Solving Sudoku Using Combined Message Passing Algorithms

Jacob Goldberger
School of Engineering, Bar-Ilan University
goldbej@eng.biu.ac.il

Abstract

In this paper we apply message-passing algorithms to solve Sudoku puzzles. We provide explicit expression for the sum-product algorithm and the max-product algorithm and analyze the difference between the algorithms in terms of performance and efficiency. The failure of the max-product algorithm when been applied to Sudoku problem is due to the existence of stopping-sets. We show empirically that the performance of the max-product algorithm in the case of Sudoku can be improved by adding redundant constraints. We show that applying the sum-product to the stopping-set obtained by the max-product, can lead to effective and efficient puzzle-solving method.

Keywords: statistical pattern recognition, graphical models,loopy belief-propagation

1 Introduction

Sudoku is a popular puzzle printed daily in newspapers all over the world. The aim of the most popular form is to fill a 9×9 matrix of cells with digits from 1 through 9. Each puzzle consists of a grid in which some digits have already been filled in and the goal is to fill in the remaining cells so that each digit appears once in each row, once in each column and once in each of nine 3×3 squares into which the grid has been subdivided. For a correctly formed Sudoku puzzle, there should be only one solution. Sudoku translates "Only single numbers allowed" in Japanese. An excellent introduction to mathematical aspects of Sudoku was published by Hayes (2006). The Sudoku, when generalized to $n \times n$ grids to be filled in by numbers from 1 to n, is known to be an NP-complete problem (Yato 2002). The proof uses a simple reduction from the Latin-Squares problem. The 3×3 version of the Sudoku problem can be efficiently solved using classical artificial intelligence methods such as depth-first search over all the values that can be filled in the empty squares. However, when n is larger the problem becomes more difficult. Sudoku can be viewed as a constraint satisfaction problem (CSP). A recent paper (Simonis 2005) compares different CSP schemes for solving Sudoku without backtracking search. In particular Sudoku puzzles can be encoded into conjunctive normal form, and subsequently be solved using propositional satisfiability (SAT) inference techniques (Lynce and Ouaknine 2006). Note that Sudoku puzzles are a special case of Latin squares; any solution to a Sudoku puzzle is a Latin square. Sudoku imposes the additional restriction on the sub-blocks. Standard solution methods for Latin squares can be modified to handle Sudoku problems.

In this paper we utilize message-passing algorithms to solve the Sudoku problem. The belief propagation (BP) algorithm (Pearl 1988) also known as the sum-product algorithm is a popular method for computing approximate marginal probabilities in loopy graphical models. Excellent approximation performance have been reported for decoding of LDPC and Turbo codes and for various computer vision applications. The max-product algorithm is a variant of the BP whose aim is to find the most probable pattern. We show that (similarly to LDPC over BEC), the failure of the max-product algorithm when applied to Sudoku is due to the existence of stopping-sets where we the algorithm shucked before the puzzle is completely solved. There are many approaches to apply message-passing algorithms to solve constraint satisfaction problems, e.g. (Braunstein et al. 2005). In this paper we show that combining the max-product and the sum-product in such a way that the sum-product is applied to the stopping-set obtained by the max-product, can yield efficient and better performed algorithm for solving Sudoku puzzles.

The paper is organized as follows. The concepts of factor graphs and message-passing algorithms are reviewed in the next section. The derivation and performance analysis of several variants of message-passing algorithms for solving Sudoku puzzles appear in Sections 3 and 4. Experimental results are presented in Section 5.

2 Factor Graphs and Message Passing Algorithms for CSP

The main concepts we utilize to obtain an algorithm for solving Sudoku puzzles are factor graphs (Kschischang et al. 2001) and message-passing algorithms (Pearl 1988). In this section we briefly review these concepts. Let $x_1, ..., x_n$ be a set of random variables and let $p(x_1, ..., x_n)$ be their joint probability. Suppose the probability function $p(x_1, ..., x_n)$ factors into a product of non-negative local functions each having some subset of $\{x_1, ..., x_n\}$ as arguments, i.e. suppose that:

$$p(x_1, ..., x_n) = \frac{1}{Z} \prod_a f_a(x_a)$$

such that $x_a \subset \{x_1, ..., x_n\}$, $f_a(x_a)$ is a non-negative function having the elements of x_a as arguments and Z is a normalization factor. Given a factorized probability function p, the associated factor-graph is a bipartite graph consisting of random variable nodes and factor nodes, connected by a set of edges in such a way that a factor f_a is connected to its arguments (which are the local random variables that are members of x_a). In the case of a CSP problem all the factors are 0-1 functions and therefore $p(x_1, ..., x_n)$ is a uniform distribution over all the patterns that satisfy all the constraints. The normalization factor Z is the number of valid patterns.

