Sudoku Solver

October 31, 2016

Advanced Analysis and Algorithms COMS3005

Report

Completed by:
Jared Naidoo - 719238
Krupa Prag - 782681
Zayyan Variawa - 852486

Abstract

A formal statement on the results of the investigation of the standard Sudoku puzzle problem, solved by the implementation of the backtracking algorithm, is contained in this report. A partially completed puzzle grid is the given input to the algorithm which outputs a unique filled grid. This report focuses on the theoretical versus experimental analysis of the backtracking algorithm on a standard Sudoku puzzle. We will further discuss the validity of our hypothesis claim, based on theoretical analysis, with the results obtained from the experimental analysis. The results achieved will illustrate various factors that will affect the algorithms performance, hence making comparisons in order to achieve an algorithm which is optimal for solving a Sudoku puzzle.

Contents

1 Introduction **5**

1.1 The Problem 5

1.2 Background and History 5

1.3 Aim 6

1.4 Objective 6

2 Analysis and Algorithm **7**

2.1 Problem analysis 7

2.2 Backtracking Algorithm 8

3 Theoretical Analysis **9**

3.1 Complexity 9

3.1.1 Best and worst cases 10

3.2 Time complexity 10

3.3 Optimality 13

3.4 Clarity and Completeness 13

4 Experimental Methodology **14**

4.1 Implementation 14

5 Implementation: Trial **15**

6 Results **15**

6.1 Presentation of results 15

6.2 Interpretation of results 15

7 Theoretical analysis relation to experimental analysis **16**

8 Conclusion **16**

9 Glossary **16**

10 Appendix **18**

10.1 Data set 18

10.1.1 Sample input 18

10.2 Results 19

10.2.1 Sample :Input board, resulting output board and time taken to solve 19

11 References 19

12 Acknowledgments 20

13 Group contribution 21

1 Introduction

1.1 The Problem

A Sudoku puzzle is completed on a polymino board or main grid of $n^2 \times n^2$ cells. The board is further outlined, demarcating $n \times n$ sub-grids. The order of the puzzle is n.
In particular, we are doing an investigation of a standard Sudoku puzzle, which is a puzzle of 81 cells in total. The order is 3, as $n = 3$, hence we get a main grid of size $3^2 \times 3^2$, with 9 sub-grids, each of size 3×3 .
The goal of this puzzle is to fill the grid with integers of the values 1 to n^2, in this case to 9, with the restriction of a single digit only appearing once in a particular row, column and sub-grid. The standard Sudoku puzzle comes partially filled, leaving the player to complete the grid whilst adhering to the restrictions placed. We note that the cells which are filled with a value is called a *clue*. A valid solution can we obtained using the clues and the definition of a Sudoku puzzle.

1.2 Background and History

Number Place - is the original name for a Sudoku Puzzle. A Sudoku Puzzle is a logic based puzzle filled with various combinations of numbers. The objective of the puzzle is to have nine 3×3 grid filled with the numbers 1 to 9, without repetition in the sub grid, it's row and it's respective column of position, outputting a unique solution. This particular number puzzle had made it's first appearance in the late 19th century's French newspapers, when puzzle setters begun amending and manipulating numbers in the magic squares.
The roots of a Sudoku Puzzle is the Magic Square, which is particularly seen as a powerful means of representing combinatoral numbers in various respects of science. Great ancient Indian mathematician, Varahamihira, used a fourth-order magic square to specify recipes for making perfumes. Even as early as (ca. 900 A.D.), in Siddhayoga, displays the oldest dated third-ordered magic square, representing Vrnda's medical work on means to ease childbirth.

Today the Sudoku-puzzle is very popular on PC's, websites and mobile phones. Many people solve these puzzles as a pass time activity.

1.3 Aim

The scenario at hand is to implement the backtracking algorithm to solve a Sudoku Puzzle. The aim of this assignment is to gain insight into the concept by means of measuring performance of the algorithm and relating these measurements to the theoretical analysis of the algorithm, through which we explore the nature of Computer Science.

