. The row alldifferent constraint and the alldifferent constraint

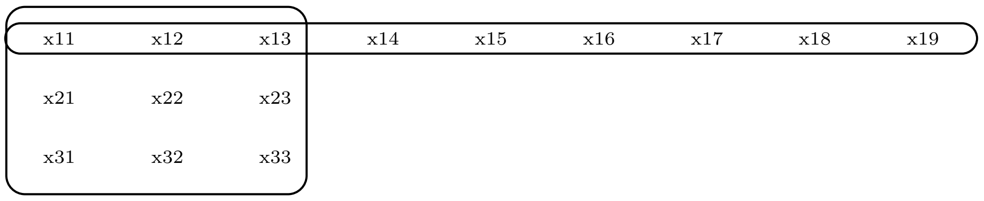

Fig. 2. Row/Block Interaction

for the major block coincide in three variables. One necessary condition is that the set of variables $B = \{x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}\}$ uses the same set of values as the set $C = \{x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}\}$ of variables. We can exploit this condition by removing from the variables in set B all values which are not supported by values in set C and vice versa. This constraint is a special case of the same with cardinality of [7], which achieves hyper arc-consistency by solving a flow problem. For this simpler variant, our method seems equivalent.

Considering all rows/columns and major blocks leads to 54 such constraints.

5.3 Rows/Blocks interaction

We can also consider the interaction of all major blocks in a line (column) with the three intersecting rows(columns), as shown in figure 3. Each value must occur

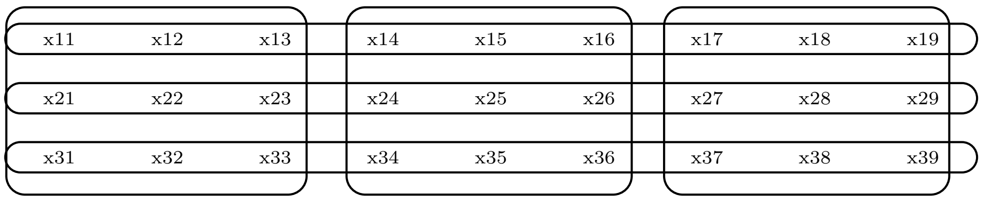

Fig. 3. Rows/Blocks interaction

exactly once in each row, but also in each block. For every value, we can express this as a matching problem between rows and blocks, which are the nodes in the bipartite graph shown in figure 4. There is an arc between a row and a block if the given value is in the domain of any of the three overlapping variables. If that edge does not belong to any matching, we can remove the value from the domain of all three variables. The algorithm works in the usual way, we first find a maximal matching, reorient the edges not in the matching and search for strongly connected components (SCC). Edges which are not in the matching and whose ends do not belong to the same SCC can be removed.

There are $6 * 4 = 54$ of these constraints.

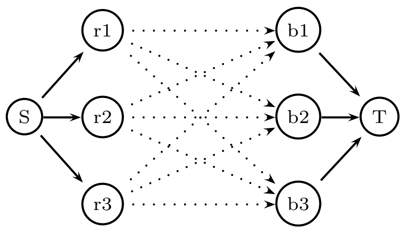

Fig. 4. Rows/Blocks matching problem

5.4 Rows/Columns/Blocks interaction

Ideally, we would like to capture the interaction of row, column and block constraints together. But we then have three sets of nodes to consider, so that a simple bipartite matching seems no longer possible. We would need a more complex flow model like the one considered in [27] to capture the interaction of the alldifferent constraints.

5.5 Shaving

Shaving[4] is a very simple, yet effective technique which considers the complete constraint set by trying to set variables to values in their domain. If the assignment fails, then this value can be removed from the domain, possibly eliminating many inconsistent values before starting search. In many (especially scheduling) problems the shaving is only applied to the minimal and maximal values in the domain. For a puzzle like Sudoku the order of the values is not significant, we therefore need to test all values in the domain. We use a simple program, which operates on a list of variables, and which removes any value in the domain of each variable which directly leads to a failure.

