Libyan Academy – Misurata

School of Applied Science and Engineering

Department of Information Technology

Application of Boltzmann Machine and other Uninformed Searching Algorithms in solving Artificial Intelligence Problems

Thesis Submitted to Department of Information Technology of Master in Information Technology

Submitted by

Wafaa Abdallah Abed Alrahman

B.Sc. in Information Technology, Sirte University, 1999

Supervised by

Dr.Mohamed Almahdi Eshtawie

Misurata – Libya

May-2017

إقرار الأمانة العلمية

أنا الطالبة وفاء عبد الله عبد الرحمن، لمسجلة بالأكاديمية الليبية – فرع مصراته بقسم تقنية المعلومات تحت رقم قيد (31357022) أقر بأنني التزمت بكل إخلاص بالأمانة العلمية المتعارف عليها لإنجاز رسالتي المعنونة ب (Application of Boltzmann Machine and other Uninformed Searching Algorithms in solving Artificial Intelligence Problems) لنيل الدرجة العلمية الماجستير وانني لم أقم بالنقل أو الترجمة من أية أبحاث أو كتب أو وسائل علمية تم نشرها داخل ليبيا أو خارجها الأ بالطريقة القانونية وبأتباع الأساليب العلمية في عملية النقل أو الترجمة وبأتباع الأساليب العلمية في عملية النقل أو الترجمة و إسناد الأعمال لأصحابها، كما انني أقر بعدم قيامي بنسخ هذا البحث من غيري وتكراره عنواناً ومضموناً.

وعلى ذلك فأنني أتحمل كامل المسئولية القانونية المترتبة على مخالفتي لذلك أن حدثت هذه المخالفة بما في ذلك سحب الدرجة العلمية الممنوحة لي.

والله على ما أقول شهيد

الاسم: .....

التوقيع: .....

التاريخ: .....

Acknowledgments

The work presented in this thesis would have been impossible without the help of ALLAH and then many other persons whom I want to recognize here.

First of all, I would like to thank Dr.Mohamed Almahdi Eshtawie for being my supervisor. Dr.Mohamed created an environment generously providing an abundance of learning opportunities. His efforts have been manifold notably those aimed at teaching me how to write up scientific research as reflected in this thesis still any mistakes remaining are my own. His work on research forms the foundation on which this research has been built during that process in constant interest in research has led to an increased understanding of the strengths and weaknesses of the algorithms.

I would like to thank the Head of Information Technology Department in The Libyan Academy-Misurata for his support. In addition, the referees for helpful comments that helped to improve the manuscript.

Finally, I want to thank my family and friends for their stimulating interest in my research. Most important of all however has been the continuous support and stimulation of my husband Dr.Emad and my daughter Shahed.

Contents

List of Tables

List of Figures

List of Terminology

Artificial intelligence (AI) is the intelligence exhibited by machines or software. It is also the name of the academic field of study, which studies how to create computers and computer software that are capable of intelligent behavior.

A search algorithm is an algorithm for finding an item with specified properties among a collection of items, which coded into a computer program.

An algorithm is an effective method that can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states.

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have an intelligent code completion.

A graph: is an ordered pair G = (V, E) comprising a set V of vertices, nodes or points together with a set E of edges, arcs or lines,

A vertex: (plural vertices) or node is the fundamental unit of which graphs are formed.

An edge: is related with two vertices, and the relation is represented as an unordered pair of the vertices with respect to the particular edge

Row: A horizontal group of 9 cells. Rows are numbered {1, 2…9} from top to bottom.

Column: A vertical group of 3 cells. Columns are numbered {1, 2…9} from left to right.

Block: A group of 9 cells arranged in a 3x3 grid.

A group: is a collection of 9 cells organized as a either a row, column or a block.

Time Complexity: refers to the number of nodes generated during a search. Measured using the "Big O" notation.

Space Complexity: refers to the maximum number of nodes stored in memory during a search. Again measured in the "Big O" notation.

Optimality: refers to the guarantee that an optimal (least-cost) solution can be found.

Completeness: refers to the guarantee that a solution can be found (if one exists).

Standard Deviation: is a number used to tell how measurements for a group are spread out from the average (mean), or expected value.

Probability density function (PDF), or density of a continuous random variable, is a function that describes the relative likelihood for this random variable to take on a given value.

Numerical integration: is the approximate computation of integral using numerical techniques.

A* algorithm: is an informed search algorithms, or best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.),

List of Symbols

RBM= Restricted Boltzmann Machine.

MP= Message Passing.

FG= Factor Graph.

BP= Belief Propagation.

BFS= Breadth First Search.

DFS= Depth First Search.

CSPs= Constraint Satisfaction Problems.

b= the branching factor of a tree structure.

d= the depth of the answer in a tree structure.

m= the maximum depth of a tree structure.

L= the depth limit of a tree structure.

CDF= cumulative distribution function.

PDF= Probability density function.

A* = algorithm

الملخص

ترتكز هذه الدراسة بشكل مباشر على دراسة قدرة خوارزميات البحث ومنها خوارزمية آلية بولتزمان المقيدة، وخوارزمية تمرير الرسالة، إضافة الى خوارزمية البحث الأفقي، أولاً وخوارزمية البحث في العمق أولاً، وذلك لحل الألغاز والمشاكل في مجال الذكاء الاصطناعي، حيث سيتم في هذا البحث استخدام هذه الأساليب أو الخوارزميات في صورها الشجرية. لكل من هذه الأساليب، ستتم كتابة خوارزمية تفصيلية وكتابة برنامج بلغة جافا طبقا لقواعدها وتبعا لطبيعتها ومن خلال هذه البرامج يمكن إجراء مقارنة بين هذه الخوارزميات من حيث الكفاءة والأداء، وحيث ان هذه الطرق مستخدمة في حل المشاكل بصورة عامة وفي مجال الذكاء الاصطناعي بصورة خاصة، تم اختيار عاملين أساسيين هما القدرة على حل المشاكل والزمن المستغرق للوصول الى هذا الحل.

في هذا البحث أختير لغز سودوكو الذي يمثل من الناحية الرياضية حل للغز في مصفوفة $N \times N$. سيتم في هذا البحث تحويل لغز سودوكو 9×9 الى التركيب الشجري الذي يناسب تطبيق الطرق المستخدمة بقراءة اللغز في التركيب الشجري الأمر الذي يمكن في نهاية الدراسة بيان فاعلية الطرق أو الخوارزميات سالفة الذكر ومقدرتها على حل اللغز. أثناء الحل سينشأ مسار يسمى بالمسار الصحيح وهو دلالة على إمكانية حل اللغز بالطريقة المستخدمة والتي تتوج بإتمام المسار الذي يعبر على كامل محتويات اللغز الذي يتم استرجاعه على شكل مصفوفة مرة أخرى. في حال عدم تمكن الطريقة من حل اللغز ستظهر رسالة فشل نتيجة المعالجة.

سيتم خلال الدراسة استخدام برنامج ماتلاب لتحليل النتائج المتحصل عليها بخصوص زمن الحل ومقدرة كل من هذه الطرق على حل اللغز إضافة الى إجراء التحليل الإحصائي لإجراء مقارنة دقيقة بينها حيث أظهرت النتائج وبينت الدراسة أن آلية بولتزمان المقيدة أفضل النتائج إذ كان لها أقصر زمن للحل وأحسن إنحراف معياري وأفضل كثافة احتمالية.

Abstract

This research primarily concerned with the ability of searching algorithms such as Restricted Boltzmann Machines, Message Passing, Breadth First Search and Depth First Search to Solve problems in the area of artificial intelligence. In this research, the different methods or algorithms will be used in tree graph form. For each of these methods, the algorithm and the software code will be written based on their rules and nature. Single Java code will be used for these different methods so that a comparison between their effecincy can be accomplished. Since these methods are applied for problem solving in general and in the area of artificial intelligence in specific, Two parameters will be investigated i.e. solving time and the ability to solve a problem.

In this research work, Sudoku puzzle that represents from mathematical point of view an $N \times N$ matrix for which a solution is to be found. The 9×9 Sudoku puzzle is chosen in this research and will be converted to tree graph form so that the above mentioned methods are proper to be applied. Hence each of these different methods will deal with Sudoku puzzle in tree graph form and apply its own procedure and basic rules together with Sudoku rules to find the correct solution for the puzzle. For each method, solution path will be established which means that the solution for the puzzle exiss. This right path will be retransferred back to a complete matrix that contain the puzzle solution. In case the method was not able to solve the puzzle, Failed message will be the result of processing.

The Matlab package will be used to analyze the numerical results and describe the solving time and the unsolved puzzles to find which of the different methods applied has the best performance. In addition, the Matlab will be used for statistical analysis to get accurate comparison results between the different methods. The results show that the Ristricted Boltezmann machine gave the best result when referring to the statistical results obtained from the Matlab and has the shortest solving time when referring to the java code results.

Chapter One

Introduction

1.1. Introduction:

Artificial Intelligence (AI) is the study and creation of computer systems that can perceive reason and act. The primary aim of AI is to produce intelligent machines. The intelligence should be exhibited by thinking, making decisions, solving problems, more importantly by learning. AI is an interdisciplinary field that requires knowledge in computer science, linguistics, psychology, biology, philosophy and so on for serious research (Russell and Norving, 2010).

AI can be defined as the area of computer science that deals with the ways in which computers can be made to perform cognitive functions ascribed to humans. However, this definition does not say what functions are performed, to what degree they are performed, or how functions are carried out. Graphs can be electronically stored in different ways. However; the way of manipulating and the nature of the graph structure directly affects the data structure used (Adedapo, *et al.*, 2015). Theoretically, one can distinguish between list and matrix structures but in concrete applications, the best structure is often a combination of both. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access for some applications but can consume huge amounts of memory (Ferozuddin and Khidir , 2011).

In tree structure, the path from initial state to the final or goal state describes the graph connectivity. However; when searching a graph the connectivity path is traced and examined. The set of possible states, together with operators defining their connectivity constitute the search space. Applying a searching algorithm will result in an output that implies the moving or traveling from the initial state to the goal or final state that satisfies the aim of searching. In real life, search usually results from a lack of knowledge (Rina and Robert, 2007). This means that lack of the relation between the initial state and any of the goal or final state.

1.2. Scope of Study:

This study focuses on the application of searching algorithms in problem solving hence the scope of study is concerned but not limited to the following points:

- ✓ Searching algorithms: There are several searching algorithms considered in this research such as Breadth First Search Algorithms, Depth First Search Algorithms and Message Passing Algorithms, and Boltzmann Machine Algorithms used to solve problems.
- ✓ Code Optimization: Java code is optimized for all algorithms are implemented; statistical analysis is applicable and the results are displayed in histograms using Matlab package
- ✓ Special Sudoku: There are several variations of Sudoku including different sizes of the grid. This study deals with the study of ordinary Sudoku, which is 9x9 grid.

1.3. Problem Statement:

Problem solving in AI may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem, solving by search algorithms is quite common technique. The real art of problem solving is in deciding the description of the states and the operators (Russell and Norvig, 2010).

In this study different kinds of searching algorithms of artificial intelligence are used for problem solving. Sudoku is considered as one of major problem addressed by researcher (Cecilia and Luciana, 2013). Sudoku is one of the interesting puzzles that gained popularity in late 1990s. It is not considered confusing and complex and may be dealt with by pencil and eraser. People started looking for tools, steps or methods to solve Sudoku using searching algorithms (Baptiste and Jean , 2014).

1.4. Problem Description:

For solving problems by computer: flesh out the task and determine what constitutes a solution. The problem is then represented in a language to compute an output that is an answer given to a user or a sequence of actions to be carried out in the environment (Russell and Norvig, 2010). The way problems are described can be formulated as follows:

1. **The Initial State**: The state at which problem will start.
2. **Successor function**: Description of possible actions and their outcomes.
3. **Goal Test**: It determines if the given state is the goal state or not. For example in chess, the goal is to reach a state called checkmate where the opponent's king is under attack and cannot escape.
4. **Path Cost:** A path cost is a function that assigns a numeric cost to each path (seconds time). The problem solving agents choose a cost that reflects its own performance measure, e.g. path distance between the cities as given in Figure1.1.

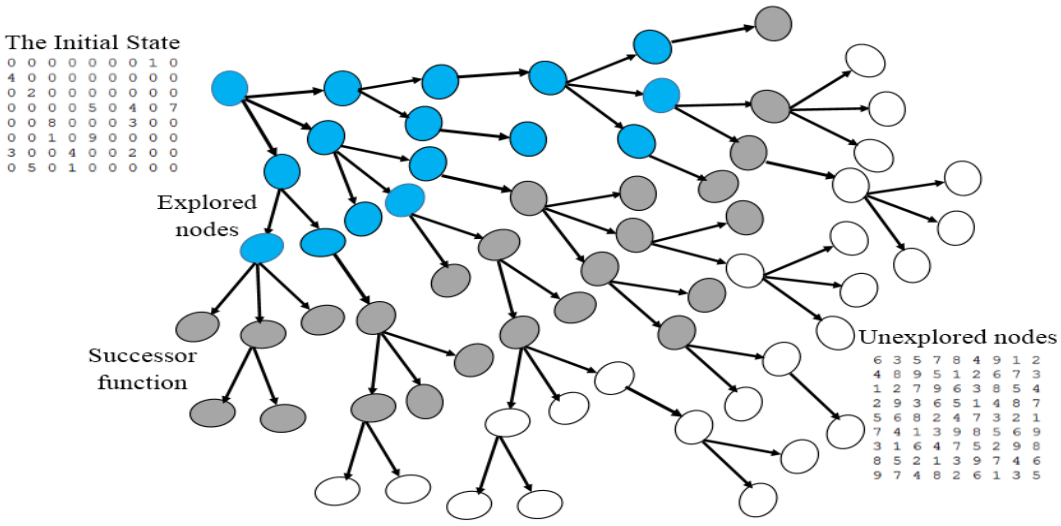


Figure 1.1: Problem solving by graph searching

Primarily the focus is to measure and analyze search algorithms according to their solving potential, and how well those algorithms are suited for parallelizing. Hence applying uninformed searching algorithms, Message Passing Algorithms and Boltzmann Machine to solving Sudoku puzzles as an example.

1.5. Aim and objectives:

This study is basically aimed at solving 9x9 Sudoku puzzle grid using searching algorithms such as Restricted Boltzmann Machine and Message Passing Algorithms, and Breadth First Search Algorithms, and Depth First Search Algorithms and there are certain objectives that will be achieved from this research which can be summarized as below:

1- Make an overview for the previous work in AI area.
2- Study and understand the four algorithms selected in this research, which are the Restricted Boltzmann Machine, Breadth First Search, Depth First Search, and Message Passing algorithm.
3- Develop the data structure appropriate with these algorithms (Tree graph structure).
4- Choose the artificial intelligence case study, 9x9 Sudoku puzzle grid and the way it can be represented to fit the data structure proposed.
5- Implement the selected algorithms to solve the Sudoku puzzle after developing the software code for each of them.
6- Compare the results of each algorithm with others and make the recommendation for the best.

1.6. Thesis Outline:

This research comes in five chapters therefore the report is organized as described below:

- **Chapter 1**: ‘‘Introduction” presented the general introduction about the research problem. In addition, it contains problem statement, field of study, problem description, and goal of study.
- **Chapter 2**: ‘‘Theoretical Background” provides the literature review about the research and basically the relevant and related recent works.
- **Chapter 3**: ‘‘Methodology” this chapter presents the description of the applied algorithms together with the Sudoku problem. Moreover, the way

the data structure is established and the software codes are written is also elaborated.

- **Chapter 4**: ‘‘Results and discussion” in this chapter, the results obtained regarding every searching algorithm is given. The numerical results were presented followed by the algorithm performance. The results obtained were discussed based on their performance from the solving time and the ability to solve point of view.
- **Chapter 5**: ‘‘Conclusions and Future Work” presents the conclusions drawn from the research and raises suggestions for future research.

Chapter Two

Theoretical Background

2.1. Introduction

This chapter describes theoretical foundation for this research. The main focus is on describing Problem Solving. It is considered as one of the corner stones of AI research within two distinguished sub-processes i.e. choosing a knowledge representation and performing a search. The term knowledge representation means including analysis conceptualization and formalization. Well-chosen representation may considerably reduce the amount of search needed to solve a problem (Russell and Norving, 2010).

In AI, problem solving is basically a search; limited search strategies is an essential aspect in this study:

2.1.1. Restricted Boltzmann Machine (RBMs):

A Boltzmann machine is a class of neural networks introduced already in late 1980's. They are based on statistical physics. In contrary to most other neural network methods, Boltzmann Machines are probabilistic graphical models that can be interpreted as stochastic neurons (Wolfgang Maass, 2014).

Restricted Boltzmann Machines are simplified versions of Boltzmann machines; Figure 2.1 shows RBMs, where connections between the hidden neurons (h) and the visible neurons (v) in the original Boltzmann, machines are removed. Only the connections between the neurons in the visible layer and the hidden layer remain (KyungHyun, *et al*., 2010). This simplification make learning in RBMs tractable compared with Boltzmann machines, where it becomes soon intractable due to the many connections except for small-scale toy problems (Marylou, *et al*., 2015).

A Restricted Boltzmann Machine have attracted much attention as building blocks for the multi-layer learning systems called deep belief networks, and variants and extensions of RBMs have found applications in a wide range of pattern recognition tasks For various supervised and unsupervised such as,

dimensionality reduction, Classification, collaborative filtering, and clustering (Asja and Christian , 2014; Hugo and Yoshua, 2008).

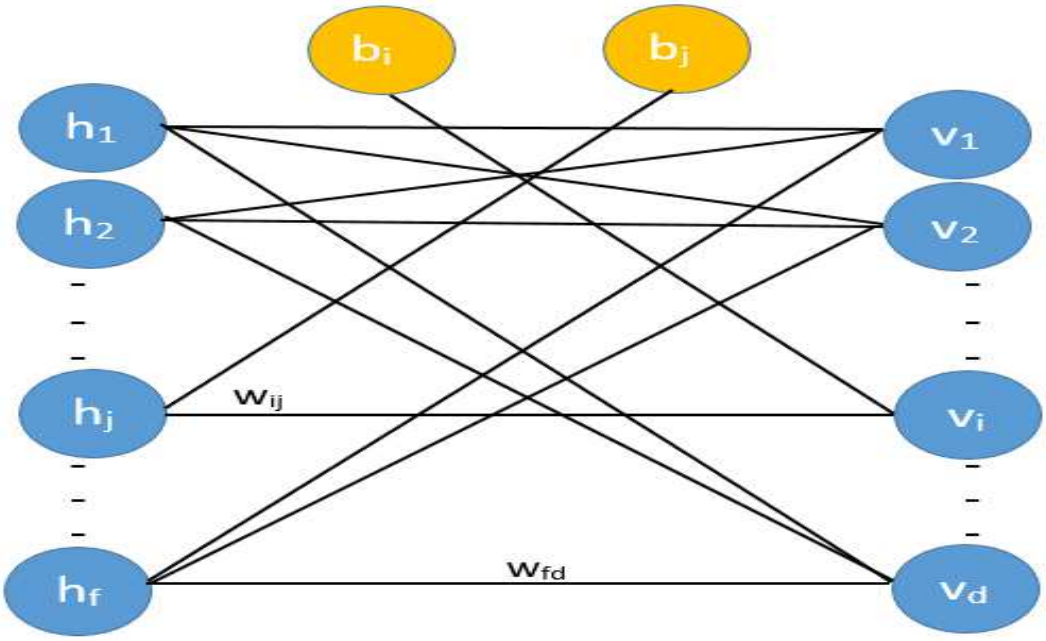


Figure 2.1 : Restricted Boltzmann Machines

RBM is an energy based model and the energy between visible and hidden units is defined by follwing equation (Ruslan, *et al.*, 2007).

$$\boldsymbol{E(v,h,\theta) = -\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} v_i\, h_j - \sum_{i=1}^{D} b_i\, v_i - \sum_{j=1}^{F} a_j\, h_j} \qquad \textbf{(2.1)}$$

Where $\theta = \{W, a, b\}$ represent the parameters of the model. W_{ij} represents the symmetric weight between i^{th} visible unit and the j^{th} hidden. The connection between bias term and the i^{th} visible unit is defined by b_i . a_j connection between bias unit and the j^{th} hidden unit. v and h represent the visible and hidden vectors respectively.

Weights are stored in a weight matrix as in equation 2.2 that satisfies the following conditions:

$$\boldsymbol{w_{ii} = \begin{cases} 0 & \forall\, \mathbf{i} \\ \mathbf{w_{ji}} & \forall\, \mathbf{i, j} \end{cases}} \qquad \textbf{(2.2)}$$

The other properties of unique rows, unique columns and unique sub-grid are encoded in the same way but processed with the whole weight matrix.

Equations 2.3 and 2.4 presents the joint probability distribution of visible and hidden units (Navdeep and Geoffrey, 2011).

$$\boldsymbol{p(v,h,\theta) = \frac{1}{Z(\theta)} e^{-E(v,h,\theta)}} \quad \textbf{(2.3)}$$

$$\mathbf{Z(\theta) = \sum_v \sum_h e^{-E(v,h,\theta)}} \quad \textbf{(2.4)}$$

Due to stochastic nature of Boltzmann machines there is a separate probability function used to determine if a single neuron should flip its state during a discrete time step (Ruslan, *et al.*, 2007). The probability that the model assigns to the visible vector v is given by the following equation:

$$\boldsymbol{p(v,\theta) = \frac{1}{Z(\theta)} \sum_h e^{-E(v,h,\theta)}} \quad \textbf{(2.5)}$$

2.1.2. Message Passing Algorithms (MP):

Message passing provides powerful approximation algorithms for problems that can be formulated in terms of, probabilistic, graphical models such as factor graphs. These methods find applications in statistical physics, inference, and combinatorial optimization.

The methods have been invented several times in different scientific communities (Frank R, *et al.*, 2001) and have applications in such diverse fields as error-correcting codes, combinatorial optimization, artificial intelligence, signal processing, statistical physics, and many more. In addition, probability theory and its applications factor graphs are used to represent factorization of a probability distribution function.

1- Factor Graphs (FG):

A factor graph is a bipartite (or bigraph) graph representing the factorization of a function. Enabling efficient computations, such as the computation of marginal distributions through the sum-product algorithm (Arunkumar and Komala, 2015).

Formally, a factor graph, G = (V, E) is a bipartite graph with two types of nodes i.e.:

- ✓ **Variable nodes**. These correspond to the vertices in the original graphical model and are represented by circle nodes in the factor graph e.g. V = {1, 2, 3, 4} be the set of four variable nodes.
- ✓ **Factor Nodes**. These correspond to the set of factors defining the joint probability distribution and are represented by square nodes in the factor graph. e.g. F = {f_1, f_2, f_3, f_4} be the set of four factor nodes.

The structure of this factorization can be expressed as in equation 2.6 (Frank R, et.al, 2001).

$$\boldsymbol{f(1,2,3,4) = f_1(1,2)f_2(1,3,4)f_3(2,4)f_4(4)} \qquad \textbf{(2.6)}$$

Figure 2.2, shows a factor graph with four variables and four constraints defined on these variables. The typical representation uses squares for factor nodes and circles for variable nodes.

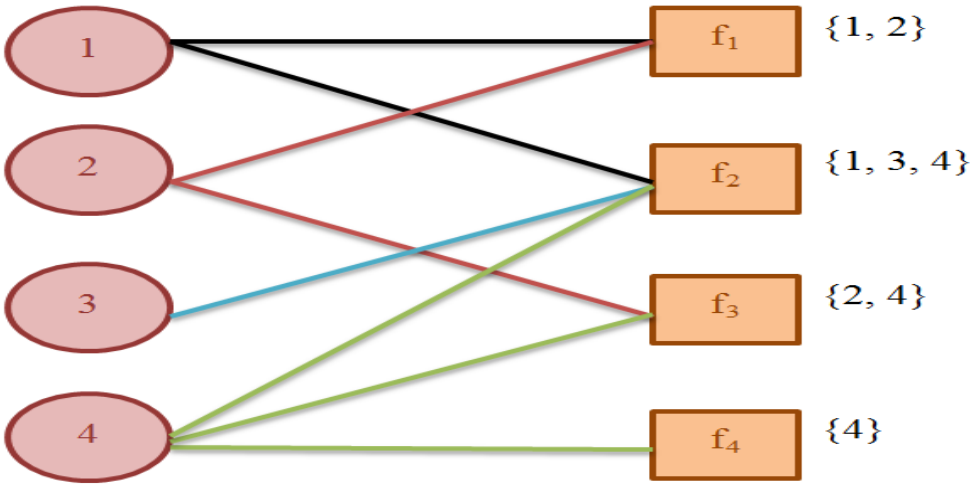


Figure 2.2 : A factor graph

2- *The Sum-Product algorithm:*

The Sum-Product algorithm (Belief Propagation (BP)) is a particular instance of a message-passing method. The sum-product algorithm is designed to calculate approximations of marginal probability distributions in loopy graphical models (Jonathan S, *et.al.,* 2005).

The Sum-Product algorithm is an efficient method for obtaining an approximate solution for the marginal probabilities p(x), which is exact in the case of loop-free factor graphs. The Sum-Product algorithm, normally presented as message update equations on a factor graph, involving messages between variable nodes and their neighboring factor nodes and vice versa is based on passing messages between adjacent nodes of the underlying factor graph (Andre, *et al.,* 2014; Frank R, *et al.,* 2001). In other word, the sum-product is defined in equations as the following:

- The constraint-to-variable messages $r_{mn}(x)$ equation 2.7 is given by:

$$\boldsymbol{r_{mn}(x) = \sum_{\substack{\{n' \in N_{m,n}\} \\ n=x\,, n'=x_n\,,\ all\ unique}} \prod_{l \in N_{m,n}} q_{lm}(x_l)} \qquad \textbf{(2.7)}$$

  Where is $n = x$ is the variable node in vector $N_{m,n}$ and $n' = x_n$ represent the neighbours of *x* in Factor Graph (FG) that all are unique.

- The Posteriori beliefs $q_n(x)$ is presented in equation 2.8 (Kristian, *et al.,* 2009) as below:

$$\boldsymbol{q_n(x) = P(n = x) \prod_{m \in M_n} r_{mn}(x)} \qquad \textbf{(2.8)}$$

  The message *m* is a vector must contain all possibility of factor node into the specific domain M_n for a variable node.

- The variable-to-constraint messages $q_{n,m}(x)$ is given in equation 2.9 as following:

$$\boldsymbol{q_{n,m}(x) = P(n = x) \prod_{m' \in M_{n,m}} r_{mn}(x)} \qquad \textbf{(2.9)}$$

Where is $P(n = x)$ Represent a priori probabilities vector of variable node (Olusegun,*et al.,*2014) if the variable node is k then the element k in probability vector is 1 and all other elements are 0. And $m' \epsilon M_{n,m}$ represent the neighbours of a message m into the specific domain $M_{n,m}$.

2.1.3. The Uninformed Searching Algorithms:

Uninformed Search is also called Brute force or blind search which is a uniformed exploration of the search space and it does not explicitly take into account either planning efficiency or execution efficiency (Farhad, *et al.*, 2012).

Uninformed Search is a general problem solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem statement or not. It uses only the information available in the problem definition. Several types of uninformed strategies are (Russell and Norving, 2010):

1- Breadth-First Search.
2- Uniform-Cost Search.
3- Depth-First Search.
4- Depth-limited Search.
5- Iterative Deepening Search.

In searching algorithms, domain specific knowledge is not required. In the case of uninformed search, the state description is required. It describes the present state from which the transition will start. In addition, a set of legal operators is also needed. These operators will direct and specify the way of movement. The initial state is necessary as a reference point for the present to next state. A final goal state is also required so that we know whether the target point is searched or not (Beamer, Krste , and David, 2012).

The only thing that a blind search can do is to differentiate between a non-goal state and a goal state. All blind search algorithms must take O (b) time and use O (d) space (Scott, *et al.*, 2012).

We have limited our implementation to solve Sudoku puzzle into Breadth First Search and Depth First Search:

1. Breadth Frist Search (BFS):

Breadth First Search is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root, selecting a node as a root. This root node will be at the top of the tree. The top is searched and it is level-by-level traversal and then the nodes below are searched for the solution. This search will be with all below level nodes. This maybe considered as a drawback of this algorithms use more memory but will always find the shortest path first; due to it is searching nature (Akanmu T, *et al*., 2010).

The properties of BFS are presented in Table 2.1. It exhaustively searches the entire graph or sequence without considering the goal node or solution until it finds it. From the standpoint of the algorithm, all child nodes obtained by expanding a node are added to a First in_First out queue. First in_First out means that nodes accessed first explored or expanded first. This elaborates the level-by-level traversal policy (Charles, 2010).

Table 2.1: The properties of Breadth First Search

Property	Description
Class	Search algorithms
Data structure	Graph
Worst case performance	$O(\|V\| + \|E\|) = O(b^d)$ every vertex and every edge will be explored in the worst case. | V | is the number of vertices and | E | s the number of edges in the graph.
Worst case space complexity	$O(\|V\|) = O(b^d)$ to find the nodes that are at distance *d* from the start node (measured in number of edge traversals) where *b* is the "branching factor" of the graph.