1.4 Objective

The objective is achieving the aim of this assignment, by the implemention of the backtracking algorithm in order to solve a partially filled Sudoku Puzzle grid such that it produces a unique solution.
Using the results of the implementation, from our experience and theoretical knowledge, we can give some insight on the performance of the algorithm that is executed. As a group, it is our objective to fulfill the aim of this project by performing theoretical analysis which will be further compared to the experimental results, and used to substantiate our theoretical conclusions, as well as, to obtain the empirical analysis of the Backtracking Algorithm. As stipulated in the project brief, we have been given free rein to choose a programming language in order to implement the algorithm. The algorithm's performance will be tested on a database of standard Sudoku puzzles which will have different input configurations.

In the field of computer science, a general notion of resolving the algorithm's variables and mapping it, is one that has great emphasis placed on it. Furthermore, from our theoretical analysis, we use this concept to establish whether the solution is feasible or not. If the solution is feasible, we investigate to some particular benchmark to where the algorithm can be implemented such that it is successfully utilized or to the point before it becomes exhaustive or demanding on resources available.

An empirical analysis will be deduced of the Backtracking algorithm with a various number of of initial starting clues. Our report will be based on only the solutions which are unique, hence streamlining our domain on which we are viewing the Backtracking algorithm. The reason for this is that the number of solutions that can be produced using the algorithm applied is not the perceptive from which the measurement of performance is desired to be taken from, however, the measurement of performance is solely based on the efficiency of the algorithm solving a Sudoku puzzle.

The procedure that we will follow :

1. Write pseudo-code which is appropriate for solving the Sudoku standard sized puzzle, incorporating the Backtracking algorithm.
2. Analysis of the pseudo-code and further research to gain the complexity
3. Implement the algorithm as Java code
4. Implement the algorithm to generate a set of results
5. Comparison of theoretical and experimental results
6. Graphical interpretation of obtained results, which represent the time required to solve the puzzle under various initial starting configurations, giving a trend as the composition of all the results on which we will base our analysis

2 Analysis and Algorithm

This section will consider the backtracking algorithm and the fundamentals of the Sudoku puzzle. We will explore the essence of various research articles, and literature in aim to gain a concrete understanding of the puzzle and the algorithm.

2.1 Problem analysis

The Sudoku puzzle could be solved by the Naive Algorithm which uses the brute-force technique. Brute-force technique is a trial and error based method, which entails proceeding through all possible combinations that could fulfill the puzzle. This primitive method - brute force, approaches the problem with the objective of filling up all candidate tiles in a random manner using integers from 1 to 9. The stopping condition in this algorithm is when a valid solution is obtained; when the Sudoku-puzzle rules are all satisfied.

This approach is seen as infallible and extremely time-consuming. As each configuration possible is executed until the correct one is found.
Sudoku by hand derives from the discovery and mastery of a myriad of subtle combinations and patterns that provide hints about the final solution. This exhaustive method could be replaced by employing intellectual strategies which could be executed by a computer program.

Computer programs use a different Sudoku-solving technique. Relying on their almost limitless capacity to solve a Sudoku puzzle, the backtracking algorithm is seen as an efficient means to solve the Sudoku puzzle.

2.2 Backtracking Algorithm

The backtracking algorithm is a powerful means of solving a constraint satisfaction problems. This manner is more effecient to that of the brute force technique as uses less memory. The algorithm can be seen as permutations which are being applied, and if a certain pattern doesn't match, we find another, until a certain pattern matches. Essentially, in an incremental fashion, the algorithm, builds candidates to the solutions, and discards each partial candidate ('backtracks') as soon as it is noted that this specific candidate cannot possibly be inserted in obtaining a valid solution. Backtracking requires recursion which can be something worse, because CPU stack space is limited and can be consumed quickly by recursion.

An overview of the algorithm:
This algorithm follows the procedure of scanning through the Sudoku-board, filling in blank cells with a valid number (ie. adhering to the Sudoku-puzzle rules: no two same numbers in any row, column or 3×3 box). This process continues to the next empty cell, filling up empty cells. When a particular cell is reached where no valid input is available (where all possible numbers from 1 to 9 cannot be placed, as Sudoku-puzzle rules are not satisfied), the algorithm moves back to the previous cell and makes amendments to that cell's value to another valid number. The procedure then continues, it moves back to the next cell and the entire process repeats till the entire board is filled with valid inputs which adhere to the contingencies.

