

Survey of the Applications of Artificial Intelligence Techniques to the Sudoku Puzzle

Fundamentals of Intelligent Systems

Dr. Gursel Serpen, Fall 2009

Robert C. Green II

December 10, 2009

Contents

1	Introduction	3
2	Problem Statement	4
3	Review of Recent Research	4
3.1	Steepest Hill Ascent	5
3.2	Genetic Algorithms	6
3.3	Harmony Search	10
3.4	Chromatic Polynomials	11
3.5	Rewriting Rules	12
3.6	Particle Swarm Optimization	13
3.7	Bee Colony Optimization	14
3.8	Artificial Immune System Optimization	14
3.9	Neural Networks	15
3.10	Simulated Annealing	16
3.11	Quantum Simulated Annealing	16
3.12	Hybrid Techniques	17
3.13	Constraint Problems	17
3.14	SAT Methods	19
3.15	Message Passing	22
3.16	Backtracking	24
3.17	Linear Programming	25
4	Open Research Problems	27

Abstract

Puzzles and games have always been two fields ripe for the application of artificial intelligence and intelligent systems techniques. Part of the allure of this field is the wide variety of games of very high difficulty (such as chess, go, backgammon, poker, and the like), some of which are NP-complete. The past few years have brought about a very popular NP-complete puzzle called sudoku.

The typical sudoku grid is a 9x9 grid split into 9 3x3 squares. The rules for filling the cells in the game are very simple: Every row, column, and square (of the 3x3 variety) must be filled with each of the numbers 1 through 9 while no number can appear more than once in any row, column, or square. At the beginning of each game enough of the squares are pre-filled in order to guarantee that there is a single solution to the problem. Besides this basic puzzle there are also many variants that combine or manipulate sudoku boards in multiple ways.

As many researchers across multiple fields (including computer science, artificial intelligence, mathematics, operations research, physics, and others) have proposed algorithms to solve sudoku puzzles in many different ways, it is only appropriate to complete a survey paper comparing and contrasting the algorithms and techniques that have been developed to generate and solve sudoku puzzles.

This paper comes to the conclusion that while many attempts have been made to solve the sudoku problem, most fall short as they do not solve the problem efficiently though some techniques, such as linear programming and constraint satisfaction, solve the problem extremely quickly and efficiently. Though the effort of solving sudoku puzzles with artificial intelligence techniques is not in vain as these investigations sometimes divulge helpful information with regards to solving other NP complete problems.

1 Introduction

Puzzles and games have always been two fields ripe for the application of artificial intelligence and intelligent systems techniques. Part of the allure of this field is the wide variety of

games of very high difficulty (such as chess, go, backgammon, poker, and the like), some of which are NP-complete. The past few years have brought about a very popular NP-complete (Yato, 2003) puzzle called sudoku.

The typical sudoku grid is a 9x9 grid split into 9 3x3 squares. The rules for filling the cells in the game are very simple: Every row, column, and square (of the 3x3 variety) must be filled with each of the numbers 1 through 9 while no number can appear more than once in any row, column, or square. At the beginning of each game enough of the squares are pre-filled in order to guarantee that there is a single solution to the problem. In total there are roughly 6,670,903,752,021,072,936,960 to these puzzles. Besides this basic puzzle there are also many variants that combine or manipulate sudoku boards in multiple ways.

2 Problem Statement

As many researchers across multiple fields (including computer science, artificial intelligence, mathematics, operations research, physics, and others) have proposed algorithms to solve sudoku puzzles in many different ways, it is only appropriate to complete a survey paper comparing and contrasting the algorithms and techniques that have been developed to generate and solve sudoku puzzles.

This paper will give an overview of the state of the art in artificial intelligence techniques that have been applied to sudoku including search based techniques, linear programming, genetic algorithms, and other AI techniques. The remainder of this paper cover both current research and open problems concerning the application of intelligent systems to sudoku.

3 Review of Recent Research

As multiple methods of solving sudoku puzzles have been proposed, this section will attempt to cover state of art methods for solving sudoku puzzles. The algorithms used will vary across

a wide range of fields that blend optimization, intelligent systems, artificial intelligence, and optimization in many ways that are both thought provoking and challenging.

3.1 Steepest Hill Ascent

(K. et al., 2007) proposes a search based solution in order to solve sudoku puzzles. The algorithm itself is a modified steepest hill ascent. Thus the algorithm must have both an objective function (a heuristic) and operators that will define the successor states.

The objective function is noted as $f = \sum_{i=1}^9 w_i f_i$ where f_i is a heuristic measuring distance to the goal and w_i is a weighted coefficient. This objective function is actually extended to include two parts $f_1 w_1$ and $f_2 w_2$. f_1 is the count of differing digits in each row and column and f_2 becomes a measure of the number of collisions in each row and column.

The heuristic functions are designed as listed in equations 7 and 8.

$$f_1 = \sum_{i=1}^9 |\{x|x = a_{ij}, j = 1..9\}| + \sum_{j=1}^9 |\{x|x = a_{ij}, i = 1..9\}| \quad (1)$$

$$f_2 = \sum_{g_{pq} \in G} |\{i|a_{iq} = g_{pq}, i = 1..9, i \neq p\} \cup \{j|a_{pj} = g_{pq}, j = 1..9, j \neq q\}| \quad (2)$$

w_1 becomes 1 and w_2 becomes -1. This should be obvious as the objective function with the highest value will be chosen for the successor state. These chosen functions infer that the value of the objective will increase when there are less collisions and less similar digits.

This basic algorithm and objective function work, but in order to improve the algorithm the initial state is adjusted. This is done through a greedy algorithm that is used to place the digits 1-9 in each 3x3 grids of the sudoku board. This process may result in many collisions - rows or columns where the same number appears more than once. Swapping of digits is then used to reduce collisions across the entire board.