As shown in Figure 2.3 an abstraction of the search order all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded (Rong and Eric, 2009). Numbers indicate the order in which nodes are expanded.

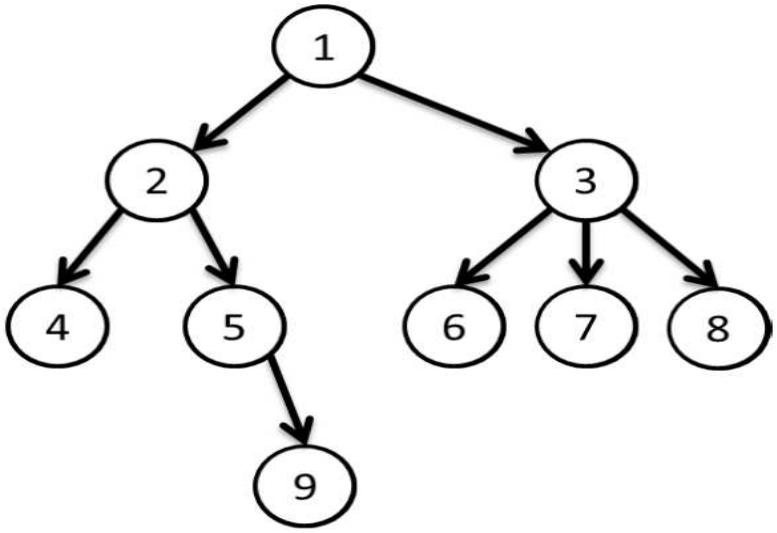


Figure 2.3: Breadth Frist Search

2. *Depth First Search (DFS):*

As searching algorithms, Depth First Search is applied for traversing graph or tree structure. One starts at the root (selecting a node as the root in the graph case) and explores as far as possible along each branch before backtracking (Farhad, *et al.*, 2012). Backtracking means traversing to the upper level since the present position is not the goal state or solution. This maybe considered as a drawback of this algorithms. Since it may go down a very long branch without ever coming to the solution node.

Table 2.2 shows the properties of the DFS algorithms. The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time O(|V| + |E|), linear in the size of the graph. In these applications it also uses space O(|V|) in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices (Sanjay and Chander , 2014).

Table 2.2: The properties of Depth First Search

Property	Description
Class	Search algorithms
Data structure	Graph
Worst case performance	$O(\|V\| + \|E\|)$ for explicit graphs traversed without repetition, $O(b^d)$ for implicit graphs with branching, factor b searched to depth d
Worst case space complexity	$O(\|V\|)$ if entire graph is traversed without repetition, $O(\text{longest path length searched})$ for implicit graphs without elimination of duplicate nodes

It searches the graph by expanding each branch to the deepest node, as depicted in Figure 2.4. Numbers indicate the order in which nodes are expanded. DFS is based on the Last in_First out principle where the last item placed on the top of the stack is the first item to be removed. A drawback from using DFS is

that the algorithm can search down a very long branch without ever coming to the solution node (Olusegun, *et al.*, 2014).

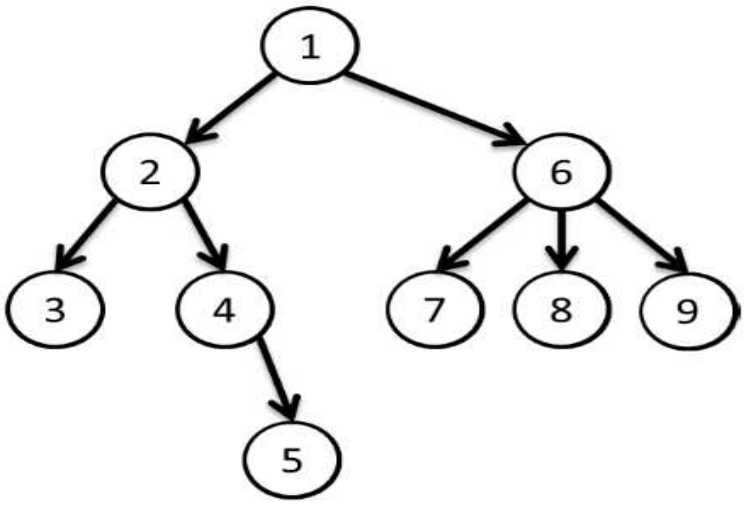


Figure 2.4: Depth First Search

2.1.4. Sudoku Puzzle Problem:

Sudoku is a kind of game problem found in Japan in the mid 80's and it become popular for researchers as an image to be deal with by scientific tools such as neural networks. However, similar games had existed before that it became widely popular around 2004 when it started to appear in newspapers where it became an appreciated alternative to the traditional crosswords. Today Sudoku does not only appear in newspapers but also as computer games and in international Sudoku championships (Behrooz , 2009).

Sudoku is a popular logic-based combinatorial puzzle game {1, 2, 3…9}. It consists of 81 cells, contained in a 9x9 grid. Each cell can contain a single integer ranging between one and nine. As shown in Figure 2.5 the grid is further split up into nine 3x3 sub-grids. The purpose of Sudoku is to fill up the entire 9x9 grid such that the following constraints are met (Bertram and Frazer, 2006):

- ✓ Each row of cells is only allowed to contain the integers one through nine exactly once.
- ✓ Each column of cells is only allowed to contain the integers one through nine exactly once.
- ✓ Each 3x3 sub-grid is also only allowed to contain the integers one through nine exactly once.

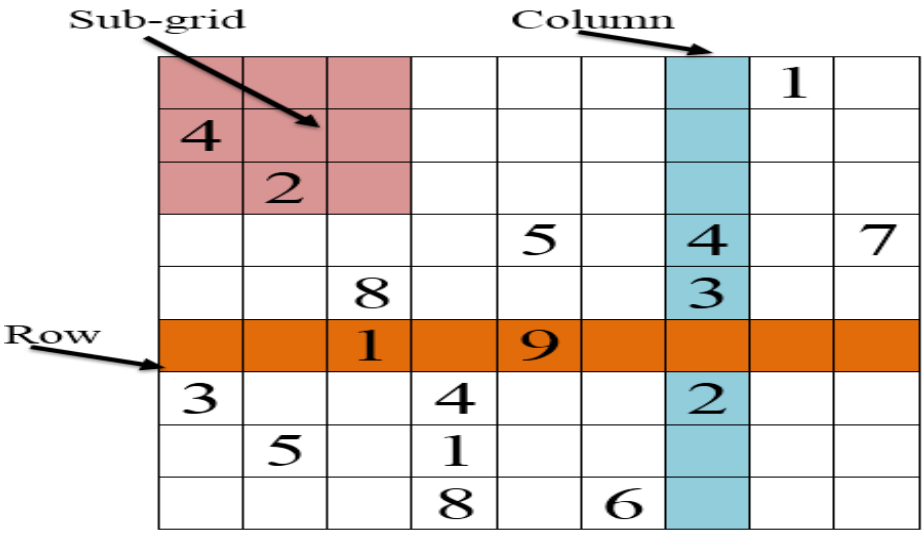


Figure 2.5 : 9x9 Sudoku Puzzle grids

The Sudoku puzzle is a special case of a more general type of problems called Constraint Satisfaction Problems, in this section; both specific Sudoku problem and the general Constrained Satisfaction Problem (CSPs) are described.

1. The specific Sudoku problem:

Specific Sudoku problem exists different clues levels (David and Steven, 2013). The difficulty is based on how many clues are given from the beginning and partially on the layout of these clues. A Sudoku usually has a minimum of 17 clues with only one, distinct, solution. Not all of them are playable by humans, at least not without guessing. A common property of the unsolvable Sudoku is that they have multiple solutions, which means that not enough clues has been given to provide a distinct solution (Xiuqin, *et al.*, 2013).

A number of cells in the grid are pre-defined by the puzzle setter, resulting in the Sudoku puzzle having a single, unique solution. Figure 2.6 depicts a typical Sudoku puzzle and these puzzles are set at different difficulties.

The puzzle in Figure 2.6 is a 9x9 puzzle since it can be solved logically and does not require guessing and its solution is shown in Figure 2.7.

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

Figure 2.6 : 9x9 Sudoku Puzzle

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

Figure 2.7 : The solution to the Sudoku Puzzle in Figure 2.6

2. *Constraint Satisfaction Problems (CSPs):*

CSPs is a collection of variables all of which have to be assigned values, subject to specified constraints. Computational problems like scheduling a collection of tasks, or interpreting a visual image, can all be seen as CSPs. We are interested in finding a satisfying assignment for the CSPs, which means that we need to assign values to each of the variables from their respective domain spaces (Todd K and Jacob H, 2006), such that all the constraints are satisfied. Figure 2.8 shows Sudoku with Constraint Satisfaction.

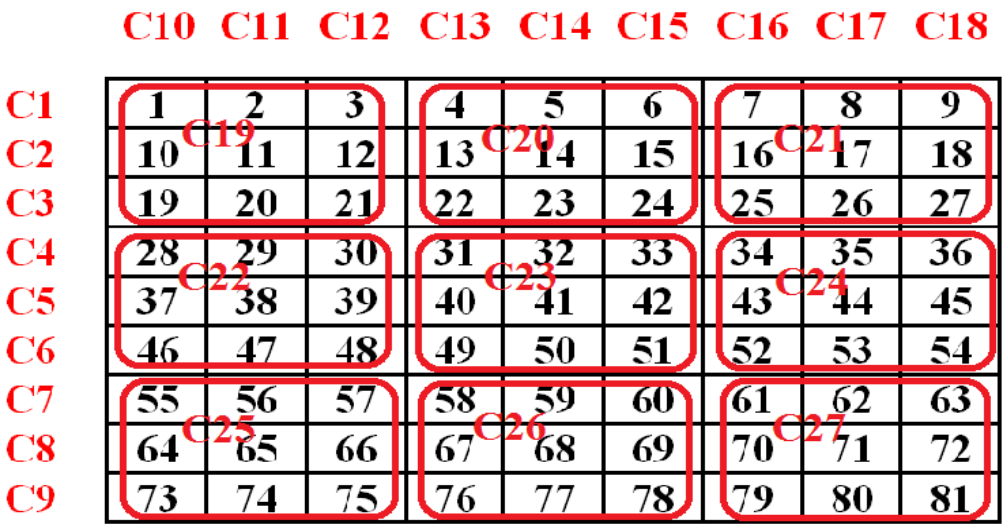


Figure 2.8 : Sudoku with Constraint Satisfaction

We can defined A Constraint Satisfaction Problem as:

A finite set of variables, $X = x_1, x_2 \ldots x_n$.

1. A domain for each variable, $D(X) = D(x_1), D(x_2) \ldots D(x_n)$, also denoted as $d_1, d_2, \ldots, d_n$, where di is the domain of xi.
2. A finite set of constraints C(X), where $c(x_1 \ldots x_n)$ denotes a constraint involving variables $x_1 \ldots x_n$.

In other words, the 9×9 Sudoku puzzles define constraints as:

1. Variables: empty slots, and Domains = {1,2,3,4,5,6,7,8,9}
2. Constraints: 27 all-different (Todd K & Jacob H, 2006).

2.2. Related Work

Geoffry, *et al.* (2006) show how to use complementary priors to eliminate the explaining away effects that make inference difficult in densely connected belief nets that have many hidden layers. With complementary priors, they derived fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. In their work, they found it to be easily trainable when the parameters initialized by greedy layer wise pre training using an RBM.

Todd K & Jacob H (2006) show the constraint satisfaction using a Tanner graph. In their work, they found that the MP paradigm is straightforward to apply to some problems with multiple constraints, with solutions obtained over discrete sets. In addition, the computational complexity is localized to each constraint. The belief propagation algorithm applied to this graph to solve Sudoku puzzle problem. The computational complexity is localized to each constraint. They conclude that cycles lead to failures in some cases, due to biases in the message passing process.

Andrea, et al. (2007) presented that the message passing algorithm is surprisingly successful in solving hard constraint satisfaction problems on sparse random graphs. Its outcome provides a heuristic to make choices at next step. Based on belief propagation (BP), they consider a simple randomized decimation algorithm, and analyze its behavior on random k-satisfiability formulae. According to their work, a vertical tangent point in the $k = 3$ function $\varphi(\theta)$ in the regime is not exist. They expect to control through the tree computation, namely $\alpha < \alpha(3) \approx -3.86$. On their work , they argued that the threshold can be computed through a tree model and evaluated via density evolution.

Heiko (2008) shows that the message-passing methods provide powerful approximation algorithm for problems that can be formulated in terms of (probabilistic) graphical models. In applications such as statistical physics, Sudoku puzzle, simple optimization problem, the message-passing algorithm can help to solve. He

examined the sum-product and max-product algorithms as two generic instances of message-passing methods that can help to solve highly difficult Sudoku puzzles. His message-passing approach to Sudoku is not superior to backtracking, the sum-product algorithm needs up to one minute to find a solution. In case of N×N Sudoku, the max-product algorithm function-node processing of the max-product algorithm can be mapped to a polynomial problem. However; for the sum-product algorithm, no such mapping is known. The fact that function-node processing's computational complexity depends on system size is a very unusual feature.

Tanya, et al. (2008) studied the convergence and stability properties of the family of reweighted sum-product algorithms or belief propagation algorithm, in which messages are adjusted with graph-dependent weights. They derive various conditions that are sufficient to ensure convergence, and provide bounds on the geometric convergence rates. The experimental simulations on various classes of graphs validate our theoretical results. Even though they have established the benefits of including observation potentials, the conditions provided are still somewhat conservative, since they require that the message updates be contractive at every update, as opposed to requiring that they be attractive in some suitably averaged sense. As an example, when averaged over multiple iterations, or over different updating sequences, an interesting direction would be to derive sharper "average-case" conditions for message-passing convergence.

Min-Quan, *et al.* (2009) proposed a puzzle solving algorithm to treat these problems. Based on the fact that most of Japanese puzzles are compact and contiguous, some logical rules are deduced to paint some cells. Then, the DFS algorithm with the "branch and bound" scheme, which is used to do early termination for those impossible paths, is used to solve those undetermined cells. The experimental results show that their algorithm can solve Japanese puzzles successfully, and the processing speed is significantly faster than that of DFS. Their method can solve those puzzles with compact black patterns quickly using branch and bound scheme, and provides correct solutions as well.

Chiung-Hsueh, *et al.* (2009) presented a puzzle solving algorithm to treat these problems. Based on the fact that most of nonograms are compact and contiguous, some logical rules are deduced to paint some cells. They used the chronological backtracking algorithm to solve those undetermined cells and logical rules to improve the search efficiently. Their experimental results show that their method can solve those puzzles with compact black patterns quickly. The DFS speed can be improved in case of puzzles with random black patterns using the pruning scheme. The algorithm proposed can solve nonograms successfully, and the processing speed is significantly faster than that of DFS. Moreover, their method can determine that a nonogram has no solution.

In (2009) David, *et al.* introduced a simple cost less modification to iterative thresholding making the sparsity-undersampling tradeoff of the new message passing algorithms equivalent to that of the corresponding convex optimization procedures. The new iterative-thresholding algorithms are inspired by belief propagation in graphical models. Their empirical measurements of the sparsity under-sampling tradeoff for the new algorithms agree with theoretical calculations. They show that a state evolution formalism correctly derives the true sparsity under-sampling tradeoff. They found a surprising agreement between earlier calculations based on random convex polytopes and this apparently very different theoretical formalism.

Todd, *et al.* (2009) In their paper, the Sudoku puzzle is a discrete constraint satisfaction problem is as the error correction decoding problem. Since the puzzle has a Tanner graph representation, the belief propagation (BP) algorithm can be adapted to solve Sudoku. They proposed an algorithm for solution to the Sinkhorn puzzle based on Sinkhorn balancing. A proof of convergence is presented, with some information theoretic connections. In addition, random generalization of the Sudoku puzzle is presented, for which the Sinkhorn-based solver is also very effective. It is interesting that the Sinkhorn balancing technique is more effective than the BP.

In (2010) Zhengbing, , *et al.* investigated the use of hardware which physically realizes quantum annealing for machine learning applications. They show how to take

advantage of the hardware in both zero- and finite-temperature modes of operation. At zero temperature their hardware is used as a heuristic minimizer of Ising energy functions. At finite temperature, the hardware allows for sampling from the corresponding Boltzmann distribution. They rely on quantum mechanical processes to perform both these tasks more efficiently than is possible through software simulation on classical computers. They show how Ising energy functions can be sculpted to solve a range of supervised learning problems. Finally, they validated the use of the hardware by constructing learning algorithms trained using quantum annealing on several synthetic and real data sets to solve Sudoku on an N×N grid rather than just a 9×9 grid. A class of polynomial problems, called P, requires a solution time that increases as a polynomial of the problem size n, e.g. n^2.

In (2010) Jussi considered an alternatives to pure depth-first search, and show that carefully chosen randomized search order, which is not strictly depth-first, allows to leverage the intrinsic strengths of CDCL better, and will lead to a planner that clearly outperforms existing planners. In their work, they didn't use the weights of decision variables obtained from conflicts as a part of variable selection. Such weights would be able to order the top-level goals and subgoals in the computation of actions based on their role in conflicts. They believe that this is the most promising area for future improvement in the implementations of their variable selection scheme.

Charles (2010) developed a multithreaded implementation of breadth-first search (BFS) of a sparse graph using a novel implementation of a multiset data structure, called a "bag," in place of the FIFO queue usually employed in serial breadth-first search algorithms. They found that reducers hide the particular nondeterministic manner in which associativity is resolved. Thereby allowing the programmer to assume specific semantic guarantees at well-defined points in the computation. This encapsulation of non-determinism simplifies the task of reasoning about the program's correctness compared to a TLS solution.

Lijuan, *et al.* (2010) presented a new GPU implementation of breadth first search that uses a hierarchical queue management technique, and found that it guarantees the same computational complexity as the fastest sequential version and can achieve up to 10 times speedup. They claim that their work is most suitable for accelerating sparse and near-regular graphs, which are widely seen in the field of EDA. In addition they conclude that both methods proposed in their work hierarchical queue management and hierarchical kernel arrangement are potentially applicable to the GPU implementations of other types of algorithms, too.

Duane,*et al.* (2011) presented a BFS parallelization focused on fine-grained task management that achieves an asymptotically optimal O(|V|+|E|) work complexity. Their implementation delivers excellent performance on diverse graphs, achieving traversal rates in excess of 3.3 billion and 8.3 billion traversed edges per Second using single and quad-GPU configurations, per second using single and quad-GPU configurations, respectively. This level of performance is several times faster than state-of-the-art implementations both CPU and GPU platforms.

Ruslan & Geoffrey E (2012) show that a Restricted Boltzmann Machines (RBMs) used for collaborative filtering. On images of 10 handwritten digits (0 to 9), Pretraining a stack of RBMs using contrastive divergence used to initialize the weights of a deep Boltzmann machine to sensible values. In addition, they found that the variational inference can be initialized sensibly by a single bottom-up pass from the data vector using twice the bottom-up weights to compensate for the lack of top-down input on the initial pass.

Farhad, *et al.* (2012) proposed new blind algorithm for solving n- queens using combination of DFS and BFS methods. Their results showed that performance and run time in this approach better then back tracking methods and hill climbing modes. As a result, the DFS algorithm is quicker than BFS algorithm and Number of extended node in DFS algorithm is less than BFS algorithm as well as the required memory for BFS is larger than DFS. Using this method, time and cost of solving n-queens problem is minimized in comparison of old methods.

Cecilia & Luciana (2013) proposed a new approach for solving Sudoku which is by modelling them as block-world problems. In block-world problems, there are a number of boxes on the table with a particular order or arrangement. The objective of this problem is to change this arrangement into the targeted arrangement with the help of two types of robots. In their work, they present three models for Sudoku and modellized Sudoku as parameterized multi-agent systems. They use Temporal Logic of Actions (TLA) for formalizing our models.

Nate, *et al.* (2013) provide a study where a description of all techniques to conclude with some general remarks, comparing message-passing algorithms based on ADMM with the family of belief propagation (BP) algorithms the sum-product version of belief propagation, used to compute marginal probabilities in graphical models, to solving Sudoku puzzle, they showed potentially solves non-convex problems much faster than Divide and Concur algorithms, have important advantages over the more widely used belief propagation algorithms, and they believed these algorithms have a promising future with many possible applications.

Stefano, *et al.* (2013) used the model Selection problem in graphical models, specifically in the context of hand-written digit recognition. A graphical model specified as a factor graph and train Restricted Boltzmann Machines (RBM). They found that the method is a massively parallelizable. In addition, anytime algorithm which can also be stopped early to obtain empirically accurate estimates that provide lower bounds with a high probability.

Nathan (2013) provide a study of the puzzle game Fling, used a number of brute-force search techniques as breadth-first search to enable designers to explore puzzles and how modifications of the puzzles influence solvability. In their work, they represent the preliminary stages of work on automated large-scale analysis of puzzles with the purposes of semi- or fully-automated design of new puzzle instances. In order to improve and understand the limits of the approach, much more work is needed, stretching the applicability of simple searches for puzzle design.

Baptiste & Jean (2014) in their work, they designed and implemented a complete solution to detect and recognize a Sudoku in an image taken from a phone-camera. The digits are recognized using a Deep Belief Network (DBN) are typically implemented as a composition of simple networks, such as a Restricted Boltzmann Machines (RBMs). They found that their method provided successful on dataset, achieving 87.5% of correct detection on the testing set. Only 0.37% of the cells were incorrectly guessed.

Wolfgang Maass (2014) discussed Boltzmann machines as computational perspective architectures for solving large constraint satisfaction problems, with Sudoku puzzle. They also described why the results were paving the way for a qualitative jump in the computational capability and learning performance of neuromorphic networks of spiking neurons with noise.

Jossy (2014) presented a research about "SUDOKU puzzles". They described the role model strategy as a convex program whose solution is the Bayesian optimal estimator in training. By running the sum-product rule and the estimator-in-training in parallel offline during a simulation, they showed that the strategy reduces to Monte Carlo integration in the non-parametric case; In fact, applications of post-processing optimization for sub-optimal estimators are burgeoning in the literature.

Caroline & Jossy (2014) discussed Codes based on Sudoku puzzles. They belief that propagation decoding introduced for the erasure channel by exchanging messages containing sets of possible values and represented by a factor graph. In their work, they showed some preliminary numerical results, listing thresholds that emerge from the density evolution recursion, alongside with a conjectured estimate for the rate of long Sudoku-type codes.

Rutuja & Payal (2014) used two methods to search the tree, the depth-first and the breadth-first search. In their work, they calculated many tree nodes in the same depth in the current game tree, which is the breadth-first search. Moreover; each cycle in the search process takes in the deepest nodes of the current game tree. The approach can take advantage of the capability of GPU to compute massive

nodes in parallel and GPUs flexibility to test by implementing it for Sudoku. The results of implementation can be compared with serial implementation of game tree search.

Sanjay & Chander (2014) used Depth First Search as solving algorithm for Sudoku puzzles. They created Sudoku puzzles in different levels respectively within tolerable time using their developed solving and generating algorithms while guarantees each of these puzzle has a unique feasible solution. Their model is very efficient, because it is able to solve every Sudoku instance in a very short time. It takes 0.2 seconds to 0.5 seconds for the diabolic ones according to their difficulty levels. This fact showed that the different difficulty between two levels affects the performance of a human solver but not that of the program. A mathematical model has been formulated.

Olusegun, *et al.* (2014) implemented Depth First Search and Breadth first search and showed that Breadth first search is complete, optimal based on some condition. In contrary, the time and space complexity is exponential. Depth first search space complexity is linear but it is neither complete nor optimal. Depth First iterative Deepening Search is a good search strategy and is better than both DFS and BFS. They concluded that most uninformed search algorithms have serious drawbacks. They came up with the results that Breadth First Search uses too much space and Depth First Search uses too much time and is not guaranteed to find a shortest part to a solution.

Taruna, *et al.* (2015) applied backtracking, brute force approach to solve Sudoku. They found that the proposed algorithm is able to solve such puzzles with any level of difficulties in a short period of time (less than one second). Moreover, the brute force algorithm seems to be a useful method to solve any Sudoku puzzles and it guarantee to find at least one solution. The algorithm does not adopt intelligent strategies to solve the puzzles.

Jossy & Joned (2015) described a non-linear codes with local permutation constraints inspired by Sudoku puzzles. Regarding the decode; they showed that permutation constraints require the evaluation of a permanent using trellis. Decoders are

specialized by them to ensure channels where the operation becomes a trellis. For the encoder, they described a universal approach to encoding a code with local constraints, and discussed its limitations when based on a sub-optimal decoder. In their work, the noticed that having started as a tool for teaching belief propagation,the study of codes with non-linear constraints is having unexpected repercussions. It has brought up some interesting technical hurdles, taught us a few things about non-linear constraints and how different they behave from linear constraints.

Chapter Three

Methodology

3.1. Introduction:

This chapter presents the methodology of the research and give details about each point in the proposed method. In addition, the modifications that might be interesting to do to each algorithm for solving Sudoku puzzle using Java programming language. The Net Beans IDE 8.0.1 for Java Development is applied to complete the coding aspect that have been described in chapter two and will be presented in the coming sections. Moreover, the Parallelising algorithms will also be discussed and the comparison between these will be in this section.

3.2. Sudoku Puzzles problem:

Earlier we show that a Sudoku grid can be filled with many configurations of symbols, only a few of which are valid solutions for which search strategies are applied. In addition, the Parallelising how to improve searching based on the structure of the problem taking into account. All search algorithms make use of the concept of a search space, which is the set of states being searched for a solution.

A previously generated puzzle from WebSudoku.com and converting them in the format of line (Series) in two text file from different clues (17 and 27) Dataset, each file contain 10 Sudoku puzzles so our program for solving can use them. In addition, save output for the solution of the current Sudoku puzzle, and the time to find the solution will be saved into text file in the format of matrix 9×9. In other hand, a separate text file will be used to save the time and one to save the solution for each method and for each case.

Figure 3.1 shows the text file as input contains the numbers 1 through 9 with a “0” indicating a blank space on the Sudoku grid. The program is set to output the solution of the current Sudoku puzzle, and the time for the solution to be found.

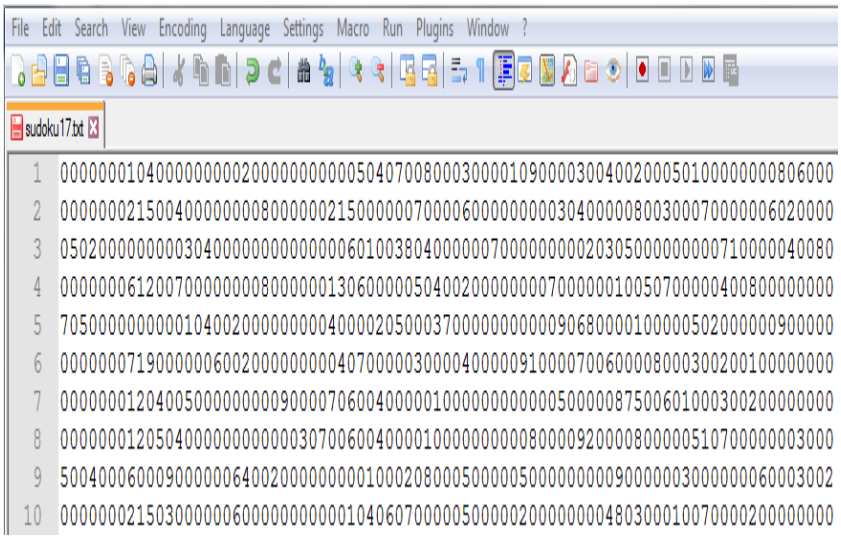


Figure 3.1: 10 Sudoku puzzles dataset text file

3.3. Data Structures for Search Strategies:

The main idea to solve the game is to try each combination of numbers in each empty spot making sure that the rules are not violated. The Sudoku puzzle will be used as a practical example of the solving algorithms discussed in previous sections.

The Search Strategies for solving Sudoku puzzles is implemented in Java. The source code for this implementation is given the appendix. The algorithm reads Sudoku puzzle from the text file.

3.3.1. Restricted Boltzmann Machines Strategy:

Restricted Boltzmann Machines are probabilistic graphical model that can be interpreted as stochastic neural networks.

We are interested in the geometry of the set of all possible marginal probability distributions of the visible variables. Which can be interpreted as stochastic neural networks. Binary RBMs, in which each variable conditioned on the others is Bernoulli distributed, are able to approximate arbitrarily well any distribution over the observable variables (Le Roux and Bengio, 2008; Montufar and Ay, 2011).