6 Propagation Schemes

The survey[3] describes different consistency methods for the alldifferent constraint. We combine them with some of the redundant constraints above to create the propagation schemes in table 3. The propagation schemes form a lattice as shown in figure 5, a dotted line indicates that in our evalution we haven't found an example to differentiate the two schemes.

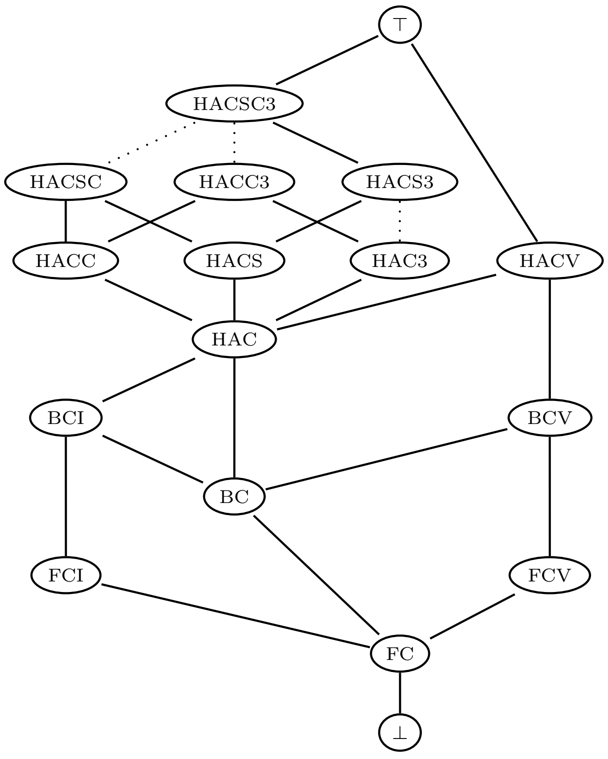

Fig. 5. Lattice of propagation schemes

FC	alldifferent with arc consistency for binary decomposition (forward checking)
FCI	forward checking with channeling
BC	alldifferent with bound-consistency
BCI	bound-consistency with channeling
HAC	alldifferent with hyper arc-consistency
HACS	HAC with same constraints
HACC	HAC with colored matrix
HAC3	HAC with 3rows/blocks interaction
HACSC	HAC with colored matrix and same constraints
HACS3	HAC with same constraints and 3rows/blocks interaction
HACC3	HAC with colored matrix and 3rows/blocks interaction
HACSC3	HAC with same constraints, colored matrix and 3rows/blocks interaction
FCV	alldifferent with forward checking plus shaving
BCV	alldifferent with bound-consistency plus shaving
HACV	alldifferent with hyper arc-consistency plus shaving

Table 3. Propagation Schemes

7 Evaluation

We decided to test our programs on different sets of published puzzle instances. In the UK, several newspapers print their daily Sudoku puzzle, and have published collections of these. We use sets from "The Times"[28], "The Daily Telegraph"[29], "The Guardian"[30], "The Independent"[31], "The Daily Mail"[32] and "The Sun"[33]. There are also magazines which specialize in Sudoku puzzles, of particular note is the Japanese puzzle publisher Nikoli[34–36], whose puzzles also appear in the UK as [37]. In addition, there are books of puzzle instances[38–41]. Usually, each instance is given a problem difficulty by the puzzle designer. In discussion boards, people often complain about the arbitrary way this difficulty is assigned. In addition we have found collections[42, 43] of 450 and 7611 puzzles which have only 17 presets, the currently smallest known number of presets for well posed problems. These are not classified by difficulty. The collection [44] contains mainly very hard problems.

In table 4 we summarize the results of our tests. Each entry is for a group of instances with the same grade. We present the source, the grade, the number of instances (Inst), and the percentage of problems solved search free for the different propagation schemes considered.