The belief-propagation (BP) algorithm is an efficient method for obtaining an approximate solution for the marginal probabilities $p(x_i)$, which is exact in the case of loop-free factor graphs. The BP algorithm is based on passing messages between adjacent nodes of the underlying factor graph. We use the notation $N(f)$ for the set of all the random variables that are connected to the factor f. Similarly, we denote the set of all the factors that are connected to the r.v. x by $N(x)$. Let $m_{x \to f}(x)$ be a message sent from variable x to factor f and let $m_{f \to x}(x)$ be a message sent from factor f to variable x. The messages are (probability) vectors defined on all the possible values of the r.v. x. The belief-propagation update rules are the following:

$$m_{x \to f}(x) \quad \propto \quad \prod_{g \in N(x) \setminus f} m_{g \to x}(x) \tag{1}$$

$$m_{f \to x}(x) \quad \propto \quad \sum_u f(u) \prod_{y \in N(f) \setminus x} m_{y \to f}(u_y) \tag{2}$$

where the summation in the last expression is performed over all the possible input values for the factor f such that x is kept fix. At the end of each iteration we can compute for each r.v. the current belief $b_x(x)$ which is the current approximation of the marginal probability $p(x)$:

$$b_x(x) \propto \prod_{f \in N(x)} m_{f \to x}(x) \tag{3}$$

In the case of loop-free graphs the beliefs converge to the exact marginal probabilities after a finite number of iterations. In the case of loopy graph we can obtain approximations that were found to be accurate in many cases.

The max-product algorithm is a variant of the sum-product algorithm. It is an efficient method for obtaining the most probable pattern, i.e. $\arg\max p(x_1, ..., x_n)$. The max-product algorithm finds an approximate solution which is exact in the case of loop-free factor graphs. The max-product is also based on passing messages along the edges of the factor graph. The update rules are similar to those of the BP algorithm, $b_x(x)$ and $m_{x\to f}(x)$ are same as expressions (3) and (1), and the sum operation appears in expression (2) is replaced by the max operation. The (approximated) most likely pattern $\hat{x}_1, ..., \hat{x}_n = \arg\max p(x_1, ..., x_n)$ can be extracted from the beliefs by $\hat{x}_i = \arg\max_v b_{x_i}(v)$ where the maximization is performed over all the possible values of x_i. The max-product messages of a CSP have the following nice property.

Theorem 1: Starting with zero-one message vectors, the max-product messages of a CSP, in both directions, are vectors of zeros and ones.

Proof: We apply the induction principle on the index of the iteration. At the beginning, the zero-ones message initialization trivially ensures the validation of the claim. Assuming the theorem is true for all the messages sent at the previous iteration. The message $m_{x\to f}(x)$ is obtained as a product of zeros and ones (see expression (1)), hence $m_{x\to f}$ is a binary vector. In a similar way, the message $m_{f\to x}(x)$ is obtained as a maximum of products of zeros and ones, hence each $m_{f\to x}$ is also a binary vector.

In the general case of a CSP (e.g. the 3-Sat problem), the max-product has to be initialized with random messages since the all-ones message is usually a fix-point. In other situations the factor-graph contains single-variable factors. This avoids the need for a random initialization of the message-passing algorithms. Instead, the single-variable factors can be used for informative initialization for the corresponding variables. We dub this case as a self-initialized CSP (SICSP). The Sudoku puzzle are examples of SICSP.

Denote the initial messages derived from the single-variable factors by $b_x(t) \in \{0, 1\}$ where 1 is indicating that t is a valid assignment to x. Following Theorem 1, each iteration of the max-product has the following simple form.

```
For each random variable $x$,
    For each $t$, such that $b_x(t) = 1$,
        For each constraint $f$ that is defined on $x$
            If the substitution $x = t$ is not consistent with the constraint $f$, set $b_x(t)$ to be zero.
```

The algorithm monotonically eliminates possible values for each variable. Therefore, even in loopy CSP cases, there is a guarantee that it will converge after a finite number of iterations. More than that in SICSP even in loopy cases, due to the informative initialization, there is a guarantee that each variable that is completely determined by the max-product is correct.

