B.Sc. in Computer Science and Engineering Thesis

Solving Sudoku as a Constraint Satisfaction Problem

Submitted by

Shahed Khan
201205055

Mahbub Alam
201205063

Sonjoy Kumar Paul
201205079

Supervised by

Abu Wasif

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

September 2017

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Solving Sudoku as a Constraint Satisfaction Problem", is the outcome of the investigation and research carried out by us under the supervision of Abu Wasif.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Shahed Khan
201205055

Mahbub Alam
201205063

Sonjoy Kumar Paul
201205079

CERTIFICATION

This thesis titled, **"Solving Sudoku as a Constraint Satisfaction Problem"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in September 2017.

Group Members:

Shahed Khan

Mahbub Alam

Sonjoy Kumar Paul

Supervisor:

Abu Wasif
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

We are thankful to

Finally,

Dhaka
September 2017

Shahed Khan

Mahbub Alam

Sonjoy Kumar Paul

Contents

List of Figures

List of Tables

List of Algorithms

ABSTRACT

Write your thesis abstract here.

Chapter 1

Introduction

Sudoku is puzzle solving game that originated in Japan. Nowadays it enjoys worldwide popularity. It appears regularly on newspapers and people with a keen interest in puzzle solving find it satisfying. There exists many techniques of solving Sudoku puzzles. We have regarded the Sudoku puzzle as a Constraint Satisfaction Problem and solved it using Constraint programming. This thesis explores different solving models and their comparative efficiency.

1.1 Background and Motivation

For the past few decades Sudoku has gained increasing popularity. It has appeared in various media such as newspapers and websites. Regular competitions are being held on Sudoku solving. Different variations such as X-Sudoku, Killer Sudoku, Jigsaw Sudoku are also gaining popularity. Different algorithms exists that solve Sudoku problem. Backtrack and Rule-based algorithms are some examples of that. However we have found out that Sudoku as a Constraint Satisfaction Problem has more grounds to cover. We have found only one type of existing model that solves Sudoku puzzles using All Different constraint. So we have tried to develop new models and compare those to already existing ones. Also Sudoku board has the property that however we rotate the board it still will have same solution. So we tried to explore this property using our models and tried to find out how the ordering of constraints affects the efficiency.

1.2 Objective

In this thesis our main goal was to find another model other than All-Different that solves the puzzle regarding it as a Constraint Satisfaction Problem. Implementing this model with different constraints we tried to find the comparative efficiency and the reason behind it. Also another of our objective was to explore the rotational property of Sudoku puzzle which means that if we rotate the board right or left, the resolution time changes. Rotation of the board basically means that the order of the constraints changes. We operated on different data and tried to find out how their resolution time changes based on different rotation.

1.3 Fundamentals Of Sudoku:

A Sudoku game consists of a 9x9 grid of numbers, where each number belongs to the range 1-9. Initially a subset of the grid is revealed and the goal is to fill the remaining grid with valid numbers. The grid is divided into 9 blocks of size 3x3. Sudoku has only one rule and that is that all regions, that is rows, columns, and blocks, contains the numbers 1-9 exactly once. In order to be regarded as a proper Sudoku puzzle it is also required that a unique solution exists, a property which can be determined by solving for all possible solutions.

1.4 Computational Perspective:

Sudoku solving is a research area in computer science and mathematics, with areas such as solving, puzzle difficulty rating and puzzle generation. The problem of solving $n^2 \times n^2$ Sudoku puzzles is NP-complete. While being theoretically interesting as an result, it has also motivated research into heuristics, resulting in a wide range of available solving methods. Some of the existing solving algorithms are backtrack, rule-based, cultural genetic with variations and Boltzmann machines.

Using all-different constraint for the solving model, the solver solves the problem by first taking the variables with single value, the clues, and reduce the domain of the candidates. It performs this repeatedly until all variables has single value.

In the model we implemented using count constraint. The approach is to take each row_block and row_column and assigning values in such a way so that each value is assigned to exactly two variables.

The main computational concern is the number of times the solver backtracks in each model. Reduction of backtracks result in a better resolution time.

Chapter 2

Problem Formulation

We take a Sudoku puzzle as a problem and solve it using Constraint Programming. We know that Sudoku is a 9 by 9 board. It has three types of group of cells. In each group every single cell must have different values. At the start, the board will have some cells with values and some empty cells. The goal is to fill the empty cells with appropriate values so each group can maintain all-different property. In order to correctly refer to the appropriate terms we define those as follows.

2.1 Sudoku Definitions:

Board: It is the whole 9 by 9 grid of the Sudoku puzzle.
Cell: Every square of the Sudoku board. There are 81 cells in total.
Row: Each group of 9 horizontal cells.
Column: Each group of 9 vertical cells.
Block: Each 3 by 3 grid of the board. They work same as the rows and columns which means every cell of the block must have different values.
Region: Each row, column and block is a region.
Row_Column: This is the combination of each row with each of its intersecting columns. Every row is taken and merged with each of the columns. The intersecting cell is counted twice. Once for the and once for the column. So each row_column has 18 cells in total. And there are 81 row_columns. This Row_Column is formed because it is needed in the second model.
Row_Block: This is another type of region that is needed in the second model. Here each row is merged with each of its intersecting blocks. Each row intersects three blocks. So there are 27 row_blocks. In this case each row block intersection has three intersecting cells. As before, each of the intersecting cells is counted twice. Once for the row and once for block. So each row_block has 18 cells in total.

Candidate: A candidate is an empty square in the board. Each Sudoku board has some candidates and the goal of the puzzle is to fill those candidates with appropriate value so that it maintains all-different property.
Clue: The squares in the board that have single values assigned to them at the start of the game are known as clues. They are unchangeable and they will be used to gain knowledge about filling up the candidates. It should be noted that only filled squares at the beginning are considered clues. Other squares that are filled by the solver are not considered clues.

2.2 CSP Definitions:

Variable: Variables have several assignable value. In Sudoku each of the 81 cells of the board is a variable. Each can a have a set of assignable values.
Domain: The set of values that are assignable to a variable is called its domain. Each candidate of the Sudoku board has domain size of nine. Where the values are in range of 1 to 9. Whereas each clue has only the pre-assigned value in its domain.
Constraint: Constraint consists a pair (scope, rel). Where scope means the a tuple of variables that participate in the constraint. Rel is a relation that defines the values those variables can take on.
Consistent Assignment: An assignment of values to variable that does not violate any constraints.
Complete Assignment: An assignment where each variable has an assignment. Solution to a Constraint Satisfaction Problem is a consistent, complete assignment.
Partial Assignment: When only some of the variables are assigned it is called a partial assignment.
Unary Constraint: It restricts the value of a single variable. An example is a lakeside house will have to be of color green.
Binary Constraint: A binary constraint relates two variables. An simple example is if a house is green then its neighboring house will have to be blue.
Global Constraint: A constraint involving an arbitrary number of variables is called a global constraint. An example is all-different constraint. We mainly use global constraint for our thesis.
All-Different Constraint: The variables on which this constraint is applied, each of those, must have different values. Since the Sudoku problem is built on this condition so the all-different constraint is naturally applicable to each of the regions of the board.
Count Constraint: This constraint is implemented by us to build a different solver other than the all-different one. The constraint is that every value from 1 to 9 must appear exactly twice in every row_column and every row_block.

Backtracking Search: It is a type of depth first search that chooses values one variable at a time and backtracks when a variable has no legal values left to assign.
Variable Ordering: In backtracking search which variable should be selected next for assignment is an important question. The intuitive idea is to take the variable with the smallest domain size. It forces the other variables to reduce their domains more effectively. This is known as Minimum Remaining Value(MRV) heuristic.
Degree Heuristic: An important heuristic for variable ordering that selects the variable that has the largest number of constraints with other unassigned variables. It reduces the domains of other related variables effectively.
Value Ordering: For choosing a value for a variable, the value that rules out the fewest number of choices for other variables is considered first. This is known as Least Constraining Value heuristic.
Node Consistency: A variable is node consistent if all the values in the all the values in the variables domain satisfy the variables unary constraints.
Arc Consistency: A variable in CSP is arc consistent if every value in its domain satisfies the variables binary constraints. More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc(X_i, X_j). A network is arc-consistent if every variable is arc-consistent with every other variable.

Chapter 3

Models

All-different model employs all-different constraint on each of the regions of the board. This constraint is applied on every row, column and block of the board. Since each of the three types of region are 9 each, there are in total 27 constraints. These constraints ensure that each cell of every region will have unique values.

Arc consistency ensures solution of the puzzle in this model. All-different constraint can be expanded into binary constraints. Because it basically says that $A_i \neq A_j$ for all i,j, $i \neq j$ in each region.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Figure 3.1: Unsolved Sudoku puzzle

We consider variable E6 from figure 3.1, the candidate between 2 and 8 in the middle box. From constraints in the box, we can remove not only 2 and 8 but also 1 and 7 from E6s domain. From the constraints in its column, we can eliminate 5, 6, 2, 8, 9 and 3. That leaves E6 with a domain of 4. So we know the answer for E6.