The final adjustment made to this method in order to improve performance is to add random a restart if the objective function remains the same for ten iterations. This is a typical methodology used when applying steepest hill ascent to a problem.

Overall the algorithm works, but as detailed in the table below, it is most likely not very usable, as are most heuristic functions that are applied to the sudoku puzzle (this is obviously the case as there are 3,546,146,300,288 goal state solutions to sudoku in general).

Level	Avg. Givens	Avg. Iterations	Avg. Restarts	Avg. Time to Solve (s)
Easy	31.12	233.96	0.12	1443.99656
Medium	25.84	392.08	1.08	1256.01920
Hard	23.44	724.60	2.80	3110.32
Hardest	20.33	3587.58	3.39	34921.31

3.2 Genetic Algorithms

(Mantere and Koljonen, 2007) suggests the design of a genetic algorithm for use in solving sudoku puzzles. The puzzle is represented by a block of chromosomes that is an array of 81 integers. This array is obviously split into 9 blocks each of length 9 to represent the sudoku puzzle. Any crossover occurs between whole 3x3 grids and any mutations occur only inside of 3x3 grids. Mutations are always of the type swap, 3-swap, or insertion. The objective function is a combination of:

- Each row and column must contain a number between 1 and 9
- Row and column sums must be 45 (Equation 1)
- Row and column products must be 9! (Equation 2)
- Each row must contain the numbers 1...9 (Equation 3)

and is defined as:

$$f(x) = 10 * (\sum_i g_{i1}(x) + \sum_j g_{j1}(x)) + \sum_i \sqrt{g_{i2}(x)} + \sum_j \sqrt{g_{j2}(x)} + 50 * (\sum_i g_{i3}(x) + \sum_j g_{j3}(x)) \quad (3)$$

where the constraints are defined using the following equations:

$$g_{i1} = \left| 45 - \sum_{j=1}^9 x_{i,j} \right| \quad g_{j1} = \left| 45 - \sum_{i=1}^9 x_{i,j} \right| \quad (4)$$

$$g_{i2} = \left| 9! - \prod_{j=1}^9 x_{i,j} \right| \quad g_{j2} = \left| 9! - \prod_{i=1}^9 x_{i,j} \right| \quad (5)$$

$$A = 1, 2, 3, 4, 5, 6, 7, 8, 9 \quad g_{i3} = |A - x_i| \quad g_{j3} = |A - x_j| \quad (6)$$

The GA is tested using a mutation probability of 0.12 with a population of 100. This formulation is run against 10 puzzles that have between 23 and 36 symmetrical givens of all difficulty levels. The stopping condition for the GA was either finding the optimal solution or reaching 100,000 iterations. An attempt was made to solve each puzzle 100 times. The results are reproduced as follows:

Difficulty rating	Givens	Count	Min	Max	Average	Median	Stdev
New	0	100	206	3824	1390.4	1089	944.67
1 star	33	100	184	23993	2466.6	917	3500.98
2 stars	30	69	733	56484	11226.8	7034	11834.68
3 stars	28	46	678	94792	22346.4	14827	24846.46
4 stars	28	26	381	68253	22611.3	22297	22429.12
5 stars	30	23	756	68991	23288.0	17365	22732.25
Easy	36	100	101	6035	768.6	417	942.23
Challenging	25	30	1771	89070	25333.3	17755	23058.94
Difficult	23	4	18999	46814	20534.3	26162	12506.72
Super difficult	22	6	3022	47352	14392	6722	17053.33

Though the algorithm does solve most soduko problems, it fails to do so efficiently.

(Mantere and Koljonen, 2008) suggests an improved genetic algorithm based on (Mantere and Koljonen, 2007). The defining changes include a population size of 21 and removing the 3-swap and insertion mutation. The objective function is also changed in order to only include:

- Each row and column must contain a number between 1 and 9
- Solutions that maintain optimal status are aged to show favoritism
- A row or column may not contain a duplicate of a given number

These pieces of the objective function are mathematically defined as:

$$P_x = \sum_{i=1}^8 \sum_{j=1}^8 \sum_{ii=i+1}^9 \sum_{jj=j+1}^9 [(x_{i,j} == x_{ii,j}) + (x_{i,j} == x_{i,jj})] \quad (7)$$

$$\text{If } Best(\text{generation}(i)) = Best(\text{generation}(i - 1)) \text{ then } Value(Best)_+ = 1 \quad (8)$$

$$P_g = \sum_{i=1}^9 \sum_{j=1}^9 (x_{ij} == g_{ij}) \quad (9)$$

An additional change to this algorithm is the addition of a belief space. This belief space is a 9x9x9 cube that represents the 9x9 sudoku board with the last dimension representing each digit that is a possibility in each of the 81 cells of the sudoku board. The digits in this cube are progressively weighted in order to favor more optimal solutions. This belief cube is then used in the generation of future generations.

This technique improves somewhat over their previous attempt, but only very little (2.63% using average solving efficiency to 4.79% using the same measure proportionally weighted).

(Nicolau and Ryan, 2006) suggests a formulation of sudoku using genetic algorithms with grammatical evolution. This type of GA adds phenotype measures to the typical chromosome representation. In this way, both the genotype and phenotype evolve together. Also, unlike a typical GA, the members of the genotype are dependent upon one another.