			Percentage Searchfree								
Source	Grade	Inst	FC	FCI	BC	BCI	HAC	HACS	HACC	HACSC	HAC3
[28]	easy	4	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[28]	mild	26	42.31	100.00	92.31	100.00	100.00	100.00	100.00	100.00	100.00
[28]	difficult	45	17.78	93.33	80.00	97.78	97.78	100.00	100.00	100.00	100.00
[28]	fiendish	25	0.00	36.00	28.00	80.00	88.00	100.00	88.00	100.00	100.00
[29]	gentle	32	21.88	100.00	93.75	100.00	100.00	100.00	100.00	100.00	100.00
[29]	moderate	66	7.58	100.00	81.82	100.00	100.00	100.00	100.00	100.00	100.00
[29]	tough	22	0.00	0.00	4.55	18.18	18.18	27.27	27.27	27.27	27.27
[29]	diabolical	12	0.00	0.00	0.00	8.33	8.33	16.67	16.67	16.67	16.67
[30]	easy	20	20.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[30]	medium	40	15.00	97.50	92.50	97.50	97.50	100.00	100.00	100.00	100.00
[30]	hard	40	0.00	45.00	42.50	90.00	92.50	100.00	97.50	100.00	100.00
[31]	elementary	10	80.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[31]	intermediate	50	48.00	98.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[31]	advanced	68	22.06	95.59	94.12	100.00	100.00	100.00	100.00	100.00	100.00
[32]	none	150	77.33	99.33	98.00	100.00	100.00	100.00	100.00	100.00	100.00
[33]	teasers	40	87.50	100.00	97.50	100.00	100.00	100.00	100.00	100.00	100.00
[33]	toughies	50	58.00	100.00	98.00	100.00	100.00	100.00	100.00	100.00	100.00
[33]	terminators	35	37.14	97.14	80.00	97.14	97.14	97.14	97.14	97.14	97.14
[34]	easy	37	48.65	100.00	97.30	100.00	100.00	100.00	100.00	100.00	100.00
[34]	medium	45	15.56	97.78	86.67	100.00	100.00	100.00	100.00	100.00	100.00
[34]	hard	17	0.00	100.00	47.06	100.00	100.00	100.00	100.00	100.00	100.00
[35, 36]	level2	18	11.11	100.00	83.33	100.00	100.00	100.00	100.00	100.00	100.00
[35, 36]	level3	23	4.35	100.00	73.91	100.00	100.00	100.00	100.00	100.00	100.00
[35, 36]	level4	24	8.33	100.00	66.67	100.00	100.00	100.00	100.00	100.00	100.00
[35, 36]	level5	25	0.00	100.00	60.00	100.00	100.00	100.00	100.00	100.00	100.00
[35, 36]	level6	28	0.00	35.71	42.86	89.29	89.29	100.00	92.86	100.00	100.00
[35, 36]	level7	29	0.00	6.90	37.93	79.31	79.31	100.00	93.10	100.00	100.00
[35, 36]	level8	29	0.00	3.45	20.69	65.52	68.97	100.00	93.10	100.00	100.00
[35, 36]	level9	20	0.00	0.00	25.00	55.00	65.00	100.00	80.00	100.00	100.00
[35, 36]	level10	14	0.00	0.00	0.00	28.57	28.57	92.86	57.14	100.00	92.86
[37]	easy	27	70.37	100.00	96.30	100.00	100.00	100.00	100.00	100.00	100.00
[37]	medium	20	10.00	95.00	80.00	95.00	95.00	100.00	95.00	100.00	100.00
[37]	hard	12	0.00	8.33	33.33	83.33	83.33	100.00	100.00	100.00	100.00
[41]	easy	10	70.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 4: Summary (cont'd)

			Percentage Searchfree								
Source	Grade	Inst	FC	FCI	BC	BCI	HAC	HACS	HACC	HACSC	HAC3
[41]	moderate	40	65.00	100.00	97.50	100.00	100.00	100.00	100.00	100.00	100.00
[41]	tricky	40	55.00	92.50	97.50	100.00	100.00	100.00	100.00	100.00	100.00
[41]	difficult	40	15.00	92.50	90.00	100.00	100.00	100.00	100.00	100.00	100.00
[41]	challenging	40	0.00	92.50	60.00	95.00	95.00	100.00	100.00	100.00	100.00
[40]	level1	40	80.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[40]	level2	40	80.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[40]	level3	40	80.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[40]	level4	40	72.50	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[40]	level5	41	24.39	100.00	92.68	100.00	100.00	100.00	100.00	100.00	100.00
[39]	easy	4	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[39]	harder	4	50.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[39]	even-harder	4	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[38]	easy	50	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[38]	medium	60	90.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
[38]	difficult	50	22.00	100.00	92.00	100.00	100.00	100.00	100.00	100.00	100.00
[38]	super-difficult	40	0.00	77.50	60.00	92.50	92.50	100.00	100.00	100.00	100.00
[42]	none	450	0.00	55.11	40.22	81.11	81.56	90.44	85.11	90.44	90.44
[43]	none	7611	0.00	48.69	29.01	69.82	70.28	85.51	75.27	85.55	85.51
[44]	none	100	0.00	0.00	3.00	5.00	5.00	14.00	10.00	14.00	14.00

