KTH Computer Science
and Communication

A study of Sudoku solving algorithms

PATRIK BERGGREN, DAVID NILSSON

Bachelor's Thesis at NADA
Supervisor: Alexander Baltatzis
Examiner: Mårten Björkman

TRITA xxx yyyy-nn

Abstract

This is a bachelor thesis that studies and compare four different Sudoku solving algorithms. Those algorithms are rule-based, backtrack, and Boltzmann machine. The comparison consisted of measuring the algorithms, including variations of the algorithms, against a database of 49 151 different 17-clue Sudoku puzzles. The results show that ...

Referat

En studie om Sudokulösningsalgorithmer

TODO:

Contents

Chapter 1

Introduction

Sudoku is a game that under recent years have gained popularity throughout the world.

The game consists of a 9 by 9 grid which shall be filled in so that each row, column and 3 by 3 square (also called a box) contains the digits 1 through 9 exactly ones. Many newspaper today contain Sudoku puzzles and there are also competitions devoted to Sudoku solving. It is therefore of interest to study how one can generate, solve and rate such puzzles by the help of computers.

1.1 Problem specification

To generate and solve sudokus one can choose from several different algorithms. Some might be limited to only generation or only solving whilst other could be used for both. The main goal of this project is to explore different aspects of Sudoku solving and generation. To be able to compare different ideas, several algorithms will be used in the comparison. The project will not only try to identify which algorithm seems to fit best for a specific aspect of Sudoku, but also conclude how each algorithm performs itself and why.
The algorithms that will be studied are backtrack solver, rule-based solver and Boltzmann machine solver. Those will be discussed and explained in more detail in section 2.1 and 3. Variations of those algorithms will also be studied so there will in effect be data for 10 or more algorithms and variations of algorithms.
The ideas that will be studied are how well each algorithm performs on a given set of puzzle. The algorithms will be compared towards each other and hopefully it will be possible to make conclusions about which kind of algorithm effectively solve which kind of puzzles. Each Sudoku solving algorithm can also be used for Sudoku generation, which is another aspect that will be studied and compared in this project. Furthermore some implementation issues regarding the algorithms will be discussed, such as how well the algorithms are suited for parallelising the work. The discussion will also touch on difficulty rating which is another aspect that often gets attention in many studies regarding Sudoku.

1.2 Scope

As this project is quite limited in time available and in expected scope, there are several limitations on what can and will be done. These are the limitations of the project:

- Limited number of algorithms: Since many of the algorithms are only available as psuedo-code and since we ourselves want to make sure that the algorithms we implement have a similar amount of optimization, we will implement the algorithms ourselves. This will however be time consuming and since each algorithm have multiple variations, the measurements will also be time consuming. The analysis would also be beyond this scope if to many algorithms were chosen.

- Optimization: As mentioned above we will put some work into optimizing the different algorithms, but it will however be limited to which extent this can be done. This will affect the conclusion as some comparisons might have some uncertianty regarding the correctness of the results. The general idea is to put the same amount of work into every algorithm, while keeping the essential parts of the program in mind.

- Special sudokus: There are several variations of Sudoku including different sizes of the grid. This thesis will however be limited to the study of ordinary Sudoku, which is 9 by 9 grids.

- The results of this thesis will be applicable to other areas apart from Sudoku related topics, but those will only be mentioned briefly and no extensive study regarding the use of the results in other areas will be done.

1.3 Purpose

As already mentioned, Sudoku is today a popular game throughout the world and it appears in multiple medias, including websites, newspapers and books. As a result, the demand of effective Sudoku related algorithms are rising. For most purposes there already exist satisfactory algorithms, but there are still some value in studying Sudoku algorithms. Sudoku could for instance be extended to a more generalized form with a n by n grid and most regular Sudoku solving algorithms could be generalized to solve those as well. Another related reason for studying Sudoku solving algorithms is that there exists several similar problems to Sudoku and an idea for Sudoku may be applicable for those other related puzzles. Latin squares are for instance a very similar puzzle that could directly benefit from Sudoku solving algorithms. There are however a broader class of problems that could benefit from these thesis and those are the NP-complete problems. Sudoku is one of those NP-complete problems [1] which briefly speaking means that no known effective algorithm exists that solve them. For now we will however only conclude that there exists some similarities between Sudoku and some of those NP-complete problems, such as latin squares and graph colouring.