For implementation (RBM) on Sudoku, The energy measure with weights and the state associated for every node of the 81×9 =729 nodes in Sudoku network, which is used to determine if the state of single neuron should be flipped, is defined. Each node has a binary state of either on or off. This is also translated to solutions, where only one of every nine neurons in groups will be in the state on. For every number being placed on the grid, there is nine possible assignments, and the probability of a neuron being activated is defined equation 3.1 as follows:

$$\boldsymbol{P_{i=on}} = \frac{\mathbf{1}}{\mathbf{1}+ \boldsymbol{e}^{-\frac{\Delta E_i}{T}}} \tag{3.1}$$

E is the summed up energy of the whole network into neuron *i*, which is a fully connected to all other neurons. *T* is a temperature constant controlling the rate of change during several evaluations with the probability $P_{i=on}$ during simulation (Wolfgang Maass, 2014). A neuron *i* shows in Figure 3.2.

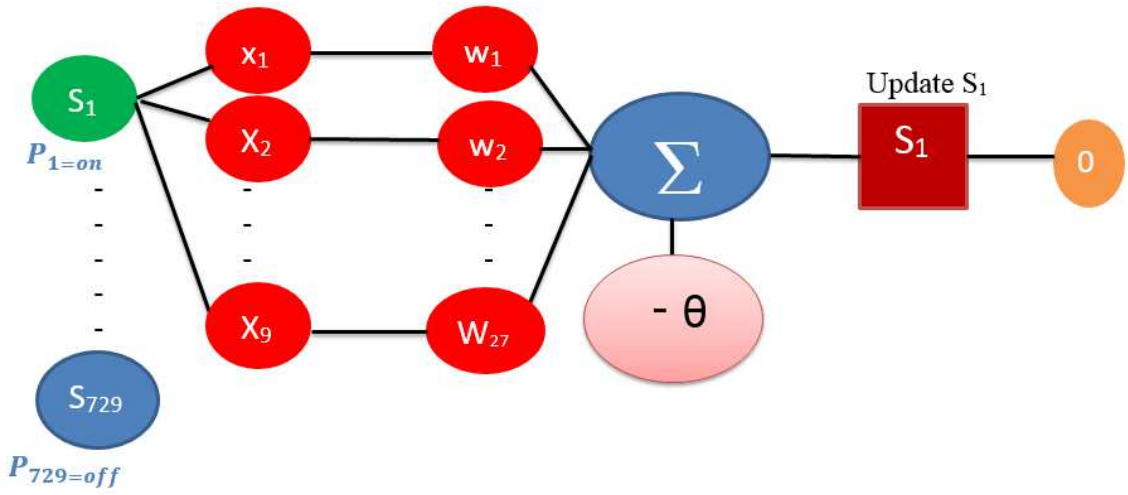


Figure 3.2: A single neuron in Sudoku grid

A single neuron is a single computation unit it begins by summing up all weighted input and thresholding the value for some constant threshold Θ. Where random weight matrix and random states is will be initialized for the 81×9 = 729 nodes in the neural network. Then the given grid values are inserted as activations of states in the associated groups of neurons with all others being set to the off state. This ensures that only the given activations are active, Encode

Sudoku restrictions into the weight matrix. Negative connections are introduced between different digits within the same group.

With all configurations done it is then the time to launch the simulation that runs in discrete time steps. At every time step, the energy function is evaluated for all nodes. The energies are then processed which renders the probabilities of nodes flipping state, which is then performed to varying degrees. Since simulation is used with a lowered temperature T there also small adjustments at certain intervals. This is done in a timely manner which allows all puzzles to be solved. A valid solution can be found by looking at all states and inspecting the requirements of Sudoku. Commonly the solution will appear at the end of the simulation when temperature has been lowered to its minimum. As in Figure 3.3.

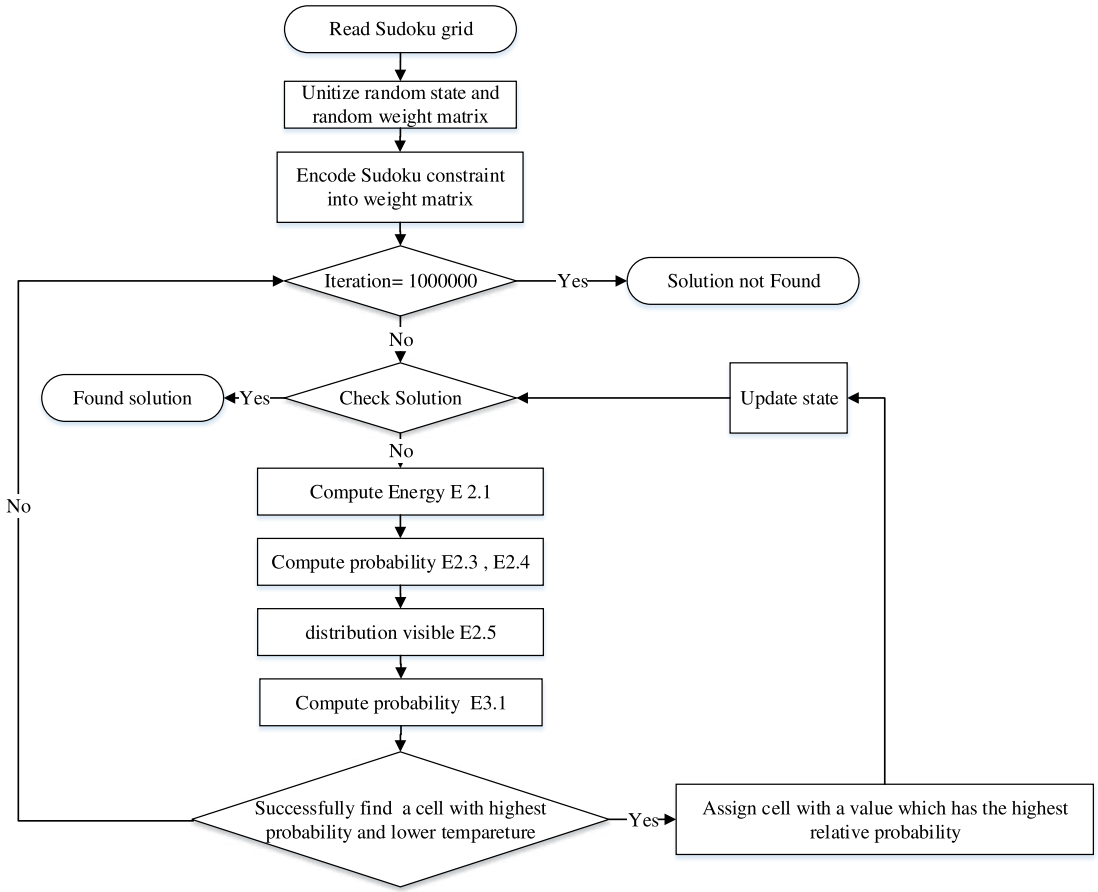


Figure 3.3: Flow chart for implemented RBM algorithm.

3.3.2. Message Passing Strategy:

Message passing is used for finding the probability distribution of the variables over the space of satisfying assignments in a Constraint Satisfaction Problems that can be factorized in terms of local functions (factors).

It involves passing local messages (according to some message update rules) over the edges of a factor graph with nodes for the variables and for the factors. Each factor node is connected to every variable node over which the factor is defined. Communication among nodes is established by sending messages over the edges. Where a node collects results from a sub-part of the graph and communicates it to the next neighbor via a message.

The Sudoku puzzle can be represented as a bipartite graph. In the graph, 9x9 Sudoku puzzle cells can be mapped to set 'S' and all the constraints that need to be satisfied can be mapped to set '*C*'. All cells (S_n) are mapped to the set '*S*' by sending probabilistic messages between adjacent nodes.

Each message *m* is a vector, must contain all possible numbers as Domain= {1, 2, 3, 4, 5, 6, 7, 8, 9}, meaning the possibility of one number can be filled into the specific cell. After several rounds of message passing, different rows, columns and sub-grid constraints (C_m) are mapped to a set '*C*' in that order. Once the two sets are defined, edges are introduced based on the relationship between the cell and the constraint. An (undirected) edge is presented between a cell node and a constraint node, if the constraint is applicable to that cell node.

Sudoku with constraints satisfaction shown in Figure 2.8 is mapped applying message passing as presented in Figure 3.4 show that the number of cell nodes is 81 and the number of constraint nodes is 27. The domain associated with each cell node is 9. Dimensions of N_m matrix is 27×9=243. Dimensions of M_n matrix is 81×3=243, that appear into flow chart will be initialize with zero as in Figure 3.5.

- The $r_{mn}(x)$ in equation 2.7 the message that constraint C_m sends to cell S_n is the probability of satisfying constraint $C_m$defined when cell S_n takes the value x in vector {1,2,3,4,5,6,7,8,9}, and the message is passed from a constraint to a cell in message vector as:

  $r_{mn}(x) = p\ (C_m\ is\ satisfied | S_n = x)$

  $r_{mn} = [\ r_{mn}(1) r_{mn}(2)\ r_{mn}(3)\ r_{mn}(4)\ r_{mn}(5)\ r_{mn}(6)\ r_{mn}(7)\ r_{mn}(8)\ r_{mn}(9)$

  $$r_{mn}(x) = \sum_{\substack{\{n' \in N_{m,n}\} \\ n=x\ , n'=x_{n'}\ ,\ all\ unique}} \prod_{l \in N_{m,n}} q_{lm}(x_l)$$

- The $q_n(x)$ in equation 2.8 the calculated cell vector value based on the messages sent and received. And the probability message cell S_n sends to constraint C_m is the probability for all other constraints associated with cell S_n besides C_m that satisfied when cell S_n takes value . The $q_{n,m}(x)$ in equation 2.9 the message that cell S_n sends to constraint C_m is defined as:

  $q_{(m,n)} = P(S_n | all\ constraints\ involving\ S_n\ except\ C_n\ are\ satisfied).$

  Also the message passed from cell S_n is a message vector $q_n(x)$.

  $$q_{n,m}(x) = P(n = x) \prod_{m' \in M_{n,m}} r_{mn}(x)$$

- The $P(n = x)$*:* is a priori probabilities vector of cell S_n if the cell is filled with number k , $k\epsilon$ { 1,2,3,4,5,6,7,8,9} then the element k in probability vector is 1 and all other elements are 0 as $p_n =$ [1 0 0 0 0 0 0 0 0].

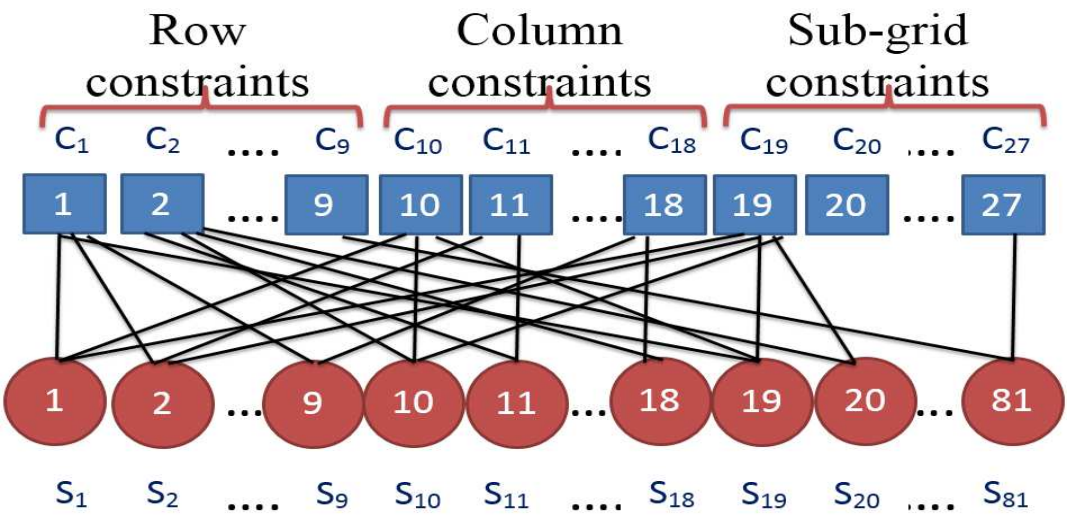


Figure 3.4: The factor graph of a 9 × 9 Sudoku solver

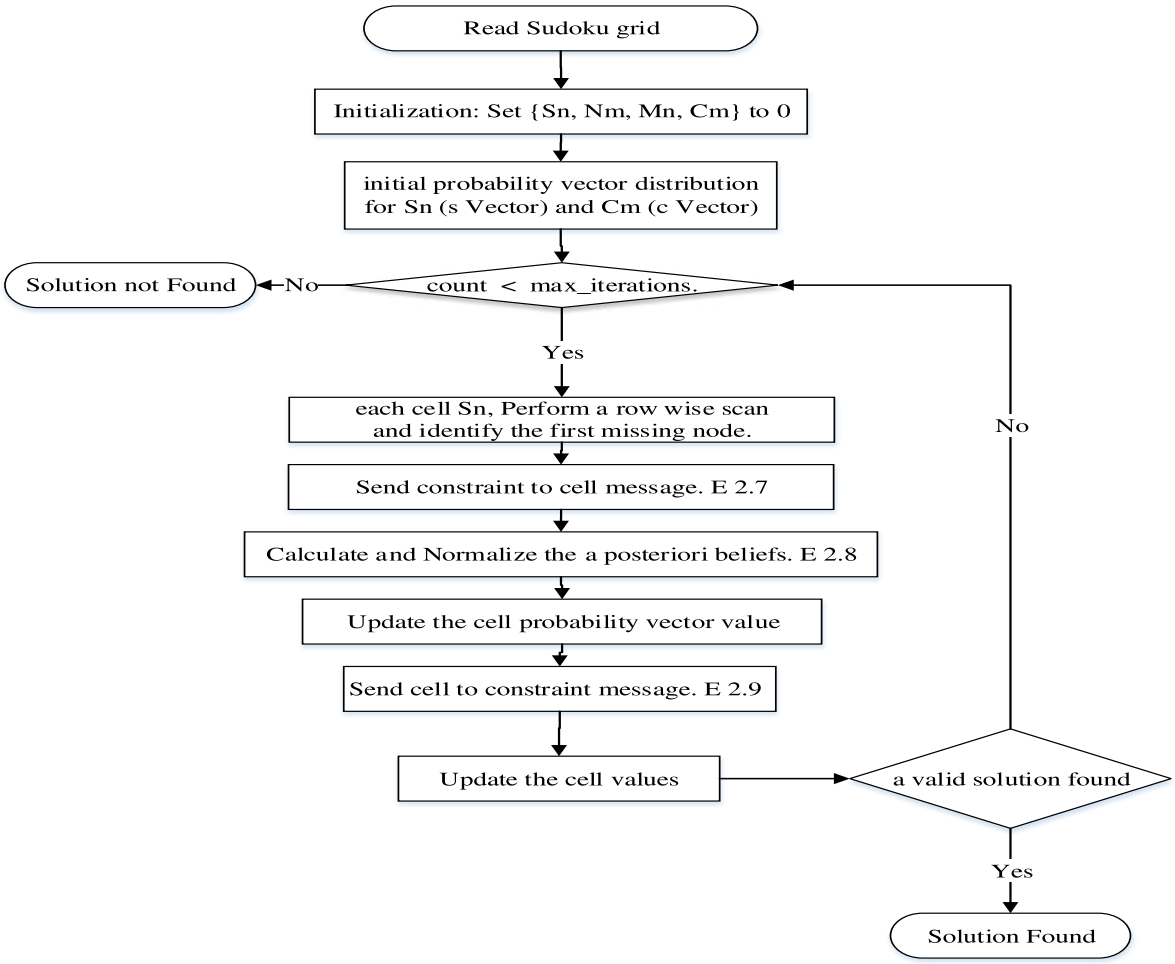
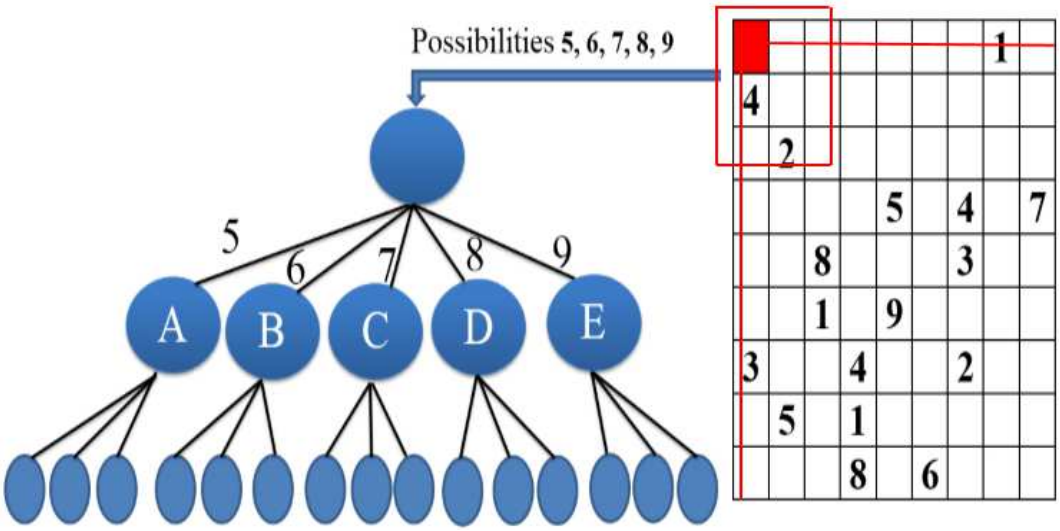


Figure 3.5: Flow chart for implemented MP algorithm.

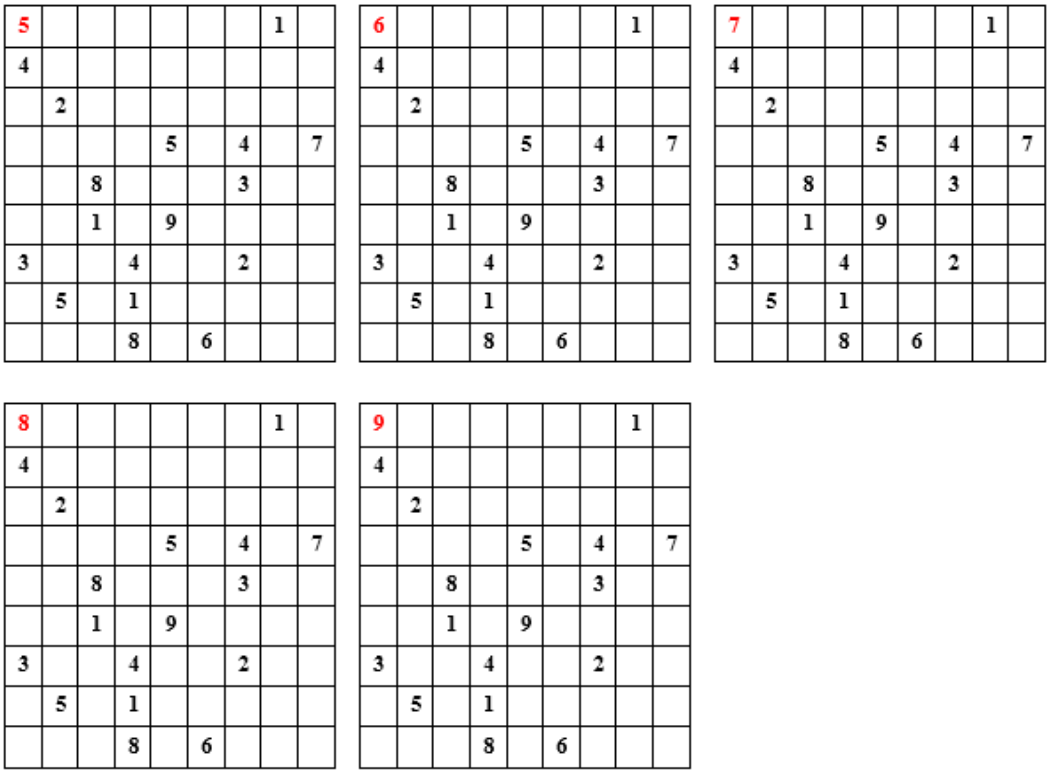
3.3.3. Breadth First Search Strategy:

BFS is a general technique of traversing a graph. A path-finding algorithm is capable of always finding the solution, if one exists. BFS expands nodes in order of their distance from the root level-by-level; each node in the search tree is expanded in a breadth wise at each level. These all expanded nodes are retained till the search is completed (Scott, *et al*., 2012).

We have used the concept of a queue to expand the states from a graph. The Breadth-First Search expands the nodes horizontally and the expansion process storing one copy of the board for each state. As shown in Figure 3.6. (a) and (b) copy of the board.



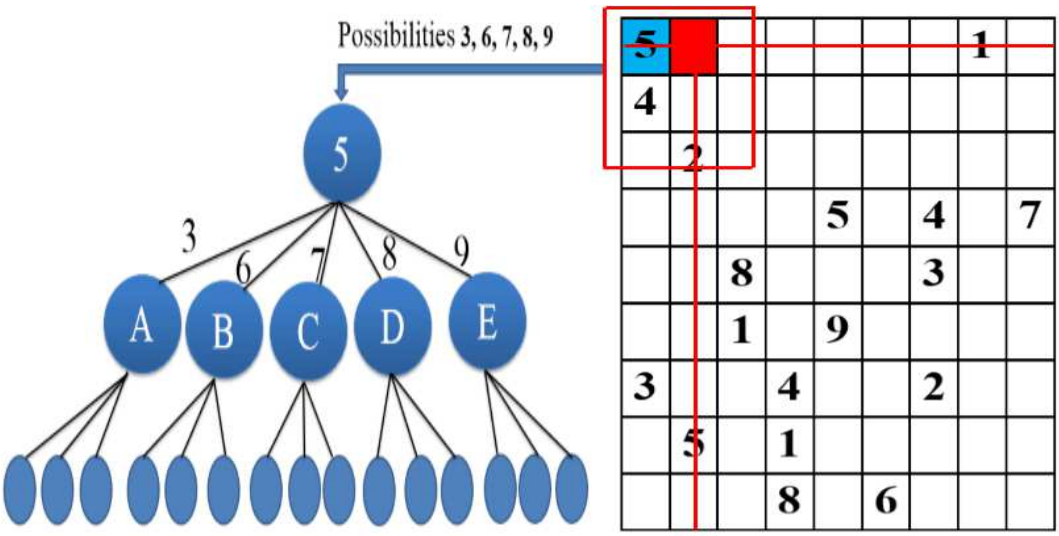
(a) The search tree is expanded



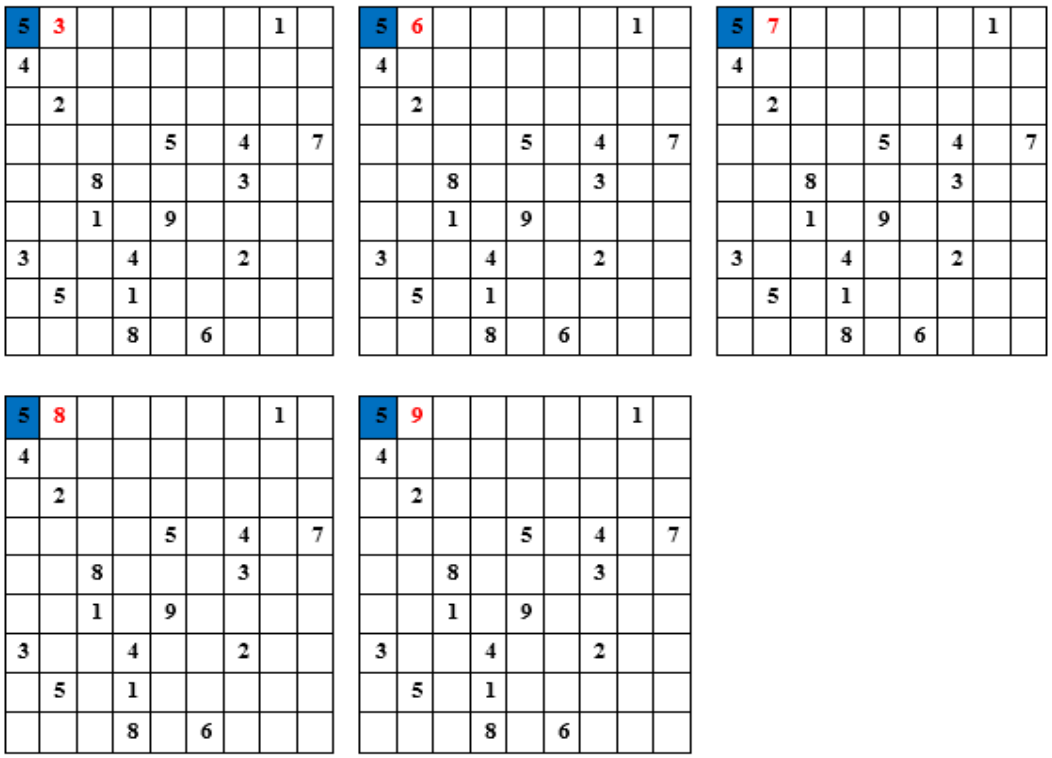
(b) Copy of the board

Figure 3.6: First step searching using Breadth-First Search

Figure 3.7 (a) shows that before expanding the next level to the bottom, it expands all the nodes from the same level first. (b) copy of the board for current state.



(a) The search tree is expanded



(b) Copy of the board

Figure 3.7: Second step searching using Breadth-First Search

Now we have nodes, we need somewhere to put them. The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy.

The appropriate data structure for this algorithm is a queue. Furthermore, BFS uses (First In_First Out) queue to guarantee that the nodes are going to be expanded in the same order that we are created. First Input_First Output queue order.

Figure 3.6 A and E nodes were already expanded, these they do not belong to the queue anymore. The expansion order is inverse to the creation order. This the five first nodes were generated in the order as follows: 1^{st} - A, 2^{nd} - B, 3^{rd} - C, 4^{th} - D and 5^{th} - E. The first node to be expanded is the first node inserted in the queue - node A.

Figure 3.8 Queues are characterized by the order in which they store the inserted nodes. Which pops the oldest element of the queue (Charles, 2010). The operations on a queue are as follows:

- ✓ **EMPTY** (Queue) returns true only if there are no more elements in the queue.
- ✓ **POP** (Queue) removes the first element of the queue and returns it.
- ✓ **INSERT** (element, Queue) inserts an element and returns the resulting queue.