The basis of this method of solving sudoku is to evolve a set of instructions that will solve the puzzle. Each member of the population generates a set of instructions that is then

passed on to the puzzle. The puzzle applies the instructions and then returns a measure of goodness. This means that the actual puzzle is never available to the algorithm. Logical instructions suggested in this work include (a full description of these techniques can be found in the paper):

- Last Remaining
- Slice and Dice
- Column Fill
- Row Fill
- Raising Numbers

Each member of the population will contain a set of 81 instructions, each of which will contain information regarding what rule to apply, which region to apply the rule to, and what number to place. The encoding scheme used uses 1620 bits for each set of instructions.

The objective function is defined as:

$$f_i = \begin{cases} k X (82 - i) & \text{if successful} \\ coverage - 9 & \text{if unsuccessful} \end{cases}$$

(10)

where k is a constant and *coverage* is a measure of how many cells were ruled out when unsuccessfully trying to place a number in a given region.

The algorithm itself seems to take between 14 and 301 generations in order to solve the sudoku puzzle, but there are some cases where it cannot solve the puzzle at all. In those cases less than half of the empty squares are filled with numbers. As no time measurement is given it seems that for a GA this algorithm performs as well as expected though it does

not solve all cases. The detailed results appear in the following table:

Puzzle	#111	#112	#113	#114	#115	#116	#117	#118	#119	#120
Avg. placements	53/53	51/51	53/53	53/53	51/51	21/53	13/53	54/54	51/51	51/51
Avg. generation	238	181	166	210	123	55	14	301	188	135
Successful runs	30	30	30	30	30	0	0	30	30	30

3.3 Harmony Search

The harmony search is a very intriguing search algorithm proposed in (Geem et al., 2001) that mimics the characteristics of a musician. The characteristics of a musician that are mimicked are memory consideration, pitch, and random consideration. (Geem, 2007) proposes a method of applying the harmony search to solving sudoku puzzles.

In order to model a sudoku puzzle using the harmony search, an objective function must first be developed. The objective function, in this case, is simplistic: Starting at the first row, all of the numbers in a row are added. 45 is then subtracted from that sum (45 is the sum of 1-9, the only digits allowed in sudoku). This is accomplished for every row and then the result of this summation over every row is summed together. This process is repeated for the columns and the mini-grids. These values are then added together and minimized. Letting x_{ij} = the cell at row i column j and B_k = the set of coordinates for block k allows this objective to be written as:

$$\text{Minimize } Z = \sum_{i=1}^9 \left| \sum_{j=1}^9 x_{ij} - 45 \right| + \sum_{j=1}^9 \left| \sum_{i=1}^9 x_{ij} - 45 \right| + \sum_{k=1}^9 \left| \sum_{(l,m) \in B_k} x_{lm} - 45 \right| \quad (11)$$

Once the objective is in place, the successor functions of the search must be defined. In this case the successor functions are successor by memory consideration, successor by random consideration, and successor by pitch consideration. Memory and random consideration each pick a new value based on a probability: the former based on values existing in the

current state and the latter based purely on stochasticity. Pitch consideration is a bit more complex as change in value is actually calculated and the applied and propagated throughout a solution in an attempt to develop a better 'harmony'.

The algorithm itself consists of 3 steps:

- Create a random harmony memory space (a solution space full of multiple states)
- Improvise a new harmony using the successor functions named above.
- Compare the new harmony against currently stored harmonies. If it is better, add it to the solution space and discard the worst solution
- Iterate until the iteration limit is reached

The performance of this algorithm is mediocre at best. In most cases it solves this puzzle in less than 300 iterations and under 40 seconds. Though there are cases, especially involving the toughest puzzles, where the algorithm gets trapped in local minima. A solid suggestion in this case would be twofold: Tweak the current heuristic so that it improves the solution time and improve the harmony search so that it has a chance to break out of local minima, possibly by using a hybrid solution.

3.4 Chromatic Polynomials

(Herzberg and Murty, 2007) suggests formulating the sudoku puzzle as a graph coloring problem. The coloring problem is solved by formulating sudoku as graph with 81 vertices where only adjacent nodes are connected. Each node represents a single cell of the sudoku board. This formulation is shown to be one of proper coloring. The number of colors needed to color such a graph is known as the chromatic number. It is then shown that there are a polynomial ways of coloring this type of graph. It is also shown that a sudoku puzzle of order n needs only n^2 colors in order to color the graph and that any sudoku puzzle that can be colored with 7 colors has no unique solution.

This paper does not actually attempt to solve the sudoku puzzle and it does not provide any measurable results. It simply poses the formulation of sudoku as a graph coloring problem. An interesting extension would be the application and systematic evaluation of this technique to actual sudoku puzzles.

3.5 Rewriting Rules

(Santos-García and Palomino, 2007) suggests a method of solving sudoku using logic to mimic human intelligence. This method is the application of rewriting rules to the game of sudoku. For the most part, this method consists of building up a set of rules that allows rewriting of the sudoku puzzle at different stages. As the author says, this is advantageous because it is a more natural representation of the puzzle. The initial step in the formulation for this solution is defining the sudoku board and the variables used in the solution. Initially, a sudoku S of order n is defined as a collection of objects C_{ij} , one for each cell at row i and column j . Each of these objects has the following attributes:

- $G_{ij} = \sqrt{n} * \text{int}((i - 1)/\sqrt{n}) + \text{int}((j - 1)/\sqrt{n}) + 1$ is the grid to which the cell belongs
- P_{ij} is the set of possible numbers that may occur in a cell.
- N_{ij} is the number of elements in P_{ij}

After this representation is defined, the following rules are used for rewriting purposes (mathematical details of the rules may be found in the full paper):

- If only one number is possible in a cell, then we remove this number from the set of possible numbers in all the other cells in the same row, column or grid.
- If two cells in the same row (column or grid) have the same set of possible numbers and its cardinality is 2, then those numbers can be removed from the sets of possible numbers of every other cell in the same row (column or grid).
- If three cells in the same row (column or grid) have the same set of possible numbers and its cardinality is 3, then those numbers can be removed from the sets of possible numbers of every other cell in the same row, column, or grid