Now we consider variable I6- the square in the bottom middle box surrounded by 1, 3 and 3. Applying arc consistency in its column, we eliminate 5, 6, 2, 4 (since we have concluded E6 must be 4), 8, 9 and 3. We eliminate 1 by arc consistency with I5, and we are left with only

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Figure 3.2: Solved Sudoku puzzle

the value 7 in domain of I6. Now there are 8 known values in column 6, so arc consistency can infer that A6 must be 1.

Inference continues along these lines and eventually, the model solves the entire puzzle, that is, all the variables have their domains reduced to a single value. It is shown in figure 3.2.

3.1 Region Ordered Model:

This model orders the constraints by region. At first the all different constraint is applied on all the rows of the board. Then the constraint is applied on all the columns of the board. And at last the constraint is applied on all the blocks. It means first the domain will be reduced by making the all the rows consistent. Then further domain reduction will occur by constraining the columns. And at last constraining all the blocks will result in a solution.

3.2 Row-Column-Block Ordered Model

This is the first and most common variant for the Sudoku solver. In this model the all-different constraint is applied such a way so that the constraint is applied on the first row, column and blocks first. Then it is applied on the second row, second column and second block. So to say it means the all different constraint is applied on sequentially on row, column and block and this is done on each of the row, column and block.

3.3 Count of Clue Ordered Model:

In order to employ this model, we explore a basic property of Sudoku board and how the domain reduction works. For each region, row, column or block, domain of a cell is reduced based upon all the clues of that region. If a candidate contains a value that is a clue in that region, then applying all-different constraint means that value will be removed from the domain of the candidate. All different constraint on a region ensures that the domain of each of the candidate of the region will be reduced by all the clues.

So we can notice that if every cell of a region is a clue then no further work is needed to be done. If there are eight clues in the region then there is only one candidate. Since eight of the nine values are assigned to eight of the cells so naturally the domain of the candidate reduces to one.

So we can deduce that the more clues a region have, the easier it is to reduce the domain of the candidates of that region. Based on this assumption, we have created our third model. This model first pre-processes all the rows, blocks and columns of the board. It takes each region and counts the number of clues in that region. Then it sorts the regions based on the count of clues in descending order. Then we apply the all-different constraint on all the 81 regions.

Since the regions are in descending order on the count of clues, applying the constraint on the regions with more clues first means their domains get reduced by a large number.

3.4 All-different-Count Model:

All-different-Count model employs a property of Sudoku to solve the puzzle. In Sudoku board each row must have a value exactly once. Each column must also have a value exactly once. So each intersecting row and column must have a value exactly twice. Except for the value in the intersecting cell of the row and column, which appears exactly once.

Now for the convenience of implementing the constraint, we did not count the intersecting cell. We make a region called row_column which has each of the value of the intersecting row and column. This means the region has 16 cells excepting the intersecting cell. There are 9 rows and 9 columns and each of the rows intersect each of the columns. So there are $9 \times 9 = 81$ row_columns in total.

Now we apply the count constraint on these regions. This constraint states that each value must appear more than once in each region. If the assignment of value is such that the region has only one or zero cell with that value then we know that it cannot find a solution and the constraint is failed. If there is assignment of variables in such way so that a value is assigned to more than one variable then we can further reduce the domain by propagating. Applying

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Figure 3.3: Row-Column Sudoku puzzle

this constraint does not ensures that each region will have each value exactly twice. It simply ensures that assignment of a value for only once or zero time will result in failure. If more than two variables are assigned the same value then the domain reduction is carried out further by all-different and count constraint on other regions.

So the count constraint being applied on the region ensures that if any assignment on a variable for a value reduces another variables domain to zero then this assignment is not consistent and the value will be removed from the variables domain.

This is applied for all the 81 one row_columns for all 9 possible values. So there are in total $81 \times 9 = 729$ constraints.

Now this constraint is used alongside the all-different constraint. We run the all-different constraint nine times for each row, column and block. We also run the count constraint on 81 regions for nine values. So in this model we impose all-different constraint for each row, column and block and impose count constraint on 81 regions in a loop from 1 to 9.

We create this model in order to analyze the effect of additional constraint along with the already existing all-different constraint. We try to find out how the number of backtracks and the resolution time changes if we add some constraint along with all-different.

Chapter 4

Experiment and Analysis:

In this section multiple results are presented together with a discussion about how the results could be interpreted.

For the experiments we used two types datasets. One set of data contains 95 Sudoku puzzles of hard rating. Most of these puzzles requires backtracks. In the case of this dataset, we consider number of backtracks for each model along with resolution time. We try to find out how the number of backtracks changes with different ordering of constraints and addition of new constraints.

Another set of data contains 500 Sudoku puzzles. We have collected these from fellow students working on Sudoku problem generation of different ratings. They are have different ratings of easy, medium and hard. We run the same tests for this dataset.

Section 4.1 is devoted to presenting how different models perform based on resolution time for the first dataset. Section 4.2 shows the result of the tests for second dataset. Section 4.3 shows how the algorithms performs relative to the each other and discusses different aspect of comparison.

4.1 Time Distributions for Dataset 1:

To get an idea of how each algorithm performs it is suitable to plot solving times in a histogram. Another way of displaying the performance is to sort the solving times and plot puzzle index versus solving time. Both of these are of interest however since they can reveal different things about the algorithms performance.

4.1.1 Region Ordered Model:

After testing the row-column-block ordered model on the 95 hard puzzle dataset, it had a mean solving time of 8.71 millisecond. its standard deviation was 6.512 milliseconds.

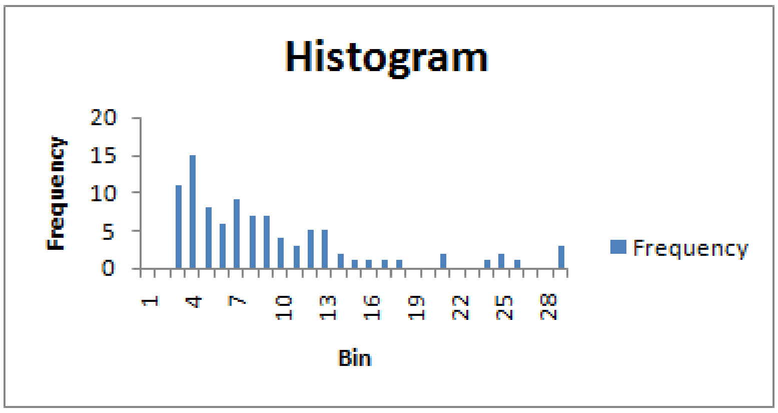

Figure 4.1: Histogram of Resolution time

At first we see from figure 4.1 the histogram for this model. From the histogram we can see that most of the puzzles had resolution time between 3 to 13 milliseconds. As was the case for the first model. But we can observe variation in this model. Although most of the puzzles are in the same range, for this model most of the puzzle lies within 4 milliseconds and as the resolution time increases number of puzzles decreases. We can also observe that while the first model had the maximum resolution time close to 40 milliseconds, this model does need more than 31 milliseconds.

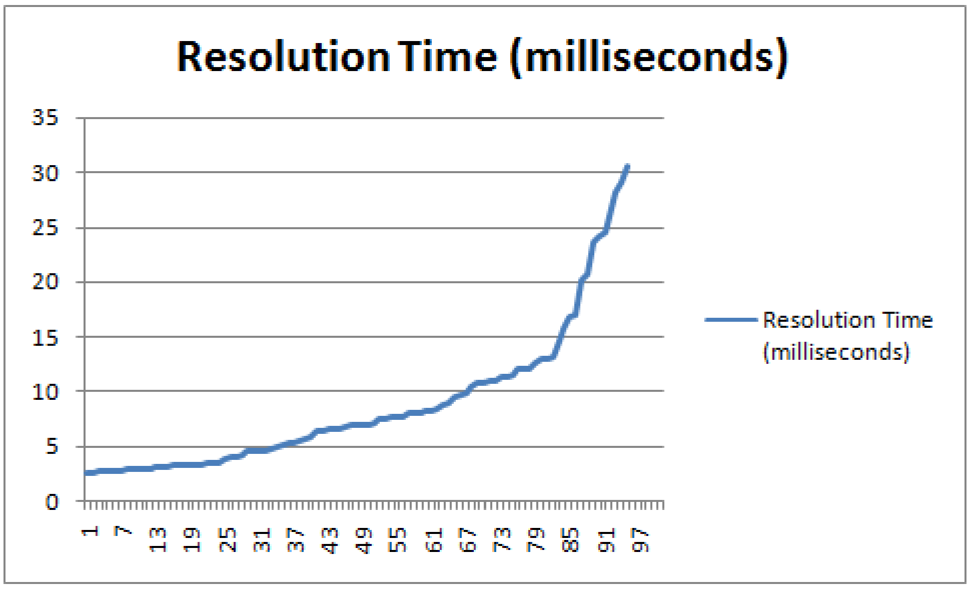

Figure 4.2: Sorted Resolution time

We can see from figure 4.2 the resolution time against puzzle indices sorted on resolution time.