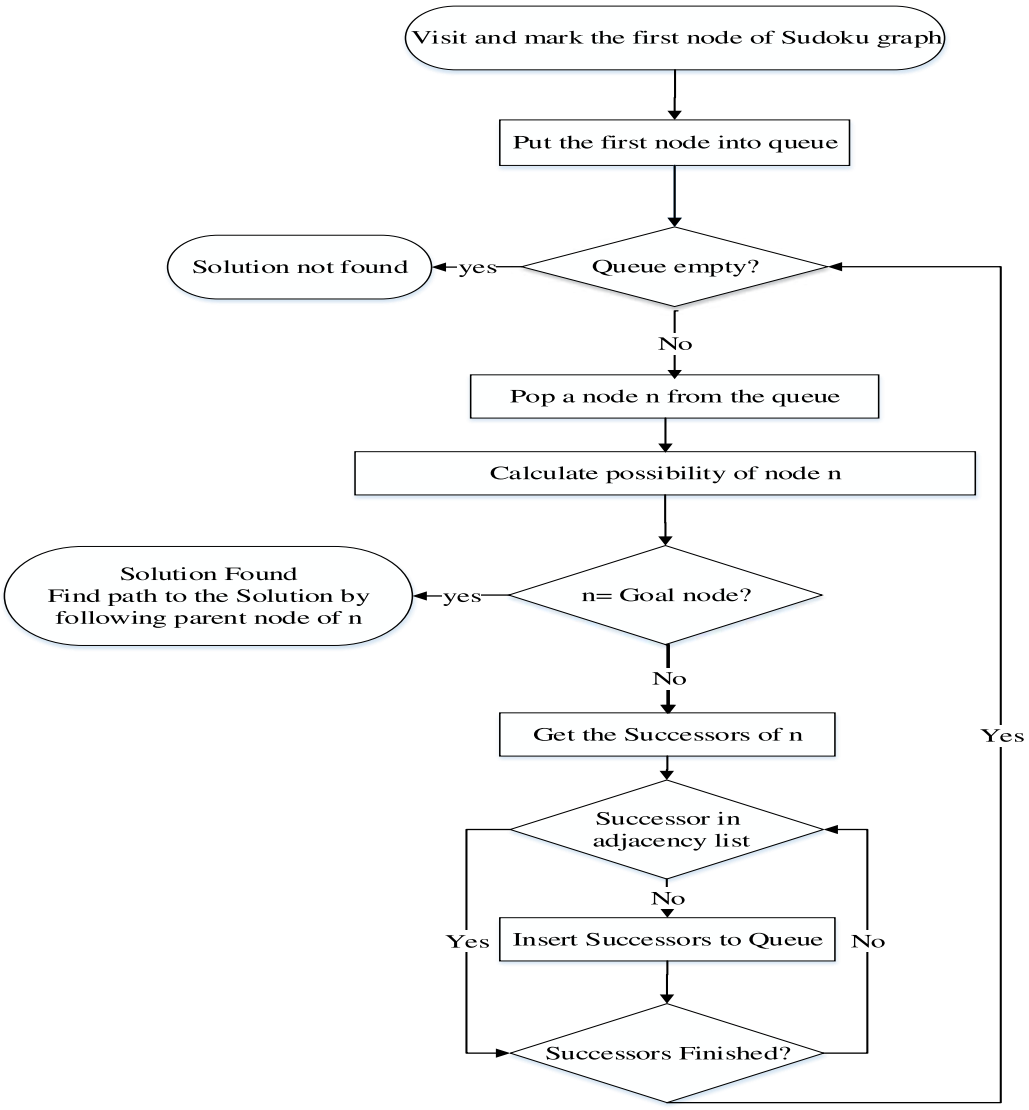
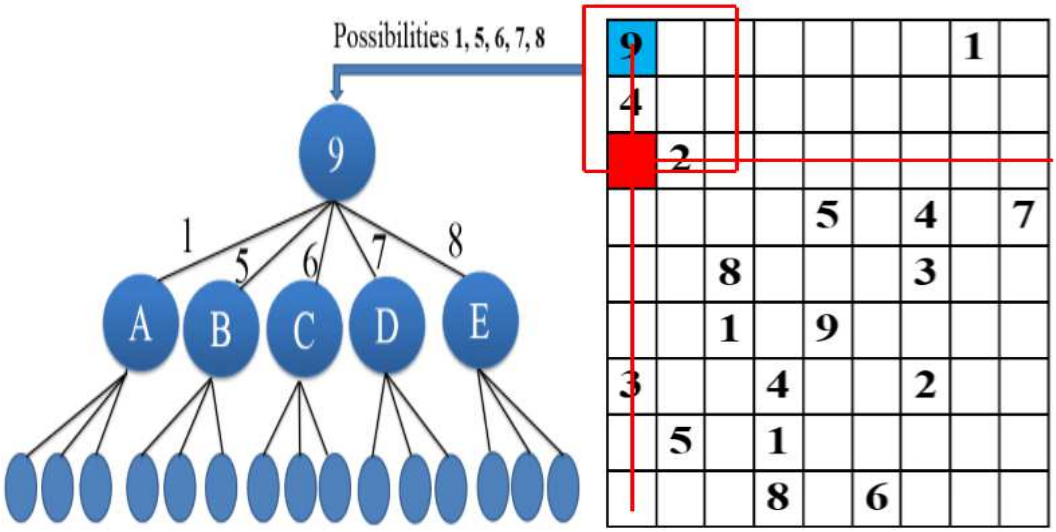


Figure 3.8 : Flow chart for implemented BFS algorithm

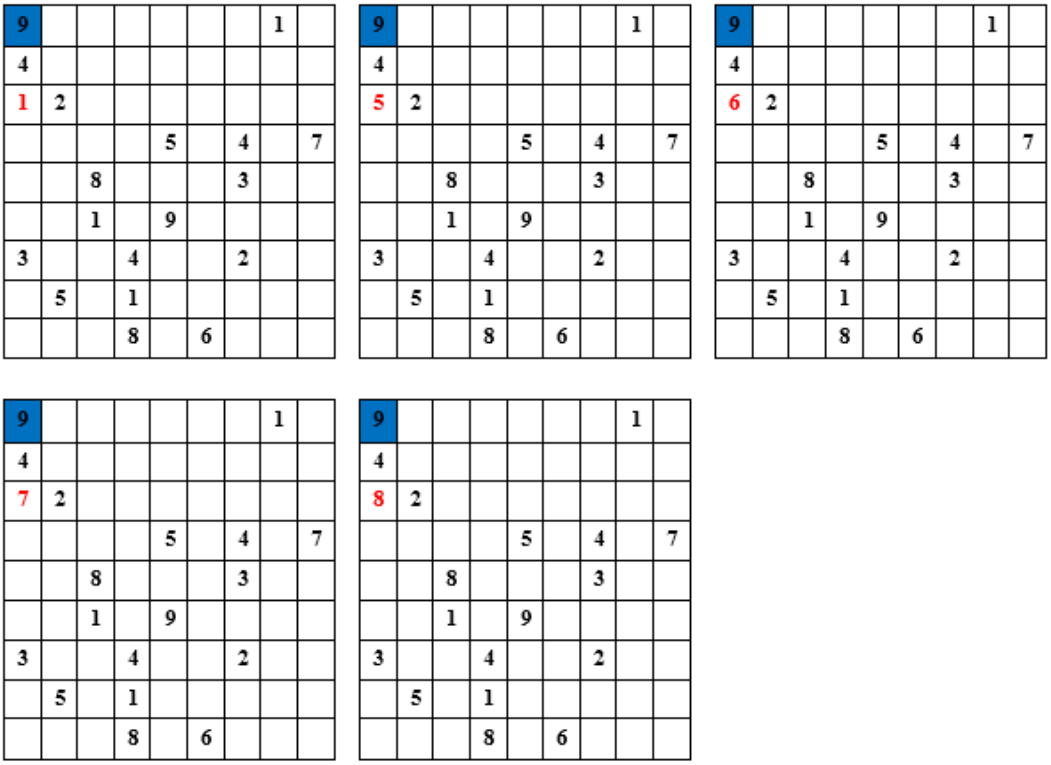
3.3.4. Depth First Search Strategy:

Depth first search is also important type of uniformed or blind search. It chooses to go deeper into the graph. DFS visits all the vertices in the graph using the concept of a stack to expand nodes vertically, and after DFS visited all the reachable vertices from particular source vertices it chooses one of the remaining undiscovered vertices and continues the search. In addition, stop increasing the depth only if a goal or a dead-end is reached. To make this mechanism possible, the algorithm uses the concept of a Last in_First out Queue as shown in Figure 3.6 (a) and (b) Copy of the board.

Similar to FIFO, the expansion order is inverse as the created order. The five first nodes were generated in the order as follows: 1st - A, 2nd - B, 3rd - C, 4th – D, and 5th – E as shown in Figure 3.6 and storing one copy of the board for each state. The first node to be expanded is the last node inserted in the queue E. and as shows in Figure 3.9.



(a) The search tree is expanded



(b) Copy of the board

Figure 3.9: Second step searching using Depth-First Search

Figure 3.10 Flow chart for implemented DFS algorithm as queue (also known as a stack) represents the next nodes that are going to be visited and consecutively expanded. In this case, the newest element of the queue is popped out. And the priority queue, which pops the element of the queue with the highest priority according to some ordering function.

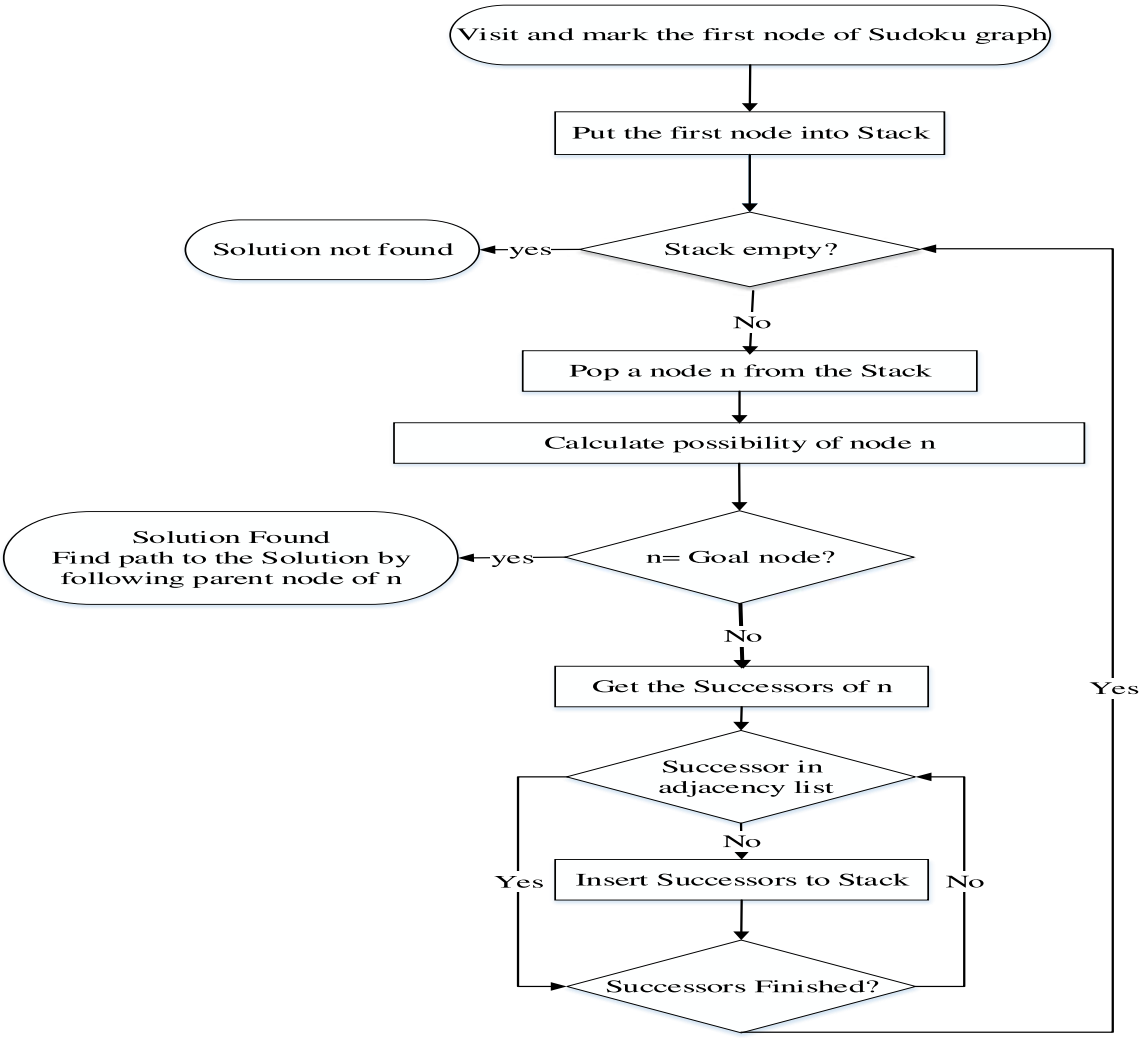


Figure 3.10: Flow chart for implemented DFS algorithm.

Chapter Four

Results and Discussion

4.1. Introduction:

The results obtained from different experiments with different strategies and algorithms will be presented and discussed in this chapter.

4.2. Numerical results:

As described in the previous chapter, the algorithms are applied for solving 20 Sudoku puzzles. Out of these 20 puzzles, some are unsolved whereas; most have been solved. The numerical results obtained are presented in table 4.1 below:

Table 4.1: Comparison of Algorithms performance to Solve Sudoku puzzles

Algorithm	Total 20 Sudoku		Time taken for Solving		Percentage of Success
	Solved	Unsolved	Minimum	Maximum	
Restricted Boltzmann Machine	18	2	6.5536×10^{-4} sec	0.0118 sec	90 %
Message passing	11	9	814481410 sec	1.1530×10^{11} sec	55 %
Breadth First Search	20	0	0.3808 sec	277.3736 sec	100 %
Depth First Search	20	0	3.1130×10^{-4} sec	36.6744 sec	100 %

The above table shows that out of 20 Sudoku puzzles two were unsolved when applying Restricted Boltzmann Machine percentage of success 90% of the total puzzles. On the other hand, percentage of success in case of Message Passing about 55%, the results for Breadth First Search and Depth First Search the percentage of solved puzzles is 100%.

The table 4.1 also shows the maximum and minimum time taken by different algorithms to solve the Sudoku puzzle. In addition, the Restricted Boltzmann Machine is the best among all four algorithms with time taken to solve the 9×9 Sudoku puzzles is the shortest about 0.0118 sec. On the other hand, the DFS comes in the second position with a solving time longer than RBM and solving all puzzles. These results will be presented graphically later.

4.3. Algorithms Performance:

It is common that different algorithms will have different performance from the point of view of efficiency and other parameters such as time taken for Solving and correctness. The representation of how different algorithms are sorted in terms of time taken for Solving so that we can plot puzzle index versus time taken for Solving distributions to get an idea about the performance of each algorithm. There are different ways by which performance can be descripted, one of those is histograms that show the frequency at which puzzles fall into a set of time intervals. Also shows possible underlying features such as if, the Sudoku solver solves the puzzle with an already known distribution and this applied to with the four algorithms. Where the first 10 puzzle of 20 case study Sudoku puzzles is 17 clues and the second is 27 clues for all algorithms.

4.3.1. Restricted Boltzmann Machine Performance:

The Boltzmann machine is applied as one of the methods for solving the 20 Sudoku problems. It is found that the solutions obtained from Boltzmann machine are different and still a valid solution. The time taken for the 20 experiments is given in Figure 4.1 below.

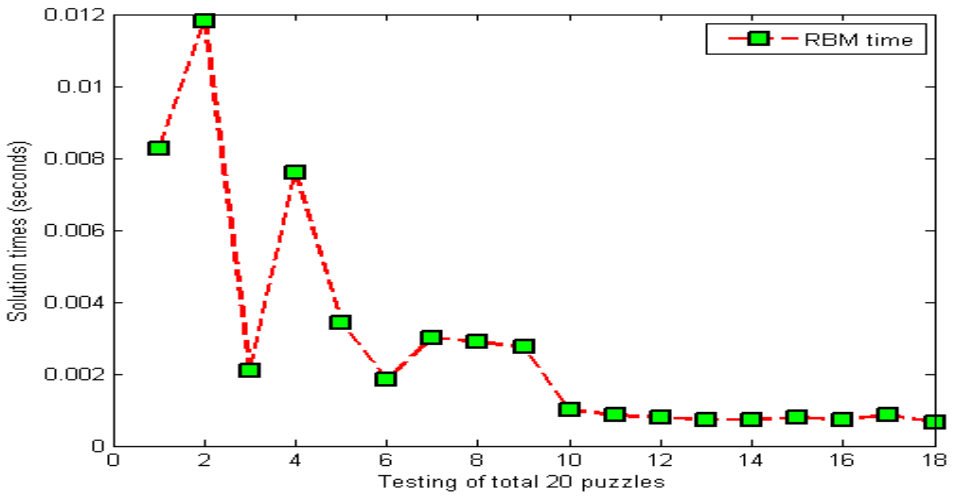


Figure 4.1: Time distributions for restricted boltzmann machine

Figure 4.1 shows the Boltzmann's machine was able to solve 18 puzzles out of 20. Two puzzles reported as unsolved one in the case of 17 clues puzzle and one in the 27 clues Sudoku. The time distribution in the above figure shows that the solving time in the case of 27 clues is less than of the 17 clues puzzle. It is noticed that puzzle 2 in the 17 clues took the longest time followed by puzzle number the eight. The time mentioned in the above figure is in seconds hence except two of the puzzles have been solved in a time less than 0.005 sec. the number of iterations taken by this method varies from 975 to 447490 iteration.

Figure 4.2 shows histogram for the time taken to solve different puzzles, which clearly shows that the common time taken lies between 0.0007 to 0.004 seconds.

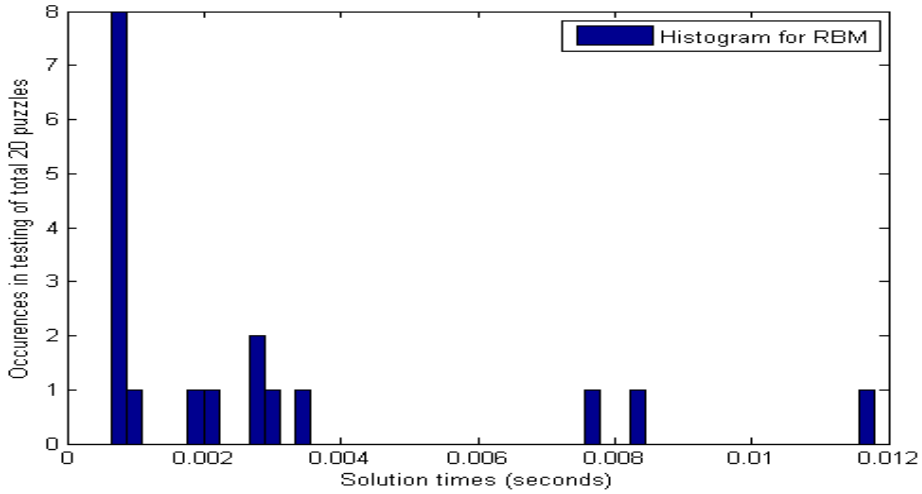


Figure 4.2: The histogram for restricted boltzmann machine

Figure 4.2 shows that the highest occurrence in testing of total 20 puzzles is near to 0.001seconds as time taken to solve the 27 clues puzzles.

4.3.2. Performance of Message Passing:

The message passing algorithm performance shows that the time taken for solving different puzzle is not convergent. Moreover, there is no difference from the solving time point of view in the case of 17 and 27 clue cases. Figure 4.3 shows that the time taking is much greater than in the case of boltzmann machine since it approaches 12×10^{10} seconds in puzzle number 6.

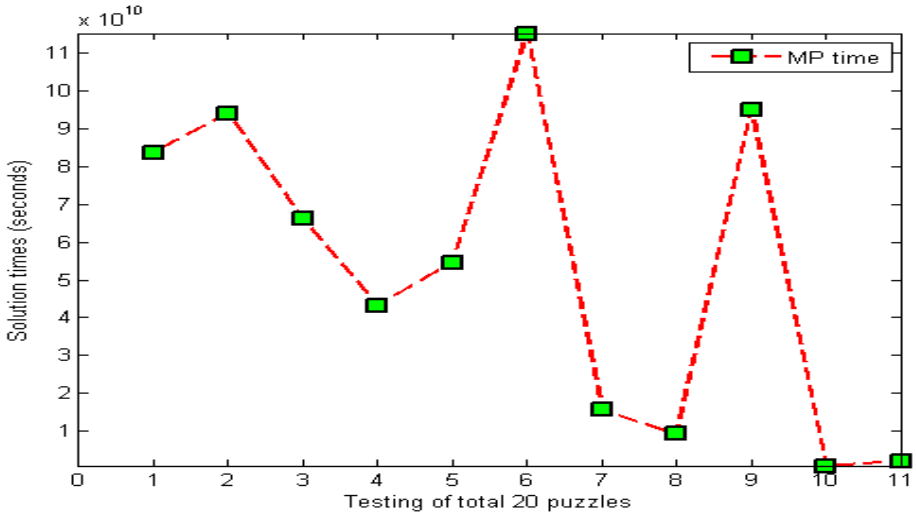


Figure 4.3: Time distributions for message passing algorithms

The following figure shows the time taken to solve any of the puzzles is different from other puzzles. Figure 4.4 clearly shows that the message passing algorithm cannot solve two puzzles with the same time. This is applicable for both the 17 and 27 puzzles.

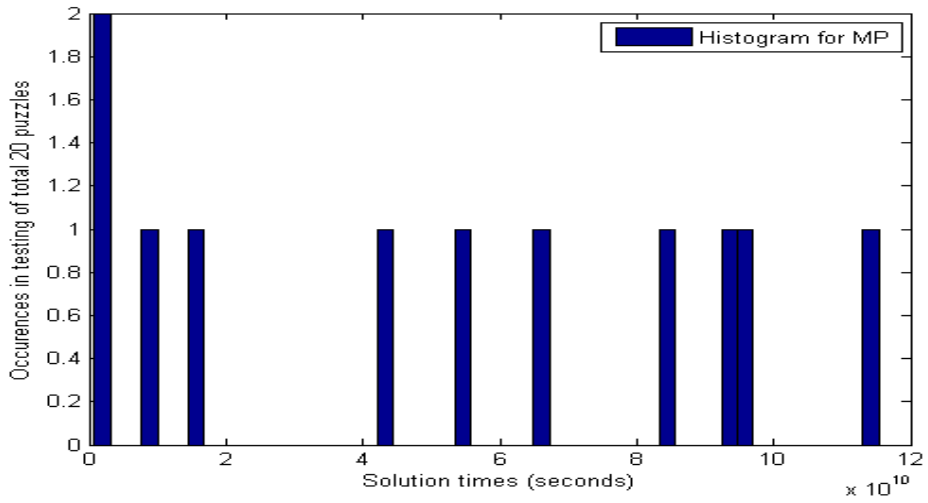


Figure 4.4: The histogram for message passing algorithms

4.3.3. Performance of Breadth First Search:

Figure 4.5. Shows the time distribution for the Breadth first search, when solving 20 Sudoku problems. It is clear that all of the 20 Sudoku problems have been solved. The time taken in the case of 17 clues puzzles is longer than in the case of 27 clues puzzles.

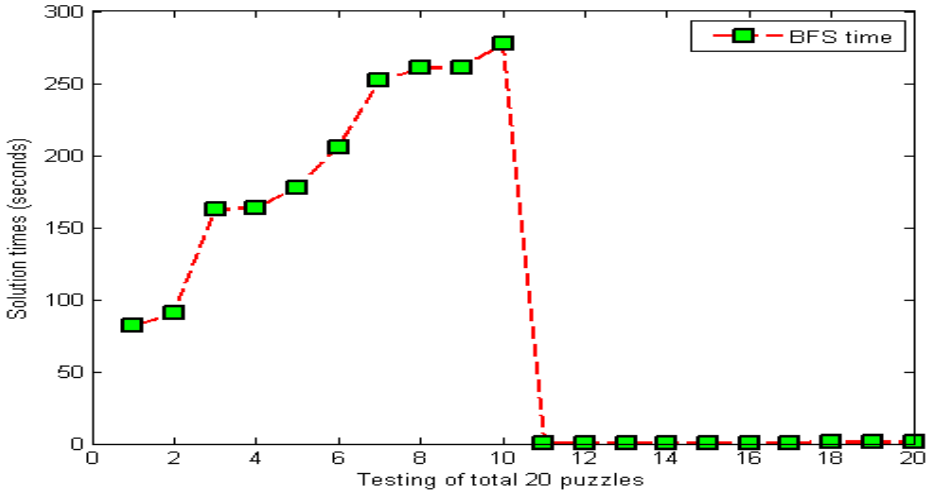


Figure 4.5: Time distributions for breadth first search

Here the best performances for algorithm is with the 27 clues Sudoku puzzles of a minimum solving time equal to 0.3808 sec. The histogram in Figure 4.6 clearly shows this result. It is worth to mention that the solving time is much less than in the case of message passing algorithm.

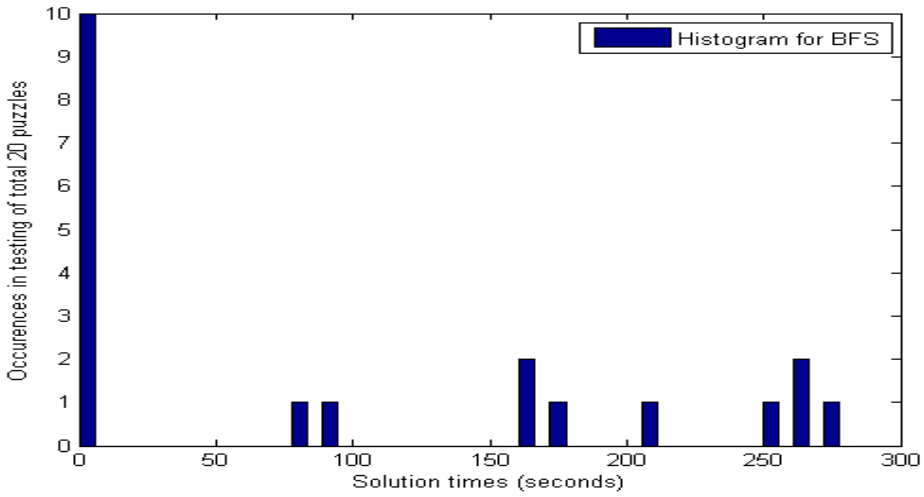


Figure 4.6: The histogram for breadth first search

4.3.4. Performance of Depth First Search:

The performance of Depth first search when applied on the same sample of 20 Sudoku problems is shown in Figure 4.7. Here in the case of 27 clues, the performance is also similar to the breadth but still faster than the Breadth in both cases.

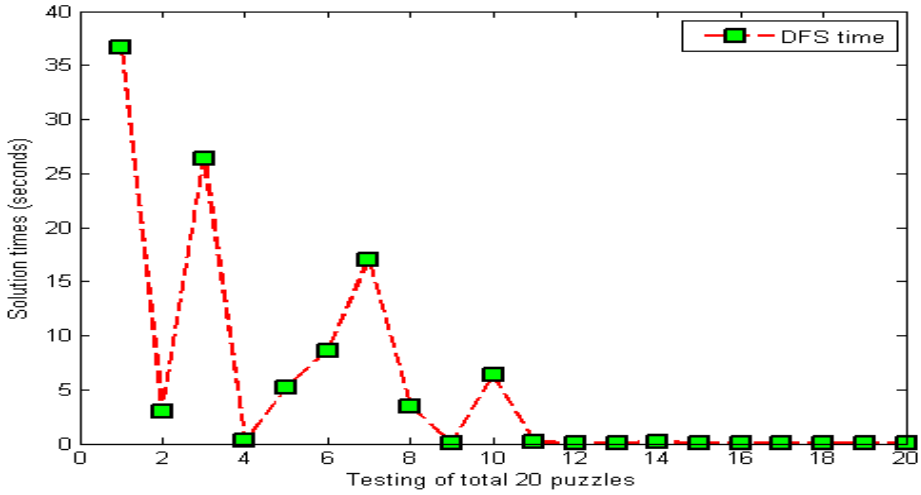


Figure 4.7: Time distributions for depth first search

Figure 4.8 shows the histogram for depth first search algorithm. It is clearly that in the case of 27 clues the solving time is small when compared with the Breadth. The depth first solving time is shorter than in the case of Breadth First Search algorithm.

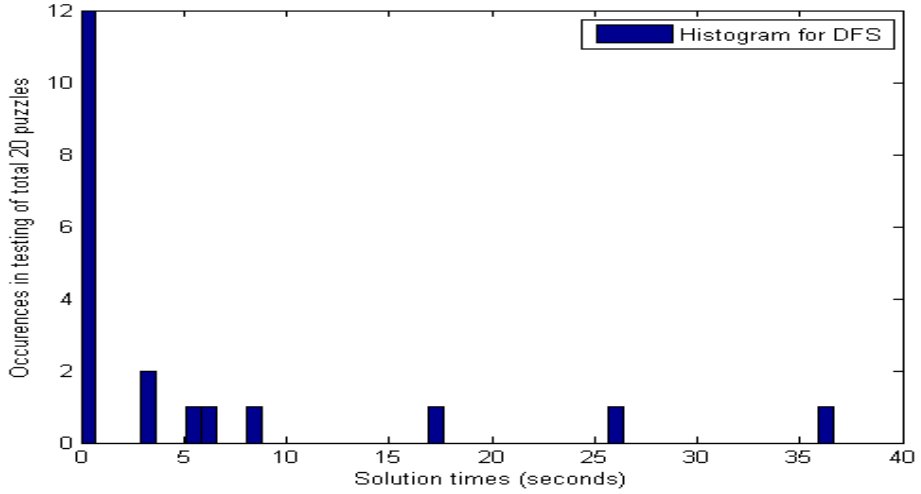


Figure 4.8: The histogram for depth first search

4.4. Discussion:

In this study, four different algorithms have been studied and applied for solving Sudoku puzzle. The main principle of each of these algorithms and their way of analysis is presented. The applied algorithms were coded in Java language tested using two different Sudoku puzzles 17 clue and 27 clues puzzle. For each of these two kinds of puzzles, ten (10) samples have been used for testing and presented to be solved by the algorithms in this study. The results obtained for these two kinds of puzzles i.e. the 17 and 27 clue puzzle when applying the four different algorithms were presented at the beginning of this chapter. The results given covers two main factors i.e. the execution time for each algorithm and the number of solved and unsolved puzzles. When comparing these results, we can see that the message passing algorithm is not homogenous with the other three algorithms i.e. the RBM, BFS, and DFS. The time distribution presented in Figure 4.9 shows the equality between the RBM, BFS, and DFS searching algorithms. On the other hand, MP has an aberrant time distribution value. The results given in this figure covers the 20 Sudoku puzzles.

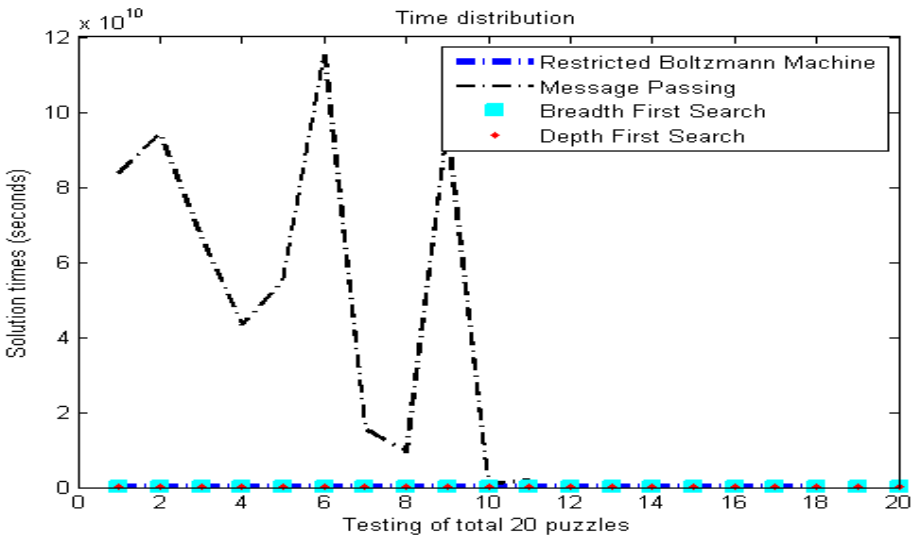


Figure 4.9: Aggregation time distributions of all algorithms

When referring to the previous figure we see that the time for RBM and DFS is closed to each other therefore, the following figure shows these three results in more details. This result shows that the Restricted Boltzmann Machine has the optimal time distribution followed by the Depth First Search.