Table 4: Summary

The programs seem to be working quite well in finding solutions without search, except for the "tough" and "diabolical" puzzles from [29] and those from [44]. These puzzles require either more powerful reasoning, or some form of search. Testing values with a shaving step works very well on these examples, as table 5 shows.

			Preset			Percentage Searchfree					
Source	Grade	Inst	Min	Avg	Max	FC	BC	HAC	FCV	BCV	HACV
[28]	easy	4	34	34.75	36	100.00	100.00	100.00	100.00	100.00	100.00
[28]	mild	26	27	29.12	32	42.31	92.31	100.00	100.00	100.00	100.00
[28]	difficult	45	23	27.09	30	17.78	80.00	97.78	97.78	100.00	100.00
[28]	fiendish	25	22	25.32	28	0.00	28.00	88.00	76.00	100.00	100.00
[29]	gentle	32	26	28.28	31	21.88	93.75	100.00	96.88	100.00	100.00
[29]	moderate	66	26	27.88	30	7.58	81.82	100.00	98.48	100.00	100.00
[29]	tough	22	26	28.05	31	0.00	4.55	18.18	95.45	100.00	100.00
[29]	diabolical	12	24	28.00	30	0.00	0.00	8.33	100.00	100.00	100.00
[30]	easy	20	24	28.05	31	20.00	100.00	100.00	100.00	100.00	100.00
[30]	medium	40	20	25.10	31	15.00	92.50	97.50	87.50	100.00	100.00
[30]	hard	40	20	22.88	27	0.00	42.50	92.50	55.00	100.00	100.00
[31]	elementary	10	28	31.20	34	80.00	100.00	100.00	100.00	100.00	100.00
[31]	intermediate	50	24	29.02	34	48.00	100.00	100.00	100.00	100.00	100.00
[31]	advanced	68	25	28.43	33	22.06	94.12	100.00	100.00	100.00	100.00
[32]	none	150	30	32.01	33	77.33	98.00	100.00	100.00	100.00	100.00
[33]	teasers	40	35	35.98	37	87.50	97.50	100.00	100.00	100.00	100.00
[33]	toughies	50	32	35.54	36	58.00	98.00	100.00	100.00	100.00	100.00
[33]	terminators	35	30	34.20	36	37.14	80.00	97.14	100.00	100.00	100.00
[34]	easy	37	24	29.54	37	48.65	97.30	100.00	100.00	100.00	100.00
[34]	medium	45	20	25.91	36	15.56	86.67	100.00	88.89	100.00	100.00
[34]	hard	17	21	25.29	29	0.00	47.06	100.00	58.82	100.00	100.00
[35, 36]	level2	18	22	24.39	28	11.11	83.33	100.00	100.00	100.00	100.00
[35, 36]	level3	23	20	23.87	26	4.35	73.91	100.00	65.22	100.00	100.00
[35, 36]	level4	24	20	23.63	29	8.33	66.67	100.00	66.67	100.00	100.00
[35, 36]	level5	25	20	23.96	30	0.00	60.00	100.00	80.00	100.00	100.00
[35, 36]	level6	28	20	23.61	28	0.00	42.86	89.29	60.71	100.00	100.00
[35, 36]	level7	29	20	24.10	32	0.00	37.93	79.31	58.62	100.00	100.00
[35, 36]	level8	29	21	23.28	29	0.00	20.69	68.97	41.38	100.00	100.00
[35, 36]	level9	20	22	23.95	27	0.00	25.00	65.00	40.00	100.00	100.00
[35, 36]	level10	14	22	24.64	29	0.00	0.00	28.57	35.71	100.00	100.00