- When a number is not possible in any cell of a row (column or grid) but one, and the cardinality of the set of possible numbers for this cell is greater than one, then this set can become a singleton set containing that number.
- When two numbers p_1 and p_2 are not possible in any cell of a row (column or grid) but two, and the sets of possible numbers for these cells have cardinality greater than two, then these sets can become p_1, p_2 .
- If, in a given grid, a number is only possible in one row (or column), then that number can be removed from the set of possible numbers in all the cells in that same row (or column) but different grid.
- This rule splits a sudoku when none of the other rules can be applied. We select a cell with a minimum number (greater than 1) of possible numbers. Then a sudoku is created with the first possible number and another one with the remaining possible numbers
- This rule is the particular case of the previous one when the number of possible numbers in a cell is equal to 2.

These rules are then implemented in a high level language called Maude that supports rewriting logic. This method of solving sudoku absolutely works but, as the author admits, does not work well when compared with other techniques. An improvement upon this method would be increasing the processing speed in order to arrive at a solution more quickly. Though, it is beneficial that a problem can be stated in such a human-like manner using rules that have been written in order to mimic human intelligence.

3.6 Particle Swarm Optimization

(Jilg and Carter, 2009) proposes a particle swarm optimization search technique known as geometric particle swarm optimization. This technique of search produces a search space of particles where in order to generate a new position, or solution in this case, the particle uses the current global optimum, its personal optimum, and its current solution in order to form

a convex hull of search spaces in which to search.

Particle Swarm optimization does not seem to work well as only 19 out of 40 grids attempted were solved. Perhaps the parameters of this algorithm could be adjusted to produce better results in the future.

3.7 Bee Colony Optimization

(Jilg and Carter, 2009) proposes a bee colony optimization technique in order to solve the sudoku puzzle. Bee colony optimization is a very intriguing search technique as it searches both locally and in parallel. To begin the algorithm, bees are randomly sent out to explore the search space. Each bees fitness is then calculated. The best of the bees remain in position. The rest of the population is redistributed so that there is a concentration of bees very near each of the best solutions. This process continues until iteration limit or convergence occurs.

Bee colony optimization does not seem to work well as only 22 out of 40 grids attempted were solved. Again, perhaps the parameters of this algorithm could be adjusted to produce better results in the future.

3.8 Artificial Immune System Optimization

(Jilg and Carter, 2009) applies the artificial immune system optimization (AIS) technique to the sudoku puzzle. While the implementation details of this method are far too complex for this paper, it should be noted that the algorithm did not perform well and solved only 6 of 40 puzzles with an average time of over 200 seconds. As AIS is an extremely complex mapping and methodology, a possibility for future work would be continued tweaking of parameters as well as applications of the newest version of AIS, vaccine-enhanced AIS (Woldemariam and Yen, 2010).

3.9 Neural Networks

(Yue and Lee, 2006) suggests an energy driven quantum neuron (Q'tron) neural network (NN) model to solve the sudoku puzzle. This method was chosen because unlike many neural networks, Q'tron NNs have the ability to escape local minima. This is important because local minima have a tendency to be an issue in sudoku puzzles. Q'tron NNs escape local minima through the use of injected noise. The value of this injected noise is typically determined via use of a cooling schedule in a similar manner to simulated annealing. The Q'tron model used in this paper also uses a known energy system. This simply means that eventually the Q'tron NN will settle down to a minima.

The puzzle itself is modeled as 9x9x9 cube. The constraints are a basic integer linear program and as such will not be covered here. The important piece is the energy function that the Q'tron NN minimizes. This energy function is the sum of the energies of the different pieces of the puzzle including rows, columns, regions, and symbols where each of these individual energies is nothing more than the summation of the integer variables affecting that particular piece of the puzzle. This is mathematically defined using the following equations:

$$E_{sudoku} = E_{row} + E_{col} + E_{rgn} + E_{sym} \quad (12)$$

$$E_{row} = \frac{1}{2} \sum_{i=1}^9 \sum_{k=1}^9 \left(\sum_{j=1}^9 Q_{ijk} - 1 \right)^2 \quad (13)$$

$$E_{col} = \frac{1}{2} \sum_{i=1}^9 \sum_{k=1}^9 \left(\sum_{j=1}^9 Q_{ijk} - 1 \right)^2 \quad (14)$$

$$E_{rgn} = \frac{1}{2} \sum_{m=0}^2 \sum_{n=0}^2 \sum_{k=1}^9 \left(\sum_{i=1}^3 \sum_{j=1}^3 Q_{3m+i,3n+j,k} - 1 \right)^2 \quad (15)$$

$$E_{sym} = \frac{1}{2} \sum_{i=1}^9 \sum_{k=1}^9 \left(\sum_{j=1}^9 Q_{ijk} - 1 \right)^2 \quad (16)$$

All of these equations can be combined into the following equation for use:

$$E_{cube} = -\frac{1}{2} \sum_{i=1}^9 \sum_{j=1}^9 \sum_{k=1}^9 \sum_{l=1}^9 \sum_{m=1}^9 \sum_{n=1}^9 (a_{ijk} Q_{ijk}) T_{ijk,lmn} (a_{lmn} Q_{lmn}) - \sum_{i=1}^9 \sum_{j=1}^9 \sum_{k=1}^9 I_{ijk} (a_{ijk} Q_{ijk}) + K \quad (17)$$

where

$$T_{ijk,lmn} = -\delta_{il} - \delta_{jm} - \delta_{kn} - \delta_{(i-1)/3,(l-1)/3} \delta_{(j-1)/3,(m-1)/3} \quad (18)$$

and

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

Further details can be seen in the paper itself.