A more structured format of the algorithm:

Starting from the first empty cell top left corner, iterating through all the columns of the row, the following steps are followed recursively.

1. If the cell is empty, add a number that is not constrained.
    - **If** it is impossible to insert a number which adheres to the Sudoku puzzle definition, report failure
    - **Else** Start a new thread on a new cell, starting from Step 2.
2. - **If** this thread reports failure, repeat Step 1, with a different number and exhaust the old number
    - **If** this thread reports success, we report success, as we make the assumption that the last cell has been successfully complete.
3. **If** the cell is filled, then go to the next empty cell
4. **If** the algorithm tries accessing ground which is out of bound, then we know we have reached success

3 Theoretical Analysis

The definition of the Sudoku puzzle must be abide by at all times ie. every cell must be filled with a unique integer along it's particular row, column and sub-grid. The backtracking algorithm ensures that if a dead-end is reached (at curr-cell), the the algorithm is executed - undoing the move and returning to the previously filled cell (prev-cell) making a possible change to prev-cell such that it would allow the next cell -curr-cell to be awarded a value that follows the puzzle's definition. Following this procedure in a recurcive manner.

3.1 Complexity

In general, the complexity of the algorithm is dependent on the number of clues given at the initial configuration.
The following cases are explored:

3.1.1 Best and worst cases

Best case:
No need to implement the backtracking algorithm, hence the path chosen from the first empty cell to the last empty cell in the grid is correct.

Worst case:
The backtracking algorithm is executed at every step. Every cell which is reached does not satisfy the Sudoku puzzle conditions, hence resulting in the backtracking algorithm being called at each cell in order to reach a final valid configuration.

3.2 Time complexity

Let k denote the number of candidate cells at the initial configuration.(Indicating the level of completeness and difficulty).

Levels

Level	**Range of clue cells**	**Upper bound of empty cells**
Easy	35-37	46
Medium	26-31	55
Hard	17-26	64

Let $n =$ number of candidate cells.
Base case: $n = 2$
We have a Sudoku grid that has two candidate cells : cell A and cell B. Cell A having a set $S_1 = a, b$ ie. Cell A has two possible options to be filled with. The first choice is taken - filled with a.
CASE 1: Then we go to cell B. Cell B has one option available in it's set $S = b$. Cell B is filled with b. The grid is then filled according to the definition and the the solution is produced with the backtracking.
CASE 2: Then we go to cell B. Cell B has no options available in it's set $S_2 = \emptyset$. The backtracking algorithm is implemented. The pointer now points to the previously filled cell : cell A. Looking at Cell A's set $S_2 = a, b$, we now assign a new numeral to the cell from the set. Assign cell A with b.

The algorithm goes to cell B which now has a non-empty set of options for this candidate cell. The cell B is now filled, completing the Sudoku puzzle according to the definition.
Inductive Hypothesis: $n = k - 1, k <= 64$
We have a Sudoku grid that has $k - 1$ candidate cells. Each of the candidate cells having at maximum 9 items in their set of possible values. We assume that the grid is filled using the backtracking algorithm.
CASE 3: $k - 1$ candidate cells filled without backtracking.
CASE 4: $k - 1$ candidate cells, some cells filled using the backtracking algorithm, while the rest are filled without using the backtracking algorithm.
CASE 5: $k - 1$ candidate cells where each cell encountered, the backtracking algorithm has to be executed to fill all the cells.
Inductive Step: $n = k, k <= 64$
We have a Sudoku grid that has k candidate cells. Each of the candidate cells having at maximum 9 items in their set of possible values. We assume that the $k - 1$ is filled using the respective cases in the inductive hypothesis.
CASE 6: From the inductive hypothesis CASE 3, $k - 1$ candidate cells filled without an backtracking, taking constant time. The k^{th} cell also having a single option which we choose, not having to back-track.
CASE 7: Following from CASE 4 of the Inductive hypothesis, of the $k - 1$ candidate cells some cells filled using the backtracking algorithm, while the rest are filled without using the backtracking algorithm. The k^{th} cell can be either filled using back tracking or could have an option that can be chosen resulting in a valid solution.
CASE 8: Following from CASE 5 of the Inductive hypothesis, $k - 1$ candidate cells are filled using the backtracking algorithm as each cell. The k^{th} cell has no possible options, resulting in backtracking to tecursively roll-bacl till the very first initial candidate cell to be amended either resulting in a valid solution or in a dead-end after exhaustive execution, hence not able to obtain a solution that adheres to the Sudoku puzzle definition.