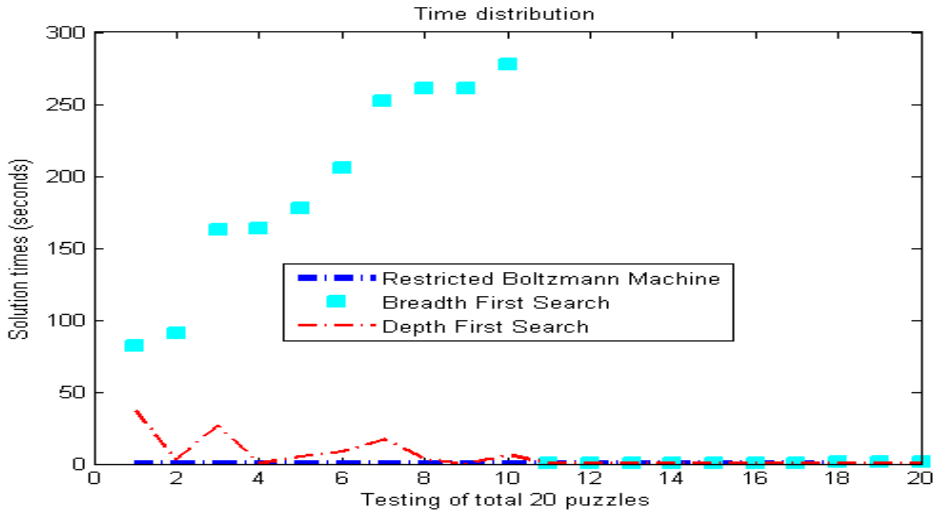


Figure 4.10: Aggregation time distributions of three of the algorithms

4.5. Evaluating problem-solving performance:

It is well known that when applying different tools to deal with or to solve the same problem, there are some factors that need to be taken into account in order to evaluate the performance of these different tools. Time, space, optimality, and completeness are the four main parameters that can lead to real evaluation when applying different tools to the same problem.

1- **Time Complexity**: defined as the maximum time taken to solve the Sudoku puzzle. This parameter differs from one algorithm to another as follows:

- For Boltzmann machine the time complexity is N^3, since it is checking a row then a column and then a block hence takes $N*N*N$ as a total time to check. Where N equal 9 and the time complexity equal $O(729)$ as a total time to check.
- In the case of message passing checking is by the puzzle itself. Therefore, the time is so long and cannot be controlled by the algorithm. There is way of checking will be row, column, block, and grid. This means that the time is $N*N*N*N$ or N^4. Where is N equal 9 and the time complexity equal $O(6561)$ as a total time to check.

- The Breadth first search complexity time is $O(b^d)$ where b is branches length generated by Sudoku and d is the depth of that branch. There are 9^{81} attempts.
- Depth first search $O(b^m)$. In this case b is 9 and m maximum depth of tree structure equals 81, there are 9^{81} attempts.

2- **Space Complexity**: Similar to the time complexity, but it represents the maximum space required for Sudoku to be solved. For the different algorithms applied, these four parameters are as follows:

- In Boltzmann, $O(dN^3)$, where d=81 is depth of network structure and N^3 is the maximum time to find a solution for the reduced 3x3 grid, the space complexity equal to $O(81\times729)$.
- Massage passing: the space complexity is calculated based on the probability where V represents the vector $max\ v \in L(V)$. When having 9 x 9 Sudoku. The total cells in the puzzle equal 81, for each cell there are depth limit of tree structure equal 27 as constraints, 9 cells in row and 9 cells in column and 9 cells in sub-grid. That will be expressed as constraint on the vector ={1,2,3,4,5,6,7,8,9} that is only one element in the probability vector can be equal 1, all others must be 0.
- In the case BFS, the space complexity is $O(b^d)$ when having regular 9 x 9 Sudoku. Here b equals 9 and d equals 81 and m is maximum depth of tree structure, hence there are 9^{81} nodes open at the deepest level.
- The space complexity in DFS, $O(bm)$,where b is branching factor of tree structure and m is maximum depth of answer in tree structure. $O(b \times m)$ there will only be $9 \times 81 = 729$ nodes open at a time.

3- **Optimality**: the optimality refers to an optimal way to find a solution. In some cases a solution can be found but not with an optimal path or way. This affects to a great extend the time taken for the solution and the memory and other parameters used. When referring to the different algorithms we found that the Boltzmann can be considered as an optimum among the four applied algorithms from the execution time point of view and it has different solution for the same Sudoku as well. On the other hand, the Depth First search is an optimal from solutions point of view but with time longer than that of

Restricted Boltzmann algorithm. The following Figure 4.11 presents this result.

Table 4.2: Algorithms performance for solving problem

Algorithm	**Time Complexity**	**Space Complexity**	**Optimality**	**Completeness**
Restricted Boltzmann Machine	$O(N^3) = O(729)$	$O(dN^3) = O(81 \times 729)$	No 2 failed	No 2 failed
Message passing	$O(N^4) = O(6561)$	$O(\|V\|\exp(\max v \in V\|L(V)\|))$	No 9 failed	No 9 failed
Breadth First Search	$O(b^d) = O(9^{81})$	$O(9^{81}) = O(9^{81})$	Yes No failed	Yes No failed
Depth First Search	$O(b^m) = O(9^{81})$	$O(bm) = O(9 \times 81)$	Yes No failed	Yes No failed

0	5	0	2	0	0	0	0	0
0	0	0	0	3	0	4	0	0
0	0	0	0	0	0	0	0	0
0	0	0	6	0	1	0	0	3
8	0	4	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
0	2	0	3	0	5	0	0	0
0	0	0	0	0	0	7	1	0
0	0	0	0	4	0	0	8	0

(a) Sudoku problem

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

(b) Boltzmann solution

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

(c) Message passing solution

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

(d) Breadth first search solution

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

(e) Depth first search solution

Figure 4.11: A sample of 17 clues Sudoku puzzles with four results

Figure 4.12 shows that Restricted Boltzmann Machine solution is different from the other two solutions and for this, the Message Passing algorithm failed to solve.

0	0	0	0	0	0	0	2	1
5	0	0	4	0	0	0	0	0
0	0	0	8	0	0	0	0	0
0	2	1	5	0	0	0	0	0
0	7	0	0	0	0	6	0	0
0	0	0	0	0	0	0	3	0
4	0	0	0	0	0	8	0	0
3	0	0	0	7	0	0	0	0
0	0	6	0	2	0	0	0	0

(a) Sudoku problem

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

(b) Boltzmann solution

7	6	8	3	9	5	4	2	1
5	1	9	8	6	2	3	8	7
2	3	4	8	1	7	9	5	6
9	2	1	5	3	6	7	4	8
8	1	3	2	4	9	6	1	5
6	4	5	7	8	1	2	3	9
4	9	7	1	5	3	8	6	2
3	0	0	0	7	0	0	0	0
0	0	6	0	2	0	0	0	0

(c) Message passing failed

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

(d) Breadth first search solution

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

(e) Depth first search solution

Figure 4.12: A sample of 17 clues Sudoku puzzles with one-failed results

Figure 4.13 illustrates that two algorithms cannot reach to the puzzle solution i.e. the Restricted Boltzmann machine and the Message Passing algorithm.

0	0	0	0	0	0	0	7	1
9	0	0	0	0	0	0	6	0
0	2	0	0	0	0	0	0	0
0	0	4	0	7	0	0	0	0
0	3	0	0	0	0	4	0	0
0	0	0	9	1	0	0	0	0
7	0	0	6	0	0	0	0	8
0	0	0	3	0	0	2	0	0
1	0	0	0	0	0	0	0	0

(a) Sudoku problem

3	4	5	8	6	3	9	7	1
9	8	1	2	5	7	3	6	4
7	2	6	4	8	9	5	3	1
6	9	4	8	7	3	1	2	5
8	3	1	5	2	6	4	9	7
2	7	5	9	1	0	0	0	0
7	0	0	6	0	0	0	0	8
0	0	0	3	0	0	2	0	0
1	0	0	0	0	0	0	0	0

(b) Boltzmann failed

4	5	6	2	3	9	8	7	1
9	1	8	7	4	5	3	6	2
3	2	7	1	6	8	5	4	9
6	9	4	8	7	3	1	2	5
8	3	0	0	0	0	4	0	0
0	0	0	9	1	0	0	0	0
7	0	0	6	0	0	0	0	8
0	0	0	3	0	0	2	0	0
1	0	0	0	0	0	0	0	0

(c) Message passing failed

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

(d) Breadth first search solution

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

(e) Depth first search solution

Figure 4.13: A sample of 17 clues Sudoku puzzles with two failed results

4- **Completeness**: if the algorithm was able to find all solutions for the Sudoku regardless the execution time, then it is said to be complete. This study shows the Depth first search and Breadth first search algorithms is complete since it was able to solve all Sudoku puzzles. In addition, the other algorithms are not.

4.6. Descriptive Statistics:

Descriptive statistics are used to provide simple summarization about the sample and the measures. Together with simple graphics analysis, they form the basis of virtually every quantitative analysis of data. It simply describes what the data shows. With inferential statistics, and trying to reach conclusions that extend beyond the immediate data alone when analyzing Mean value and Standard Deviation value after sorted data (the time taken to solve puzzles) for all algorithms and comparing as follows:-

4.6.1. Restricted Boltzmann Machine time solving:

The algorithm solves a set of Sudoku puzzles with a Mean value of 2.818048×10^{-03} seconds and Variation in solving time was also small with a Standard Deviation value of 3.182617×10^{-03} seconds, which means that most of the numbers are very close to the average as show Figure 4.14.

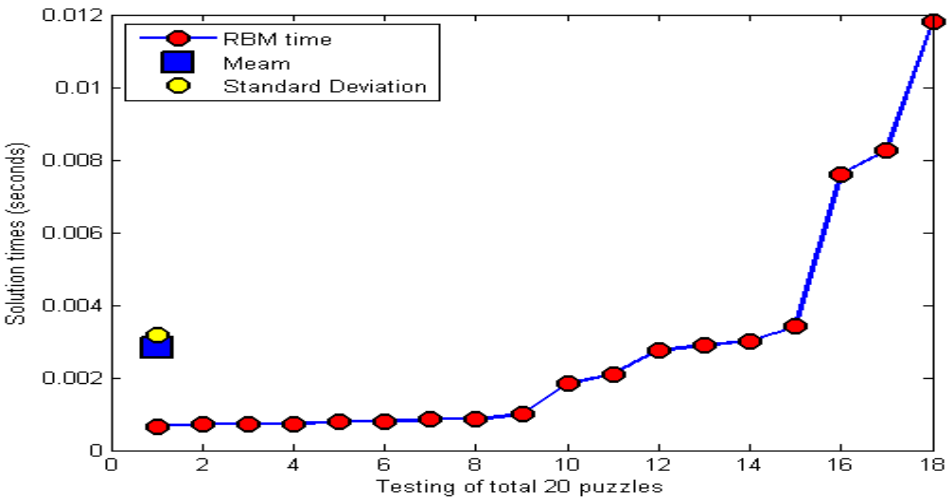


Figure 4.14: The solving time for restricted Boltzmann machine

4.6.2. Message Passing time solving:

Figure 4.15 shows that Message Passing algorithm solves a set of Sudoku puzzles with a Mean value of $5.274069\times10^{+10}$ seconds and a high Standard Deviation value of $4.145572\times10^{+10}$ seconds that the numbers are spread out (from the Mean).

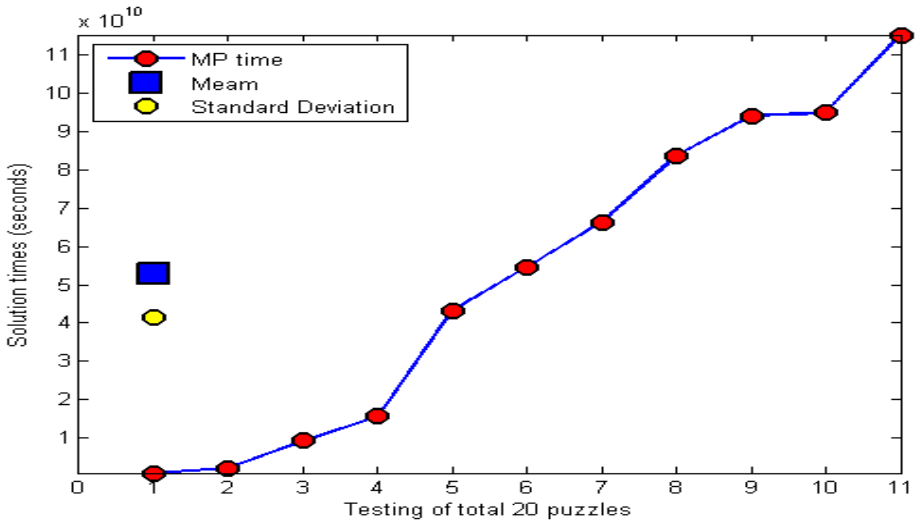


Figure 4.15: The solving time for message passing

4.6.3. Breadth First Search time solving:

In this case, the Breadth First Search algorithm solves a set of Sudoku puzzles with a Mean value of $9.697644\times10^{+1}$ seconds and here a high Standard Deviation value of $1.100881\times10^{+2}$ seconds which means that the numbers are spread out as show in Figure 4.16.

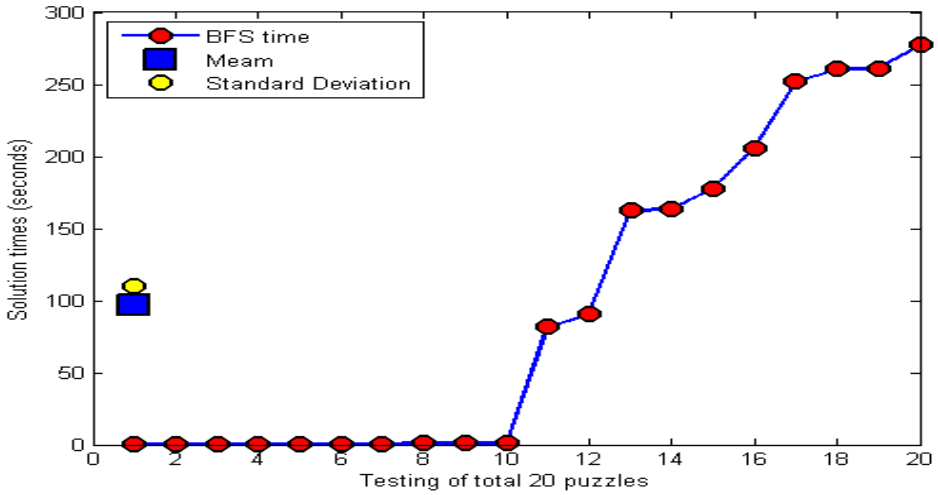


Figure 4.16: The solving time for breadth first search

4.6.4. Depth First Search time solving:

The algorithms solves a set of Sudoku puzzles with a Mean value of 5.370067×10 seconds and a high Standard Deviation value of $1.004221\times10^{+1}$ seconds which means that the numbers are spread out as show in Figure 4.17.

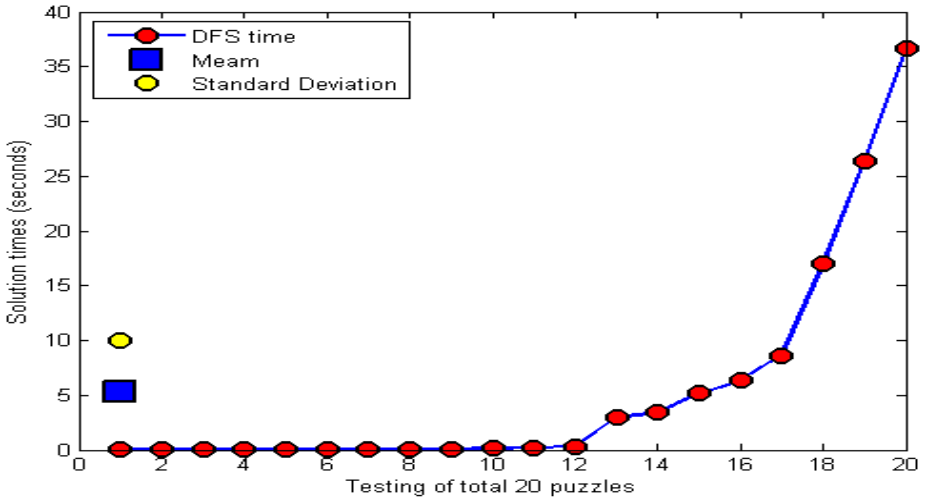


Figure 4.17: The solving time for depth first search

4.7. Statistical Analysis:

Section (4.3) gives a comparison between the algorithms for solving time point of view. Because test data was random numbers, we can use the statistical analysis on testing data to determine the underlying probability distribution of the data by comparing it using a probability density function after taken approximate integral that includes what computational limitations were present and how this affected the results.

In equation 4.1 the probability density function for a normal distribution (Gang, *et al*., 2013) with Mean μ, Standard Deviation σ, and variance σ^2 is

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]} \qquad 4.1$$

Because density function, has the property that it integrates close to one. Its integration property will be used to conclude the results obtained in terms of the solving time as cumulative distribution function as below.

4.7.1. Cumulative distribution for Restricted Boltzmann Machine:

Density function has the property that it integrates equal to 0.9992 and attempt get the best the cumulative distribution function (CDF) as show in Figure 4.18.

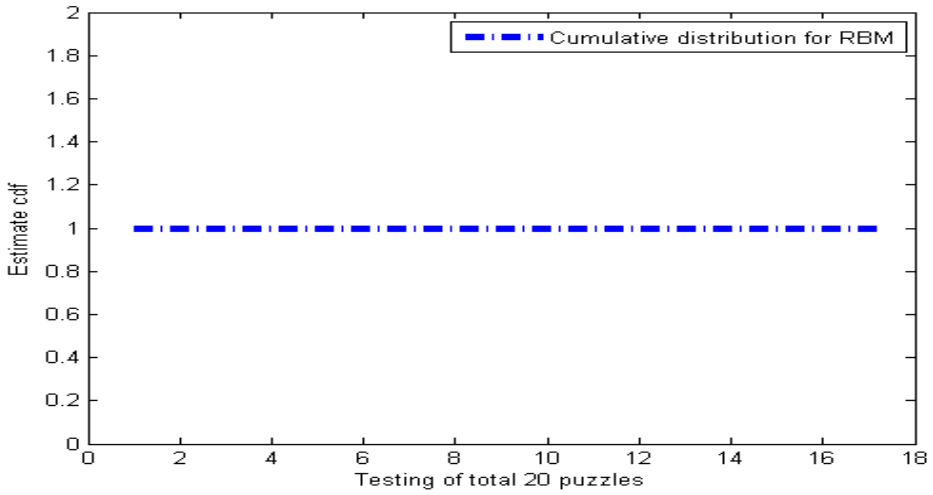


Figure 4.18: Cumulative distribution for RBM

It is clear that the distribution has a constant probability function, since the time taken to reach the solution occurs regularly. If the variable, time taken to solve the puzzle, is a random variable with a uniform distribution, and the integration represents the area below the curve that expresses the sum of all possibilities, so that its integral approaches 1 as given in Figure 4.18. In section 4.6.1, it was discussed that RBM has a small Standard Deviation value of 3.182617×10^{-03} seconds. This means that most of the values of the time taken to solve the puzzle are very close to the average value as shown in Figure 4.14.

4.7.2. Cumulative distribution for Message Passing:

Here the density function has the worst property because it integrates equal to 0.1813 and it is far from one. The cumulative distribution explains how the original data (time taken to solve puzzles) are spare out form each other as show in Figure 4.19.

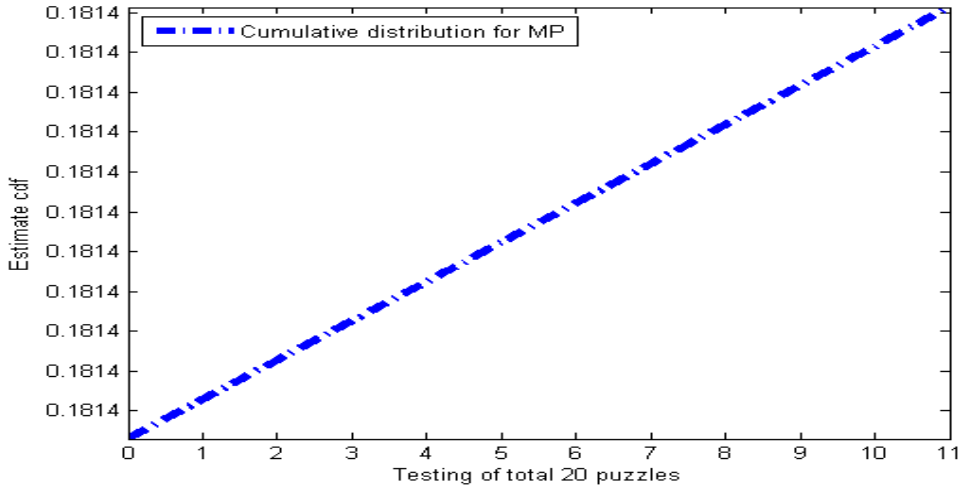


Figure 4.19: Cumulative distribution for MP

In this Figure 4.19, it can be seen that the distribution for the time taken to solve the puzzle is irregular. In other words, the time taken does not change in an orderly manner. Figure 4.19 shows that the distribution is not limited and when integrated is not close to 1 which means that the deficiency of this

algorithm in solving the puzzle. Earlier, in section 4.6.2 we show that the message passing has highest Standard Deviation value i.e. $4.145572\times10^{+10}$ seconds and the values are spread out from the Mean.

4.7.3. Cumulative distribution for Breadth First Search:

Density function has a good property that it integrates equal to 0.3029 it is far from one and attempt get the cumulative distribution similar Message Passing algorithm as show in Figure 4.20.

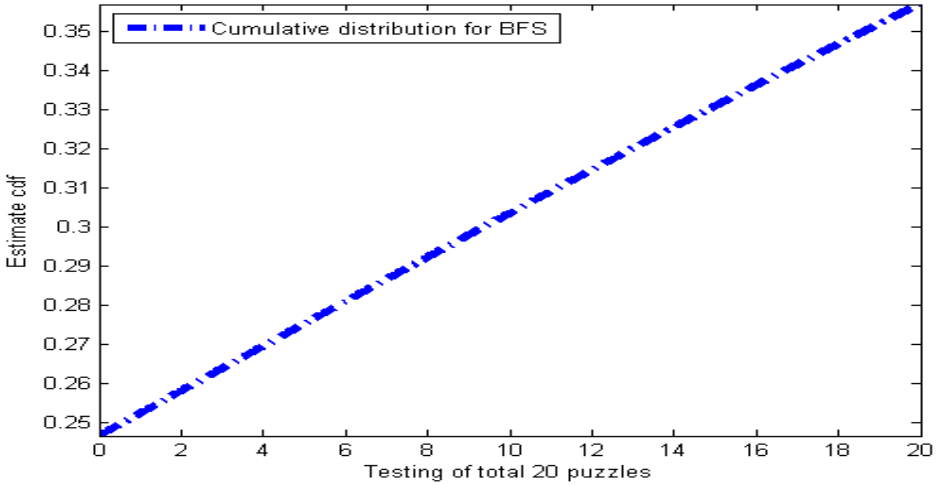


Figure 4.20: Cumulative distribution for BFS

Figure 4.20 shows that the distribution is also irregular for the breadth first search algorithm. The time taken to solve the puzzle is not changing in regular manner. However; when comparing the values of time taken to solve, standard deviation, and the integration with those obtained for message passing we can see that BFS is better. It is equal to 0.3029 BSF whereas; 0.1813 in the case of message passing. In section 4.6.3, it was given that the BFS has a high Standard Deviation value of $1.100881\times10^{+2}$ seconds that ensures the distribution given in Figure 4.16.

4.7.4. Cumulative distribution for Depth First Search:

Figure 4.21 the density function has a Better property and the cumulative distribution explains that it integrates equal to 0.7873.

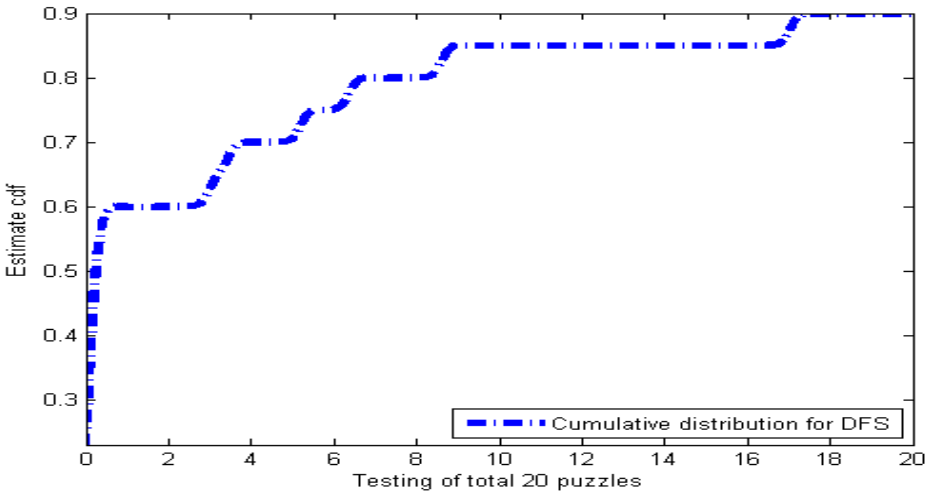


Figure 4.21: Cumulative distribution for DFS

In this case, the probability density function can be described as a histogram, which represents the relative frequencies within the fields of the graphical results. Figure 4.21 shows that for the puzzles from 9 to 17 the distribution is regular, and hence can be said that the algorithm performance is stable. Figure shows that for puzzle from 1 to 8 and from 18 to 20 has irregular distribution. With these distributions, the DFS came the second in rank after the RBM. In section 4.6.4, the standard deviation for the DFS was discussed where it shows that the value for the DFS is $1.004221\times10^{+1}$ seconds that reflects the distribution given in Figure 4.17.

Chapter Five

Conclusions and Future work

5.1. Conclusion:

This chapter presents the conclusions drawn from the research and raises the suggestions for future work. The study is primarily concerned with Searching Algorithms i.e. Breadth First Search and Depth First Search for Problem Solving mechanisms in the field of artificial intelligence. In addition, Restricted Boltzmann Machine as a kind of neural network and Message passing Algorithm have been studied and applied for problem solving. In order to determine their ability for problem solving so that their performance, significance, and hence a comparison between them can be made, 9×9 Sudoku puzzles is used as a case study in this research. Java programming language is used to code these different algorithms so that the previous mentioned parameters found. For each of these four algorithms, two main parameters were examined i.e. the solving time and successfulness.

The results show that the Restricted Boltzmann Machine algorithm is the best among all four algorithms covered in this study. The time taken by RBM to solve the 9×9 Sudoku puzzles is the shortest time equal to 0.0118 sec with a solving rate of 18 out of 20 puzzle. On the other hand, the DFS comes in the second position with a solving time longer than RBM and solving all puzzles. The restricted Boltzmann machine presents the lowest standard deviation values equals to 3.182617×10^{-3} second and with a higher probability density function approaches 1 i.e. 0.9992 Regarding the Depth first search algorithm the standard deviation value is $1.004221\times10^{+1}$ second and the probability density function is 0.7873 which means that the solving time differs based on the problem to be solved itself.

The message passing algorithm shows the worst results from the solving time point of view and the unsuccessfulness to solve the puzzles 11 out of 20. This algorithm came with a standard deviation equals to $4.145572\times10^{+10}$ second and a cumulative probability density function of 0.1813.

5.2. Future Work:

As a future work, different searching algorithms might be applied as:

1- Uniform-Cost Search
2- Depth-Limited Search
3- Iterative Deepening Search
4- A* algorithm
5- Genetic algorithm
6- Minimax algorithm

In this research, a 9×9 Sudoku puzzle is chosen as a study case, as future work, different types of Sudoku puzzles can be chosen. Killer Sudoku puzzle, Even Sudoku puzzle or X Sudoku puzzle, which is known as diagonal puzzle, different data structure might be used with a puzzles that can be adopted in the future.