We can see from this model that resolution time starts to increase exponentially after around 13 milliseconds. We can see that only a few problems show high amount of resolution time with all the other puzzles being closer.

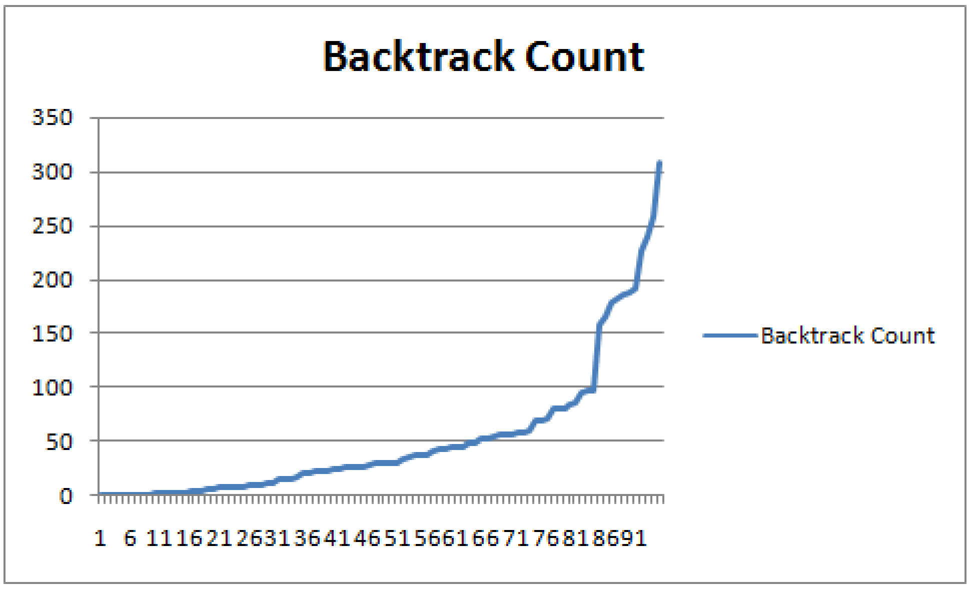

Figure 4.3: Sorted Backtrack Count

We see the same result in figure 4.3 where sorted puzzle indices are plotted against their backtracking time. We see that number of backtracks increases exponentially after it crosses 100 backtracking count. Whereas in the previous model backtrack increased exponentially near 75. Also we can notice that it has a bit sideways curve near 180 backtrack which was not present in the previous model. This model shows a more stable and relatively better result than the first one for the hard dataset.

4.1.2 Row-Column-Block Ordered Model:

The region-ordered solver had a mean solving time of 8.863 milliseconds with a standard deviation of 6.946 milliseconds. It solved all 95 hard puzzles that was in the first dataset used for testing.

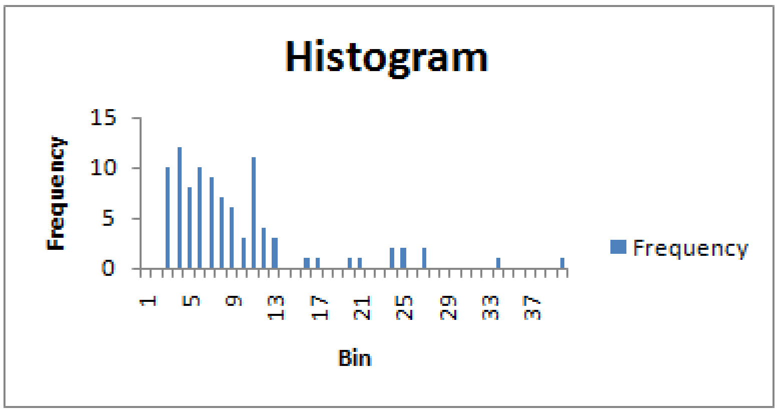

Figure 4.4: Histogram of Resolution time of Row-Column-Block Ordered model

Figure 4.4 is a histogram on a with the resolution time shown in milliseconds that shows how the region ordered model performed over all test puzzles. It is observable that there is a quite small time interval at which most puzzles are solved. This is observed from the histogram. Most of the solved puzzles resolution time lies within 3 to 13 milliseconds.

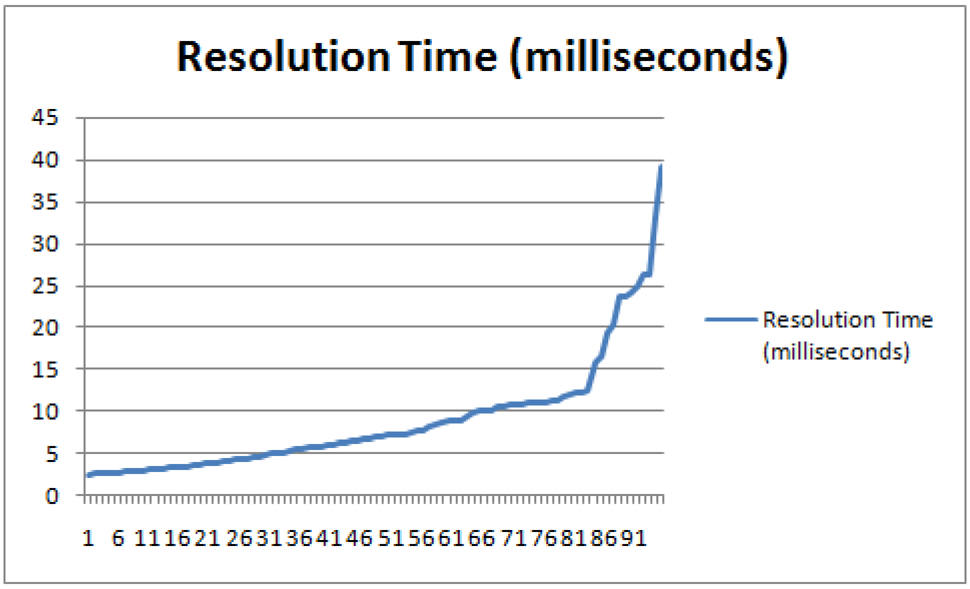

Figure 4.5: Sorted Resolution time of Row-Column-Block Ordered Model

Another way to visualize the result is shown in figure 4.5. The figure have plotted the puzzle indices sorted after solving time against their resolution times. Note that the y-axis is the solving time in milliseconds. As in figure 4.1, only a few puzzles had relatively high solving times. This picture also more clearly illustrates the idea explored above. Namely that the algorithm will increase its solving times fast at a certain point. From that statement, it can be concluded that only a small portion of all Sudoku puzzles are difficult, in the sense that the constraints that this model uses in this particular order is enough.

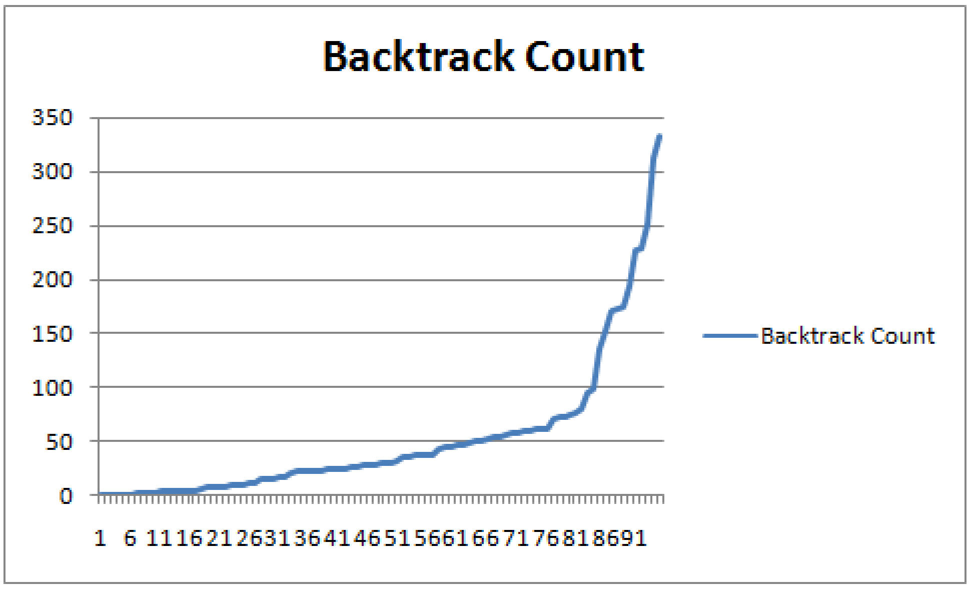

Figure 4.6: Sorted Backtrack Count of Row-Column-Block Ordered Model

In figure 4.6 the number of backtracks needed is shown against the puzzle indices after sorting them on the number of backtracks. The y-axis is the number of backtracks and the x-axis is the puzzle index. We can see from this graph that most of the puzzles take backtracks lower than 50. After the backtrack count crosses 75, number of backtracks rises exponentially for the last 10-15 puzzles. Which is consistent with our previous figures where the resolution time rises exponentially for 10-15 puzzles.