This method of solving sudoku works very well as long as noise is injected. In fact, with noise injected a solution can be found in less than one second. When run in the noise free mode this solver fails. This means that local minima are very problematic for the sudoku puzzle and should be taken into consideration whenever designing an algorithm that involves this puzzle.

3.10 Simulated Annealing

(Jilg and Carter, 2009) proposes a solution to the sudoku puzzle based on simulated annealing, or the mimicking of metals as they cool. This method does not seem to work well as only 14 out of 40 grids attempted were solved. Perhaps the parameters of this algorithm could be adjusted to produce better results in the future.

3.11 Quantum Simulated Annealing

(Jilg and Carter, 2009) proposes a solution to the sudoku puzzle based on quantum simulated annealing. This method is different from regular simulated annealing except that a field strength variable is used that determines the search radius. This value is used in the same method as the temperature in simulated annealing though temperature is also used in

the quantum simulated annealing, though in a secondary capacity. This method does not seem to work well as only 7 out of 40 grids attempted were solved. Perhaps the parameters of this algorithm could be adjusted to produce better results in the future.

(Perez and Marwala, 2008) also proposes a quantum simulated annealing approach. Results reported were that in the best case the algorithm took 65 seconds to run and found the solution after 42,700 iterations.

3.12 Hybrid Techniques

(Perez and Marwala, 2008) suggests a hybrid method combining simulated annealing with genetic algorithms in order to solve the sudoku puzzle. This algorithm uses the genetic algorithm to get close to an optimal solution and then leverages simulated annealing in order to find the optimal solution. Over 20 runs this algorithm took between 1.447 seconds and 3 minutes to solve the sudoku puzzle leaving its performance lacking at best.

3.13 Constraint Problems

(Simonis, 2005) introduces a constraint problem formulation of the sudoku puzzle. The formulation itself consists of a $n^2 * n^2$ matrix of variables across the domain of 1 to n^2 . $3 * n^2$ AllDifferent Constraints are created for each row, column and major block. In order to improve efficiency, a dual set of variables is used to take advantage of channeling. These variables are formulated as an inverse constraint.

This solution also takes advantage of redundant constraints. Four types of redundant constraints are implemented:

- Row/Column interaction - Allows a value to appear only once in row/column by mapping possible solutions to a bipartite graph. This adds 9 constraints to the model.
- Row/Block interaction - Variables of the rows are also shared with each block in the puzzle. This constraint works by removing those assignments which overlap and cause failures. This adds 54 constraints to the model.

- Rows/Blocks - This characterizes interaction of rows and columns across sets of 3 blocks. This adds 54 constraints to the model.
- Rows/Columns/Blocks - This method is not implemented, only mentioned. As it needs a more complex model (bigger than bipartite graph), it is left for future work.

This model also takes advantage of shaving - removing any assignments that lead to a failure from the solution space. This is basically a method of shaving possible assignments.

Concerning propagation schemes, this paper applies multiple which include:

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

As compared to other methods, this method does not solve these problems very efficiently. In fact, the basic model struggles to solve the most difficult puzzles, though once shaving is introduced all problems are solved with an average time between 51 and 765 seconds. Further and more detailed results can be seen in the paper itself. They are not reproduced here because they are very long.

3.14 SAT Methods

SAT solvers are fairly straight forward and easy to implement, especially in the case of sudoku. SAT solvers, in general, rely upon the use of propositional logic in order to satisfy boolean constraints.

(Weber, 2005) suggests a formulation where each cell of the sudoku grid is assigned a variable p_{ij}^d where i is the row, j is the column, and d is each possible value (1-9) of the current variable. Then $1 \leq i, j, d \leq 9$ represents the truth value of $x_{ij} = d$. This basic clause is used to build up a complete constraint satisfaction problem that allows for constraint propagation while using less than 729 variables and 11745 clauses. This implementation is quite successful and solves sudoku problems in milliseconds. The actual constraints used in this algorithm are as follows:

$$\bigvee_{d=1}^9 p_{ij}^d \quad (19)$$

$$\bigwedge_{1 \leq d < d'} \neg p_{ij}^d \vee \neg p_{ij}^{d'} \quad (20)$$

(Lynce and Ouaknine, 2006) suggests a formulation of sudoku as a SAT problem that contains 729 propositional variables to represent each square of the grid and its assignment. Once all of the logical constraints are added the minimal encoding contains 8,829 clauses and the extended encoding contains 11,988 clauses (both excluding unit clauses for pre-assigned variables). The authors test both encodings using multiple inference techniques. Both the minimal and extended coding are defined as follows:

- There is at least one number in each entry:

$$\bigwedge_{x=1}^9 \bigwedge_{y=1}^9 \bigwedge_{z=1}^9 s_{xyz}$$

- Each number appears at most once in each row:

$$\bigwedge_{y=1}^9 \bigwedge_{z=1}^9 \bigwedge_{x=1}^8 \bigwedge_{i=x+1}^9 (\neg s_{xyz} \vee \neg s_{iyz})$$

- Each number appears at most once in each column:

$$\bigwedge_{x=1}^9 \bigwedge_{z=1}^9 \bigwedge_{y=1}^8 \bigwedge_{i=y+1}^9 (\neg s_{xyz} \vee \neg s_{xiz})$$

- Each number appears at most once in each 3x3 sub-grid:

$$\bigwedge_{z=1}^9 \bigwedge_{i=0}^2 \bigwedge_{j=0}^2 \bigwedge_{x=1}^3 \bigwedge_{y=1}^3 \bigwedge_{k=y+1}^3 (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$$

$$\bigwedge_{z=1}^9 \bigwedge_{i=0}^2 \bigwedge_{j=0}^2 \bigwedge_{x=1}^3 \bigwedge_{y=1}^3 \bigwedge_{k=x+1}^3 \bigwedge_{l=1}^3 (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+l)z})$$