Moreover a different kinds of puzzles such as N-Queen and Travelling Salesman Problem, Rubik's Cube are kinds of puzzles that may be applied with the previous mentioned data structures and searching algorithms.

References

Adedapo O, A., Ganiyu R, A., Olabiyisi S, O., Omidiora E, O., & Sijuade A, A. (2015). Procedural Cognitive Complexity Measure of Tree Search Algorithms. *International Journal of Advanced Research in Computer Science and Software Engineering, 7*, pp. 31-35.

Akanmu T, A., Olabiyisi S, O., Omidiora E, O., Oyeleye C, A., Mabayoje M, A., & Babatunde A, O. (2010). Comparative Study of Complexities of Breadth- First Search and Depth-First Search Algorithms using Software Complexity Measures. *the World Congress on Engineering 2010*, *I.* London, U.K.

Andre, M., Florent , K., Eric W., T., & Lenka, Z. (2014, Jun 17). Sparse Estimation with the Swept Approximated Message-Passing Algorithm. *Proceedings of the 32nd International Conference on Machine Learning, v1*, pp. 1123-1132.

Andrea, M., Federico, R.-T., & Guilhem , S. (2007, Sep 11). Solving Constraint Satisfaction Problems through Belief Propagation-guided decimation. Artificial Intelligence (cs.AI); Disordered Systems and Neural Networks (cond-mat.dis-nn).

Arunkumar , B., & Komala, R. (2015, July 7). Applications of Bipartite Graph in diverse fields including cloud computing. *International Journal Of Modern Engineering Research (IJMER)*(5).

Asja ., F., & Christian , I. (2014, January). Training restricted Boltzmann machines: An introduction,. *Pattern Recognitation archive journal, 47*(1), 25-39.

Baptiste, W., & Jean , H. (2014). Camera-based Sudoku recognition with Deep Belief Network. *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference* (pp. 83 – 88). Fribourg: IEEE.

Beamer, S., Krste , A., & David, P. (2012). Direction-Optimizing Breadth-First Search. *International Conference on High Performance Computing, Networking, Storage and Analysis (SC).* Salt Lake City, Utah: IEEE.

Behrooz , P. (2009). Motivating Computer Engineering Freshmen Through Mathematical and Logical Puzzles. *TRANSACTIONS ON EDUCATION. VOL. 52, NO. 3.* IEEE.

Caroline , A., & Jossy , S. (2014, Jul). Density Evolution for SUDOKU codes on the Erasure Channel. *Funded in part by the European Research Council under ERC grant agreement 259663 and by the FP7 Network of Excellence NEWCOM## under grant agreement 318306*. University of Cambridge.

Cecilia , N., & Luciana, A. (2013). Modelling Sudoku Puzzles as Block-world Problems. *International Journal of Computer, Control, Quantum and Information Engineering, 7*(8), pp. 487- 493.

Charles, E. (2010). A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers). *the twenty-second annual ACM symposium on Parallelism in algorithms and architectures conference*, (pp. 303-314).

Chiung-Hsueh , Y., Hui-Lung , L., & Ling-Hwei, C. (2009, November 13). An efficient algorithm for solving nonograms. © *Springer Science+Business Media, LLC 2009*.

David, D., Arian , M., & Andrea, M. (2009, September 11). Message-passing algorithms for compressed sensing. *The National Academy of Sciences of the United States of America (PNAS)*.

David, H., & Steven, C. (2013). EA-EMA Optimization Applied to Killer Sudoku Puzzles. *Conference Organized by Missouri University of Science and Technology* (pp. 58 – 64). online at : www.sciencedirect.com.

Duane, M., Michael , G., & Andrew , G. (2011). *High Performance and Scalable GPU Graph Traversal.* Technical Report CS-2011-05, University of Virginia, Department of Computer Science.

Farhad , S., Bahareh , S., & Golriz , F. (2012, September). A New Solution for N-Queens Problem using Blind Approaches: DFS and BFS Algorithms. *International Journal of Computer Applications (0975 – 8887), Volume 53– No.1*.

Ferozuddin, R., & Khidir , M. (2011). Applications of Graph Theory in Computer Science. *Computational Intelligence, Communication Systems and Networks (CICSyN).* IEEE Xplore.

Frank R, K., Brendan J, F., & Hans-An. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory, 47(2), 498–519*.

Gang, Z., Hugh , S., Lixin , W., & Alistair, D. (2013, DECEMBER 2013). A Statistical Assessment of the Performance of FSV. *ACES JOURNAL, 28*(12), 1179-1186.

Geoffrey E, H., Simon, O., & Yee-Whye, T. (2006, July). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), pp. 1527–1554.

Heiko, B. (2008). Passing messages to lonely numbers. *Computing in Science and Engineering, 2*(10), pp. 32 – 40.

Hugo, L., & Yoshua , B. (2008). Classification using Discriminative Restricted Boltzmann Machines. *Appearing in Proceedings of the 25 International Conference on Machine Learning.* Helsinki, Finland.

Jonathan S, Y., William T, F., & YairWeiss. (2005, July). Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory, 51(7)*.

Jossy , S. (2014). The Role Model Estimator Revisited. *University of Cambridge, Funded in part by the European Research Council under ERC grant agreement 259663 and by the FP7 Network of Excellence NEWCOM# under grant agreement 318306*.

Jossy , S., & Joned , S. (2015, Apr 16). An investigation of SUDOKU-inspired non-linear codes with local constraints. *University of Cambridge, U.K., arXiv:1504.03946v2 [cs.IT]*.

Jussi, R. (2010). Heuristic Planning with SAT: Beyond Uninformed Depth-First Search. *Australasian joint conference on artificial intelligence; AI 2010: advances in artificial intelligence.* Berlin: by Springer.

Kristian , K., Babak , A., & Sriraam , N. (2009). Counting Belief Propagation. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI2009).* Uncertainty in Artificial Intelligence.

KyungHyun , C., Tapani, R., & Alexander, I. (2010). Parallel Tempering is Efficient for Learning Restricted Boltzmann Machines. *the International Joint Conference on Neural Networks.* Barcelona, Spain.

Le Roux, N., & Bengio, Y. (2008). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation Journal, 20(6)*.

Lijuan , L., Martin , W., & Wen-mei, H. (2010). An Effective GPU Implementation of Breadth-First Search. *Design Automation Conference (DAC) 47th ACM/IEEE* (pp. 52 - 55). IEEE.

Marylou , G., Eric W., T., & Florent, K. (2015, Jun 15). Training Restricted Boltzmann Machines via the Thouless-Anderson-Palmer Free Energy. *Advances in Neural Information Processing Systems, 28*, 640--648.

Mehmet , D., Kamer , K., Bora, U., & Umit , V. (2013, Mar 6). GPU accelerated maximum cardinality matching algorithms for bipartite graphs. *Distributed Parallel, and Cluster Computing (cs.DC)*.

Min-Quan , J., Chiung-Hsueh, Y., Hui-Lung , L., & Ling-Hwei , C. (2009). Solving Japanese puzzles with logical rules and depth first search algorithm. *the Eighth International Conference on Machine Learning and Cybernetics. 5*, pp. 2962-2967. IEEE.

Montufar, G., & Ay, N. (2011). Refinements of universal approximation results for deep belief networks and restricted Boltzmann machines. *Neural Computation journal*.

Nate , D., Jose, B., Veit , E., & Jonathan , S. (2013, May 8). An Improved Three-Weight Message-Passing Algorithm. *arXiv:1305.1961v1 [cs.AI]*.

Nathan, R. (2013). An Argument for Large-Scale Breadth-First Search for Game Design and Content Generation via a Case Study of Fling! *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment; Ninth Artificial Intelligence.* Organizing agency, location.

Navdeep, J., & Geoffrey, H. (2011). Learning a better representation of speech soundwaves using restricted boltzmann machines. *Speech and Signal Processing (ICASSP) 2011 IEEE International Conference* (pp. 5884–5887). IEEE.

Olusegun, O. A., Babatunde, A. N., Omotehinwa, T. O., Aremu, D. R., & Balogun, B. F. (2014, May – June). An Appropriate Search Algorithm for Finding Grid Resources. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 3, Issue 3*.

Rina , D., & Robert, M. (2007, January 17). AND/OR search spaces for graphical models,. *Artificial Intelligence*(171), pp. 73–106.

Rong , Z., & Eric , A. (2009). Combining Breadth-First and Depth-First Strategies in Searching for Treewidth. *The twenty-First International Joint AAAI Conferece on Artifical Intelligence*, (pp. 641-645).

Ruslan , S., Andriy, M., & Geoffrey E., H. (2007). Restricted Boltzmann machines or collaborative filtering. *In Proceedings of the 24th international conference on Machine learning* (pp. 791–798). New York, NY, USA: ACM: (ICML 2007).

Ruslan, S., & Geoffrey E., H. (2012). An efficient learning procedure for deep Boltzmann machines. *Neural Computation*(24), pp. 1967–2006.

Russell, S., & Norvig, P. (2010). Chapters 3 and 4. In *Artificial Intelligence: A Modern Approach* (Vol. 3rd Edition).

Russell, S., & Norving, P. (2010). Chapters 2. In *Artificial Intelligence – A Modern Approach* (Vol. 3rd edition). New Jersey: Pearson.

Rutuja, U., & Payal , S. (2014). A Review on Parallelization of Node based Game Tree Search Algorithms on GPU. *(IJCSIT) International Journal of Computer Science and Information Technologies, 5*(6), pp. 7385-7388.

Sanjay , J., & Chander , S. (2014). Mathematical and C Programming Approach for Sudoku Game. *Journal of Game Theory 2014, 3*(1), pp. 1-6.

Scott, K., Ethan, B., & Wheeler, R. (2012). Abstraction-Guided Sampling for Motion Planning. *Proceedings of the Fifth Annual Symposium Combinatorial Search.* Association for the Advancement of Artificial Intelligence (www.aaai.org).

Stefano, E., Carla P, G., Ashish, S., & Bart , S. (2013, Feb 2). Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization. *arXiv:1302.6677v1 [cs.LG]*.

Tanya, G. R., Martin, J. W., & Shankar, S. S. (2008). Convergence Analysis of Reweighted Sum-Product Algorithms. *IEEE TRANSACTIONS ON SIGNAL PROCESSING, 56*, pp. 4293-4305.

Taruna , K., Preeti, Y., & Lavina. (2015, September - October). Study Of Brute Force and Heuristic Approach to solve sodoku. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 5*(2), pp. 052-055.

Todd K, M., & Jacob H, G. (2006). Multiple constraint satisfaction by belief propagation: An example using sudoku. *Adaptive and Learning Systems* (pp. 122 – 126). IEEE Mountain Workshop on 2006 .

Todd, K. M., Jacob, H. G., & Joseph, J. K. (2009, Aprial). Sinkhorn Solves Sudoku. *IEEE Transactions on Information Theory, 55*(4), pp. 1714- 1746.

WebSudoku. (2009, April). *Sudoku for Webmasters.* Retrieved Feb 24, 2016, from http://www.free-sudoku.com/webmaster.php

Wolfgang Maass. (2014, May). Noise as a Resource for Computation and Learning in Networks of Spiking Neurons. *Proceedings of the IEEE, 102*(5), pp. 861-880.

Xiuqin , D., Junhao, L., & Guangqing, L. (2013, March). Research on Sudoku Puzzles Based on Metaheuristics Algorithm. *Journal of Modern Mathematics Frontier*(2).

Zhengbing, B., Fabian , C., William G, M., & Geordie, R. (2010). *The Ising model: teaching an old problem new tricks.* D-Wave Systems Technical Report.

The appendix A

Boltzmann Source code

```
package simulated.algorithms.sudokuimplementation;

import simulated.algorithms.TemperatureFunction;

import boltzmann.Sudoku;

import boltzmann.SudokuSimulation;

private static final double STOP_ENERGY = 2.0;


public class DefaultSudokuSimulation extends SudokuSimulation {

public DefaultSudokuSimulation(double randomCoefficient,
     TemperatureFunction energyFunction, Sudoku initialSudoku)

     { super(randomCoefficient, energyFunction, initialSudoku);  }
     protected Sudoku getNeighbour(Sudoku currentSudoku)

     { return currentSudoku.getNeighbour();}

protected boolean optimalChanged(Sudoku current, double currentEnergy,
long iterations)

   {          if (currentEnergy == STOP_ENERGY)

       {       int emptyCount = 0;

               for (boolean[] notEmpty : current.getInitialFillMask())
     {

                 for (boolean b : notEmpty) {

                     if (!b) { emptyCount++; }  }  }

System.out.print(""); emptyCount);  current.print();
current.WriteFiles();

            return true; }

    if (current.isValid())

        current.print();

        return false;   }
```

Message passing Source code

```
package message_passing;

import java.io.BufferedReader;

import java.io.BufferedWriter;

public class SudukuPGM {

public static int PROPAGATION_Size = 0;

public static void main(String args[]) {

SudukuPGM p = new SudukuPGM(); p.GetSolution(); }

public void GetSolution() {

     Date start = new Date();  SudukuPGM sudoku = new SudukuPGM();
     SudukuPGM.PROPAGATION_Size = i;

// initialize the matrix, S, N, M

public void initialMatrix() {  incompleteCount = 81;  int index = 0;

    for (int i = 0; i < 27; i++) { for (int j = 0; j < 9; j++) {
     N_Current[i][j] = -1; N[i][j] = -1;          }     }

    for (int i = 0; i < 9; i++) { for (int j = 0; j < 9; j++) {
     S[index] = matrix[i][j];

            int boxN = getBoxNumber(i, j);  M[index][0] = i;
     M[index][1] = j + 9; M[index][2] = boxN; addToN(index, i);
     addToN(index, j + 9); addToN(index, boxN);

                if (S[index] != 0) { modifyN_Current(index);  }

      for (int v = 0; v < 9; v++) {   // initialize the probability of

                    for (int C = 0; C < 3; C++) {

                    Q[M[index][C]][index][v] = Math.log10(1);

                    R[M[index][C]][index][v] = Math.log10(1); }}
     index++;}}}

public double permutate(int n, String pre, String last, String[]
     position,

    double sum, int m) {if (last.length() == 0) { double product =
     Math.log10(1);
```

```
public void getProbabilityForQ(int indexS) {

    for (int C = 0; C < 3; C++) { for (int v = 0; v < 9; v++)
     {Q[M[indexS][C]][indexS][v] = Math.log10(1); }

    for (int C_other = 0; C_other < 3; C_other++) { if (C_other != C)
     {

     getProbabilityFromEachConstraints_Q(indexS,
     M[indexS][C],M[indexS][C_other]);  }  }

   for (int i = 0; i < 9; i++) {   if (Q[M[indexS][C]][indexS][i] >
     Math.log10(0)) { vCount++; }  }

    if (vCount != 0) { for (int i = 0; i < 9; i++) {

            Q[M[indexS][C]][indexS][i] = Q[M[indexS][C]][indexS][i]-
     Math.log10(vCount);    }    }}

public void getProbabilityFromEachConstraints_Q(int index, int m,int
     realConstraintNum) {

    int num = 0; while (N_Current[realConstraintNum][num] != -1) {

        int indexS = N_Current[realConstraintNum][num];

        Q[m][index][S[indexS] - 1] = Math.log10(0); num++; } }

public void getProbabilityFromEachConstraints_PRODUCT(int index,int
     constraintNum) {

// get the number of values in the constraints

    while (N_Current[constraintNum][num] != -1) {

        int indexS = N_Current[constraintNum][num];// index of S

        P[index][S[indexS] - 1] = Math.log10(0); num++; }

        public void modifyN_Current(int index) {

    incompleteCount -= 1; for (int C = 0; C < 3; C++) {
     addToN_Current(M[index][C], index);  } }

public double sum_log(double a, double b) {

    if (a == Double.NEGATIVE_INFINITY) {  return b;  } else if (b ==
     Double.NEGATIVE_INFINITY) {

        return a;  } else { if (a > b) {   x = a;  y = b;     }
     double decide = Math.pow(10, x - y);

        if ((decide + 1) == Double.POSITIVE_INFINITY) {// overflow  }
     else {  decide += 1;  c = y + Math.log10(decide); return c;   }
```

Breadth First Search Source code

```
package breadth_first_search;

import java.util.Queue;

public class SudokuSolver { SudokuField f = new SudokuField();

     public Guess(int x, int y, int val) { this.x = x; this.y = y;
     this.val = val; }   }

     public SudokuSolver(String file,int count) throws IOException {

     this.fileName = file; load(file,count); sudokuCounter++;      }

        public static void GetSolution() { String file =
     "./sudoku17.txt";

            while (counter < 10) { try {  SudokuSolver s = new
     SudokuSolver(file,sudokuCounter);

                    s.solve(); end = System.nanoTime();
     System.out.println("time: "+ (end-start) +"ns\n\n");    }
     catch(Exception e) { System.out.println(e); } try { public static
     void main(String[] args) throws IOException {
     SudokuSolver.GetSolution(); }  private void solve() {  boolean
     check = false;      f.print(check); solve(0, 0, 1); // x 0, y 0,
     val 1;  check = true;  f.print(check);}

     private boolean solve(int startX, int startY, int startVal) {
     recursions++;

      Queue<Guess> candidates = new Queue <> (); int x, y, val;

     for (x = startX; x < 9; x++) { y = (x == startX) ? startY : 0;
     for (; y < 9; y++) {

     if (f.getVal(x, y) == 0) {      val = (x == startX && y == startY)
     ? startVal : 1;

     for (; val <= 9; val++) { boolean free = f.check(x, y, val);

     if (!free) continue; Guess guess = new Guess(x, y, val);
     candidates.add(guess);    }

     if (candidates.size() == 1) {Guess guess = candidates.remove();
                         f.set(guess.x, guess.y, guess.val);
     candidates.clear(); } else if (candidates.size() == 0) {

     return false; } else {    for (int i = candidates.size() - 1; i >=
     0; i--) {                  Guess guess = candidates.remove();
     SudokuField backup = new SudokuField(f);

      f.set(guess.x, guess.y, guess.val); boolean result =
     solve(guess.x, guess.y, guess.val);}
```

```
private void load(String fileName,int lineNum) throws IOException {

try (BufferedReader br = new BufferedReader(new FileReader(fileName)))
     {

int count=0;    for (int a = 0; a < 9; a++) { for (int b = 0; b <
9;b++) {int num= (int)numbers[count]-'0'; f.set(a,b,num);
f.set_start_sudoku(a, b, num);  count++;  }      checkAll();  return;
} counter++;        }} catch (IOException e) {  e.printStackTrace();
}}

int s_x = x / 3; // square id in x direction, starting at 0

int s_x_off = s_x * 3; // square x offset

int s_y = y / 3; int s_y_off = s_y * 3; // square y offset

for (int i = 0; i < 3; i++) { for (int j = 0; j < 3; j++) {       int
     i_pos = i + s_x_off;

int j_pos = j + s_y_off; if (i_pos != x && j_pos != y &&
f.getVal(i_pos, j_pos) == val) {

System.out.printf("check: x:%d, y:%d, val:%d\n", i_pos, j_pos,
f.getVal(i_pos, j_pos)); return false;    } } return true;  }

private void checkRange(int x) { if (x < 0 || x > 8) { throw new
     IllegalArgumentException(); }

private void checkValue(int val) { if (val < 1 || val > 9) { throw new
     IllegalArgumentException();
```

Depth first search Source code

```
package depth_first;

import java.util.Stack;

public class depth_first_search {

class SudokuState {

public SudokuState( int cellnum , int board[][] )

{  for( int i = 0 ; i < 9 ; i++ )   { this.possiblity[i] = true ;
this.cellnum = cellnum ;

 this.board = board ; }

  public int[] getPossiblity() {
```

```
 int num = numberofchoices(); if( num == 0 )  return null ; int
     pchoices[] = new int[num] ;

for(int i = 0 ; i < 9 ; i++) { if(possiblity[i] == true) {pchoices[j]
     = i+1 ; j++ ;} return pchoices ;}

 public int numberofchoices(){ int j = 0 ; for( int i = 0 ; i < 9 ;
     i++ ) {  if( possiblity[i] == true )

  return j ;  }

 public void setPossiblity(boolean choicestate , int choice) {

this.possiblity[choice-1] = choicestate ; }

public int getcellnum(){return this.cellnum ;  }

public void setBoardCell( int val , int cellnum ) {

int row = (cellnum-1)/9 ; int col = cellnum%9 - 1; if( col == -1 )
     this.board[row][8] = val ;

else this.board[row][col] = val ; }

public int getBoardCell( int cellnum )

public int[][] getBoard() {  return this.board; }

private boolean possiblity[] = new boolean[9] ; private int board[][]
     = null ;

public void calculatePossiblities( SudokuState state , int board[][] )

{ int cellnum = state.getcellnum() ; int row = (cellnum - 1) /
     board.length ;

int col = ( cellnum % board.length ) - 1 ;

if( col == -1 ) col = 8 ; rowPossiblities( state , board , row ) ;
     colPossiblities( state , board , col ) ;

subPossiblities( state , board , row , col ) ; }

public void rowPossiblities( SudokuState state , int board[][] , int
     row )

for( int i = 0 ; i < board.length ; i++ )  { if( board[row][i] != -1 )

state.setPossiblity(false, board[row][i] ); }  }

public void colPossiblities( SudokuState state , int board[][] , int
     col )

{for( int i = 0 ; i < board.length ; i++ ) {if( board[i][col] != -1 )
```

```
 state.setPossiblity(false, board[i][col] ) ; }   }

public void subPossiblities( SudokuState state , int board[][] , int
     row , int col )

{  if( row < 3 && col < 3 ) {for( int i = 0 ; i < 3 ; i++ ) for( int j
     = 0 ; j < 3 ; j++ )

  if( board[i][j] != -1 )   state.setPossiblity(false , board[i][j]) ;
     return ;    }

       public int[][] solveSudoku( int board[][] ) {

        Stack<SudokuState> statestack = new Stack<SudokuState>();

        SudokuState svar = new SudokuState(nextemptycell(board),
     board);

        while( true ) { calculatePossiblities(svar , board) ; if(
     svar.numberofchoices() == 0 )

        { svar = statestack.pop();  svar.setPossiblity(false ,
     svar.getBoardCell(svar.getcellnum()) ) ;

       svar.setBoardCell( -1 , svar.getcellnum()) ; board =
     svar.getBoard() ; }Else {
     svar.setBoardCell(svar.getPossiblity()[0] , svar.getcellnum()) ;
     statestack.push(svar) ;             board = svar.getBoard() ;

      if( nextemptycell(board) == -1 ) break ; svar = new
     SudokuState(nextemptycell(board), board) ;  }  }  return board ;
     }

    public void printboard(int[][] board) {for( int i = 0 ; i <
     board.length ; i++ )

    { for( int j = 0 ; j < board.length ; j++ )  System.out.print( " "
     + board[i][j]);  } }

    public void FinalSolution() { int [][] original_puzzle = new
     int[9][9];

     int[][] problem = new int[9][9]; try (BufferedReader br = new
     BufferedReader(new FileReader(fileName))) {while ((line =
     br.readLine()) != null) { char[] numbers = line.toCharArray();

        depth_first_search ssolve = new depth_first_search() ;

        int sboard[][] = ssolve.solveSudoku( board ) ;   }
     System.out.println(e); }

       public static void main(String[] args) { depth_first_search d =
     new depth_first_search(); d.FinalSolution();   } }
```

Matlab Source code

```
Exacusion_time
bt = [];

Min_time = min(bt)
Max_time = max(bt)

figure (1)
plot(bt,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)

xlabel({'Testing of total 20 puzzles'});% Create xlabel
ylabel('Solution times (seconds)');% Create ylabel
legend('RBM time');
figure (2)
plot(1:length(bt),sort(bt),'-ob','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','r',...
                'MarkerSize',10);

hold on;
Meam_Boltzmann=mean(bt);
St = std(bt);%standard deviation
fprintf('standard deviation for Boltzmann = %d\n Meam for Boltzmann =
%d\n',St,Meam_Boltzmann);

plot(Meam_Boltzmann,'rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','b',...
                'MarkerSize',15)
hold on;

plot(St,'ok','LineWidth',1.5,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','y',...
                'MarkerSize',10)
hold on;
xlabel({'Testing of total 20 puzzles'});% Create xlabel
ylabel('Solution times (seconds)');% Create ylabel
legend('RBM time','Meam','Standard Deviation','Location','NorthWest');

figure (3)
hist(bt,50);% a histogram are 50 intervalls.

xlabel({'Solution times (seconds)'});% Create xlabel
ylabel({'Occurences in testing of total 20 puzzles'});% Create ylabel
legend('Histogram for RBM');

figure (4)
% % returns Exponential probability density function and Mean of
probability distribution

y = exppdf(bt,mean(bt));
plot(bt,y,'--ok','LineWidth',1.5,...
                'MarkerEdgeColor','b',...
                'MarkerFaceColor','m',...
```

```
                'MarkerSize',10)

xlabel({'Testing of total 20 puzzles'});% Create xlabel
ylabel({'pdf of the exponential distribution'});% Create ylabel
legend('Probability density function for RBM');
figure (5)

xexp = 1:1.01:20;
[fbt bt] = ksdensity(bt,xexp,'function','cdf');
% Cumulative Density Function
approximate_integral =trapz(fbt)/length(bt)% returns the approximate
integral of fbt

plot(bt,fbt,'-.b','LineWidth',4);

xlabel({'Testing of total 20 puzzles'});% Create xlabel
ylabel({'Estimate cdf'});% Create ylabel
legend('Cumulative distribution for RBM');
```

The appendix B

The solutions for 17 clues Sudoku problems

Restricted Boltzmann Machine	Message Passing	Breadth First Search	Depth First Search
Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1 0
4 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0	4 0 0 0 0 0 0 0 0
0 2 0 0 0 0 0 0 0	0 2 0 0 0 0 0 0 0	0 2 0 0 0 0 0 0 0	0 2 0 0 0 0 0 0 0
0 0 0 0 5 0 4 0 7	0 0 0 0 5 0 4 0 7	0 0 0 0 5 0 4 0 7	0 0 0 0 5 0 4 0 7
0 0 8 0 0 0 3 0 0	0 0 8 0 0 0 3 0 0	0 0 8 0 0 0 3 0 0	0 0 8 0 0 0 3 0 0
0 0 1 0 9 0 0 0 0	0 0 1 0 9 0 0 0 0	0 0 1 0 9 0 0 0 0	0 0 1 0 9 0 0 0 0
3 0 0 4 0 0 2 0 0	3 0 0 4 0 0 2 0 0	3 0 0 4 0 0 2 0 0	3 0 0 4 0 0 2 0 0
0 5 0 1 0 0 0 0 0	0 5 0 1 0 0 0 0 0	0 5 0 1 0 0 0 0 0	0 5 0 1 0 0 0 0 0
0 0 0 8 0 6 0 0 0	0 0 0 8 0 6 0 0 0	0 0 0 8 0 6 0 0 0	0 0 0 8 0 6 0 0 0
Solution	Solution	Solution	Solution
8 9 5 7 6 3 2 1 4	6 9 3 7 8 4 5 1 2	6 9 3 7 8 4 5 1 2	6 9 3 7 8 4 5 1 2
4 3 6 9 1 2 8 7 5	4 8 7 5 1 2 9 3 6	4 8 7 5 1 2 9 3 6	4 8 7 5 1 2 9 3 6
1 2 7 5 8 4 6 3 9	1 2 5 9 6 3 8 7 4	1 2 5 9 6 3 8 7 4	1 2 5 9 6 3 8 7 4
9 6 3 2 5 8 4 1 7	9 3 2 6 5 1 4 8 7	9 3 2 6 5 1 4 8 7	9 3 2 6 5 1 4 8 7
5 7 8 6 4 1 3 9 2	5 6 8 2 4 7 3 9 1	5 6 8 2 4 7 3 9 1	5 6 8 2 4 7 3 9 1
2 4 1 3 9 7 5 8 6	7 4 1 3 9 8 6 2 5	7 4 1 3 9 8 6 2 5	7 4 1 3 9 8 6 2 5
3 8 9 4 7 5 2 6 1	3 1 9 4 7 5 2 6 8	3 1 9 4 7 5 2 6 8	3 1 9 4 7 5 2 6 8
6 5 2 1 3 9 7 4 8	8 5 6 1 2 9 7 4 3	8 5 6 1 2 9 7 4 3	8 5 6 1 2 9 7 4 3
7 1 4 8 2 6 9 5 3	2 7 4 8 3 6 1 5 9	2 7 4 8 3 6 1 5 9	2 7 4 8 3 6 1 5 9