Table 5: Shave Summary (cont'd)

			Preset			Percentage Searchfree					
Source	Grade	Inst	Min	Avg	Max	FC	BC	HAC	FCV	BCV	HACV
[37]	easy	27	24	29.41	37	70.37	96.30	100.00	100.00	100.00	100.00
[37]	medium	20	20	25.15	30	10.00	80.00	95.00	85.00	100.00	100.00
[37]	hard	12	20	24.58	28	0.00	33.33	83.33	58.33	100.00	100.00
[41]	easy	10	28	31.10	34	70.00	100.00	100.00	100.00	100.00	100.00
[41]	moderate	40	26	29.50	34	65.00	97.50	100.00	100.00	100.00	100.00
[41]	tricky	40	24	28.48	33	55.00	97.50	100.00	100.00	100.00	100.00
[41]	difficult	40	24	28.27	34	15.00	90.00	100.00	100.00	100.00	100.00
[41]	challenging	40	24	28.32	32	0.00	60.00	95.00	97.50	100.00	100.00
[40]	level1	40	32	34.58	36	80.00	100.00	100.00	100.00	100.00	100.00
[40]	level2	40	31	32.02	33	80.00	100.00	100.00	100.00	100.00	100.00
[40]	level3	40	29	29.93	31	80.00	100.00	100.00	100.00	100.00	100.00
[40]	level4	40	26	28.25	30	72.50	100.00	100.00	100.00	100.00	100.00
[40]	level5	41	24	25.80	31	24.39	92.68	100.00	92.68	100.00	100.00
[39]	easy	4	32	35.00	36	100.00	100.00	100.00	100.00	100.00	100.00
[39]	harder	4	29	30.50	32	50.00	100.00	100.00	100.00	100.00	100.00
[39]	even-harder	4	24	24.50	25	0.00	100.00	100.00	50.00	100.00	100.00
[38]	easy	50	35	43.32	48	100.00	100.00	100.00	100.00	100.00	100.00
[38]	medium	60	26	35.48	40	90.00	100.00	100.00	100.00	100.00	100.00
[38]	difficult	50	24	27.84	30	22.00	92.00	100.00	98.00	100.00	100.00
[38]	super-difficult	40	24	27.68	31	0.00	60.00	92.50	92.50	100.00	100.00
[42]	none	450	17	17.00	17	0.00	40.22	81.56	1.56	100.00	100.00
[43]	none	7611	17	17.00	17	0.00	29.01	70.28	0.62	99.89	100.00
[44]	none	100	17	21.51	31	0.00	3.00	5.00	4.00	93.00	100.00

Table 5: Shave Summary

We can see that all problems can be solved by using shaving techniques combined with hyper arc-consistency, and nearly all by combining bound-consistency with shaving. Using only forward checking together with shaving works well as long as enough presets are given. On the minimal problems it nearly always fails to find solutions.

Table 6 shows the execution times needed for solving the puzzles. We use a simple most-constrained/indomain, complete search routine (L) together with our different propagation schemes. The models with channeling are penalized by our naive hyper arc-consistent implementation of the inverse constraint. The times show the minimal, maximal, average and median times over all examples from the data sets above. All times are in milliseconds with ECLiPSe 5.8 on a 1.5 GHz/1Gb laptop. The best results are obtained using bound-consistent alldifferent constraints together with one shaving step before search.

Program	Min	Max	Avg	Median
HACV	0	7281	349	321
FCL	0	25077	745	140
BCL	0	3715	51	40
FCIL	340	10686	761	741
BCIL	190	2053	702	730
HACL	0	1812	322	330
FCVL	0	25026	725	111
BCVL	0	771	66	40

Table 6. Runtime (ms)

We were also interested if the published puzzles were locally minimal, i.e. did not contain redundant hints. Table 7 shows the results on some tests. Most of the published puzzles are not locally minimal, they often contain more than 10 redundant hints which can be removed without loosing the uniqueness of the solution. The instances from [42, 43] are an exception, they were collected to be locally minimal. Most of the hard puzzles from [44] are also minimal. We computed this reduction with a simple, committed choice program. We search for a hint that can be eliminated; if one is found, we commit to this choice and try to find additional redundant hints. The method does not guarantee minimality of the reduction, but leads to locally minimal instances.