Best case:
The first input to a void cell does not infringe the stipulation of the puzzle. This statement holds true for all empty cells in the grid. Hence, resulting in the algorithm never having to backtrack - regardless of the initial configuration of the grid. Therefore since it always produces a valid digit on

the first encounter with the cell, it will never have to backtrack to previous nodes in the tree of 'to-fill nodes'. Thus the order is constant, such that $n = O(1^k)$, where k is the upper bound being $k = 64$. However, $n = 1^{64}$ is still 1.
Hence, the complexity of the best case is $O(1)$, which means it takes constant time and produces a straight line function.

Average case:
An acyclic tree is created as an analogy for our valid output. We assign an arbitrary root node 'A'. For each empty cell, there are 9 possibilities (numerals from 1 to 9). We add 9 nodes to the tree where each node represents a eligible digit for the first cell. We then connect these nodes to the root node ny adding 9 edges. We then consider the second candidate cell. We again denote the 9 possible inputs and add another layer to the tree. We continue in this manner, which leads to k levels of the tree, which excludes the root node. This tree now represents each possible solution that could be used for the puzzle as produced in the light of brute force technique. The backtracking algorithm will follow a similar approach to that of a depth first search. It goes as far as possible along a branch of the tree, stopping at the point it violates any of the contingencies. At this point, we backtrack and try and find another branch. The complexity of a depth-first-search is $O(n + m)$, where n = number of nodes and m = the number of edges. In this instance, $n = 9^k$ and k is the number of candidate cells.

Worst Case:
Similar to that of the average case, in this scenario, we backtrack at every encounter with a candidate cell, and making the maximum number of this recursive process in order to output a valid solution to the puzzle, or to have exhausted all means to find a solution resulting in the conclusion that no solution is possible for the particular puzzle. In the worst case, we would have the least number of clues; 17 clues at the initial configuration, hence, the grid having 64 candidate cells ($k = 81 - 17 = 64$). Thus we are certain that our k can have a maximum value of 64. In the absolute worst case, we will have a order $O(9^{64})$, where $n = 9$ and $k = 64$. We can further claim that this is the upper bound for our worst case which has a general exponential trend.

3.3 Optimality

Viewing various reports completed on the topic of the solving of a standard Sudoku puzzle, we have compared various manners in which the puzzle has been solved. Among these methods were : human intuition of solving, genetic programming combined with heuristic moves - these strategies implemented the techniques such as naked single method, hidden single method. Another approach to solving the puzzle investigated in *Aref-Fiorella-KexJobb-sist report* this was the pencil-paper-method. This method entails several methods which are used by human players in solving the puzzle, which are difficult to apply to a computer program, as a human player having a greater overall view of the Sudoku grid. Human solving includes the usage of a combination techniques namely: Unique missing candidate, Naked single method and backtracking. From the results in this particular report, we note that the pencil-paper-method performs better than the brute force method. Furthermore, when we compare this method to the backtracking method, there is a point at which they co-inside, however for lower level puzzles, the pencil-paper-method still dominates in favour of a better time complexity.

From *Patrik-Berggren-David-Nilsson's-report* a method that was implemented to solve the puzzle is named rule-based algorithm. This algorithm's performance was seen as the best, performing better that the backtracking solver. The backtracking solver and the rule-based algorithm are both deterministic, hence reducing a depiction of the execution time as a distribution which is precise with a low variance. This claim indicates that although the backtracking method is feasible, it is not optimal as the rule-based solver performs better than that of the backtracking solver in the respect of performance measure. We thus note that there is room for improvement to the backtracking approach for solving the udoku puzzle.