4.1.3 Count of Clue Ordered Model:

This model employs the sorted region on number of clues technique. It has mean resolution time 8.36 milliseconds and standard deviation of 6.445 milliseconds.

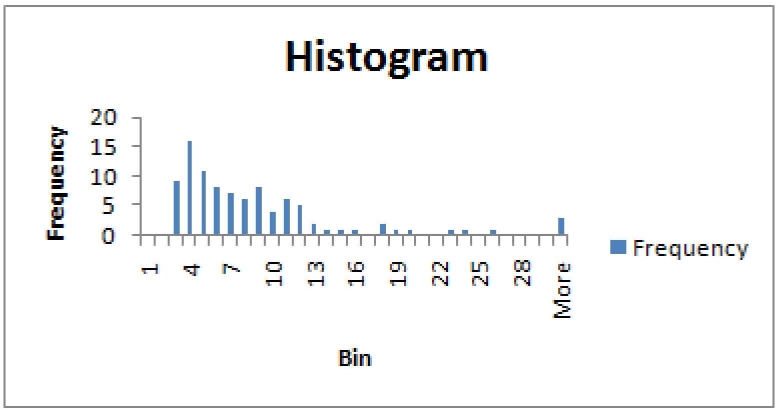

Figure 4.7: Histogram of Count of Clue Ordered Model

n figure 4.7 the histogram is shown. We can see most of the value lies within 12 milliseconds with the most puzzle being solved between the range of 3 to 4 milliseconds. This model hardly have very few puzzles being solved after 13 milliseconds. But the frequency suddenly rises a bit after 30 milliseconds.

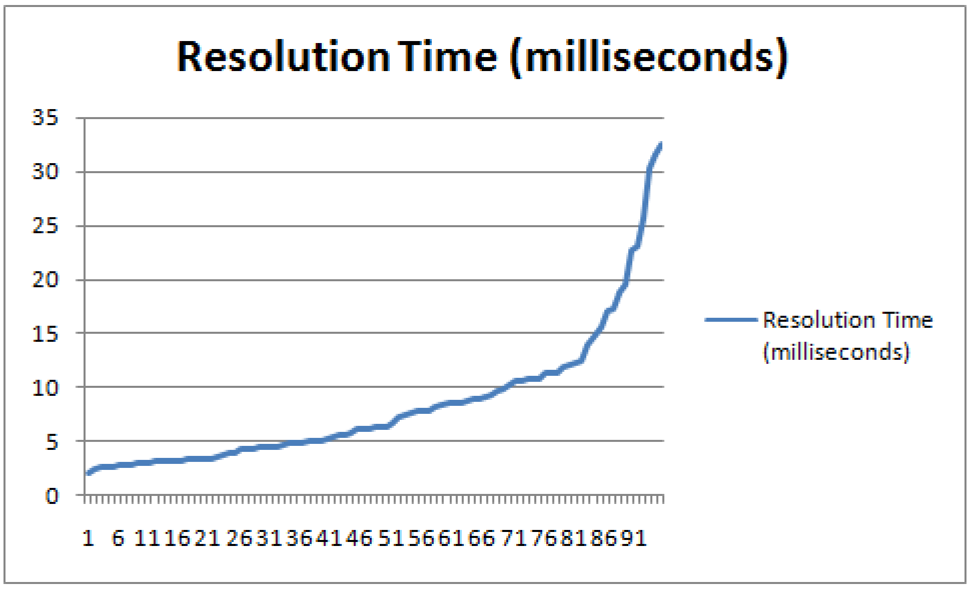

Figure 4.8: Sorted Resolution time of Count of Clue Ordered Model

From figure 4.8 where the sorted puzzle indices are plotted against the resolution time we see the resolution time rises exponentially after 12 milliseconds. Also we notice there is a slight curve at the top of the graph, which indicates the frequency rises a bit at the highest range.

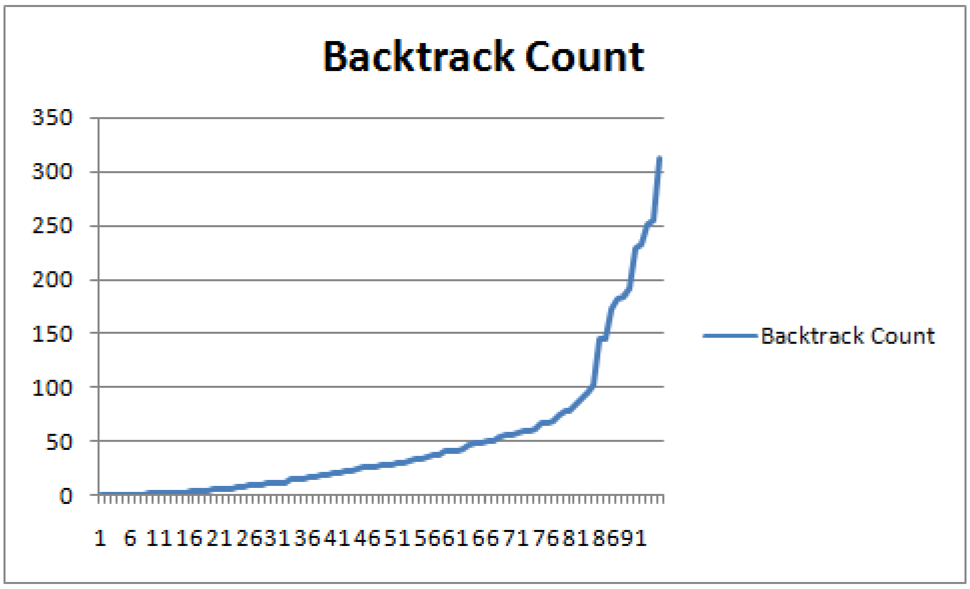

Figure 4.9: Sorted Backtrack Count of Count of Clue Ordered Model

From figure 4.9 we see the relation between backtrack count and puzzle indices. We see that it has a gradual increasing shape with less variance than the previous two. It start rising exponentially around 100 backtrack count and the highest number of backtracks are over 300.

4.1.4 All-Different-Count model:

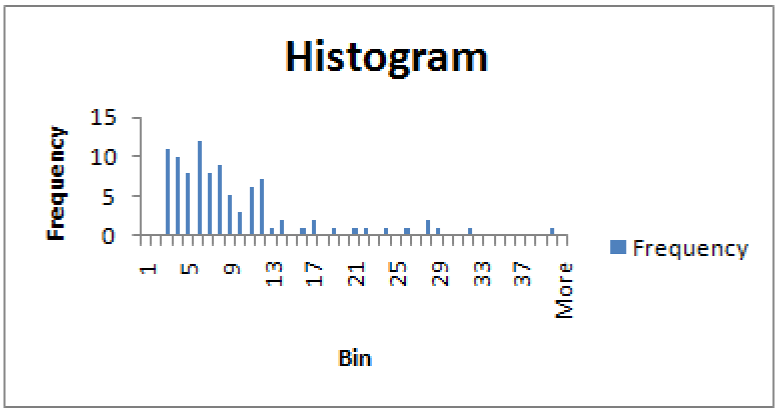

Figure 4.10: Histogram of All-Different-Count Model

For the model in which we implement additional count constraint alongside all-different, we can see that it has higher frequency of values within 12 milliseconds with the most being solved within the range of 4 to 5 milliseconds. Although it has fewer frequency, after that some of the problems it solves near 40 milliseconds. Which means it takes a bit more time for the harder problem than previous two models. It has a mean resolution time of 8.619 milliseconds and standard deviation of 6.407 milliseconds.