			Preset			Locally	Reduced		
Source	Grade	Inst	Min	Avg	Max	Minimal	Min	Avg	Max
[28]	easy	4	34	34.75	36	0	23	23.75	24
[28]	mild	26	27	29.12	32	0	22	23.62	25
[28]	difficult	45	23	27.09	30	1	22	23.89	25
[28]	fiendish	25	22	25.32	28	6	21	23.80	26
[29]	gentle	32	26	28.28	31	0	23	24.03	26
[29]	moderate	66	26	27.88	30	0	22	23.77	26
[29]	tough	22	26	28.05	31	0	22	24.32	26
[29]	diabolical	12	24	28.00	30	0	23	24.42	26
[30]	easy	20	24	28.05	31	0	22	22.95	25
[30]	medium	40	20	25.10	31	6	19	22.30	25
[30]	hard	40	20	22.88	27	14	19	21.80	26
[31]	elementary	10	28	31.20	34	0	21	23.80	26
[31]	intermediate	50	24	29.02	34	0	22	23.94	27
[31]	advanced	68	25	28.43	33	0	22	23.79	26
[32]	none	150	30	32.01	33	0	22	23.84	26
[33]	teasers	40	35	35.98	37	0	22	24.35	26
[33]	toughies	50	32	35.54	36	0	22	24.68	27
[33]	terminators	35	30	34.20	36	0	22	24.91	27
[34]	easy	37	24	29.54	37	0	21	23.11	25
[34]	medium	45	20	25.91	36	4	20	22.64	26
[34]	hard	17	21	25.29	29	3	20	22.82	25
[35, 36]	level2	18	22	24.39	28	2	20	22.72	25
[35, 36]	level3	23	20	23.87	26	7	20	22.35	24
[35, 36]	level4	24	20	23.63	29	6	19	22.13	25
[35, 36]	level5	25	20	23.96	30	2	19	22.40	25
[35, 36]	level6	28	20	23.61	28	9	20	22.54	24
[35, 36]	level7	29	20	24.10	32	6	19	22.38	25
[35, 36]	level8	29	21	23.28	29	10	19	22.24	24
[35, 36]	level9	20	22	23.95	27	7	22	22.90	25
[35, 36]	level10	14	22	24.64	29	4	22	23.36	25
[37]	easy	27	24	29.41	37	1	20	23.26	26
[37]	medium	20	20	25.15	30	2	20	22.20	24
[37]	hard	12	20	24.58	28	2	20	22.83	25
[41]	easy	10	28	31.10	34	0	22	23.70	25
[41]	moderate	40	26	29.50	34	0	22	24.07	27
[41]	tricky	40	24	28.48	33	1	21	23.57	25
[41]	difficult	40	24	28.27	34	0	22	24.13	26
[41]	challenging	40	24	28.32	32	0	22	24.05	26
[40]	level1	40	32	34.58	36	0	21	23.55	26
[40]	level2	40	31	32.02	33	0	21	23.55	25
[40]	level3	40	29	29.93	31	0	22	23.40	25
[40]	level4	40	26	28.25	30	0	21	23.02	25
[40]	level5	41	24	25.80	31	0	20	22.68	24
[39]	easy	4	32	35.00	36	0	23	24.00	25
[39]	harder	4	29	30.50	32	0	23	24.00	26
[39]	even-harder	4	24	24.50	25	1	21	22.75	25
[38]	easy	50	35	43.32	48	0	22	24.22	28

Table 7: Reduction Summary (cont'd)