From Theorem 1 we also obtain that the max-product algorithm in this context is equivalent to the arc-consistency algorithm, see (Decheter et al. 2003) for further discussion. The stopping-set is actually the situation where after constraint propagation only several more squares can be filled although all squares have at least some possibilities eliminated.

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

a

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

b

			48		48			
		89	89					
	89	89						
	489	489	489					
		49			49			

c

Figure 1: (a) An example of a Sudoku puzzle. (b) The digits that are revealed as a result of the max-product algorithm. (c) The undetermined cells in the stopping-set obtained after the max-product algorithm has converged.

3 Applying Message-Passing Algorithms to Sudoku

In this section we shall describe how message-passing algorithms can be utilized to solve a Sudoku puzzle. Although the following approach is general, to simplify the presentation we concentrate on the popular case of 9×9 grids. We can associate discrete random variables with the 81 entries, denoted by $x_{11}, x_{12}, ..., x_{99}$, each r.v. can obtain the values 1 to 9. A particular Sudoku puzzle is formed by revealing the values of some of those random variables. Figure 1a shows a typical (difficult) Sudoku problem. Denote the indexes of the given random variables by $I \subset \{(1,1), ..., (9,9)\}$ and the known value of x_{ij} by $c_{ij} \in \{1, ..., 9\}$. The twenty-seven row, column and sub-block constraints can be also presented using the random variables. For example the constraint that each of the nine digits should appear exactly once in the first row is translated to the constraint that the values of the nine random variables $x_{11}, ..., x_{19}$ should form a permutation on $1, ..., 9$. The Sudoku problem can be presented using the following factor graph for the distribution $p(x_{11}, ..., x_{99})$:

$$\frac{1}{Z} \prod_{ij \in I} 1_{\{x_{ij}=c_{ij}\}} \cdot \prod_{\text{constraints}} 1_{\{\text{constraint is satisfied}\}} \tag{4}$$

The factor graph defines a uniform probability over all the valid solutions that are consistent with the given values. Since all the factors in equation (4) are indicator functions of probabilistic events, the normalized scalar Z in this case is exactly the number of valid solutions. For the case of properly formed Sudoku puzzle we know that $Z = 1$.

Given the factor-graph (4) we can apply the message-passing algorithms to solve the Sudoku problem defined by the graph. We can consider using either the sum-product or the max-product algorithm. We start analyzing the behavior of the sum-product algorithm. The computation of the message $m_{x \to f}(x)$ is simple and straight-forward. The computation of the sum-product message $m_{f \to x}(x)$ is more involved. To simplify notation we concentrate on a single factor f corresponding to the row-constraint on $x_{i1}, ..., x_{i9}$. The message from i-th row constraint f to x_{ij}, $m_{f \to x_{ij}}(t)$, is the information provided by the constraint on the validity of assigning the digit t to x_{ij}. According to the general sum-product message definition (2), the message from f to x_{ij} is:

$$m_{f\to x_{ij}}(t) = \sum_{\{\pi\in S_9 \mid \pi_j = t\}} \prod_{k\neq j} m_{x_{ik}\to f}(\pi_k) \tag{5}$$

such that S_9 is the set of all the permutations on the set $\{1, ..., 9\}$. Define the following 9×9 matrix A:

$$A = \begin{pmatrix} m_{x_{i1}\to f} \\ \vdots \\ m_{x_{i9}\to f} \end{pmatrix} \tag{6}$$

Denote the minor of A obtained by removing the j-th row and t-th column by A_{jt}. It can be easily verified from equation (5) that $m_{f\to x_{ij}}(t)$ is exactly the permanent of the matrix A_{jt}, i.e.,

$$m_{f\to x_{ij}}(t) = \text{Permanent}(A_{jt}) = \sum_{\pi\in S_8} \prod_{k=1}^{8} A_{jt}(k,\pi_k) \tag{7}$$

where S_8 is the set of all permutations on eight elements. The sum-product algorithm can be utilized iteratively until the hard-decision version of the current set of beliefs satisfies all the Sudoku constraints (or the number of iterations exceeds a pre-defined number). However, computing the permanent of a matrix, when generalized to size $n\times n$ is known to be an NP-complete problem which makes the usage of sum-product algorithm for larger Sudoku grids non tractable.