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 2 1	0 0 0 0 0 0 0 2 1	0 0 0 0 0 0 0 2 1	0 0 0 0 0 0 0 2 1
5 0 0 4 0 0 0 0 0	5 0 0 4 0 0 0 0 0	5 0 0 4 0 0 0 0 0	5 0 0 4 0 0 0 0 0
0 0 0 8 0 0 0 0 0	0 0 0 8 0 0 0 0 0	0 0 0 8 0 0 0 0 0	0 0 0 8 0 0 0 0 0
0 2 1 5 0 0 0 0 0	0 2 1 5 0 0 0 0 0	0 2 1 5 0 0 0 0 0	0 2 1 5 0 0 0 0 0
0 7 0 0 0 0 6 0 0	0 7 0 0 0 0 6 0 0	0 7 0 0 0 0 6 0 0	0 7 0 0 0 0 6 0 0
0 0 0 0 0 0 0 3 0	0 0 0 0 0 0 0 3 0	0 0 0 0 0 0 0 3 0	0 0 0 0 0 0 0 3 0
4 0 0 0 0 0 8 0 0	4 0 0 0 0 0 8 0 0	4 0 0 0 0 0 8 0 0	4 0 0 0 0 0 8 0 0
3 0 0 0 7 0 0 0 0	3 0 0 0 7 0 0 0 0	3 0 0 0 7 0 0 0 0	3 0 0 0 7 0 0 0 0
0 0 6 0 2 0 0 0 0	0 0 6 0 2 0 0 0 0	0 0 6 0 2 0 0 0 0	0 0 6 0 2 0 0 0 0
Solution	Failed	Solution	Solution
9 6 8 7 3 5 4 2 1	7 6 8 3 9 5 4 2 1	7 6 8 3 9 5 4 2 1	7 6 8 3 9 5 4 2 1
5 3 2 4 1 6 7 8 9	5 1 9 8 6 2 3 8 7	5 1 9 4 6 2 3 8 7	5 1 9 4 6 2 3 8 7
1 4 7 8 9 2 3 5 6	2 3 4 8 1 7 9 5 6	2 3 4 8 1 7 9 5 6	2 3 4 8 1 7 9 5 6
6 2 1 5 4 3 9 7 8	9 2 1 5 3 6 7 4 8	9 2 1 5 3 6 7 4 8	9 2 1 5 3 6 7 4 8
4 7 3 2 8 9 6 1 5	8 1 3 2 4 9 6 1 5	8 7 3 2 4 9 6 1 5	8 7 3 2 4 9 6 1 5
8 5 9 1 6 7 2 3 4	6 4 5 7 8 1 2 3 9	6 4 5 7 8 1 2 3 9	6 4 5 7 8 1 2 3 9
4 9 2 3 5 1 8 6 7	4 9 7 1 5 3 8 6 2	4 9 7 1 5 3 8 6 2	4 9 7 1 5 3 8 6 2
3 8 5 6 7 4 1 9 2	3 0 0 0 7 0 0 0 0	3 5 2 6 7 8 1 9 4	3 5 2 6 7 8 1 9 4
7 1 6 9 2 8 5 4 3	0 0 6 0 2 0 0 0 0	1 8 6 9 2 4 5 7 3	1 8 6 9 2 4 5 7 3

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 5 0 2 0 0 0 0 0	0 5 0 2 0 0 0 0 0	0 5 0 2 0 0 0 0 0	0 5 0 2 0 0 0 0 0
0 0 0 0 3 0 4 0 0	0 0 0 0 3 0 4 0 0	0 0 0 0 3 0 4 0 0	0 0 0 0 3 0 4 0 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 6 0 1 0 0 3	0 0 0 6 0 1 0 0 3	0 0 0 6 0 1 0 0 3	0 0 0 6 0 1 0 0 3
8 0 4 0 0 0 0 0 0	8 0 4 0 0 0 0 0 0	8 0 4 0 0 0 0 0 0	8 0 4 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0	7 0 0 0 0 0 0 0 0	7 0 0 0 0 0 0 0 0	7 0 0 0 0 0 0 0 0
0 2 0 3 0 5 0 0 0	0 2 0 3 0 5 0 0 0	0 2 0 3 0 5 0 0 0	0 2 0 3 0 5 0 0 0
0 0 0 0 0 0 7 1 0	0 0 0 0 0 0 7 1 0	0 0 0 0 0 0 7 1 0	0 0 0 0 0 0 7 1 0
0 0 0 0 4 0 0 8 0	0 0 0 0 4 0 0 8 0	0 0 0 0 4 0 0 8 0	0 0 0 0 4 0 0 8 0
Solution	Solution	Solution	Solution
4 5 6 2 1 9 3 7 8	4 5 9 2 1 8 6 3 7	4 5 9 2 1 8 6 3 7	4 5 9 2 1 8 6 3 7
1 8 2 5 3 7 4 6 9	2 8 1 7 3 6 4 5 9	2 8 1 7 3 6 4 5 9	2 8 1 7 3 6 4 5 9
9 3 7 8 6 4 5 2 1	6 7 3 4 5 9 1 2 8	6 7 3 4 5 9 1 2 8	6 7 3 4 5 9 1 2 8
2 9 5 6 7 1 8 4 3	5 9 2 6 7 1 8 4 3	5 9 2 6 7 1 8 4 3	5 9 2 6 7 1 8 4 3
8 6 4 9 2 3 1 5 7	8 1 4 5 9 3 2 7 6	8 1 4 5 9 3 2 7 6	8 1 4 5 9 3 2 7 6
7 1 3 4 5 8 6 9 2	7 3 6 8 2 4 5 9 1	7 3 6 8 2 4 5 9 1	7 3 6 8 2 4 5 9 1
6 2 1 3 8 5 9 3 4	1 2 7 3 8 5 9 6 4	1 2 7 3 8 5 9 6 4	1 2 7 3 8 5 9 6 4
5 4 8 7 9 2 7 1 6	3 4 8 9 6 2 7 1 5	3 4 8 9 6 2 7 1 5	3 4 8 9 6 2 7 1 5
3 7 9 1 4 6 2 8 5	9 6 5 1 4 7 3 8 2	9 6 5 1 4 7 3 8 2	9 6 5 1 4 7 3 8 2

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 6 1	0 0 0 0 0 0 0 6 1	0 0 0 0 0 0 0 6 1	0 0 0 0 0 0 0 6 1
2 0 0 7 0 0 0 0 0	2 0 0 7 0 0 0 0 0	2 0 0 7 0 0 0 0 0	2 0 0 7 0 0 0 0 0
0 0 0 8 0 0 0 0 0	0 0 0 8 0 0 0 0 0	0 0 0 8 0 0 0 0 0	0 0 0 8 0 0 0 0 0
0 1 3 0 6 0 0 0 0	0 1 3 0 6 0 0 0 0	0 1 3 0 6 0 0 0 0	0 1 3 0 6 0 0 0 0
0 5 0 4 0 0 2 0 0	0 5 0 4 0 0 2 0 0	0 5 0 4 0 0 2 0 0	0 5 0 4 0 0 2 0 0
0 0 0 0 0 0 7 0 0	0 0 0 0 0 0 7 0 0	0 0 0 0 0 0 7 0 0	0 0 0 0 0 0 7 0 0
0 0 0 0 1 0 0 5 0	0 0 0 0 1 0 0 5 0	0 0 0 0 1 0 0 5 0	0 0 0 0 1 0 0 5 0
7 0 0 0 0 0 4 0 0	7 0 0 0 0 0 4 0 0	7 0 0 0 0 0 4 0 0	7 0 0 0 0 0 4 0 0
8 0 0 0 0 0 0 0 0	8 0 0 0 0 0 0 0 0	8 0 0 0 0 0 0 0 0	8 0 0 0 0 0 0 0 0
Solution	Solution	Solution	Solution
5 7 8 3 2 4 9 6 1	5 7 4 3 2 9 8 6 1	5 7 4 3 2 9 8 6 1	5 7 4 3 2 9 8 6 1
2 3 6 7 9 1 8 4 5	2 8 6 7 5 1 3 4 9	2 8 6 7 5 1 3 4 9	2 8 6 7 5 1 3 4 9
1 4 9 8 5 6 3 2 7	1 3 9 8 4 6 5 7 2	1 3 9 8 4 6 5 7 2	1 3 9 8 4 6 5 7 2
9 1 3 7 6 2 5 8 4	4 1 3 2 6 7 9 8 5	4 1 3 2 6 7 9 8 5	4 1 3 2 6 7 9 8 5
6 5 7 4 8 3 2 1 9	9 5 7 4 8 3 2 1 6	9 5 7 4 8 3 2 1 6	9 5 7 4 8 3 2 1 6
4 8 2 1 9 5 7 3 6	6 2 8 1 9 5 7 3 4	6 2 8 1 9 5 7 3 4	6 2 8 1 9 5 7 3 4
3 9 4 2 1 7 6 5 8	3 4 2 9 1 8 6 5 7	3 4 2 9 1 8 6 5 7	3 4 2 9 1 8 6 5 7
7 6 1 5 3 8 4 9 2	7 6 1 5 3 2 4 9 8	7 6 1 5 3 2 4 9 8	7 6 1 5 3 2 4 9 8
8 2 5 6 4 9 1 7 3	8 9 5 6 7 4 1 2 3	8 9 5 6 7 4 1 2 3	8 9 5 6 7 4 1 2 3

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
7 0 5 0 0 0 0 0 0	7 0 5 0 0 0 0 0 0	7 0 5 0 0 0 0 0 0	7 0 5 0 0 0 0 0 0
0 0 0 0 1 0 4 0 0	0 0 0 0 1 0 4 0 0	0 0 0 0 1 0 4 0 0	0 0 0 0 1 0 4 0 0
2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0
0 4 0 0 0 0 2 0 5	0 4 0 0 0 0 2 0 5	0 4 0 0 0 0 2 0 5	0 4 0 0 0 0 2 0 5
0 0 0 3 7 0 0 0 0	0 0 0 3 7 0 0 0 0	0 0 0 3 7 0 0 0 0	0 0 0 3 7 0 0 0 0
0 0 0 0 0 0 0 9 0	0 0 0 0 0 0 0 9 0	0 0 0 0 0 0 0 9 0	0 0 0 0 0 0 0 9 0
6 8 0 0 0 0 1 0 0	6 8 0 0 0 0 1 0 0	6 8 0 0 0 0 1 0 0	6 8 0 0 0 0 1 0 0
0 0 0 5 0 2 0 0 0	0 0 0 5 0 2 0 0 0	0 0 0 5 0 2 0 0 0	0 0 0 5 0 2 0 0 0
0 0 0 9 0 0 0 0 0	0 0 0 9 0 0 0 0 0	0 0 0 9 0 0 0 0 0	0 0 0 9 0 0 0 0 0
Solution	Failed	Solution	Solution
7 1 5 2 4 9 3 8 6	7 1 5 4 2 9 3 8 6	7 1 5 4 2 9 3 8 6	7 1 5 4 2 9 3 8 6
9 8 6 7 1 3 4 5 2	3 9 8 6 1 5 4 2 7	3 9 8 6 1 5 4 2 7	3 9 8 6 1 5 4 2 7
2 3 4 6 5 8 9 1 7	2 6 8 4 3 7 5 1 9	2 6 4 8 3 7 5 1 9	2 6 4 8 3 7 5 1 9
8 4 7 1 9 6 2 3 5	8 4 7 1 6 9 2 3 5	8 4 7 1 9 6 2 3 5	8 4 7 1 9 6 2 3 5
1 2 9 3 7 5 8 6 4	0 0 0 3 7 0 0 0 0	9 5 2 3 7 8 6 4 1	9 5 2 3 7 8 6 4 1
5 6 3 8 2 4 7 9 1	0 0 0 0 0 0 0 9 0	1 3 6 2 5 4 7 9 8	1 3 6 2 5 4 7 9 8
6 8 5 4 3 7 1 2 9	6 8 0 0 0 0 1 0 0	6 8 9 7 4 3 1 5 2	6 8 9 7 4 3 1 5 2
4 9 1 5 8 2 6 7 3	0 0 0 5 0 2 0 0 0	4 7 1 5 8 2 9 6 3	4 7 1 5 8 2 9 6 3
3 7 2 9 6 1 5 4 8	0 0 0 9 0 0 0 0 0	5 2 3 9 6 1 8 7 4	5 2 3 9 6 1 8 7 4

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 7 1	0 0 0 0 0 0 0 7 1	0 0 0 0 0 0 0 7 1	0 0 0 0 0 0 0 7 1
9 0 0 0 0 0 0 6 0	9 0 0 0 0 0 0 6 0	9 0 0 0 0 0 0 6 0	9 0 0 0 0 0 0 6 0
0 2 0 0 0 0 0 0 0	0 2 0 0 0 0 0 0 0	0 2 0 0 0 0 0 0 0	0 2 0 0 0 0 0 0 0
0 0 4 0 7 0 0 0 0	0 0 4 0 7 0 0 0 0	0 0 4 0 7 0 0 0 0	0 0 4 0 7 0 0 0 0
0 3 0 0 0 0 4 0 0	0 3 0 0 0 0 4 0 0	0 3 0 0 0 0 4 0 0	0 3 0 0 0 0 4 0 0
0 0 0 9 1 0 0 0 0	0 0 0 9 1 0 0 0 0	0 0 0 9 1 0 0 0 0	0 0 0 9 1 0 0 0 0
7 0 0 6 0 0 0 0 8	7 0 0 6 0 0 0 0 8	7 0 0 6 0 0 0 0 8	7 0 0 6 0 0 0 0 8
0 0 0 3 0 0 2 0 0	0 0 0 3 0 0 2 0 0	0 0 0 3 0 0 2 0 0	0 0 0 3 0 0 2 0 0
1 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0
Failed	Failed	Solution	Solution
3 4 5 8 6 3 9 7 1	4 5 6 2 3 9 8 7 1	4 5 6 2 3 9 8 7 1	4 5 6 2 3 9 8 7 1
9 8 1 2 5 7 3 6 4	9 1 8 7 4 5 3 6 2	9 1 8 7 4 5 3 6 2	9 1 8 7 4 5 3 6 2
7 2 6 4 8 9 5 3 1	3 2 7 1 6 8 5 4 9	3 2 7 1 6 8 5 4 9	3 2 7 1 6 8 5 4 9
6 9 4 8 7 3 1 2 5	6 9 4 8 7 3 1 2 5	6 9 4 8 7 3 1 2 5	6 9 4 8 7 3 1 2 5
8 3 1 5 2 6 4 9 7	8 3 0 0 0 0 4 0 0	8 3 1 5 2 6 4 9 7	8 3 1 5 2 6 4 9 7
2 7 5 9 1 0 0 0 0	0 0 0 9 1 0 0 0 0	2 7 5 9 1 4 6 8 3	2 7 5 9 1 4 6 8 3
7 0 0 6 0 0 0 0 8	7 0 0 6 0 0 0 0 8	7 4 2 6 5 1 9 3 8	7 4 2 6 5 1 9 3 8
0 0 0 3 0 0 2 0 0	0 0 0 3 0 0 2 0 0	5 6 9 3 8 7 2 1 4	5 6 9 3 8 7 2 1 4
1 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0	1 8 3 4 9 2 7 5 6	1 8 3 4 9 2 7 5 6

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 1 2	0 0 0 0 0 0 0 1 2	0 0 0 0 0 0 0 1 2	0 0 0 0 0 0 0 1 2
0 4 0 0 5 0 0 0 0	0 4 0 0 5 0 0 0 0	0 4 0 0 5 0 0 0 0	0 4 0 0 5 0 0 0 0
0 0 0 0 0 9 0 0 0	0 0 0 0 0 9 0 0 0	0 0 0 0 0 9 0 0 0	0 0 0 0 0 9 0 0 0
0 7 0 6 0 0 4 0 0	0 7 0 6 0 0 4 0 0	0 7 0 6 0 0 4 0 0	0 7 0 6 0 0 4 0 0
0 0 0 1 0 0 0 0 0	0 0 0 1 0 0 0 0 0	0 0 0 1 0 0 0 0 0	0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 5 0	0 0 0 0 0 0 0 5 0	0 0 0 0 0 0 0 5 0	0 0 0 0 0 0 0 5 0
0 0 0 0 8 7 5 0 0	0 0 0 0 8 7 5 0 0	0 0 0 0 8 7 5 0 0	0 0 0 0 8 7 5 0 0
6 0 1 0 0 0 3 0 0	6 0 1 0 0 0 3 0 0	6 0 1 0 0 0 3 0 0	6 0 1 0 0 0 3 0 0
2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0
Solution	Solution	Solution	Solution
5 6 8 7 3 4 9 1 2	5 9 8 4 6 3 7 1 2	5 9 8 4 6 3 7 1 2	5 9 8 4 6 3 7 1 2
1 4 9 2 5 6 8 7 3	7 4 2 8 5 1 6 3 9	7 4 2 8 5 1 6 3 9	7 4 2 8 5 1 6 3 9
7 2 3 8 1 9 6 4 5	3 1 6 7 2 9 8 4 5	3 1 6 7 2 9 8 4 5	3 1 6 7 2 9 8 4 5
8 7 2 6 9 5 4 3 1	1 7 5 6 3 2 4 9 8	1 7 5 6 3 2 4 9 8	1 7 5 6 3 2 4 9 8
4 3 5 1 7 8 2 6 9	8 6 9 1 4 5 2 7 3	8 6 9 1 4 5 2 7 3	8 6 9 1 4 5 2 7 3
9 1 6 4 2 3 7 5 8	4 2 3 9 7 8 1 5 6	4 2 3 9 7 8 1 5 6	4 2 3 9 7 8 1 5 6
3 9 4 3 8 7 5 2 6	9 3 4 2 8 7 5 6 1	9 3 4 2 8 7 5 6 1	9 3 4 2 8 7 5 6 1
6 5 1 9 4 2 3 8 7	6 8 1 5 9 4 3 2 7	6 8 1 5 9 4 3 2 7	6 8 1 5 9 4 3 2 7
2 8 7 5 6 1 1 9 4	2 5 7 3 1 6 9 8 4	2 5 7 3 1 6 9 8 4	2 5 7 3 1 6 9 8 4

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 1 2	0 0 0 0 0 0 0 1 2	0 0 0 0 0 0 0 1 2	0 0 0 0 0 0 0 1 2
0 5 0 4 0 0 0 0 0	0 5 0 4 0 0 0 0 0	0 5 0 4 0 0 0 0 0	0 5 0 4 0 0 0 0 0
0 0 0 0 0 0 0 3 0	0 0 0 0 0 0 0 3 0	0 0 0 0 0 0 0 3 0	0 0 0 0 0 0 0 3 0
7 0 0 6 0 0 4 0 0	7 0 0 6 0 0 4 0 0	7 0 0 6 0 0 4 0 0	7 0 0 6 0 0 4 0 0
0 0 1 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0
0 0 0 0 8 0 0 0 0	0 0 0 0 8 0 0 0 0	0 0 0 0 8 0 0 0 0	0 0 0 0 8 0 0 0 0
9 2 0 0 0 0 8 0 0	9 2 0 0 0 0 8 0 0	9 2 0 0 0 0 8 0 0	9 2 0 0 0 0 8 0 0
0 0 0 5 1 0 7 0 0	0 0 0 5 1 0 7 0 0	0 0 0 5 1 0 7 0 0	0 0 0 5 1 0 7 0 0
0 0 0 0 0 3 0 0 0	0 0 0 0 0 3 0 0 0	0 0 0 0 0 3 0 0 0	0 0 0 0 0 3 0 0 0
Solution	Solution	Solution	Solution
8 4 7 3 5 6 9 1 2	3 6 4 9 7 8 5 1 2	3 6 4 9 7 8 5 1 2	3 6 4 9 7 8 5 1 2
3 5 9 4 2 1 6 7 8	1 5 2 4 3 6 9 7 8	1 5 2 4 3 6 9 7 8	1 5 2 4 3 6 9 7 8
2 1 6 9 7 8 5 3 4	8 7 9 1 2 5 6 3 4	8 7 9 1 2 5 6 3 4	8 7 9 1 2 5 6 3 4
7 8 2 6 3 5 4 9 1	7 3 8 6 5 1 4 2 9	7 3 8 6 5 1 4 2 9	7 3 8 6 5 1 4 2 9
6 3 1 7 9 4 2 8 5	6 9 1 2 4 7 3 8 5	6 9 1 2 4 7 3 8 5	6 9 1 2 4 7 3 8 5
5 9 4 1 8 2 3 6 7	2 4 5 3 8 9 1 6 7	2 4 5 3 8 9 1 6 7	2 4 5 3 8 9 1 6 7
9 2 3 8 4 7 8 5 6	9 2 3 7 6 4 8 5 1	9 2 3 7 6 4 8 5 1	9 2 3 7 6 4 8 5 1
4 6 8 5 1 9 7 2 3	4 8 6 5 1 2 7 9 3	4 8 6 5 1 2 7 9 3	4 8 6 5 1 2 7 9 3
1 7 5 2 6 3 1 4 9	5 1 7 8 9 3 2 4 6	5 1 7 8 9 3 2 4 6	5 1 7 8 9 3 2 4 6

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
5 0 0 4 0 0 0 6 0	5 0 0 4 0 0 0 6 0	5 0 0 4 0 0 0 6 0	5 0 0 4 0 0 0 6 0
0 0 9 0 0 0 0 0 0	0 0 9 0 0 0 0 0 0	0 0 9 0 0 0 0 0 0	0 0 9 0 0 0 0 0 0
6 4 0 0 2 0 0 0 0	6 4 0 0 2 0 0 0 0	6 4 0 0 2 0 0 0 0	6 4 0 0 2 0 0 0 0
0 0 0 0 0 1 0 0 0	0 0 0 0 0 1 0 0 0	0 0 0 0 0 1 0 0 0	0 0 0 0 0 1 0 0 0
2 0 8 0 0 0 5 0 0	2 0 8 0 0 0 5 0 0	2 0 8 0 0 0 5 0 0	2 0 8 0 0 0 5 0 0
0 0 0 5 0 0 0 0 0	0 0 0 5 0 0 0 0 0	0 0 0 5 0 0 0 0 0	0 0 0 5 0 0 0 0 0
0 0 0 0 9 0 0 0 0	0 0 0 0 9 0 0 0 0	0 0 0 0 9 0 0 0 0	0 0 0 0 9 0 0 0 0
0 0 3 0 0 0 0 0 0	0 0 3 0 0 0 0 0 0	0 0 3 0 0 0 0 0 0	0 0 3 0 0 0 0 0 0
0 6 0 0 0 3 0 0 2	0 6 0 0 0 3 0 0 2	0 6 0 0 0 3 0 0 2	0 6 0 0 0 3 0 0 2
Solution	Failed	Solution	Solution
5 8 2 4 7 9 1 6 3	5 1 2 4 3 7 8 6 9	5 1 2 4 3 7 8 6 9	5 1 2 4 3 7 8 6 9
3 7 9 1 6 5 4 2 8	3 8 9 5 1 6 2 4 7	3 8 9 1 5 6 2 4 7	3 8 9 1 5 6 2 4 7
6 4 1 3 2 8 7 9 5	6 4 7 8 2 9 1 3 5	6 4 7 8 2 9 1 3 5	6 4 7 8 2 9 1 3 5
5 9 4 7 8 1 2 3 6	4 3 5 9 6 1 7 2 8	4 3 5 9 6 1 7 2 8	4 3 5 9 6 1 7 2 8
2 7 8 9 3 6 5 4 1	2 9 8 3 7 4 5 1 6	2 9 8 3 7 4 5 1 6	2 9 8 3 7 4 5 1 6
1 3 6 5 4 2 8 7 9	1 7 3 5 8 2 9 4 3	1 7 6 5 8 2 3 9 4	1 7 6 5 8 2 3 9 4
8 2 5 6 9 7 3 1 4	7 2 6 1 9 5 4 8 3	7 2 1 6 9 5 4 8 3	7 2 1 6 9 5 4 8 3
9 1 3 2 5 4 6 8 7	9 5 3 2 8 6 4 7 1	9 5 3 2 4 8 6 7 1	9 5 3 2 4 8 6 7 1
4 6 7 8 1 3 9 5 2	8 6 4 7 1 3 5 9 2	8 6 4 7 1 3 9 5 2	8 6 4 7 1 3 9 5 2

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 2 1	0 0 0 0 0 0 0 2 1	0 0 0 0 0 0 0 2 1	0 0 0 0 0 0 0 2 1
5 0 3 0 0 0 0 0 0	5 0 3 0 0 0 0 0 0	5 0 3 0 0 0 0 0 0	5 0 3 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0	6 0 0 0 0 0 0 0 0	6 0 0 0 0 0 0 0 0	6 0 0 0 0 0 0 0 0
0 0 0 1 0 4 0 6 0	0 0 0 1 0 4 0 6 0	0 0 0 1 0 4 0 6 0	0 0 0 1 0 4 0 6 0
7 0 0 0 0 0 5 0 0	7 0 0 0 0 0 5 0 0	7 0 0 0 0 0 5 0 0	7 0 0 0 0 0 5 0 0
0 0 0 2 0 0 0 0 0	0 0 0 2 0 0 0 0 0	0 0 0 2 0 0 0 0 0	0 0 0 2 0 0 0 0 0
0 0 0 4 8 0 3 0 0	0 0 0 4 8 0 3 0 0	0 0 0 4 8 0 3 0 0	0 0 0 4 8 0 3 0 0
0 1 0 0 7 0 0 0 0	0 1 0 0 7 0 0 0 0	0 1 0 0 7 0 0 0 0	0 1 0 0 7 0 0 0 0
2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0	2 0 0 0 0 0 0 0 0
Solution	Solution	Solution	Solution
8 9 4 5 1 6 7 2 1	8 7 9 5 4 3 6 2 1	8 7 9 5 4 3 6 2 1	8 7 9 5 4 3 6 2 1
5 2 3 7 4 3 8 9 6	5 2 3 7 1 6 4 8 9	5 2 3 7 1 6 4 8 9	5 2 3 7 1 6 4 8 9
6 7 1 9 2 8 4 3 5	6 4 1 8 2 9 7 3 5	6 4 1 8 2 9 7 3 5	6 4 1 8 2 9 7 3 5
3 8 9 1 5 4 2 6 7	3 8 5 1 9 4 2 6 7	3 8 5 1 9 4 2 6 7	3 8 5 1 9 4 2 6 7
7 4 2 8 6 9 5 1 3	7 9 2 6 3 8 5 1 4	7 9 2 6 3 8 5 1 4	7 9 2 6 3 8 5 1 4
1 6 5 2 3 7 9 4 8	1 6 4 2 5 7 8 9 3	1 6 4 2 5 7 8 9 3	1 6 4 2 5 7 8 9 3
9 5 6 4 8 1 3 7 2	9 5 6 4 8 1 3 7 2	9 5 6 4 8 1 3 7 2	9 5 6 4 8 1 3 7 2
4 1 8 3 7 2 6 5 9	4 1 8 3 7 2 9 5 6	4 1 8 3 7 2 9 5 6	4 1 8 3 7 2 9 5 6
2 3 7 6 9 5 1 8 4	2 3 7 9 6 5 1 4 8	2 3 7 9 6 5 1 4 8	2 3 7 9 6 5 1 4 8

The solutions for 27 clues Sudoku problems

Restricted Boltzmann Machine	Message Passing	Breadth First Search	Depth First Search
Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
6 3 0 0 0 0 0 0 0	6 3 0 0 0 0 0 0 0	6 3 0 0 0 0 0 0 0	6 3 0 0 0 0 0 0 0
0 0 0 5 0 0 0 0 8	0 0 0 5 0 0 0 0 8	0 0 0 5 0 0 0 0 8	0 0 0 5 0 0 0 0 8
0 0 5 6 7 4 0 0 0	0 0 5 6 7 4 0 0 0	0 0 5 6 7 4 0 0 0	0 0 5 6 7 4 0 0 0
0 0 0 0 2 0 0 0 0	0 0 0 0 2 0 0 0 0	0 0 0 0 2 0 0 0 0	0 0 0 0 2 0 0 0 0
0 0 3 4 0 1 0 2 0	0 0 3 4 0 1 0 2 0	0 0 3 4 0 1 0 2 0	0 0 3 4 0 1 0 2 0
0 0 0 0 0 0 3 4 5	0 0 0 0 0 0 3 4 5	0 0 0 0 0 0 3 4 5	0 0 0 0 0 0 3 4 5
0 0 0 0 0 7 0 0 4	0 0 0 0 0 7 0 0 4	0 0 0 0 0 7 0 0 4	0 0 0 0 0 7 0 0 4
0 8 0 3 0 0 9 0 2	0 8 0 3 0 0 9 0 2	0 8 0 3 0 0 9 0 2	0 8 0 3 0 0 9 0 2
9 4 7 1 0 0 0 8 0	9 4 7 1 0 0 0 8 0	9 4 7 1 0 0 0 8 0	9 4 7 1 0 0 0 8 0
Solution	Solution	Solution	Solution
6 3 9 2 1 8 4 5 7	6 3 9 2 1 8 4 5 7	6 3 9 2 1 8 4 5 7	6 3 9 2 1 8 4 5 7
4 7 1 5 3 9 2 6 8	4 7 1 5 3 9 2 6 8	4 7 1 5 3 9 2 6 8	4 7 1 5 3 9 2 6 8
8 2 5 6 7 4 1 3 9	8 2 5 6 7 4 1 3 9	8 2 5 6 7 4 1 3 9	8 2 5 6 7 4 1 3 9
5 6 4 8 2 3 7 9 1	5 6 4 8 2 3 7 9 1	5 6 4 8 2 3 7 9 1	5 6 4 8 2 3 7 9 1
7 9 3 4 0 1 0 2 0	7 9 3 4 5 1 8 2 6	7 9 3 4 5 1 8 2 6	7 9 3 4 5 1 8 2 6
2 1 8 0 0 0 3 4 5	2 1 8 7 9 6 3 4 5	2 1 8 7 9 6 3 4 5	2 1 8 7 9 6 3 4 5
0 0 0 0 0 7 0 0 4	3 5 2 9 8 7 6 1 4	3 5 2 9 8 7 6 1 4	3 5 2 9 8 7 6 1 4
0 8 0 3 0 0 9 0 2	1 8 6 3 4 5 9 7 2	1 8 6 3 4 5 9 7 2	1 8 6 3 4 5 9 7 2
9 4 7 1 0 0 0 8 0	9 4 7 1 6 2 5 8 3	9 4 7 1 6 2 5 8 3	9 4 7 1 6 2 5 8 3