			Preset			Locally	Reduced		
Source	Grade	Inst	Min	Avg	Max	Minimal	Min	Avg	Max
[38]	medium	60	26	35.48	40	0	22	23.50	25
[38]	difficult	50	24	27.84	30	0	22	24.22	26
[38]	super-difficult	40	24	27.68	31	1	22	24.40	29
[42]	none	450	17	17.00	17	450	17	17.00	17
[44]	none	100	17	21.51	31	82	17	21.14	26

Table 7: Reduction Summary

8 Problem Generator

We have seen that with different variants of our model we can solve puzzle instances of varying difficulty. Can we use the same techniques to generate puzzles as well? We can consider three possible approaches to the puzzle generation:

1. We start with a random partial assignment of values to the grid and check if this is a well posed puzzle, and whether it is search free for some strategy. This does not give a guarantee to find solutions of a predetermined difficulty, in addition the partial assignment may be inconsistent and we have to decide a priori how many preset values we want to generate.
2. We start with an empty grid and add preset values one by one until there is a unique solution or we detect inconsistency. If required, we can add redundant hints until the problem is search free for a given propagation scheme. We will need a backtracking mechanism to escape inconsistent assignments.
3. We start with a full grid and remove values until the problem is no longer well posed or no longer search free for some propagation scheme. This will lead to problems which are locally minimal and well posed. Generating initial starting solutions is quite simple, but may still require some backtracking.

The third approach can either generate well posed, locally minimal problems or can be used to find search-free puzzles of a given difficulty grade that can not be further reduced without loosing the search free property. This bottom-up problem generation has been used in [16] to generate solvable problem instances of quasi-group completion for CSP or SAT solvers. In that case one starts with a completed quasi-group, removes a number of values from the grid, and is left with a feasible quasi-group completion problem. The generated problem may have multiple solutions, but that is ok in the given context. As we are interested in well posed problems, we have to make the removal steps one by one as long as the solution stays unique.

9 Conclusions

In this paper we have discussed a constraint formulation of the Sudoku puzzle, and have seen that we can use different modelling techniques to find puzzle solutions without search. The problem is closely related to the quasi-group completion problem, which has been extensively studied in the constraint community. The additional alldifferent constraints on the major blocks give us more chances to find redundant constraints that can help in the solution process. Our methods can solve many, but not all, published examples without search, some requiring a shaving technique. We can use the model to generate puzzles of a given difficulty, again following techniques developed for quasi-group completion. Sudoku puzzles are not only an interesting addition to the problem set for developing constraint techniques, but also provide a unique opportunity to make more people interested in constraint programming.

Acknowledgment

We want to thank J. Schimpf for suggesting a more elegant shaving implementation, and G. Stertenbrink for pointing out additional references and data sets.

References

1. various: Sudoku wikipedia entry. `http://en.wikipedia.org/wiki/Sudoku` (2005)
2. Apt, K.: Principles of Constraint Programming. Cambridge University Press (2003)
3. van Hoeve, W.: The alldifferent constraint: A survey. `sherry.ifi.unizh.ch/article/vanhoeve01alldifferent.html` (2001)
4. Torres, P., Lopez, P.: 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). (2000) 181–186
5. Beldiceanu, N., Carlsson, M., Rampon, J.: Global constraint catalog. Technical Report T2005:08, SICS (2005)
6. Regin, J., Gomes, C.: The cardinality matrix constraint. In: CP 2004. (2004)
7. Beldiceanu, N., Katriel, I., Thiel, S.: GCC-like restrictions on the same constraint. In: Recent Advances in Constraints (CSCLP 2004). Volume 3419 of LNCS., Springer-Verlag (2005) 1–11
8. Lauriere, J.: A language and a program for stating and solving combinatorial problems. Artificial Intelligence **10** (1978) 29–127
9. Haralick, R., Elliott, G.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence **14** (1980) 263–313
10. Smith, B., Brailsford, S., Hubbard, P., Williams, H.: The progressive party problem: Integer linear programming and constraint programming compared. Constraints **1** (1996) 119–138
11. Harvey, W.: The fully social golfer problem. In: SymCon'03: Third International Workshop on Symmetry in Constraint Satisfaction Problems. (2003) 75–85
12. Jefferson, C., Miguel, A., Miguel, I., Tarim, A.: Modelling and solving English peg solitaire. In: Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'03). (2003) 261–275
13. Simonis, H.: Interactive problem solving in ECLiPSe. ECLiPSe User Group Newsletter (2003)
14. Stergiou, K., Walsh, T.: The difference all-difference makes. In: IJCAI-99. (1999)
15. Shaw, P., Stergiou, K., Walsh, T.: Arc consistency and quasigroup completion. In: ECAI-98 workshop on binary constraints. (1998)