Unlike the sum-product, the computation of the exact max-product message $m_{f\to x}(x)$ is not NP-complete. The max-product message $m_{f\to x_{ij}}(t)$ from the i-th row-constraint f to x_{ij} is:

$$\max_{\{\pi\in S_9 \mid \pi_j = t\}} \prod_{k\neq j} m_{x_{ik}\to f}(\pi_k) = \max_{\pi\in S_8} \prod_{k=1}^{8} A_{jt}(k,\pi_k) \tag{8}$$

The problem of finding the maximum generalized diagonal of a square matrix is equivalent to the problem of finding maximum weight bipartite matching. The nodes of the bipartite graph are the rows and the columns of the matrix and the weight on the edge connecting row i and column j is the ij entry of the matrix. The problem of finding maximum weight bipartite matching (assignment problem) can be solved in a polynomial time using, for example, the classical Hungarian method (Kuhn, 1955).

Following Theorem 1 we can state a simpler equivalent formulation for the max-product algorithm. The natural initial values for the beliefs are:

$$b_{ij}(t) = \begin{cases} 1_{\{t=c\}} & \text{if the value of } x_{ij} \text{ is given to be } c \\ 1 & \text{if } x_{ij} \text{ is not given} \end{cases} \tag{9}$$

Each iteration of the max-product algorithm is as follows:

```
For each entry $ij$ of the Sudoku grid,
    For each $t$, such that $b_{ij}(t) = 1$,
        For all the (three) constraints $f$ that are defined on the $ij$ cell.
            If the substitution $x_{ij} = t$ is not consistent with the constraint $f$, set $b_{ij}(t)$ to be zero.
```

The algorithm monotonically eliminates possible values for each cell of the Sudoku grid and there is a guarantee that it will converge after a finite number of iterations. Unlike the case of the belief-propagation, all the results obtained by the max-product algorithm are guarantied to be correct. If a digit was found by the algorithm to be invalid assignment for a particular cell, then it is indeed invalid.

The loopy max-product algorithm can not completely solve the NP-complete Sudoku problem. The algorithm can halt before a complete solution was found in situations where no single constraint can eliminate any currently valid digit from any grid-cell. In other words, the max-product can prematurely stop without recovering all the cells.

This drawback of the max-product algorithm in the Sudoku problem resembles the stopping-sets which determine the performance of LDPC codes under iterative decoding over erasure channels (Di 2002). Hence we use the terminology stopping-set problem to coin the early convergence problem that can occurs in our case. A stopping-set in a Sudoku problem is a set of random variables (grid cell) S such that, even if all the other cells are given (or correctly found), for each $x \in S$ there are at least two digits that satisfy all the constraints on x. Figure 1 shows an example of a Sudoku problem and the stopping-set obtained as a result of the max-product algorithm.

The sum-product and max-product show totaly different behaviors when they are derived from the factor graph of the Sudoku problem. Computing the sum-product messages is an NP-complete problem. The algorithm is not always converging and when it fails to solve a Sudoku problem there is no guarantee that any of the digits filled by the algorithm is correct. In contrast the max-product algorithm is much more computationally efficient. In case the max-product algorithm fails, there is still guarantee that each digit in the stopping-set is correct. A natural approach is to combine the two algorithms. First apply the max-product algorithm. Then (if needed) apply the sum-product to the stopping-set obtained by the max-product. Experimental results (Section 6) justify combining the two methods. Note that this approach can not be applied to LDPC, since in that case the max-product and the sum-product are the same.

4 Redundant Constraints

Utilizing the max-product algorithm to solve a Sudoku problem can succeed if and only if the given initial values do not cause stopping-sets. We can overcome the stopping-set situations by performing several "guesses" on the unsolved places. In this way we can combine the standard back-tracking approach with the message-passing based approach. Application of this combined method in the case of LDPC over binary erasure channels can be found in (Pishro-Nik and Fekri 2004). Yet another approach is to use additional constraints. The 27 constraints that appear in the factor graph (4) completely define the Sudoku problem and any additional constraint (e.g. $x_{11} \neq x_{12}$) is redundant. However, adding more constraints can improve the performance of Sudoku solution that is based on the max-product algorithm. Using the additional constraints we can sometimes avoid the stopping-set situation.