Chapter 2

Background

The background gives a introduction of Sudoku solving and the various approaches to creating efficient solvers.

2.1 Sudoku fundamentals

A Sudoku game consists of a 9x9 grid of numbergs, each belonging to the range 1-9. Initially a subset of the grid is revealed and the goal is to fill the reamining grid with valid numbers. The grid is guarded by certain rules restricting which values that are valid insertions, with the initial subset always being valid. The three main rules are: row and columns can only contain all 1-9 digits exactly once, which also applies to each one of the nine 3x3 subgrids [2]. In order to be regarded as a proper Sudoku puzzle it is also required that a unique solution exists, a property which can be analyzed by studying the size of the initial subset and solving for all possible solutions.

The size of the given subset, typically referred to as the number of clues, determine the difficculty of finding all grid values and thereby solving the Sudoku. Commonly the number of clues are reduced for increased difficulty with difficulty levels such as "easy", "medium" and "hard" being common ratings in newpapers. Rating puzzles is however not just that simple but requires more extensive analysis, something which has been studied separately (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.6

There is however a lower limit on the number of clues given that results in a unique solution. This limit was proven to be 17 [2], limiting the interesting number of clues to the range of 17-80.

2.2 Computational perspective

Sudoku solving is an research area in computer science and mathematics, with areas such as solving, puzzle difficulty rating and puzzle generation being researched [6](http://www.springerlink.com/index/L14T86X63XQ7402T.pdf, "Research on Construting of Sudoku Puzzles") [11].

The problem of solving $n^2 * n^2$ Sudoku puzzles is NP-complete [1]. While being theoretically interesting a an result it has also motivated research into heurstics, resulting in a wide range of available solving methods. Some of these algorithms include backtrack [9], rule-based [4], cultural genetic with variations[6], and Boltzmann machines [5]. Given the large variety of solvers available it is interesting to group them together with similar features in mind and try to make generic statements about their performance and other aspects. One selection critiera is their underlying method of traversing the search space, in this case deterministic and stochastic methods. Deterministic solvers include backtrack and rule-based. The typical layout of these is a predetermined selection of rules and a deterministic way of traversing all possible solutions. They can be seen as performing discrete steps and at every moment some transformation is applied in a deterministic way. Stochastic solvers include genetic algorithms and Boltzmann machines. They are typically based different stochastic selection criteria that decides how candidate solutions are constructed and how the general search path is built up. While providing more flexibility and more a more generic approach to Sudoku solving there might be performance problems and wider distributions of execution times (todo: ref).

2.3 Evaluated algorithms

Given the large amount of different algorithms available it is necessary to reduce the candidates, while still providing a quantitative study with broad results. With these requirements in mind, three different algorithms were chosen: backtrack, rule-based and Boltzmann machine. These represent different groups of solvers and were all possible to implement within a reasonable timeframe. A short description is given below with further in depth studies in the following subsections.

- Backtrack: Backtrack is probably the most basic Sudoku solving strategy for computer algorithms. It is a kind of a bruteforce method which tries different numbers and if it fails it backtracks and try a different number.

- Rule-based: This method consists of using several rules that logically proves that a square either must have a certain number or roles out numbers that are impossible (which for instance could lead to a square with only one possible number). This method is very similar to how humans solve Sudoku and the rules used is in fact derived from human solving methods.

- Boltzmann machine: Modeling a Sudoku by using a constraint solving artificial neural network. The most natural way of encoding constraints is with a reccurent stochastic boltzmann network. By overlaying the input grid over all neural network nodes and then simulating until it is stable, a valid solution can be found. This way of solving constraint problems can be seen as a natural way of encoding data and constraints. It is however difficult to reason about performance.