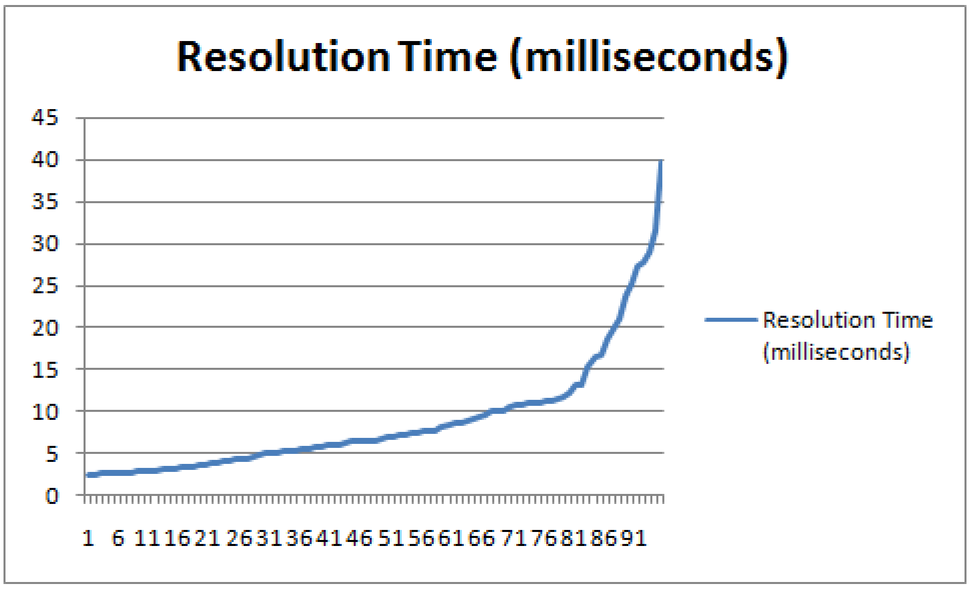

Figure 4.11: Sorted Resolution Time of All-Different-Count Model

If we analyze the resolution time in figure 4.11 we see that it has less variation while it rises towards 14 milliseconds and after that it rises exponentially. Also we can see it crosses the 5 milliseconds threshold near 31st index, which is better than the previous model which crosses the 5 milliseconds index near 36th index.

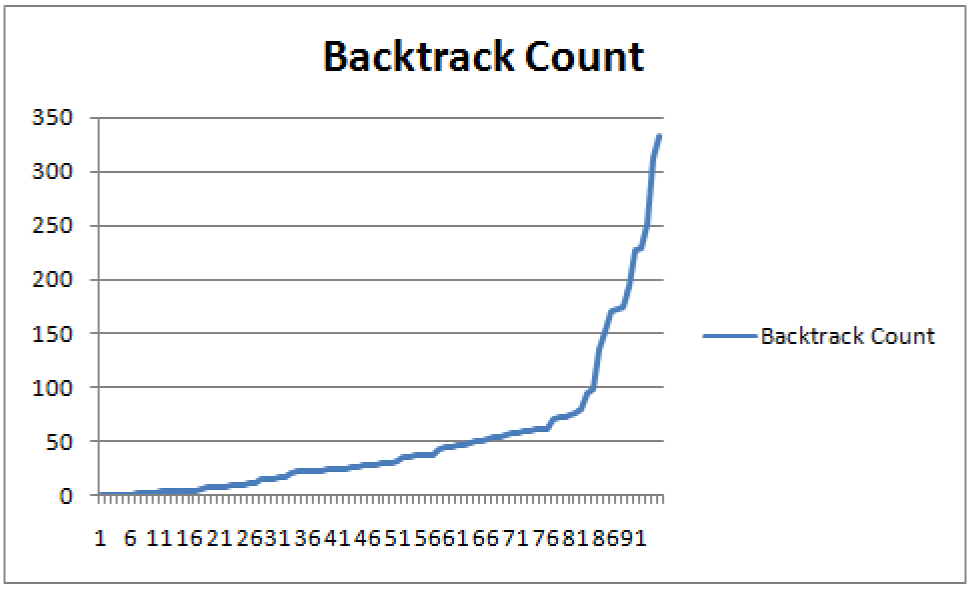

Figure 4.12: Sorted Backtrack Count of All-Different-Count Model

In figure 4.12 we see the backtrack graph and it shows that it starts to rise exponentially after 100 backtracks. And the variation is relatively small.

4.2 Time Distribution for Dataset 2

We take the same approach while performing the tests on dataset 2. We take the histogram, resolution time graph and backtrack count graph and analyze each dataset.

4.2.1 Region Ordered Model:

The row-column-block ordered solver had a mean solving time of 2.541 milliseconds. It had a standard deviation of 1.121 milliseconds.

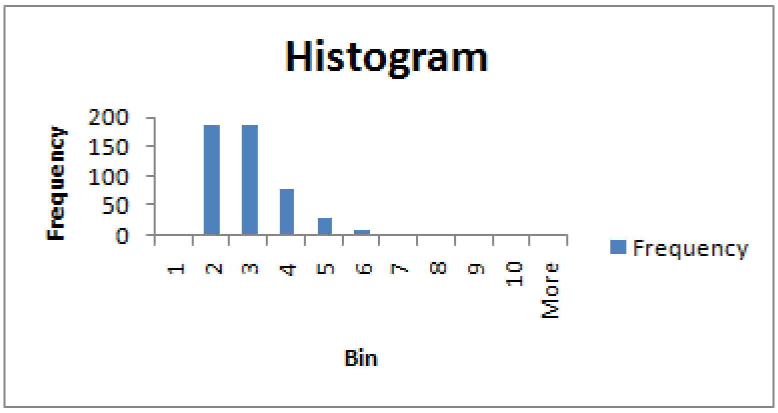

Figure 4.13: Histogram of Region Ordered Model

From figure 4.13 we can observe that most of the problems are solved within 2 and 3 milliseconds. The frequency gradually decreases as resolution time increases with very few puzzle after 6 seconds. The difference of this model with the previous one is that it has same number of puzzles solved in the 2 and 3 milliseconds range, whereas the previous one has more in the 2 milliseconds range in 3 milliseconds range.

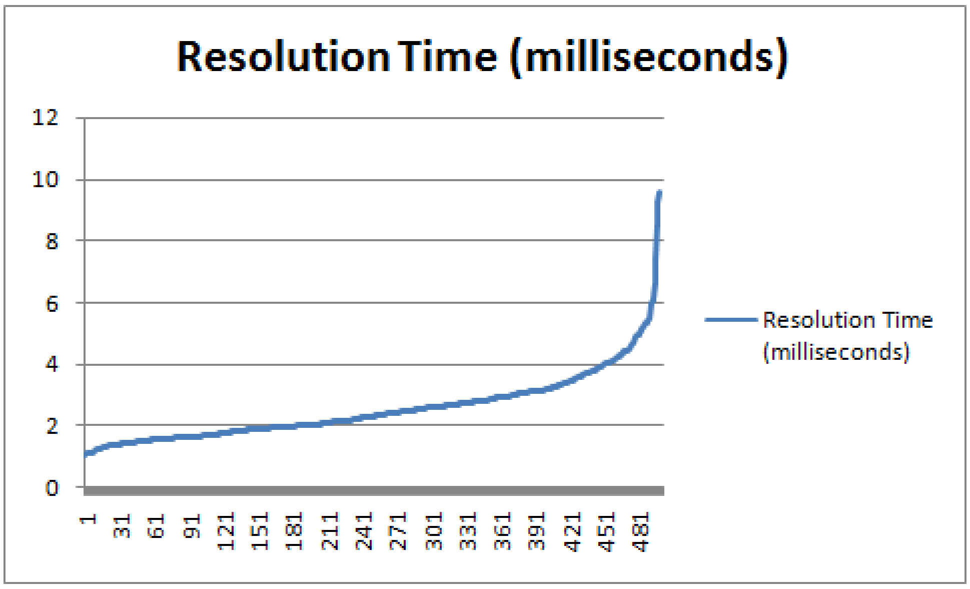

Figure 4.14: Sorted Resolution Time of Region Ordered Model

Figure 4.14 shows the result with the resolution time being plotted against the puzzle indices. We can see from the figure that close to 470 problems are solved under just 5 milliseconds. And the resolution time increases by small amount till that. But after 5 milliseconds it increases exponentially. We can see the difference with the previous model that the previous model solved the highest difficulty puzzle in close to 25 milliseconds, but this models most computationally expensive puzzle takes only about 9 millisecond to solve.

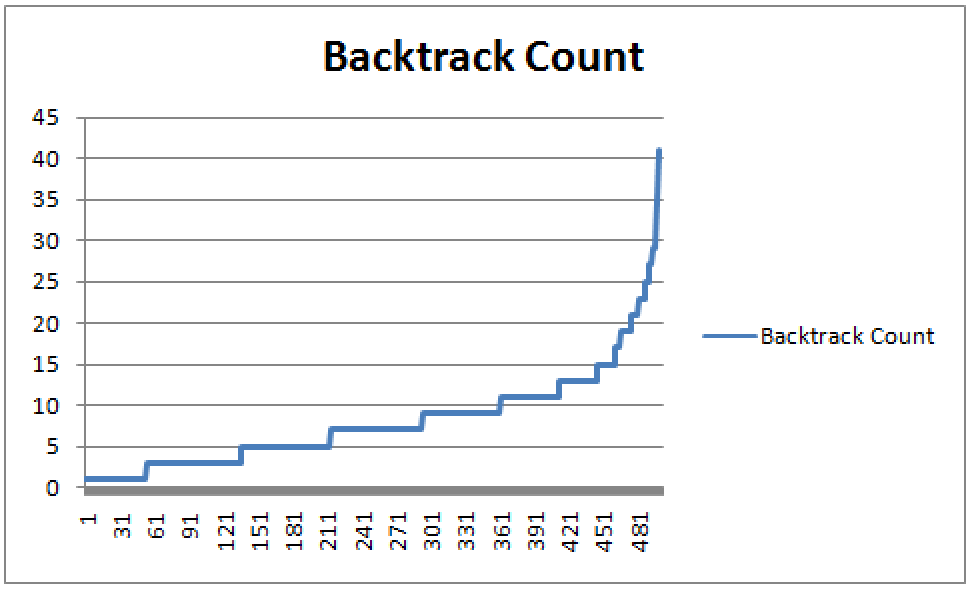

Figure 4.15: Sorted Backtrack Count of Region Ordered Model

In figure 4.15 the number of backtracks needed is shown against the puzzle indices after sorting them on the number of backtracks. We can see that the number of backtracks increases stepwise and with small steps until 15 milliseconds. We can see the highest number of backtrack is close to 40. For the previous model it was close to 45.