16. Achlioptas, D., Gomes, C., Kautz, H., Selman, B.: Generating satisfiable problem instances. In: AAAI-2000. (2000)
17. Dincbas, M., Simonis, H.: APACHE - a constraint based, automated stand allocation system. In: Advanced Software Technology in Air Transport (ASTAIR'91). (1991) 267–282
18. Simonis, H.: Building industrial applications with constraint programming. In Common, H., Marche, C., Treinen, R., eds.: Constraints in Computational Logics - Theory and Applications, Springer Verlag (2001)
19. Eppstein, D.: Nonrepetitive paths and cycles in graphs with application to Sudoku. ACM Computing Research Repository (2005)
20. Wallace, M., Novello, S., Schimpf, J.: ECLiPSe : A platform for constraint logic programming. ICL Systems Journal **12** (1997)
21. Cheadle, A.M., Harvey, W., Sadler, A.J., Schimpf, J., Shen, K., Wallace, M.G.: ECLiPSe: An introduction. Technical Report IC-Parc-03-1, IC-Parc, Imperial College London (2003)
22. Dincbas, M., Simonis, H., Van Hentenryck, P.: Solving large combinatorial problems in logic programming. J. Log. Program. **8** (1990) 75–93
23. Puget, J.F.: A fast algorithm for the bound consistency of alldiff constraints. In: AAAI. (1998) 359–366
24. Lopez-Ortiz, A., Quimper, C.G., Tromp, J., van Beek, P.: A fast and simple algorithm for bounds consistency of the alldifferent constraint. In: IJCAI. (2003)
25. Regin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: AAAI. (1994) 362–367
26. Cheng, B.M.W., Lee, J.H.M., Wu, J.C.K.: Speeding up constraint propagation by redundant modeling. In Freuder, E.C., ed.: CP. Volume 1118 of Lecture Notes in Computer Science., Springer (1996) 91–103
27. Beldiceanu, N., Katriel, I., Thiel, S.: Filtering algorithms for the same constraint. In: Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP'AI'OR) 2004. (2004)
28. Gould, W.: The Times Su Doku, Book 1. The Times (2005)
29. Mepham, M.: The Daily Telegraph Sudoku. The Daily Telegraph (2005)
30. n/a: The Guardian Sudoku Book 1. Guardian Books (2005)
31. Norris, H.: The Independent Book of Sudoku Volume 1. The Independent (2005)
32. n/a: Daily Mail Sudoku. Associated Newspapers Ltd (2005)
33. Perry, J.: The Sun Doku. The Sun (2005)
34. various: Sudoku 1. Nikoli (1988) In Japanese.
35. various: Gekikara Sudoku 1. Nikoli (2004) In Japanese.
36. various: Gekikara Sudoku 2. Nikoli (2004) In Japanese.
37. n/a: Sudoku - the original hand-made puzzles. Puzzler **1** (2005)
38. Vorderman, C.: Carol Vorderman's How to do Sudoku. Random House (2005)
39. Wilson, R.: How to solve Sudoku. The Infinite Ideas Company (2005)
40. Sinden, P.: The Little Book of Sudoku Volume 1. Michael O'Mara Books (2005)
41. Huckvale, M.: The Big Book of Su Doku. Orion (2005)
42. Royle, G.: Minimum Sudoku. `http://www.csse.uwa.edu.au/~gordon/sudokumin.php` (2005)
43. Royle, G.: Minimum Sudoku. `http://www.csse.uwa.edu.au/~gordon/sudoku17` (2005)
44. Stertenbrink, G., Meyrignac, J.C.: 100 Sudoku problems. `http://magictour.free.fr/top100` (2005)