2.3.1 Backtrack

The backtracking algorithm for solving Sudoku is a bruteforce method. One might view it as guessing which numbers goes where. When a deadend is reached, the algorithm backtracks to a earlier guess and tries something else. This means that the backtracking algorithm does an extensive search to find a solution. The main drawback to this algorithm is that it is exponential in time complexity, but will on the other hand always find a solution sooner or later if one exists. This method may also be used to determine if a solution is unique for a puzzle as the algorithm can easily be modified to continue searching after finding one solution. As a result it could be used to generate valid Sudoku puzzles (with unique solutions).
Parallelising is something that could improve this algorithm considerable. The way this could be done is by searching different searchpaths in parallel. This is possible as the search three is fixed and only determined by the puzzle.
There are several interesting variations of this algorithm that might prove to vary in effiecency. The main variations regards in which order the squares are visited. One might for instance search the squares in random order or in some order that is defined by the structure of the Sudoku. One might for instance first fill in all possible numbers in each empty square and then begin with the least number of candidates, or one might want to finish nearly completed rows, columns or boxes. One might also vary how the guesses are done. Some rules regarding how the guesses are done might lead to a higher probability to find a solution which in turn will make the whole algorithm faster as the solutions will be found earlier in the search.

2.3.2 Rule-based

This algorithm relies on the logic rules that humans use to solve Sudoku. Those rules works as proofs and either determines that a square must be a certain number or prooves that a certain square can not contain a certain number. The rules that are used are as follows:

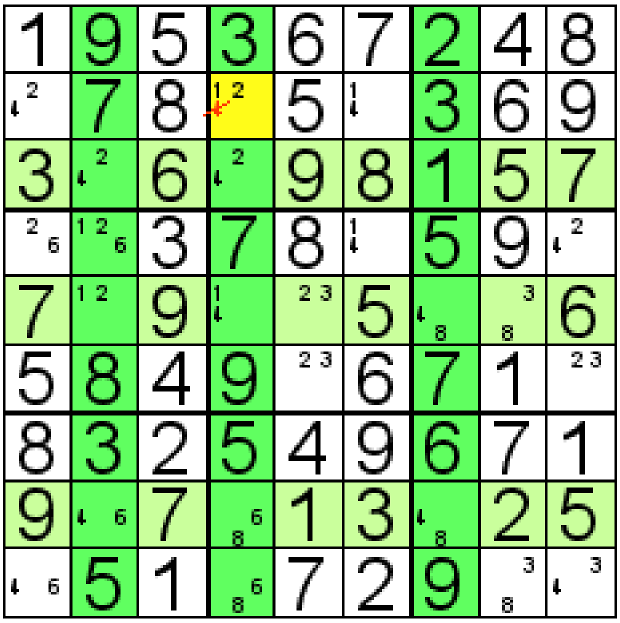
Naked Single This means that a square only have one candidate number.

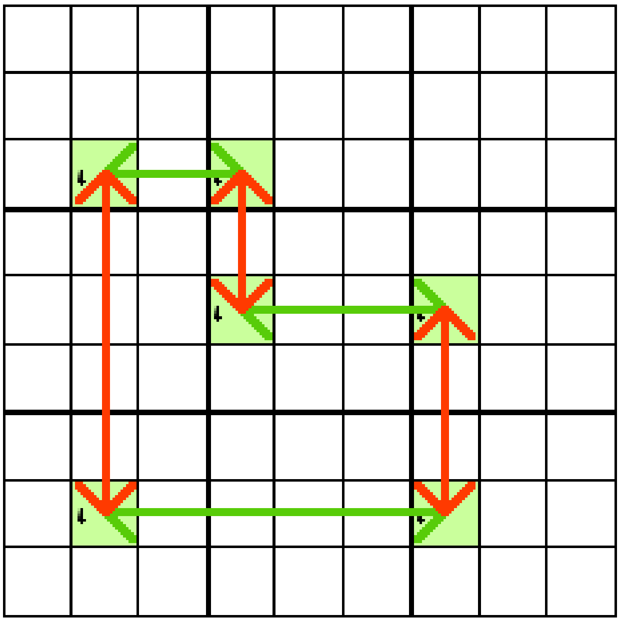
Hidden Single If a row, column or box contains only one square which can hold a specific number then that number must go into that square.