4.2.2 Row-Column-Block Ordered Model:

The region-ordered solver had a mean solving time of 2.560 milliseconds. It had a standard deviation of 1.583 milliseconds.

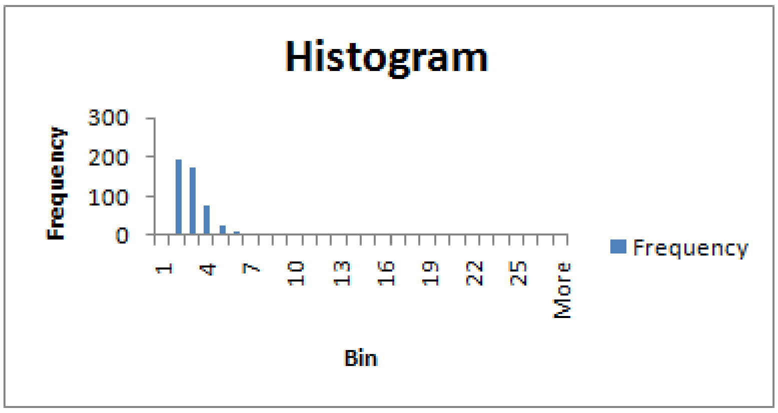

Figure 4.16: Histogram of Row-Column-Block Ordered Model

Figure 4.16 that shows how the region ordered model performed over all test puzzles. We can observe that most of the problems are solved within 2 and 3 milliseconds with the range

2 millisecond having the most puzzle. The frequency gradually decreases as resolution time increases with very few puzzle after 6 seconds.

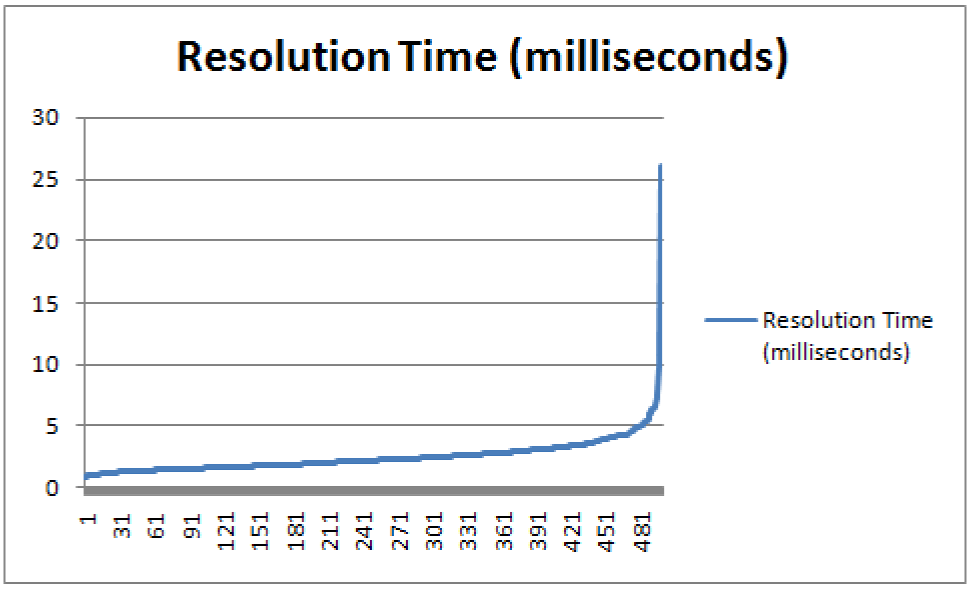

Figure 4.17: Sorted Resolution Time of Row-Column-Block Ordered Model

Figure 4.17 shows the result with the resolution time being plotted against the puzzle indices. We can see from the figure that close to 470 problems are solved under just 5 milliseconds. And the resolution time increases by small amount till that. But after 5 milliseconds it increases exponentially.

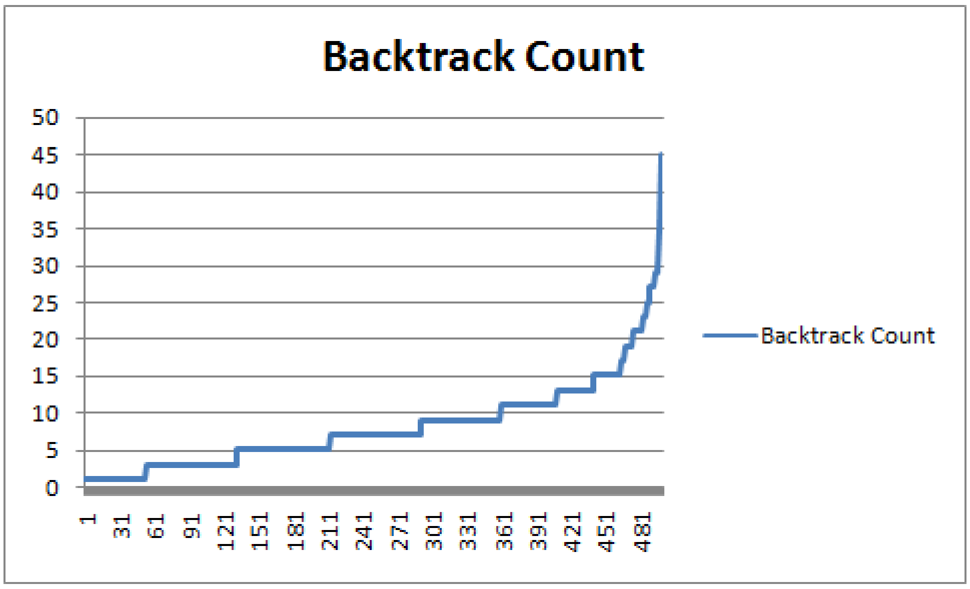

Figure 4.18: Sorted Backtrack Count of Row-Column-Block Ordered Model

In figure 4.18 the number of backtracks needed is shown against the puzzle indices after sorting them on the number of backtracks. We can see that the number of backtracks increases stepwise and with small steps until 15 milliseconds. The reason this count backtrack increases stepwise because the problems were generated by predetermining specific numbers of backtracks. We can see the highest number of backtrack is close to 45. And most of the puzzles take backtracks lower than 12.

4.2.3 Count of Clue Ordered Model:

This model employs the sorted region on number of clues technique. It has a mean solving time of 2.437 milliseconds. It had a standard deviation of .981 milliseconds.

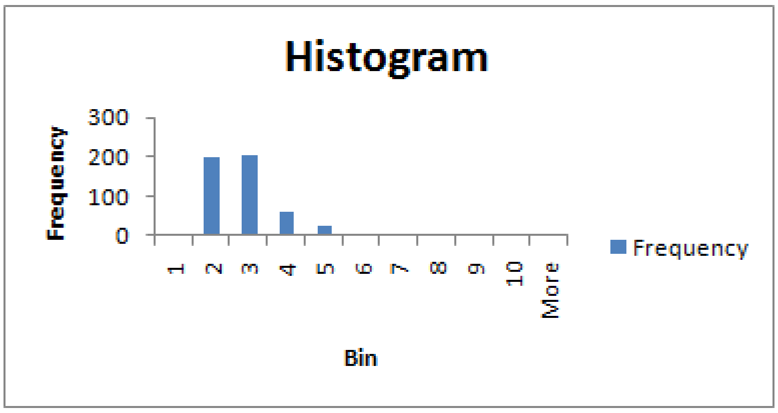

Figure 4.19: Histogram of Count of Clue Ordered Model

From figure 4.19 we can observe that most of the problems are solved within 2 and 3 milliseconds as before. Its difference with the previous models is that it solves more puzzle in 3 milliseconds than in 2 milliseconds.