3.4 Clarity and Completeness

As a group, we feel that the backtracking algorithm is not the optimal manner in which to go about solving a Sudoku puzzle. Theoretically the puzzle can be solved in this recursive manner. The concept can be followed through, thus we can claim it is a fairly clear algorithm, although not seen as efficient in terms of performance measure.

4 Experimental Methodology

As a joint team effort, we worked toward producing theoretical as well as experimental analysis of the standard Sudoku Puzzle. We have implemented the algorithm using the eclipse IDE, writing in the Java programming language. A database of test samples (of varying difficulty and initial configurations) were produced, which had been stored and solved by our algorithm. The results obtained were used to draw conclusions in this report. The results depict the time used to solve the puzzle with our implemented algorithm in milliseconds for each grid. The experiment was carried out ????????? number of times to obtain a trend to classify our findings as. A graph representing the results of number of candidate cells as the independent variable, and time measured in milliseconds as the dependent variable.
With the aid of the following hardware and software we were able to complete this experiment. The following items were used :

- Windows 10
- Lenovo
- RAM
- Eclipse IDE Java

4.1 Implementation

Noting the power that the Backtracking Algorithm posses, we have implemented a version of the Backtracking algorithm in our algorithm which solves a Sudoku-puzzle.
The Sudoku-solver algorithm starts off with a data set which is a Sudoku-grid with a set number of initial values entered.The Sudoku-solver starts scanning the Sudoku-grid from the very first empty cell from the top left-hand corner. The solver views all the values that have already been entered in that cell's particular row, column and it's sub-grid. If there is a value that is then applicable for that cell, it takes the first value of the possible list of values applicable for the cell, and fills it in that cell. The solver keeps scanning the Sudoku-grid in an iterative manner, viewing empty cells row-by-row. If at any particular point the solver notes that a cell - let's call this cell B, has only one possible value that it can be filled with, but that value was previously

allocated to another cell (not a initial value in that cell's particular row or column)- let's call this cell A, then the solver 'backtracks' - meaning, that the solver undoes all previous steps taken to the point where cell A is allocated a value. Then, cell A is allocated another value from the list possible values it could take on, and the Sudoku-solver starts its process from this point, with the aim to satisfy cell B with an eligible input. The solver recursively carries out this process until obtaining it's goal of solving the Sudoku-puzzle.

5 Implementation: Trial

Searches horizontally, then vertically then sub-grid. Best case: solved matrix Worst case: at each iteration to back track. from second potential tile.

Explanation:

Rule 1: The values that are eligible for the matrix are integers from 1 to 9.
Rule 2: in each sub-grid, there are multiple possible values which would be acceptable, looking at the other sub-grids along that sub-grid's row and column, we choose a value that is not already used in any of these sub-grid's.
Rule 3: Similar to that of rule 2. We just apply the same concept to each row and column.

6 Results

6.1 Presentation of results

6.2 Interpretation of results

As a result of performing analysis and experiments, we have gained insight into the backtracking algorithm. The complexity of the backtracking algorithm is illustrated in the graphs above, which depict that the time complexity is in some relation proportional to the number of candidate cells with which the puzzle is initialized.
From the graphical representations and analysis performed we see the exponential base instinct of the backtracking algorithm. As the number of empty tiles increase, so does the time taken to solve the puzzle.

We can deduce from the results that the algorithm utilized is not efficient i scenarios with large sets of data. The result of this is that it is demanding on time when the number of tiles to be filled on the board increase. However, for much easier leveled puzzles, the algorithm performs in an efficient manner. This can be further substantiated by the graphs produced. The complexity of this case has a linear nature, which tends toward a quadratic base nature.
In particularly more difficult puzzles, the results resemble the trends of exponential graphs. As the number of candidate cells increase, the time taken to solve the puzzle increases in an exponential manner. We can support this claim by mere interpretation that the algorithm having to perform backtracking at each step, resulting in performing worse than that of the best case which entails no roll backs.

7 Theoretical analysis relation to experimental analysis

8 Conclusion

In conclusion, we can state that comparing the Brute Force technique to the Backtracking algorithm, we have reduced that the backtracking technique is comparatively more efficient then to that of the Brute Force Approach, as the running time is less as a result of not exploring every possible path. The Sudoku puzzle will thus be the best at finding the path taking to satisfy the definition of the puzzle.