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
3 6 0 0 2 0 0 8 9	3 6 0 0 2 0 0 8 9	3 6 0 0 2 0 0 8 9	3 6 0 0 2 0 0 8 9
0 0 0 3 6 1 0 0 0	0 0 0 3 6 1 0 0 0	0 0 0 3 6 1 0 0 0	0 0 0 3 6 1 0 0 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
8 0 3 0 0 0 6 0 2	8 0 3 0 0 0 6 0 2	8 0 3 0 0 0 6 0 2	8 0 3 0 0 0 6 0 2
4 0 0 6 0 3 0 0 7	4 0 0 6 0 3 0 0 7	4 0 0 6 0 3 0 0 7	4 0 0 6 0 3 0 0 7
6 0 7 0 0 0 1 0 8	6 0 7 0 0 0 1 0 8	6 0 7 0 0 0 1 0 8	6 0 7 0 0 0 1 0 8
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 0 0 4 1 8 0 0 0	0 0 0 4 1 8 0 0 0	0 0 0 4 1 8 0 0 0	0 0 0 4 1 8 0 0 0
9 7 0 0 3 0 0 1 0	9 7 0 0 3 0 0 1 0	9 7 0 0 3 0 0 1 0	9 7 0 0 3 0 0 1 0
Solution	Failed	Solution	Solution
3 6 1 7 2 5 4 8 9	3 6 1 5 2 4 7 8 9	3 6 1 5 2 4 7 8 9	3 6 1 5 2 4 7 8 9
7 8 9 3 6 1 3 2 5	7 8 9 3 6 1 2 5 4	7 8 9 3 6 1 2 5 4	7 8 9 3 6 1 2 5 4
2 4 5 8 9 4 7 6 1	2 5 5 8 9 7 6 1 3	2 4 5 8 7 9 3 6 1	2 4 5 8 7 9 3 6 1
8 9 3 1 5 7 6 4 2	8 9 3 1 5 7 6 4 2	8 9 3 1 5 7 6 4 2	8 9 3 1 5 7 6 4 2
4 1 2 6 8 3 5 9 7	4 0 0 6 0 3 0 0 7	4 1 2 6 8 3 5 9 7	4 1 2 6 8 3 5 9 7
6 5 7 2 4 9 1 3 8	6 0 7 0 0 0 1 0 8	6 5 7 9 4 2 1 3 8	6 5 7 9 4 2 1 3 8
1 3 8 9 7 6 2 5 4	0 0 0 0 0 0 0 0 0	1 3 4 7 9 5 8 2 6	1 3 4 7 9 5 8 2 6
5 2 6 4 1 8 9 7 3	0 0 0 4 1 8 0 0 0	5 2 6 4 1 8 9 7 3	5 2 6 4 1 8 9 7 3
9 7 4 5 3 2 8 1 6	9 7 0 0 3 0 0 1 0	9 7 8 2 3 6 4 1 5	9 7 8 2 3 6 4 1 5

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 7 2 5 6 4 0 0	0 0 7 2 5 6 4 0 0	0 0 7 2 5 6 4 0 0	0 0 7 2 5 6 4 0 0
4 0 0 0 0 0 0 0 5	4 0 0 0 0 0 0 0 5	4 0 0 0 0 0 0 0 5	4 0 0 0 0 0 0 0 5
0 1 0 0 3 0 0 6 0	0 1 0 0 3 0 0 6 0	0 1 0 0 3 0 0 6 0	0 1 0 0 3 0 0 6 0
0 0 0 5 0 8 0 0 0	0 0 0 5 0 8 0 0 0	0 0 0 5 0 8 0 0 0	0 0 0 5 0 8 0 0 0
0 0 8 0 6 0 2 0 0	0 0 8 0 6 0 2 0 0	0 0 8 0 6 0 2 0 0	0 0 8 0 6 0 2 0 0
0 0 0 1 0 7 0 0 0	0 0 0 1 0 7 0 0 0	0 0 0 1 0 7 0 0 0	0 0 0 1 0 7 0 0 0
0 3 0 0 7 0 0 9 0	0 3 0 0 7 0 0 9 0	0 3 0 0 7 0 0 9 0	0 3 0 0 7 0 0 9 0
2 0 0 0 0 0 0 0 4	2 0 0 0 0 0 0 0 4	2 0 0 0 0 0 0 0 4	2 0 0 0 0 0 0 0 4
0 0 6 3 1 2 7 0 0	0 0 6 3 1 2 7 0 0	0 0 6 3 1 2 7 0 0	0 0 6 3 1 2 7 0 0
Solution	Solution	Solution	Solution
3 8 7 2 5 6 4 1 9	3 8 7 2 5 6 4 1 9	3 8 7 2 5 6 4 1 9	3 8 7 2 5 6 4 1 9
4 6 2 8 9 1 3 7 5	4 6 9 7 8 1 3 2 5	4 6 9 7 8 1 3 2 5	4 6 9 7 8 1 3 2 5
5 1 9 7 3 4 8 6 2	5 1 2 4 3 9 8 6 7	5 1 2 4 3 9 8 6 7	5 1 2 4 3 9 8 6 7
1 2 3 5 4 8 9 6 7	1 2 3 5 4 8 9 7 6	1 2 3 5 4 8 9 7 6	1 2 3 5 4 8 9 7 6
7 5 8 9 6 3 2 4 1	7 5 8 9 6 3 2 4 1	7 5 8 9 6 3 2 4 1	7 5 8 9 6 3 2 4 1
6 9 4 1 2 7 5 8 3	6 9 4 1 2 7 5 8 3	6 9 4 1 2 7 5 8 3	6 9 4 1 2 7 5 8 3
8 3 1 4 7 5 2 9 6	8 3 5 6 7 4 1 9 2	8 3 5 6 7 4 1 9 2	8 3 5 6 7 4 1 9 2
2 7 5 6 8 9 1 3 4	2 7 1 8 9 5 6 3 4	2 7 1 8 9 5 6 3 4	2 7 1 8 9 5 6 3 4
9 4 6 3 1 2 7 5 8	9 4 6 3 1 2 7 5 8	9 4 6 3 1 2 7 5 8	9 4 6 3 1 2 7 5 8

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
0 7 9 0 5 0 1 8 0	0 7 9 0 5 0 1 8 0	0 7 9 0 5 0 1 8 0	0 7 9 0 5 0 1 8 0
8 0 0 0 0 0 0 0 7	8 0 0 0 0 0 0 0 7	8 0 0 0 0 0 0 0 7	8 0 0 0 0 0 0 0 7
0 0 7 3 0 6 0 0 0	0 0 7 3 0 6 0 0 0	0 0 7 3 0 6 0 0 0	0 0 7 3 0 6 0 0 0
4 5 0 7 0 8 0 9 6	4 5 0 7 0 8 0 9 6	4 5 0 7 0 8 0 9 6	4 5 0 7 0 8 0 9 6
0 0 3 5 0 2 7 0 0	0 0 3 5 0 2 7 0 0	0 0 3 5 0 2 7 0 0	0 0 3 5 0 2 7 0 0
7 0 0 0 0 0 0 0 5	7 0 0 0 0 0 0 0 5	7 0 0 0 0 0 0 0 5	7 0 0 0 0 0 0 0 5
0 1 6 0 3 0 4 2 0	0 1 6 0 3 0 4 2 0	0 1 6 0 3 0 4 2 0	0 1 6 0 3 0 4 2 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0
Solution	Solution	Solution	Solution
3 4 1 8 2 7 5 6 9	3 4 5 8 7 1 2 6 9	3 4 5 8 7 1 2 6 9	3 4 5 8 7 1 2 6 9
2 7 9 6 5 3 1 8 4	2 7 9 6 5 3 1 8 4	2 7 9 6 5 3 1 8 4	2 7 9 6 5 3 1 8 4
8 6 5 9 4 1 2 3 7	8 6 1 4 2 9 5 3 7	8 6 1 4 2 9 5 3 7	8 6 1 4 2 9 5 3 7
1 9 7 3 4 6 8 5 2	1 9 7 3 4 6 8 5 2	1 9 7 3 4 6 8 5 2	1 9 7 3 4 6 8 5 2
4 5 2 7 1 8 3 9 6	4 5 2 7 1 8 3 9 6	4 5 2 7 1 8 3 9 6	4 5 2 7 1 8 3 9 6
6 8 3 5 9 2 7 4 1	6 8 3 5 9 2 7 4 1	6 8 3 5 9 2 7 4 1	6 8 3 5 9 2 7 4 1
7 3 4 2 8 9 6 1 5	7 3 8 2 6 4 9 1 5	7 3 8 2 6 4 9 1 5	7 3 8 2 6 4 9 1 5
9 1 6 7 3 5 4 2 8	5 1 6 9 3 7 4 2 8	5 1 6 9 3 7 4 2 8	5 1 6 9 3 7 4 2 8
5 2 8 1 6 4 9 7 3	9 2 4 1 8 5 6 7 3	9 2 4 1 8 5 6 7 3	9 2 4 1 8 5 6 7 3

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 1 0 2 0 9 0 0	0 0 1 0 2 0 9 0 0	0 0 1 0 2 0 9 0 0	0 0 1 0 2 0 9 0 0
0 8 0 9 6 0 0 1 0	0 8 0 9 6 0 0 1 0	0 8 0 9 6 0 0 1 0	0 8 0 9 6 0 0 1 0
4 0 0 0 0 0 0 5 7	4 0 0 0 0 0 0 5 7	4 0 0 0 0 0 0 5 7	4 0 0 0 0 0 0 5 7
0 0 8 0 0 0 4 0 1	0 0 8 0 0 0 4 0 1	0 0 8 0 0 0 4 0 1	0 0 8 0 0 0 4 0 1
0 0 0 6 0 3 0 0 0	0 0 0 6 0 3 0 0 0	0 0 0 6 0 3 0 0 0	0 0 0 6 0 3 0 0 0
2 0 9 0 0 0 8 0 0	2 0 9 0 0 0 8 0 0	2 0 9 0 0 0 8 0 0	2 0 9 0 0 0 8 0 0
7 4 0 0 0 0 0 0 5	7 4 0 0 0 0 0 0 5	7 4 0 0 0 0 0 0 5	7 4 0 0 0 0 0 0 5
0 2 0 0 1 8 0 6 0	0 2 0 0 1 8 0 6 0	0 2 0 0 1 8 0 6 0	0 2 0 0 1 8 0 6 0
0 0 5 0 7 0 0 0 0	0 0 5 0 7 0 0 0 0	0 0 5 0 7 0 0 0 0	0 0 5 0 7 0 0 0 0
Solution	Solution	Solution	Solution
3 6 1 7 2 5 9 4 8	3 6 1 7 2 5 9 4 8	3 6 1 7 2 5 9 4 8	3 6 1 7 2 5 9 4 8
5 8 7 9 6 4 2 1 3	5 8 7 9 6 4 2 1 3	5 8 7 9 6 4 2 1 3	5 8 7 9 6 4 2 1 3
4 9 2 8 3 1 6 5 7	4 9 2 8 3 1 6 5 7	4 9 2 8 3 1 6 5 7	4 9 2 8 3 1 6 5 7
6 3 8 2 5 9 4 7 1	6 3 8 2 5 9 4 7 1	6 3 8 2 5 9 4 7 1	6 3 8 2 5 9 4 7 1
1 7 4 6 8 3 5 9 2	1 7 4 6 8 3 5 2 9	1 7 4 6 8 3 5 2 9	1 7 4 6 8 3 5 2 9
2 5 9 1 4 7 8 3 6	2 5 9 1 4 7 8 3 6	2 5 9 1 4 7 8 3 6	2 5 9 1 4 7 8 3 6
7 4 6 3 9 6 1 8 5	7 4 6 3 9 2 1 8 5	7 4 6 3 9 2 1 8 5	7 4 6 3 9 2 1 8 5
9 2 3 5 1 8 7 6 4	9 2 3 5 1 8 7 6 4	9 2 3 5 1 8 7 6 4	9 2 3 5 1 8 7 6 4
8 1 5 4 7 2 3 2 9	8 1 5 4 7 6 3 9 2	8 1 5 4 7 6 3 9 2	8 1 5 4 7 6 3 9 2

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 5 0 8 0 0 0 2 0	0 5 0 8 0 0 0 2 0	0 5 0 8 0 0 0 2 0	0 5 0 8 0 0 0 2 0
6 0 0 0 1 0 0 9 0	6 0 0 0 1 0 0 9 0	6 0 0 0 1 0 0 9 0	6 0 0 0 1 0 0 9 0
7 0 0 0 0 0 0 0 6	7 0 0 0 0 0 0 0 6	7 0 0 0 0 0 0 0 6	7 0 0 0 0 0 0 0 6
0 7 0 0 2 0 3 0 0	0 7 0 0 2 0 3 0 0	0 7 0 0 2 0 3 0 0	0 7 0 0 2 0 3 0 0
5 0 4 0 0 0 9 0 8	5 0 4 0 0 0 9 0 8	5 0 4 0 0 0 9 0 8	5 0 4 0 0 0 9 0 8
1 0 3 0 8 0 0 0 0	1 0 3 0 8 0 0 0 0	1 0 3 0 8 0 0 0 0	1 0 3 0 8 0 0 0 0
9 0 0 0 7 0 2 0 0	9 0 0 0 7 0 2 0 0	9 0 0 0 7 0 2 0 0	9 0 0 0 7 0 2 0 0
0 6 0 0 9 0 0 0 0	0 6 0 0 9 0 0 0 0	0 6 0 0 9 0 0 0 0	0 6 0 0 9 0 0 0 0
0 8 0 1 0 3 0 4 0	0 8 0 1 0 3 0 4 0	0 8 0 1 0 3 0 4 0	0 8 0 1 0 3 0 4 0
Solution	Failed	Solution	Solution
4 5 9 8 3 7 4 2 1	3 5 9 8 4 6 1 2 7	3 5 9 8 4 6 1 2 7	3 5 9 8 4 6 1 2 7
6 3 8 2 1 6 7 9 5	6 4 2 3 1 7 8 9 5	6 4 2 3 1 7 8 9 5	6 4 2 3 1 7 8 9 5
7 1 2 5 4 9 8 3 6	7 1 8 2 5 9 4 3 6	7 1 8 2 5 9 4 3 6	7 1 8 2 5 9 4 3 6
8 7 6 9 2 5 3 1 4	8 7 6 9 2 5 3 1 4	8 7 6 9 2 5 3 1 4	8 7 6 9 2 5 3 1 4
5 2 4 3 6 1 9 7 8	5 2 4 6 3 1 9 7 8	5 2 4 6 3 1 9 7 8	5 2 4 6 3 1 9 7 8
1 9 3 7 8 4 5 6 2	1 9 3 7 8 4 6 5 2	1 9 3 7 8 4 6 5 2	1 9 3 7 8 4 6 5 2
9 4 1 6 7 8 2 5 3	9 3 5 4 7 8 2 0 0	9 3 5 4 7 8 2 6 1	9 3 5 4 7 8 2 6 1
3 6 5 4 9 2 1 8 7	0 6 0 0 9 0 0 0 0	4 6 1 5 9 2 7 8 3	4 6 1 5 9 2 7 8 3
2 8 7 1 5 3 6 4 9	0 8 0 1 0 3 0 4 0	2 8 7 1 6 3 5 4 9	2 8 7 1 6 3 5 4 9

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 4 0 0 0 0 0 5 0	0 4 0 0 0 0 0 5 0	0 4 0 0 0 0 0 5 0	0 4 0 0 0 0 0 5 0
0 0 1 9 4 3 6 0 0	0 0 1 9 4 3 6 0 0	0 0 1 9 4 3 6 0 0	0 0 1 9 4 3 6 0 0
0 0 9 0 0 0 3 0 0	0 0 9 0 0 0 3 0 0	0 0 9 0 0 0 3 0 0	0 0 9 0 0 0 3 0 0
6 0 0 0 5 0 0 0 2	6 0 0 0 5 0 0 0 2	6 0 0 0 5 0 0 0 2	6 0 0 0 5 0 0 0 2
1 0 3 0 0 0 5 0 6	1 0 3 0 0 0 5 0 6	1 0 3 0 0 0 5 0 6	1 0 3 0 0 0 5 0 6
8 0 0 0 2 0 0 0 7	8 0 0 0 2 0 0 0 7	8 0 0 0 2 0 0 0 7	8 0 0 0 2 0 0 0 7
0 0 5 0 0 0 2 0 0	0 0 5 0 0 0 2 0 0	0 0 5 0 0 0 2 0 0	0 0 5 0 0 0 2 0 0
0 0 2 4 0 6 7 0 0	0 0 2 4 0 6 7 0 0	0 0 2 4 0 6 7 0 0	0 0 2 4 0 6 7 0 0
0 3 0 0 0 0 0 4 0	0 3 0 0 0 0 0 4 0	0 3 0 0 0 0 0 4 0	0 3 0 0 0 0 0 4 0
Solution	Solution	Solution	Solution
3 4 8 2 6 7 9 5 1	3 4 8 2 6 7 9 5 1	3 4 8 2 6 7 9 5 1	3 4 8 2 6 7 9 5 1
5 7 1 9 4 3 6 2 8	5 7 1 9 4 3 6 2 8	5 7 1 9 4 3 6 2 8	5 7 1 9 4 3 6 2 8
2 6 9 1 8 5 3 7 4	2 6 9 1 8 5 3 7 4	2 6 9 1 8 5 3 7 4	2 6 9 1 8 5 3 7 4
6 9 7 6 5 1 4 3 2	6 9 7 3 5 1 4 8 2	6 9 7 3 5 1 4 8 2	6 9 7 3 5 1 4 8 2
1 2 3 8 7 4 5 9 6	1 2 3 8 7 4 5 9 6	1 2 3 8 7 4 5 9 6	1 2 3 8 7 4 5 9 6
8 5 4 3 2 9 1 8 7	8 5 4 6 2 9 1 3 7	8 5 4 6 2 9 1 3 7	8 5 4 6 2 9 1 3 7
4 1 5 7 9 8 2 6 3	4 1 5 7 9 8 2 6 3	4 1 5 7 9 8 2 6 3	4 1 5 7 9 8 2 6 3
9 8 2 4 3 6 7 1 5	9 8 2 4 3 6 7 1 5	9 8 2 4 3 6 7 1 5	9 8 2 4 3 6 7 1 5
7 3 6 5 1 2 8 4 9	7 3 6 5 1 2 8 4 9	7 3 6 5 1 2 8 4 9	7 3 6 5 1 2 8 4 9

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 2 0 0 3 0 0 9 0	0 2 0 0 3 0 0 9 0	0 2 0 0 3 0 0 9 0	0 2 0 0 3 0 0 9 0
0 0 0 0 0 7 0 0 0	0 0 0 0 0 7 0 0 0	0 0 0 0 0 7 0 0 0	0 0 0 0 0 7 0 0 0
9 0 0 2 0 8 0 0 5	9 0 0 2 0 8 0 0 5	9 0 0 2 0 8 0 0 5	9 0 0 2 0 8 0 0 5
0 0 0 8 0 0 5 0 0	0 0 0 8 0 0 5 0 0	0 0 0 8 0 0 5 0 0	0 0 0 8 0 0 5 0 0
6 0 7 0 0 0 2 0 8	6 0 7 0 0 0 2 0 8	6 0 7 0 0 0 2 0 8	6 0 7 0 0 0 2 0 8
0 0 3 1 0 2 9 0 0	0 0 3 1 0 2 9 0 0	0 0 3 1 0 2 9 0 0	0 0 3 1 0 2 9 0 0
8 0 0 6 0 5 0 0 7	8 0 0 6 0 5 0 0 7	8 0 0 6 0 5 0 0 7	8 0 0 6 0 5 0 0 7
0 0 0 3 0 9 0 0 0	0 0 0 3 0 9 0 0 0	0 0 0 3 0 9 0 0 0	0 0 0 3 0 9 0 0 0
0 3 0 0 2 0 0 5 0	0 3 0 0 2 0 0 5 0	0 3 0 0 2 0 0 5 0	0 3 0 0 2 0 0 5 0
Solution	Failed	Solution	Solution
7 2 8 5 3 6 4 9 1	1 2 8 5 3 4 7 9 6	1 2 8 5 3 4 7 9 6	1 2 8 5 3 4 7 9 6
3 1 5 4 9 7 6 8 2	3 6 5 9 1 7 4 3 2	3 6 5 9 1 7 4 8 2	3 6 5 9 1 7 4 8 2
9 6 4 2 1 8 7 3 5	9 7 4 2 6 8 1 8 5	9 7 4 2 6 8 1 3 5	9 7 4 2 6 8 1 3 5
2 9 1 8 7 4 5 6 3	2 4 1 8 9 6 5 6 3	2 4 1 8 9 6 5 7 3	2 4 1 8 9 6 5 7 3
6 4 7 9 5 3 2 1 8	6 9 7 4 5 3 2 7 8	6 9 7 4 5 3 2 1 8	6 9 7 4 5 3 2 1 8
5 8 3 1 6 2 9 7 4	5 8 3 1 7 2 9 1 4	5 8 3 1 7 2 9 6 4	5 8 3 1 7 2 9 6 4
8 7 9 6 4 5 1 2 7	8 1 9 6 4 5 3 4 7	8 1 9 6 4 5 3 2 7	8 1 9 6 4 5 3 2 7
1 5 2 3 8 9 3 4 6	7 5 2 3 8 9 6 1 1	7 5 2 3 8 9 6 4 1	7 5 2 3 8 9 6 4 1
4 3 6 7 2 1 8 5 9	4 3 6 7 2 1 8 5 9	4 3 6 7 2 1 8 5 9	4 3 6 7 2 1 8 5 9

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 8 0 0 0 5 0 0 0	0 8 0 0 0 5 0 0 0	0 8 0 0 0 5 0 0 0	0 8 0 0 0 5 0 0 0
0 0 0 0 0 3 0 5 7	0 0 0 0 0 3 0 5 7	0 0 0 0 0 3 0 5 7	0 0 0 0 0 3 0 5 7
0 0 0 0 7 0 8 0 9	0 0 0 0 7 0 8 0 9	0 0 0 0 7 0 8 0 9	0 0 0 0 7 0 8 0 9
0 6 0 4 0 0 9 0 3	0 6 0 4 0 0 9 0 3	0 6 0 4 0 0 9 0 3	0 6 0 4 0 0 9 0 3
0 0 7 0 1 0 5 0 0	0 0 7 0 1 0 5 0 0	0 0 7 0 1 0 5 0 0	0 0 7 0 1 0 5 0 0
4 0 8 0 0 7 0 2 0	4 0 8 0 0 7 0 2 0	4 0 8 0 0 7 0 2 0	4 0 8 0 0 7 0 2 0
9 0 1 0 2 0 0 0 0	9 0 1 0 2 0 0 0 0	9 0 1 0 2 0 0 0 0	9 0 1 0 2 0 0 0 0
8 4 0 3 0 0 0 0 0	8 4 0 3 0 0 0 0 0	8 4 0 3 0 0 0 0 0	8 4 0 3 0 0 0 0 0
0 0 0 1 0 0 0 8 0	0 0 0 1 0 0 0 8 0	0 0 0 1 0 0 0 8 0	0 0 0 1 0 0 0 8 0
Solution	Failed	Solution	Solution
7 8 6 9 4 5 3 1 2	7 8 3 9 6 5 2 1 4	7 8 3 9 6 5 2 1 4	7 8 3 9 6 5 2 1 4
5 1 9 2 6 3 4 5 7	2 1 9 8 4 3 6 5 7	2 1 9 8 4 3 6 5 7	2 1 9 8 4 3 6 5 7
3 2 4 8 7 1 8 6 9	6 5 4 2 7 1 8 3 9	6 5 4 2 7 1 8 3 9	6 5 4 2 7 1 8 3 9
1 6 5 4 8 2 9 7 3	1 6 5 4 8 2 9 7 3	1 6 5 4 8 2 9 7 3	1 6 5 4 8 2 9 7 3
2 3 7 6 1 9 5 4 8	3 2 7 6 1 9 5 4 8	3 2 7 6 1 9 5 4 8	3 2 7 6 1 9 5 4 8
4 9 8 5 3 7 1 2 6	4 9 8 3 5 7 6 2 1	4 9 8 5 3 7 1 2 6	4 9 8 5 3 7 1 2 6
9 5 1 7 2 8 6 3 4	9 3 1 7 2 8 4 6 5	9 3 1 7 2 8 4 6 5	9 3 1 7 2 8 4 6 5
8 4 2 3 5 6 7 9 1	8 4 6 3 5 6 7 9 1	8 4 2 3 5 6 7 9 1	8 4 2 3 5 6 7 9 1
6 7 3 1 9 4 2 8 5	5 7 2 1 9 4 3 8 2	5 7 6 1 9 4 3 8 2	5 7 6 1 9 4 3 8 2

Sudoku Problem	Sudoku Problem	Sudoku Problem	Sudoku Problem
0 0 0 5 0 2 9 0 0	0 0 0 5 0 2 9 0 0	0 0 0 5 0 2 9 0 0	0 0 0 5 0 2 9 0 0
0 0 0 0 4 0 0 0 0	0 0 0 0 4 0 0 0 0	0 0 0 0 4 0 0 0 0	0 0 0 0 4 0 0 0 0
1 0 6 0 0 0 3 0 5	1 0 6 0 0 0 3 0 5	1 0 6 0 0 0 3 0 5	1 0 6 0 0 0 3 0 5
0 0 0 2 5 1 0 0 8	0 0 0 2 5 1 0 0 8	0 0 0 2 5 1 0 0 8	0 0 0 2 5 1 0 0 8
0 7 0 4 0 8 0 3 0	0 7 0 4 0 8 0 3 0	0 7 0 4 0 8 0 3 0	0 7 0 4 0 8 0 3 0
8 0 0 0 0 0 0 0 1	8 0 0 0 0 0 0 0 1	8 0 0 0 0 0 0 0 1	8 0 0 0 0 0 0 0 1
3 0 8 0 0 0 1 0 4	3 0 8 0 0 0 1 0 4	3 0 8 0 0 0 1 0 4	3 0 8 0 0 0 1 0 4
0 0 0 0 2 0 0 0 0	0 0 0 0 2 0 0 0 0	0 0 0 0 2 0 0 0 0	0 0 0 0 2 0 0 0 0
0 0 5 1 0 4 8 0 0	0 0 5 1 0 4 8 0 0	0 0 5 1 0 4 8 0 0	0 0 5 1 0 4 8 0 0
Solution	Failed	Solution	Solution
4 3 7 5 1 2 9 8 6	4 3 7 5 1 2 9 8 6	4 3 7 5 1 2 9 8 6	4 3 7 5 1 2 9 8 6
5 8 9 3 4 6 2 1 7	5 8 9 3 4 6 2 1 7	5 8 9 3 4 6 2 1 7	5 8 9 3 4 6 2 1 7
1 2 6 7 8 9 3 4 5	1 2 6 7 8 9 3 4 5	1 2 6 7 8 9 3 4 5	1 2 6 7 8 9 3 4 5
6 4 3 2 5 1 7 9 8	6 4 3 2 5 1 7 9 8	6 4 3 2 5 1 7 9 8	6 4 3 2 5 1 7 9 8
9 7 1 4 6 8 5 3 2	9 7 1 4 6 8 5 3 2	9 7 1 4 6 8 5 3 2	9 7 1 4 6 8 5 3 2
8 5 2 9 7 3 4 6 1	8 5 2 0 0 0 0 0 1	8 5 2 9 3 7 4 6 1	8 5 2 9 3 7 4 6 1
3 9 8 6 9 5 1 7 4	3 0 8 0 0 0 1 0 4	3 9 8 6 7 5 1 2 4	3 9 8 6 7 5 1 2 4
2 1 4 8 2 7 6 5 3	0 0 0 0 2 0 0 0 0	7 1 4 8 2 3 6 5 9	7 1 4 8 2 3 6 5 9
7 6 5 1 3 4 8 2 9	0 0 5 1 0 4 8 0 0	2 6 5 1 9 4 8 7 3	2 6 5 1 9 4 8 7 3