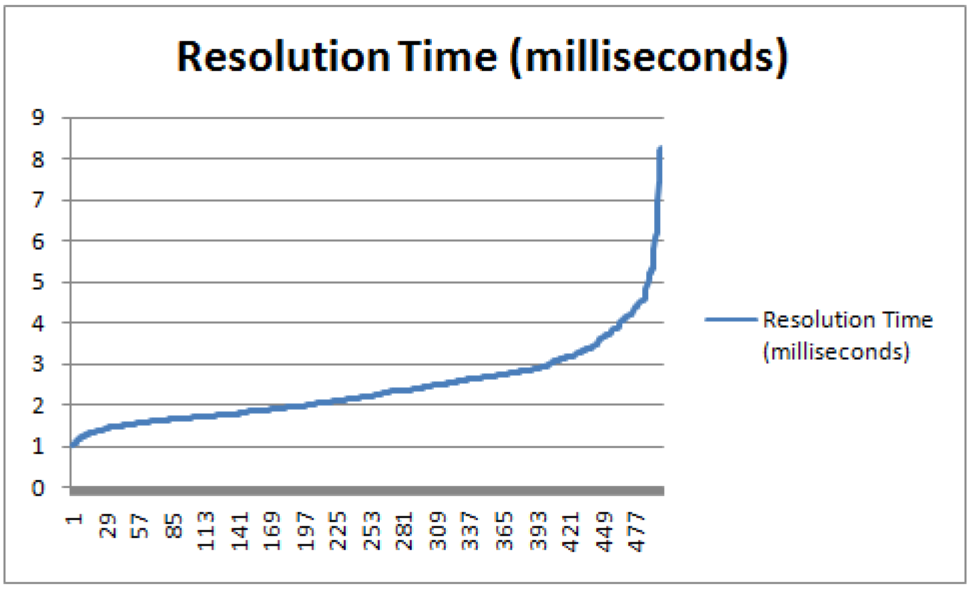

Figure 4.20: Sorted Resolution Time of Count of Clue Ordered Model

From figure 4.20 we can see that close to 393 puzzles are within 3 milliseconds threshold which is relatively better than the previous model. Also it solves the most computationally expensive problem close to 8 milliseconds, which is better than the previous models.

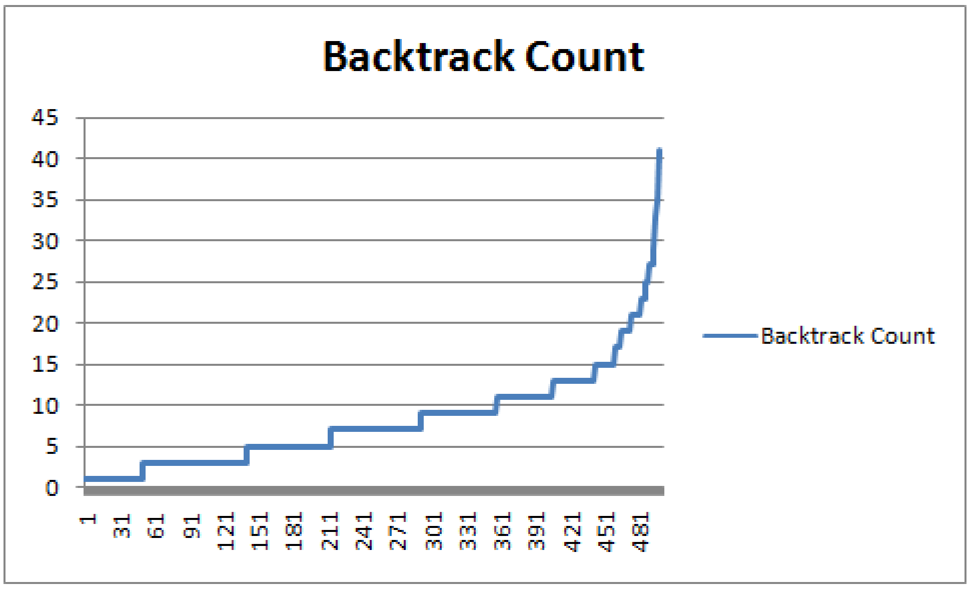

Figure 4.21: Sorted Backtrack Count of Count of Clue Ordered Model

In figure 4.21 the number of backtracks needed is shown against the puzzle indices after sorting them on the number of backtracks. We can see that the number of backtracks increases stepwise and with small steps until 15 milliseconds. We can see the highest number of backtrack is close to 40.

4.2.4 All-Different-Count model:

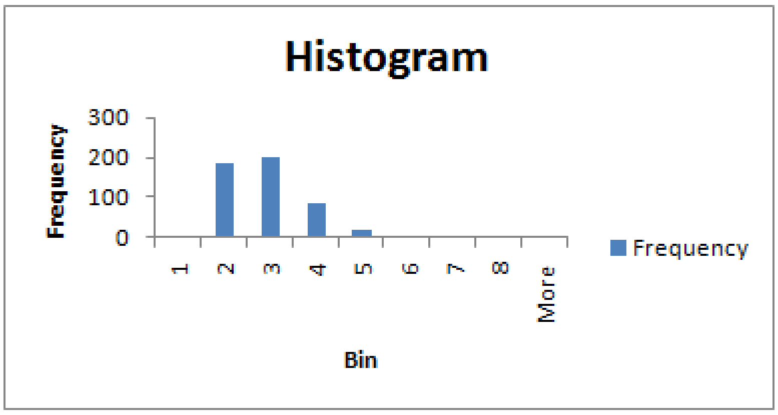

Figure 4.22: Histogram of All Different Count Model

From figure 4.22 we can observe that the proportion of problems solved within 3 milliseconds is more than the proportion of problems solved within 3 milliseconds, while other ranges contributing smaller amounts. Its mean resolution time is 2.469 milliseconds and standard deviation is .980 milliseconds.

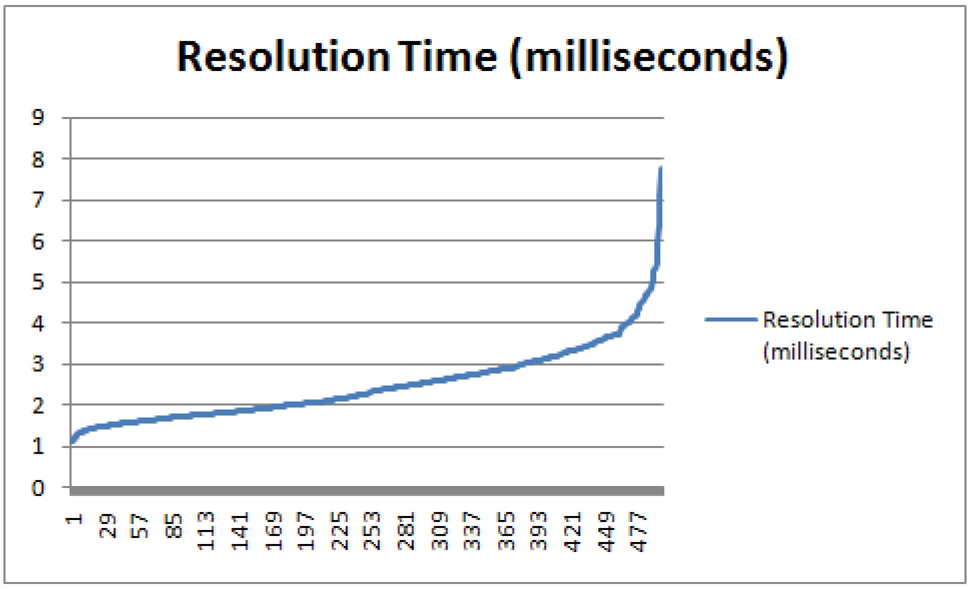

Figure 4.23: Resolution Time of All Different Count Model

From figure 4.23 we can see that it puzzles resolution time starts to increase exponentially within 5 milliseconds. But it solves the most computationally expensive problem under 8 milliseconds, which is the best among the models.

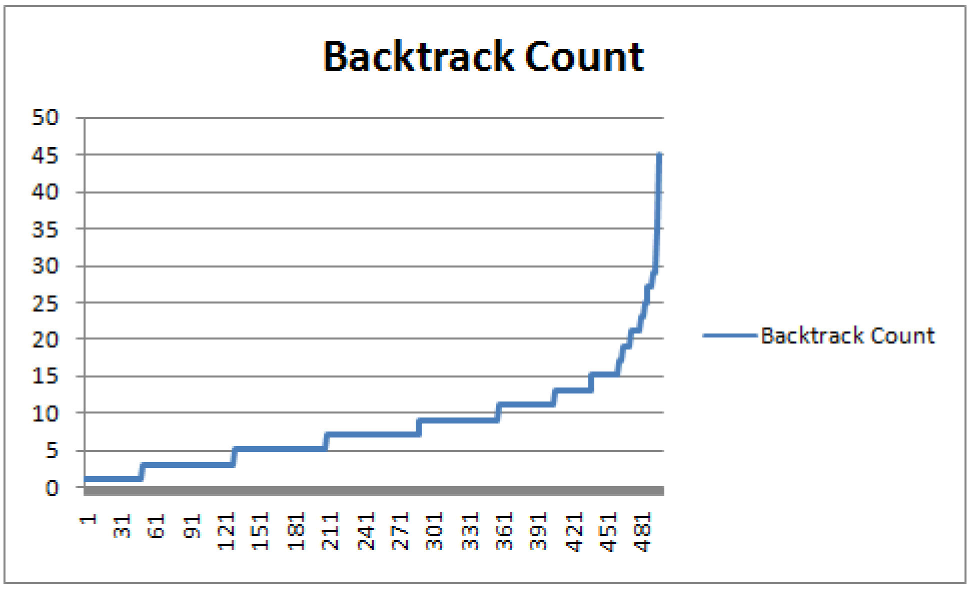

Figure 4.24: Backtrack Count of All Different Count Model

In figure 4.24 the number of backtracks needed is shown against the puzzle indices. It shows properties like the previous models.