Many of the rules suggested in commercial guides to sudoku are derived from the max-product basic rule presented in the previous section e.g. the rule "If k cells in a group contain k candidates that are not found in any other cells in that group, then other candidates in those cells can be excluded".

Consider the following set of redundant constraints that is based on interactions of a row (or column) with a square (see e.g. Eppstein 2005). Consider the first row and the first square constraints. The values of x_{11}, x_{12}, x_{13} appear in both constrains, Hence the two sets of variables $\{x_{14}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19}\}$ and $\{x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}\}$ should have the same sets of values. Considering all possible interactions, we obtain 54 such constraints. Using the max-product algorithm, the constraints are transformed to the following decision rule: If the cells of a square that can contain a digit t all lie in a single row or column,

x_{11} x_{12} x_{13}	x_{14} x_{15} x_{16}	x_{17} x_{18} x_{19}
x_{21} x_{22} x_{23}		
x_{31} x_{32} x_{33}		

we eliminate positions for t that are outside the square but inside that row or column. Similarly, if the cells that can contain t within a row or column all lie in a single square, we eliminate positions that are inside that square but outside the row or column.

Another set of decision rules is as follows. For each digit $t = 1, ..., 9$ there should be exactly nine places on the 9×9 Sudoku grid, having the value t and there should be exactly one t in each row and in each column. In other words all the grid cells having the value t should form a generalized diagonal. This constraint can be translated to the following decision rule. if the assignment of a digit t to a cell ij can not be extended to a placement of nine copies of t covering each row and column of the grid exactly once, we exclude the assignment of t to x_{ij}. This decision rule can be efficiently implemented in the following way. Define the following 9×9 matrix:

$$B(i,j) = 1_{\{t \text{ is a possible value for } x_{ij}\}} \tag{10}$$

and denote the ij minor of B by B_{ij}. To find if the digit t in a cell ij can not be extended to a placement of nine copies of t we need to check if B_{ij} contains a non-zero generalized diagonal.

In this section we have described two sets of decision rules. There are other popular rules for Sudoku that can be use to exclude candidate digits from a cell e.g. X-ray, Swordfish and Jellyfish. [1]

5 Experimental Results

We have checked our approach on a set of difficult sudoku problems. There is a correlation between the number of given values and the difficulty level of the problem. At the moment, the smallest number of revealed values in a Sudoku puzzle that has a unique completion is 17. There are no known 16-given Sudoku examples that have a unique completion. Many examples of 17-given Sudoku puzzles that has a unique completion were collected by Gordon Royle and can be found in his website (Royle, 2005). We have used

[1]http://www.palmsudoku.com/pages/techniques-overview.php

35000 such examples in the following experiments. Denote the 27 max-product derived decision rules by MP, the additional 54 constraints, defined on the previous section, by C_1 and the additional set of nine permutation constraints by C_2. Table 1 shows the percentage of Sudoku puzzles that can be complectly solved using various combinations of the constraints. Table 1 also shows for the failure cases the average number of cells that were revealed until we arrived to a stopping-set. The last column presents the average processing time of solving a single Sudoku puzzle, measured in mili-seconds. It can be seen that using the decision rules derived from the factor graph representation and the first rule-set described in the this section, the contribution of second rule-set is neglected. The second part of Table 1 shows the performance results of

Table 1: Comparative performance results for different rule combinations. The abbreviations are: MP for max-product, SP for sum-product and C_1, C_2 are the first and second constraint-sets described in Section 5.

method	percentage of success	average size of stopping-set	run time
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+MP	76.8		269
SP+MP+C_2	80.6		260
SP+MP+C_1+C_2	89.5		233

the sum-product algorithm and the results of applying the sum-product algorithm on stopping-sets obtained from several combinations of decision rules. Table 1 shows that the best strategy in term of performance (and also in term of computational complexity comparing with the sum-product) is applying the sum-product algorithm on the stopping-set obtained by the (extended version of) max-product.

The short cycles that appear in the Sudoku factor-graph cause that the sum-product algorithm can quickly amplify non-correct assignments. If a non-correct digit appears in much more patterns that are consistent with a loop of constraints, it can cause the belief of the true assignment to be very small. To avoid numerical instability we force each non-zero entry of the message to be above a pre-defined value.