Naked pair If a row, column or box contains two squares which each only have two specific candidates. If one such pair exists, then all occurences of these two candidates may be removed from all squares that share a row, column or box with both of the squares in the pair. This concept can also be extended to three or more squares.

Hidden pair If a row, column or box contains only two squares which can hold two specific candidates, then those squares are a hidden pair. It is hidden because those squares might also include several other candidates. Since one already know which two numbers have to go into the two squares one might remove other candidates for those two squares. Those squares will now be a naked pair and one could therefore apply the rule for removal of further candidates. Similar to naked pairs this concept may also be extended to three or more squares.

X-wing/swordfish/jellyfish/squirmbag These are all names of a more general rule that concerns a loop. Below is a example of a so called swordfish (figure 1 and 2). In figure 1 the squares which have the digit 4 as a candidate have been marked. The so called swordfish is shown in figure 2. The logical reasoning one might use with a swordfish is that since the configurations of the fours in the swordfish can only take two configurations and both of them will cover the same set of columns and rows. The other squares in those rows and columns with the digit 4 as a candidate may therefore remove 4 as a candidate. Swordfish is the special case of this concept where 3 rows and columns are involved in the loop, but the concept also applies to loops within 2 rows and columns (X-wing) and even loops with more than 3 rows and columns involved (jellyfish and squirmbag).





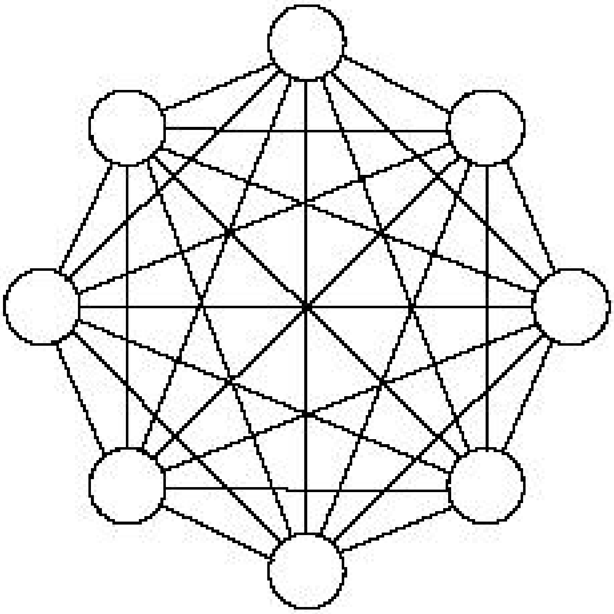
There are other rules that could be used but due to limitations these are the only implemented ones. There are several different aspects that are interesting to note about this algorithm. First of all the time complexity of the algorithm will be polynomial given that a solution can be found using these rules. This is a simple consequence of the fact that each rule only take polynomial time to check and there are only polynomial many squares to fill in. There are however no guarantee that a solution will be found and if no rule match it is impossible for the algorithm to continue. One way around this is to combine this algorithm with the backtracking algorithm. Other aspects that shall be studied are which rules shall be applied in which order and even if some rules are to time consuming to check compared to the extra puzzles that can be solved. The variations of this algorithm therefore consists of different sets of rules (including guessing rules for backtracking) that are applied in different orders.

2.3.3 Boltzmann machine

In order to fully understand the artifical neural network approach of solving Sudoku puzzles, it is necessary to have some basic theoretical background before introducting the implementation.

Theory

The artificial neural network solver is based on reccurrent stochastic networks that encode all properties of Sudoku. In the core there is a Boltzmann machine, which can be seen as a network of fully connected nodes, as pictured below.



The recurrent property implies that output connections are fed back into to neurons, or in this case, all nodes are fully connected. This definition often implies a hopfield network which is a somewhat limited version of the Boltzmann machine. What differentiates them and makes the latter more powerfull is the stochatic property, meaning state transitions of individual nodes are stochastic.