4.3 Comparison between Models

4.3.1 95 Hard Puzzle Dataset

	Avg Resolution Time	Avg Backtrack Count	Std Dev of Time	Std Dev of Backtrack
Region Ordered Model	8.710	52.010	6.512	64.063
Row-Column-Block Ordered Mode	8.863	52.557	6.946	66.565
Count of Clue Ordered Model	8.360	50.515	6.945	65.152
All-Different-Count model	8.619	49.787	6.407	61.168

Table 4.1: Comparison between different models for hard dataset

From the table we can see that in terms of average resolution time, count-of-clue ordered model is the best with the lowest resolution time of 8.360 milliseconds. The all-different-count model that employs cont constraint along with all-different constraint is the second best. The region ordered model comes third and row-column-block ordered model comes last in terms of average resolution time. So we can see that ordering constraints in terms of count of clue makes for the best model in terms of average resolution time.

Now if we consider the standard deviation of resolution time we can see that model 4 has the lowest standard deviation of 6.407. Then comes model 3, then 1 and then 4.
If we think in terms of backtracking count, we can see that model 4 need the lowest number of average backtracks. Model 3 need lesser than the other two models. And model 2 needs the most. So we can say that adding count constraint with the all-different constraint reduces the number of backtracks best.

Standard deviation of backtracks is the lowest in model 4. Model 1 comes in second. Model 2 deviates the most.

4.3.2 5000 Puzzle Dataset:

	Avg Resolution Time	Avg Backtrack Count	Std Dev of Time	Std Dev of Backtrack
Region Ordered Model	2.54	8.012	1.122	5.941
Row-Column-Block Ordered Mode	2.56	8.030	1.583	5.993
Count of Clue Ordered Model	2.437	8.022	.980	6.038
All-Different-Count model	2.469	8.070	.900	5.980

Table 4.2: Comparison between different models for 5000 Easy Dataset

From this table we see that the average resolution time of model 3 is the lowest. As was the case for 95 hard puzzle dataset. Model 4 comes in second in terms of low average resolution time. Among the other two model 1 performs better than model 2.

The standard deviation of resolution time for this second dataset shows us that model 4 deviates the lowest. But the standard deviation of model 3 is pretty close. Among the other two models model 2 deviates more than model 1.

Now for the case of backtrack count we can see that model 4 needs the lowest number of backtracks in average. Model 1 whose average number of backtracks are 8.021 is second. But model 3 is almost equal with average number of backtracks being 8.22. Model 3 need the most average number of backtracks.

The standard deviation of backtrack is lowest in model 1 for this dataset. Model 4 has the second lowest standard deviation. Among model 2 and 3, model deviates less.

So from both of the dataset we can see that average resolution time, standard deviation of resolution time and average backtrack count is consistent among the datasets. For the case of standard deviation of backtrack count model 1 becomes better than model 4 in second dataset. Otherwise all other comparisons are consistent.

Chapter 5

Conclusion

All different is a common constraint for solving Sudoku puzzle. The common approach taken to solve Sudoku puzzle using all-different constraint is that it works on the row, column and blocks serially or it works on the regions serially. Which the cases for model 2 and 1 respectively. In this thesis we tried to take different approach other than these two. One approach was to count the number of clues in each region and sort the regions based on the counts in descending order. Then we imposed the constraint on the regions in sorted order. Another of the approach was to add another constraint with the all-different constraint and see if it works better or not.

From the analysis and comparison of the four models on two different datasets, we can see that these two approaches have positive result over the common approaches.

The count-clue-ordered model reduces the average resolution time the most. In terms of average resolution time both of the approaches, model 3 and model 4 shows reduction of average resolution time over model 1 and 2. Ordering the constraints so as to they work first on the regions with most clues seems to give the best result. The addition of count constraint with all-different model also gives better average resolution time than the other two model. So in terms of average resolution time we can say that those two approaches were better than the common ones.

While model 3 has the lowest average resolution time, model 4 deviates the least. And in this case too, the deviation of model 3 and 4 is less than model 1 and 2. So we can say that both clue-count-order and all-different-count model work better than model 1 and 2 in terms of average resolution time and standard deviation of resolution time.

In case of average number of backtracks, all four models are pretty close. Among these, model 4, the clue-count-order model has the least average number of backtracks. In this case the average number of backtracks for model 3 is slightly higher than model 4, but it still performs better than the other two models.

In case of standard deviation of backtracks model 4 is better in one dataset and model 1 is better in the other one. In both cases model 3 deviates more than model 1. So in terms of deviation model 3 and 4 does not bring any noticeable improvement.

In conclusion we can say that the ordering of constraints in count-clue-order model results in better average resolution time. And adding count constraint along with the all-different constraint gives us better average backtrack count. Both of these models give better result than row-column-block order and region-order models in terms of average resolution time and average backtrack count.

References

Appendix A

Algorithms

A.1 Sample Algorithm

In Algorithm 1 we show how to calcute $y = x^n$.

Algorithm 1 Calculate $y = x^n$

```
Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  if $n < 0$ then
    $X \leftarrow 1/x$
    $N \leftarrow -n$
  else
    $X \leftarrow x$
    $N \leftarrow n$
  end if
  while $N \neq 0$ do
    if $N$ is even then
      $X \leftarrow X \times X$
      $N \leftarrow N/2$
    else $\{N$ is odd$\}$
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while
```

Appendix B

Codes

B.1 Sample Code

We use this code to find out…

```c
#include <stdio.h>
int Fibonacci(int);

main()
{
  int n, i = 0, c;

  printf("Enter␣the␣value␣of␣n:␣");
  scanf("%d",&n);

  printf("\nFibonacci␣series\n");

  for (c = 1 ; c <= n ; c++)
    {
      printf("%d\n", Fibonacci(i));
      i++;
    }

  return 0;
}

int Fibonacci(int n)
{
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return (Fibonacci(n-1) + Fibonacci(n-2));
}
```

B.2 Another Sample Code

```
SELECT associations2.object_id, associations2.term_id,
       associations2.cat_ID, associations2.term_taxonomy_id
FROM (SELECT objects_tags.object_id, objects_tags.term_id,
     wp_cb_tags2cats.cat_ID, categories.term_taxonomy_id
FROM (SELECT wp_term_relationships.object_id,
     wp_term_taxonomy.term_id, wp_term_taxonomy.term_taxonomy_id
FROM wp_term_relationships
LEFT JOIN wp_term_taxonomy ON
     wp_term_relationships.term_taxonomy_id =
     wp_term_taxonomy.term_taxonomy_id
ORDER BY object_id ASC, term_id ASC)
AS objects_tags
LEFT JOIN wp_cb_tags2cats ON objects_tags.term_id =
     wp_cb_tags2cats.tag_ID
LEFT JOIN (SELECT wp_term_relationships.object_id,
     wp_term_taxonomy.term_id as cat_ID,
     wp_term_taxonomy.term_taxonomy_id
FROM wp_term_relationships
LEFT JOIN wp_term_taxonomy ON
     wp_term_relationships.term_taxonomy_id =
     wp_term_taxonomy.term_taxonomy_id
WHERE wp_term_taxonomy.taxonomy = 'category'
GROUP BY object_id, cat_ID, term_taxonomy_id
ORDER BY object_id, cat_ID, term_taxonomy_id)
AS categories on wp_cb_tags2cats.cat_ID = categories.term_id
WHERE objects_tags.term_id = wp_cb_tags2cats.tag_ID
GROUP BY object_id, term_id, cat_ID, term_taxonomy_id
ORDER BY object_id ASC, term_id ASC, cat_ID ASC)
AS associations2
LEFT JOIN categories ON associations2.object_id =
     categories.object_id
WHERE associations2.cat_ID <> categories.cat_ID
GROUP BY object_id, term_id, cat_ID, term_taxonomy_id
ORDER BY object_id, term_id, cat_ID, term_taxonomy_id
```

Generated using Undegraduate Thesis LaTeX Template, Version 1.3. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 23rd January, 2021 at 7:30am.