- There is at most one number in each entry:

$$\bigwedge_{x=1}^9 \bigwedge_{y=1}^9 \bigwedge_{z=1}^8 \bigwedge_{i=z+1}^9 (\neg s_{xyz} \vee \neg s_{xyi})$$

- Each number appears at least once in each row:

$$\bigwedge_{y=1}^9 \bigwedge_{z=1}^9 \bigwedge_{x=1}^9 s_{xyz}$$

- Each number appears at least once in each column:

$$\bigwedge_{x=1}^9 \bigwedge_{z=1}^9 \bigwedge_{y=1}^9 s_{xyz}$$

- Each number appears at least once in each 3x3 sub-grid:

$$\bigwedge_{z=1}^9 \bigwedge_{i=0}^2 \bigwedge_{j=0}^2 \bigwedge_{x=1}^3 \bigwedge_{y=1}^3 s_{(3i+x)(3j+y)z}$$

The authors use both of these encodings and combine them with multiple unit propagation and resolution techniques including:

- Unit propagation (up)
- Unit propagation + failed literal rule (up+flr)
- Unit propagation + hyper-binary resolution (up+hypre)
- Unit propagation + binary failed literal rule (up+binflr)

in order to produce the results in the following two table which details the percentage of puzzles solved:

	up	up+flr	up+hypre	up+ binflr
Minimal	0%	1%	25%	69%
Extended	47%	100%	100%	100%

Of the puzzles solved, the authors state that most were solved in under 10 milliseconds. Though all combinations do not solve all puzzles it is found that the extended encoding using unit propagation with a combination of either the failed literal rule, binary failed literal rule, or hyper-binary resolution solves 100% of puzzles successfully. These are excellent results and are expected as SAT solvers in general solve this type of problem easily.

(Kwon and Jain, 2006) proposes formulating sudoku as a propositional satisfiability problem(SAT) using conjunctive normal form(CNF) through the introduction of sophisticated encoding techniques in order to remove redundant and extra constraints. This technique offers up to a reduction factor of between 12 and 79 by leveraging the knowledge of fixed cells in the puzzle. This is important as large sudoku puzzles can create CNF files that are far too large to process by state of the art SAT solvers.

The actual technique for removing extra constraints is rather novel. Fixed cells are first inspected. The inspection of these cells can lead to the inference of many facts, mainly that some variables in the formulation must be either true or false. Thus any variables in the formulation that represent those values determined to be true and false can be removed from the formulation.

The example given in the paper discusses a cell (1,3) whose value is fixed at 6. The following inferences are made from this fact:

- Variable(1,3,6) representing the cell is true;
- Other variables (1,3,1),(1,3,2),(1,3,3),(1,3,4),(1,3,5),(1,3,7),(1,3,8),(1,3,9) representing the cell are false;
- Clause (1,3,1),(1,3,2),(1,3,3),(1,3,4),(1,3,5),(1,3,6),(1,3,7),(1,3,8),(1,3,9) representing the cell definedness is true;
- Clauses $\neg(1,3,1), \neg(1,3,2), \neg(1,3,1), \neg(1,3,3), \neg(1,3,1), \neg(1,3,4), \dots$ representing the cell uniqueness are true; etc

The only weakness of this methodology is that it depends strongly on the amount of fixed cells in any given sudoku puzzle, though it does greatly improve performance, especially space complexity as compared to other SAT formulations as it removes extra constraints. The following table details the performance of this proposed encoding as compared to a typical extended encoding. Note that the typical time to solve a puzzle is typically less than one second and always less than two seconds.

		Extended Encoding			Proposed Encoding		
size	level	vars	clauses	time	vars	clauses	time
9x9	easy	729	12013	0.00	200	1761	0.00
9x9	hard	729	12018	0.00	164	1070	0.00
16x16	easy	4096	124008	0.01	648	5598	0.00
16x16	hard	4096	124002	0.01	797	8552	0.00
25x25	easy	15625	752792	0.07	1792	19657	0.04
25x25	hard	15625	752778	0.21	1990	24137	0.05
36x36	easy	46656	3271748	0.50	4186	57595	0.06
36x36	hard	46656	3271748	0.67	3673	45383	0.08
49x49	easy	117649	11305189	1.47	7642	112444	0.13
64x64	easy	262144	33048912	stack	11440	169772	0.04
81x81	easy	531441	85060787	stack	17793	266025	0.06

3.15 Message Passing

Message passing algorithms are used to solve the sudoku puzzle in (Goldberger, 2007; Khan et al., 2009).

(Goldberger, 2007) formulates the sudoku problem as a constraint satisfaction problem that is solved using message passing algorithms, particularly max-product and sum-product algorithms. The algorithm itself builds a factor-graph representation of the puzzle (a bipartite graph) that is then solved by passing messages of different types between the vertices of the graph using the max-product (MP) and sum-product (SP) algorithms combined with

two different constraint sets, C_1 and C_2 . C_1 contributes 54 additional constraints to the formulation while C_2 contributes nine extra constraints to the problem. The details of both of these constraint sets are detailed in this work. This paper also shows that the max-product algorithm suffers from the development of stopping sets, though the paper also describes methods for overcoming this difficulty. The performance of this algorithm is detailed as follows:

Method	Percentage of Success	Average size stopping-set	run time(ms)
MP	70.7	44.7	6
MP+ C_1	85.5	52.0	6
MP+ C_2	75.5	45.6	6
MP+ C_1+C_2	85.6	52.0	6
SP	71.3		2810
SP+ C_1	76.8		269
SP+ C_2	80.6		260
SP+ C_1+C_2	89.5		233

This algorithm, in general, performs reasonably well as the longest amount of processing time used to solve a sudoku puzzle is 2.8 seconds. The failure of this algorithm is the fact that it does not solve all sudoku puzzles as can be seen in the table above.