In the case of Sudoku there are a total of 81*9 nodes; for every number being placed on the grid, there is nine possible assignments. When a solution is present in the network there will only be one node active in every group and the solution can be extracted by simply assigning the node index within the group as the resulting grid number.

Given that a network of nodes has been defined it is also required to encode constraints. This is done by introducing weights on all edges in the fully connected network. All weights are stored in a weight matrix that satisfies the following conditions:

$$w_{ii} = 0 \quad \forall i$$

$$w_{ij} = w_{ji} \quad \forall i, j$$

At any moment in the simulation there is a state present in the network, which is coded separately. Each node has a binary state s_i of either on or off. This also translates to solutions, where only one of every nine neurons in groups will be in the state on. With weights and the state associated with every node of the network it is also useful to define an energy measure which is used to determine if the state of single neuron should be flipped. The difference in enery activation for a single node flipping state is given by the following equation, where theta is a offset used as an adjustable performance parameter [13]:

$$\Delta E_i = \sum_j w_{ij} s_j - \theta$$

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. [13] Here another adjustable performance parameter is introduced, T. In order to have convergence in the network towards a valid Sudoku solution it is necessary to control the probablity of state flipping as the simulation goes on. This is done by utilising simulated annealing which controls the temperature T over time. By lowering the value in a controlled fashion it is possible to maximize the probability of a final correct solution.

$$p_{i=on} = \frac{1}{1 + e^{-\frac{\Delta E_i}{T}}}$$

Using the previously introduced expressions it is now possible to describe the actual implementation and the steps necessary to solve Sudoku puzzles.

Implementation

The first step is to take initialize a random weight matrix and random states for all 81*9 nodes in the neural network. Then all given grid values are inserted as activiations 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.

The second step is to encode Sudoku restrictions into the weight matrix. Negative connections are introduced between different digits on within the same group, in other words only one digit should be active on every place in the grid at the same time. The other properties of unique rows, unique columns and unique subsquares are encoded in the same way but processed with the whole weight matrix.

With all configuration done it is then time to launch the simulation, which is run in discrete time steps. At every time step the energy function is evaluated at all nodes. The energies are then processed which renders the probablities of nodes flipping state, which is then performed to varying degrees. Since simulated annealing 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 but still doesn't waste any time for uneccessary computation.
A valid solution can be found by simple looking at all states and insepecting the requirements of Sudoku. Commonly the solution will appear in the end of the simulation when temperaure has been lowered to its minimum.

Chapter 3

Method

Since this report have several aims, this section have been divided into different parts to clearly depict what aspects have been considered regarding the different aims. Those sections will also describe in detail how the results was generated. Section 3.1 is devoted to explaining the test setup which includes hardware specifications but also an overview picture of the setup. Section 3.2 focuses on how and what aspects of the algorithms where analysed. Section 3.3 explains the process of choosing test data. The last subsection 3.4 gives an overview of the statistical analyses which was performed on the test data. This also includes what computational limitations was present and how this effect the results.

3.1 Test setup

The central part of the test setup was the test framework which timed and tested each algorithm on different puzzles. This was done by implementing each algorithm as separate classes that all inherited from a superclass for all sudokusolvers. This made sure the implementations was consistent which made the test process easier. The last part of the setup consisted of the test puzzles which was tested on the algorithms. The test consisted of timing the algorithms on each puzzle. Since there might be variations in processor performance multiple tests where performed with each puzzle. The mean value was then saved for each puzzle and algorithm. As the variance in processor performance might vary depending on the puzzle tested a confidence intervall for the mean value was calculated for the mean value using bootstraping. This will be described in more detail in section 3.4.

3.2 Comparison Methods

Multiple aspects of the results was considered when analysing and comparing the algorithms. The following three subsections describes those aspects in more detail.