A more detailed performance description can be obtained by analyzing the sizes of the stopping-sets.

The situation can be viewed as more difficult when the size of the stopping-set is smaller. Figure 2 shows histograms of Sudoku puzzles (out of 35000) as a function of the stopping-set size. The histograms is shown for both the max-product method and for the max-product combined with all the additional rules. Comparing the two histograms, it can be seen that the additional rules can help where the size of the stopping-set is relatively large. In those cases the max-product can already reveal some of the cells. However in difficult cases where the max-product algorithm almost doesn't help at all (i.e. the stopping-set size is near 17) the additional rules don't contribute much.

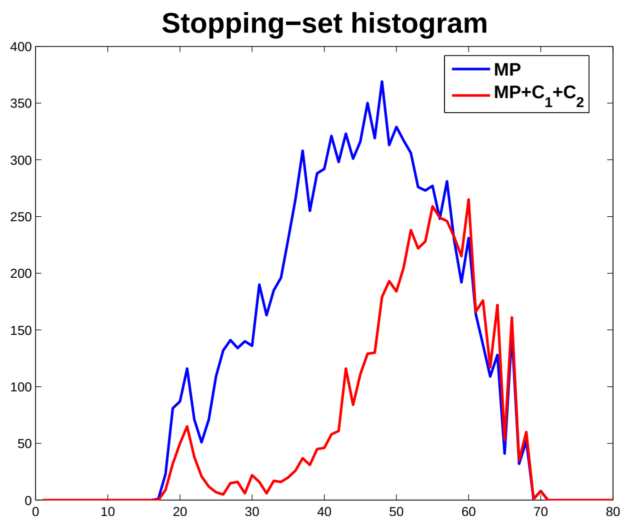

Figure 2: Histogram of stopping-sets frequencies as a function of the stopping-set size.

6 Conclusions

In this study we showed the different nature of the sum-product and the max-product algorithm and presented a combined message-passing approach for solving Sudoku puzzles. For future work we intend to carry out a systematic comparison of the message-passing based techniques described in the paper with other known approaches for solving Sudoku. The comparison can be mainly relevant for larger and more difficult sudoku puzzles and also for variants of the Sudoku such as the Kakuro puzzle.

References

[1] A. Braunstein, M. Mezard and R. Zecchina (2005). Survey propagation: an algorithm for satisfiability. Random Structures and Algorithms 27, 201-226.

[2] F. Bertram and F. Jarvis. Enumerating possible Sudoku grids. preprint.

[3] R. Dechter, and R. Mateescu. A Simple Insight Into Properties Of Iterative Belief Propagation (2003). Uncertainty in Artificial Intelligence (UAI).

[4] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke (2002). Finite length analysis of low-density parity-check codes on the binary erasure channel. IEEE Trans. on Info. Theory, pp. 1570-1579.

[5] D. Eppstein (2005). Nonrepetitive paths and cycles in graphs with application to Sudoku. ACM Computing Research Repository, cs.DS/0507053.

[6] R. G. Gallager (1963). Low density parity check codes, MIT Press,Cambridge, MA.

[7] B. Hayes (2006). Unwed numbers, the mathematics of Sudoku, a puzzle that boasts 'No math required'. American Scientist 94(1):12.

[8] F. R. Kschischang, B. J. Frey and H. Loeliger (2001). Factor graphs and the sum-product algorithm. IEEE Trans. on Info. Theory.

[9] H. W. Kuhn (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2, pp. 83-97.

[10] I. Lynce and J. Ouaknine (2006). Sudoku as a SAT problem. International Symposium on Artificial Intelligence and Mathematics.

[11] J. Pearl (1988). Probabilistic reasoning in intelligent systems. San Mateo CA: Morgan Kaufman.

[12] H. Pishro-Nik and F. Fekri (2004). On decoding of low-density parity-check codes over binary erasure channel. IEEE Trans. on Info. Theory, pp. 429-454.

[13] Gordon Royle (2005). Sudoku Patterns. http://www.csse.uwa.edu.au/gordon/sudokupat.php.

[14] H. Simonis (2005). Sudoku as a constraint problem. CP Workshop on Modeling and Reformalating Constraint satisfaction Problems.

[15] Y. Yato and T. Seta (2002). Complexity and completeness of finding another solution and its application to puzzles. Proc. of the National Meeting of the Information Processing Society of Japan (IPSJ).