(Khan et al., 2009) develops a probabilistic graphical model of sudoku that is then solved using message passing algorithms. One of the main contributions is the relaxation of the all-different constraints which gives a significant complexity savings ($O(N!N^4)$ vs $O(N^4)$). The other is the application of Sinkhorn balancing to the formation of the sudoku puzzle. This is detailed in the paper and will not be explained here.

The algorithm itself is called solution check and is defined as follows:

- Calculate relative probability of each unassigned cell.
- The cell with maximum relative probability is selected for that value

- Delete this assigned value from all other cells in this constraints
- Redistribute the deleted probability in those cells to other cells evenly
- Repeat until puzzle is solved

The algorithm performs in a very similar manner to the previously discussed message passing algorithm.

3.16 Backtracking

In (Norvig, 2009), Russel Norvig proposes a solution to the sudoku puzzle and also provides the python code necessary to solve the puzzle. The algorithm presented is simply a combination of constraint propagation and direct search. Constraint propagation is first implemented in such a manner that two different constraints are propagated:

- When a value is assigned to a square that same value is removed as a possible assignment in all related squares
- If a square is left with a single value for possible assignment, that value is immediately assigned

This makes the initial algorithm quite simple:

The steps in this algorithm are as follows:

- Assign a value to a square on the grid
- Propagate constraints to other squares

This algorithm, though it is simplistic, solves simple sudoku puzzles with ease, though it is unable to solve more difficult sudoku puzzles. In order to solve the more difficult puzzles the algorithm is modified to include a recursive depth-first search with backtracking. The algorithm now becomes:

- Make sure a solution or a contradiction has not been found

- Choose the square with the least possibilities for assignment and consider all its possible values by assigning each value one at a time
- Search from the resulting position

This algorithm performs in a stellar fashion. It turns out that after going through a list of 95 hard puzzles, the algorithm considers an average of 64 possibilities per puzzle and does not search more than 16 squares. It is also capable of solving roughly 8 puzzles per second or roughly 0.125 seconds per puzzle. This is, perhaps, the best performing of all algorithms for the sudoku search and, as it turns out, seems to be one of the simplest and most straightforward: backtracking with constraint propagation.

3.17 Linear Programming

(Bartlett and Langville, 2007) proposes a method for solving sudoku puzzles as an binary integer linear program (BILP). The formulation is quite simple as it simply uses binary variables to choose which digit will be placed in which cell. The decisions variables are defined as

$$x_{ijk} = \begin{cases} 1 & \text{if element } i,j \text{ is integer } k \\ 0 & \text{otherwise} \end{cases}$$

The constraints for this model are also typical:

- Only one k in each column
- Only one k in each row
- Only one k in each sub matrix
- Every position in the puzzle must be filled

These same constraints written mathematically are:

$$\sum_{i=1}^n x_{ijk} = 1, j = 1 : n, k = 1 : n \tag{21}$$

$$\sum_{j=1}^n x_{ijk} = 1, i = 1 : n, k = 1 : n \quad (22)$$

$$\sum_{j=mq-m+1}^{mq} \sum_{i=mp-m+1}^{mp} x_{ijk} = 1, k = 1 : n, p = 1 : m, q = 1 : m \quad (23)$$

$$\sum_{k=1}^n x_{ijk} = 1, i = 1 : n, k = 1 : n \quad (24)$$

$$x_{ijk} = 1 \forall (i, j, k) \in G \quad (25)$$

$$x_{ijk} = 1 \in \{0, 1\} \quad (26)$$

The objective function is also typical in that, being a feasibility problem, no objective function is actually required. The objective used in this case is $\min 0^T x$.

The authors do mention that the search space of a soduko puzzle using the BILP methodology is 2^{n^3} . This means that a puzzle with no givens has a search space size of 2.82×10^{219} . While this number is quite large, it drastically shrinks as more and more given cells are added into the puzzle.

The model is solved in MatLab using the classic branch and bound technique. The constraints are formed according to each puzzle (fixed cells are constrained to certain values). The method works quite well and is very fast (roughly 16 seconds per solution). Worthy of note is that the sudoku puzzle can also be modeled as in integer linear program where x_{ijk} take on the values 1-9 instead of 0 and 1. This paper also discusses the application of integer programming to sudoku variants as well as the generation of sudoku puzzles, but those discussions are beyond the scope of this paper.

(Koch, 2006) also proposes using linear programming as a technique for solving sudoku puzzles. The difference in this study is that the puzzle is solved using a library known as ZIMPL. The paper initially uses ZIMPL to develop an integer linear program based upon the all-different constraints of constraint programming that are typically used when solving sudoku puzzles. This formulation of the problem took an exhausting amount of time to solve. The problem was then reformulated as a binary linear program. This formulation has

solved every puzzle tested rather quickly.

Generally speaking, linear programming is a very useful technique for solving sudoku puzzles. It should be noted that, as is the case with all linear programming, it is often the formulation that is the most important aspect of building and solving any model. Integer linear programming has a tendency to stall when attempting to solve these puzzles as the search space becomes too large and is intractable. As such is not a useful tool. On the other hand, binary linear programs are quite successful and should be pursued if this technique is chosen as a tool to solve sudoku puzzles.

4 Open Research Problems

Open research problems and future work concerning sudoku puzzles seem to fall into two possible categories: Performance tweaking and hybrid techniques.