3.2.1 Solving

3.2.2 Difficulty rating

One can often find difficulty ratings associated to Sudoku puzzles in puzzle books etc. Those are often based on the level of human solving techniques that are needed to solve the puzzle in question. [14] This study will similarly measure the puzzles difficulty, but will not rely on which level of human solving techniques that are needed, but instead on how well each algorithm performs at solving each puzzle. This comparison will be limited to the rule-based and backtracking algorithms and the goal will be to see if there is a correlation between how hard the puzzles are. Since only two algorithms will be used, the results will be somewhat uncertain but it will be a beginning in determining if certain puzzles are inherently difficult independent on which algorithms are used to solve it.

3.2.3 Generation and parallelization

This is a more practical aspect of the comparison and no tests will be done. It is however still possible to discuss how well the algorithms are suited for generating puzzle and how well they can be parallelisised. Generation of puzzles is obviously interesting because that is mainly what one want to do if constructing a puzzle collection. Parallelising is however not entirely obvious why it is of interest. Normal Sudoku puzzles can be solved in a matter of milliseconds by the best Sudoku solvers and one might therefore struggle to see the need for parallelising those solvers. And truly this topic is quite unrelevant for normal Sudoku puzzles but the discussion that will be held about the algorithms is however still of practical interest since one might want to solve n by n puzzles which can get extremely difficult fast when n grows. Since the algorithms to some extent also can be applied to other NP-complete problems, the discussion is also relevant in determining which type of algorithms might be useful in other areas. This last remark has already been studied, but not for Sudoku solving algorithms and we therefore take the chance to start the discussion.

3.3 Benchmark puzzles

The test data consisted of multiple puzzles that was chosen beforehand. Since the set of test puzzles can affect the outcome of this study it is appropriate to motivate the choice of puzzles. As was discovered during the study the Boltzmann machine algorithm did not perform as well as the other algorithms and some modifications to which puzzles was used was therefore done. The backtracking and rule-based algorithms was however both tested on a set of 49151 17-clue puzzles. Those was found on [12] and is claimed by the author Royle to be a collection of all 17-clue puzzles that he has been able to find on the Internet. The reason for chosing this specific database is because the generation of the puzzles does not involve a specific algorithm but is rather a collection of puzzles found by different puzzle generating algorithms. The puzzles are therefore assumed to be represantative of all 17-clue puzzles. This assumption is the main motivating factor for chosing this set of puzzles, but there is however also other factors that makes this set of puzzles suitable. As recently discovered by Tugemann and Civario, no 16-clue puzzle exists which means that puzzles must contain 17 clues to have unique solutions. [2] As discussed under section 3.2.2 difficutly rating is poorly measured by the number of clues in a puzzle, but one can however see a correlation between the number of clues and the difficulty. [14] This means that the choosen set of 17-clue puzzles shall contain some of the hardest Sudoku puzzles that exists. This is ofcourse a wanted feature since one then can see how the algorithms performs at puzzles of all difficulties.

3.4 Statistical analysis

Due to several reasons statistical analyses is required to make a rigouros statement about the results. This is mainly due to two reason. Firstly the results contain a very large dataset and secondly there are some randomness in the test results which can only be dealt with by using statistical models. Most statistical tests give a confidence in the results to depict how surely one can be about the results of the statistical test. Naturally a higher confidence and more precise results leads to higher requirements on the statistical test. As described in section 3.4.2 some of the statistical tests have been limited by computational constraints and a lower confidence level in combination with a more inprecise result have therefore been needed for those tests.

3.4.1 Statistical tests

This section explains which statistical tests and methods are used in the study. The first statistical method that is applied is to make sure that variance in processor performance does not affect the results considerable. This is done by measuring a specific algorithms solving time for a specific puzzle multiple times. The mean value of those times are then calculated and bootstraping are used to attain a 95% confidence interval of 0.05 seconds. The reason bootstraping is used is because it does not require the stochastic variable to be a certain distribution. This is necessary since the distribution of the processor perfomance is unkown and also since the distribution might vary between different puzzles.
The meanvalues are then saved as described in section 3.1. It is now that the real analyses of the algorithms begins. Even if the representation of the results does not really classify as a statistical method it is appropriate to mention that the results are displayed as histograms which means that the data are sorted and devided into bars of equal width. For this study this means each bar represents a fixed size solution time intervall. The height of the bars are proportional to the frequency data points falls into that bar's time intervall. After the histogram are displayed