9 Glossary

- Algorithm: a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
- Sudoku Puzzle : a puzzle in which players insert the numbers one to nine into a grid consisting of nine squares subdivided into a further nine smaller squares in such a way that every number appears once in each horizontal line, vertical line, and square.

- Sudoku grid: a puzzle in which several numbers are to be filled into a 9x9 grid of squares so that every row, every column, and every 3x3 box contains the numbers 1 through 9
- Sudoku sub-grid: the 3X3 square contained in the 9X9 grid.
- Magic squares: a square that is divided into smaller squares, each containing a number, such that the figures in each vertical, horizontal, and diagonal row add up to the same value
- Backtracking: Backtracking is a method of solving a problem that involves undoing the operations in a problem to work backward from an output to an input.
- Candidate: Potential value for a cell.
- Contingency:A condition limiting the location of a value
- Scanning: The process of working through a puzzle to look for or eliminate values.
- Counting: Process of stepping through the values for a row, column or block to see where they can or cannot be used.
- Polyomino: A shape composed of equal sized, side-adjacent squares. Often used for Sudoku region variants.
- Clue: The cells which are filled with a value at the start configuration.
- Depth-first search (DFS): is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.
- Naked Single method: in a specific cell only one digit remains possible (the last remaining candidate has no other candidates to hide behind and is thus naked). The digit must then go into that cell.
- Hidden Single method: for a given digit and house only one cell is left to place that digit. The cell itself has more than one candidate left, the correct digit is thus hidden amongst the rest.

10 Appendix

10.1 Data set

10.1.1 Sample input

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

- 0 = candidate cell
- Integer of value from 1 to 9 = clue
- * = demarcating of the grids

10.2 Results

10.2.1 Sample :Input board, resulting output board and time taken to solve

Input board

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

Output board

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

Time taken to solve: 7.3320672 miliseconds

11 References

1. `https://www.google.com/search?client=ubuntu&channel=fs&q=Magic+square+&ie=utf-8&oe=utf-8(2016-09-23)`

2. https://www.google.com/search?client=ubuntu&channel=fs&q=Magic+square+&ie=utf-8&oe=utf-8#channel=fs&q=sudoku+definition(2016-09-23)

3. http://timesofindia.indiatimes.com/home/sunday-times/What-is-the-history-of-S
articleshow/9169332.cms(2016-09-23)

4. https://illuminations.nctm.org/Lesson.aspx?id=655(2016-09-23)

5. http://searchsecurity.techtarget.com/definition/brute-force-cracking(2016-09-

6. https://www.learner.org/courses/learningmath/algebra/keyterms.html(2016-10-01)

7. http://byteauthor.com/2010/08/sudoku-solver/(2016-10-07)

8. https://codemyroad.wordpress.com/2014/05/01/solving-sudoku-by-backtracking/ (2016-10-25)

9. https://en.wikipedia.org/wiki/GlossaryofSudoku (2016-10-25)

10. http://www.diva-portal.org/smash/get/diva2:721641/FULLTEXT01.pdf (2016-10-29)

11. http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand13/Group1Vahid/report/Aref-
Fiorella-KexJobb-sist.pdf (2016-10-29)

12. http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group6Alexander/final/Pa
(2016-10-30)

13. https://projecteuler.net/project/resources/p096sudoku.txt (2016-10-30)

12 Acknowledgments

- DR H.Vadapalli for class notes
- Fiorella Grados and Aref Mohammadi for the experimental comparisons used from their report 'The Sudoku Solver'
- Patrik Berggren and David Nilsson for the results in comparison found in the report 'A study of a Sudoku solving algorithm'
- The dataset used to generate results refernce[13]

13 Group contribution

Group contribution		
Jared Naidoo: 719238	Zayyan Variawa: 852486	Krupa Prag: 782681
Implementation of backtracking algorithm	Implementation of database generation	Theoretical research
Implementation of solver algorithm	Theoretical research	Complexity proof
Theoretical research		Experimental analysis
Empirical analysis		Comparison between theoretical and experimental analysis
Write-up implementation		Conclusion