Many of the algorithms listed in this paper take far too long to solve sudoku problems while methods such as SAT/CNF encoding, constraint programming, and backtracking are able to solve the puzzle almost instantly and methods like linear programming guarantee solutions to a feasible puzzle. It could be quite beneficial to continually tweak and re-design many methodologies and algorithms in order to increase performance. This will mean developing new formulations of the sudoku puzzle, developing new heuristics are far better than current ones, tweaking parameters of population based search techniques in order to improve convergence time, and doing systematic comparisons of the best techniques.

Parallel, distributed, and threaded computing should also be used in order to improve the performance of many of these techniques. In this age of multi-core computers, Beowulf clusters, cloud computing, and grid computing there is no excuse for not enhancing the performance of algorithms and computational techniques by leveraging these technologies to the fullest even though developing new techniques and algorithms may prove quite challenging.

The challenges of taking on this task may, in fact, lead to breakthroughs that aid in the development of algorithms and techniques that may be mapped to even more difficult problems.

Hybrid techniques may also be used in the future to improve performance and convergence when solving sudoku puzzles. Though (Perez and Marwala, 2008) has already posed a hybrid technique that was rather unsuccessful, a further opportunity can always be found by combining, refining, and inventing new search techniques.

One further area of research that is open to the sudoku community is the application of sudoku to other research problems. The message passing techniques developed and discussed in (Goldberger, 2007; Khan et al., 2009) both reference low-density parity-check (LDPC) codes proving that sudoku algorithms and modeling techniques will surely have applications in other difficult problems.

(Jilg and Carter, 2009) reiterates what has been suggested above and adds that grid communication topologies may also be a fresh ground for improving sudoku search techniques and performance. In any case, it should be recognized that all studies reviewed in this paper could and should be improved upon.

References

- BARTLETT, A. AND LANGVILLE, A. 2007. An integer programming model for the sudoku problem.
- GEEM, Z. W. 2007. Harmony search algorithm for solving sudoku. *In* KES (1), pp. 371–378.
- GEEM, Z. W., KIM, J. H., AND LOGANATHAN, G. V. 2001. A new heuristic optimization algorithm: Harmony search. *SIMULATION* 76:60–68.
- GOLDBERGER, J. 2007. Solving sudoku using combined message passing algorithms. Technical report trbiu-eng-2007-05-03, Engineering School, Bar-Ilan Univ.
- HERZBERG, A. M. AND MURTY, M. R. 2007. Sudoku squares and chromatic polynomials. *Notices of the AMS* 54.

- JILG, J. AND CARTER, J. 2009. (c - international conference) sudoku evolution - solving sudoku with ai-related algorithms. *In* ICE-GIC International Consumer Electronics Society Games Innovation Conference, Imperial College, London.
- K., J. S., ROACH, P. A., AND PERKINS, S. 2007. Construction of heuristics for a search based approach to solving sudoku. *In* Research and Development in Intelligent Systems XXIV: Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Artificial Intelligence, Springer-Verlag.
- KHAN, S., JABBAR, S., JABBARI, S., AND GHANBARINEJAD, M. 2009. Solving sudoku using probabilistic graphical models. Unpublished.
- KOCH, T. 2006. Rapid mathematical programming or how to solve sudoku puzzles in a few seconds. *In* H.-D. Haasis, H. Kopfer, and J. Schönberger (eds.), Operations Research Proceedings 2005, pp. 21–26. ZIB-Report 05-51.
- KWON, G. AND JAIN, H. 2006. Optimized cnf encoding for sudoku puzzles. *In* The 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Short Paper Proceedings.
- LYNCE, I. AND OUAKNINE, J. 2006. Sudoku as a sat problem. *In* International Symposium on Artificial Intelligence and Mathematics.
- MANTERE, T. AND KOLJONEN, J. 2007. Rating and generating sudoku puzzles with ga. *In* 2007 IEEE Congress on Evolutionary computation CEC2007, pp. 1382–1389, Singapore.
- MANTERE, T. AND KOLJONEN, J. 2008. Sudoku solving with cultural swarms. *In* AI and Machine Consciousness, Proceedings of the 13th Finnish Artificial Intelligence Conference STeP 2008, pp. 160–67, Espoo, Finland.
- NICOLAU, M. AND RYAN, C. 2006. Solving sudoku with the GAuGE system. *In* P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt (eds.), Proceedings of the 9th European Conference on Genetic Programming, volume 3905 of *Lecture Notes in Computer Science*, pp. 213–224, Budapest, Hungary. Springer.
- NORVIG, R. 2009. Solving every sudoku puzzle. <http://norvig.com/sudoku.html>.
- PEREZ, M. AND MARWALA, T. 2008. Stochastic optimization approaches for solving sudoku. *CoRR* abs/0805.0697.
- SANTOS-GARCÍA, G. AND PALOMINO, M. 2007. Solving sudoku puzzles with rewriting

- rules. *Electron. Notes Theor. Comput. Sci.* 176:79–93.
- SIMONIS, H. 2005. Sudoku as a constraint problem. *Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems* pp. 13–27.
- WEBER, T. 2005. A SAT-based Sudoku solver. In G. Sutcliffe and A. Voronkov (eds.), LPAR-12, The 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Short Paper Proceedings, pp. 11–15.
- WOLDEMARIAM, K. M. AND YEN, G. G. 2010. Vaccine-enhanced artificial immune system for multimodal function optimization. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics* 40:218–228.
- YATO, T. 2003. Complexity and completeness of finding another solution and its application to puzzles. Master’s thesis, University of Tokyo.
- YUE, T.-W. AND LEE, Z.-C. 2006. Sudoku solver by q’tron neural networks. In ICIC (1), pp. 943–952.