one can easily compare the results between different algorithms and also consider the distribution of the solution times of individual algorithms. The usefullness of this somewhat unrigouros approach shall not negleted, but shall also use statistical methods to rigourosly prove some relations.
The first thing one think of looking for is probably how the different algorithms compare in solving ability. This means that one want to find out if one algorithm is better than other algorithms. Since the distribution again is unknown one has to rely on more general statistical tests. One of those are wilcoxons sign test. n This makes use of the fact that the difference in solving times between two algorithms will have a mean value of 0 if there is no difference between the two algorithms. The tests uses the binomial theorem to see if the sign of the difference is unevenly distributed. One can then determine if an algorithm performs better than an other algorithm with a certain confidence. Since the distribution of the solution times are unknown it is not possible for wilcoxons sign test to give any measurement of how much better an algorithm is than another algorithm. One might however make a histogram of the difference which leads us to the next aspect, namely difficulty rating and distribution. Difficulty rating for each algorithm can easily be calculated by making histograms of the solution times for each algorithm. One aim of this study have however been to determine if there is puzzles that are inherently difficult, that is regardless of solution method. To determine if that is the case some statistical methods are required. The reader shall note that this test is only conducted on the backtracking and rule-based algorithms and the result can therefore not speak for other solution methods. The method used is however

3.4.2 Computational constraints

3.5 Benchmark puzzles

This thesis relies highly on measuring how long it takes different algorithms to solve puzzles. To measure this, puzzles are needed and consideration shall therefore be taken to choose those puzzles wisely. The puzzles that have been chosen to test the solving capabilities of the algorithms are a collection consisting of 49 151 puzzles which each have 17 clues (meaning 17 squares are filled in). [12] Those have been collected by Gordon Royle from different sites as well as from his own database. The reason for choosing this collection is because it is believed not to be biased towards a specific solving idea. This is because it is a collection that comes from collecting as many of the 17-clue puzzles that have been found as possible rather than using a specific generation method. The collection also have other benefits. The puzzles are all qualitativly different as no pair of puzzles is within the same equivalence class. Equivalent puzzles are defined as puzzles that could be transformed into each other using some predefined operations that doesnt change the solution. One might for instance interchange the digits in a puzzle, but the solution would then be the transformed solution and no real change to the puzzle have therefore been done. Some other operations includes interchanging rows, columns and stacks (three boxes in a row,column). The downside of choosing this collection is that some algorithms may perform poorly because of the low nr of clues given and some analysis of this will therefore be carried out. This will be done by solving the given puzzles and adding some of the numbers in the solution to the puzzle. The reason for this approach is ones again that one does not want to rely too much on a specific generation method.

3.6 Comparison methods

As already mentioned several comparisons and measurements will be carried out to analyse different aspects of the algorithms. Those will include statistically tests to for instance determine with what confidence one can say that one algorithm performs better than another. The following subsections will however explain more clearly what will be measured regarding each aspect. Two of the general problems one most confront when doing this kind of study is how one shall make sure that the measured time for running a algorithm is correct. There might for instance be delays due to slow input and output and one must also account to variations in runningtime. The input/output delays can be avoided by communicating with the algorithm through a direct interface. Variations in runningtime is however difficult to predict, but one might however use a meanvalue as a approximation. To some extent the variations of the runningtimes will be reduced as the algorithms are tested against a large set of puzzles.

3.6.1 Solving

The main interest of this thesis is the algorithms ability to solve sudokus and most focus will therefore also be on measuring their solving abilities. As already mentioned, there is a fixed collection of puzzles that shall be tested on each algorithm and also on each variation of the algorithms. The data needed for the analysis will consist of measuring the time it takes for each algorithm to solve each puzzle. This will result in quite a big data set and consideration must therefore be taken regarding what statisticall methods are used to analyse it. (It is however at this point not decided which stasticall tests s

3.6.2 Generation

Sudoku solving algorithms may also be used to generate puzzles. One way this could be done in is by randomly placing numbers in squars and then using the solver to determine if there is a Sudoku and if it is unique. Some solvers may however be better suited for this as some will for example not be vary suited for determining if a solution is unique. There are however also other methods for generating puzzles as explained in [11]. The algorithms will therefore be compared at theirs ability to generate puzzles in any way that the authors thinks suitable. The comparison will regard how correct the generated puzzles are (if there is always a unique solution) and also how fast the generation was done. As the data from the generation testing will be very similar to the data from the solving tests, the same statistical tests will be used.

3.6.3 Parallelising

3.6.4 Difficulty rating

3.7 Comparison

At this point, the conclusions that can be drawn are very limited. This is largerly due to the fact that the test system have not yet been implemented even thought the first draft of the algorithms have been implemented. Some tests so far however show that the rule-based solver seems to be able to solve roughly 50% of the puzzles with only hidden and naked single rules. Since puzzles that only requires these rules are generally regarded as very easy puzzles, one can conclude that most Sudoku puzzles probably are easy. An hypothesis that is related to that but not yet tested is that it will probably be difficult to generate difficult problems. Backtracking algorithms with some variations have also been tested, but at a smaller scale. It seems so far that backtracking seems to be a less fit algorithm for solving 17-clue puzzles, but some optimization are yet to be done. Those are the subsections expected to exist in the finished report.

3.7.1 Solving

This will probably be the most extensive subsection and it will probably be divided into smaller parts. It is however difficult to tell which those will be now since they greatly depend on what results we get.

3.8 Generation

3.9 Parallelising

3.10 Difficulty ratings

Chapter 4

Analysis

Chapter 5

Conclusions

Chapter 6

References

Bibliography

[1] Takayuki Y, Takahiro S. Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. [homepage on the Internet]. No date [cited 2012 Mar 8]. Available from: The University of Tokyo, Web site: http://www-imai.is.s.u-tokyo.ac.jp/ yato/data2/SIGAL87-2.pdf

[2] Tugemann B, Civario G. There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem. [homepage on the Internet]. 2012 [cited 2012 Mar 8]. Available from: University College Dublin, Web site: http://www.math.ie/McGuire_V1.pdf

[3] Felgenhauer B, Jarvis F. Enumerating possible Sudoku grids. [homepage on the Internet]. 2005 [cited 2012 Mar 8]. Available from: University of Sheffield, Web site: http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf

[4] Astraware Limited. Techniques For Solving Sudoku. [homepage on the Internet]. 2008 [cited 2012 Mar 8]. Available from:, Web site: http://www.sudokuoftheday.com/pages/techniques-overview.php

[5] Ekeberg. Boltzmann Machines. [homepage on the Internet]. 2012 [cited 2012 Mar 8]. Available from:, Web site: http://www.csc.kth.se/utbildning/kth/kurser/DD2432/ann12/forelasningsanteckningar/07-boltzmann.pdf

[6] Marwala T. Stochastic Optimization Approaches for Solving Sudoku. [homepage on the Internet]. 2008 [cited 2012 Mar 8]. Available from:, Web site: http://arxiv.org/abs/0805.0697

[7] . An Incomplete Review of Sudoku Solver Implementations. [homepage on the Internet]. 2011 [cited 2012 Mar 8]. Available from:, Web site: http://attractivechaos.wordpress.com/2011/06/19/an-incomplete-review-of-sudoku-solver-implementations/

[8] Harvey W, Ginsberg M. Limited discrepancy search. [homepage on the Internet]. No date [cited 2012 Mar 13]. Available from: University of Oregon, Web site: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.2426&rep=rep1&type=pdf

[9] Cazenave Cazenave T. A search based Sudoku solver. [homepage on the Internet]. No date [cited 2012 Mar 13]. Available from: Université Paris, Dept. Informatique Web site: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.459&rep=rep1&type=pdf

[10] Mantere T, Koljonen J. Sudoku Solving with Cultural Swarms. AI and Machine Consciousness [homepage on the Internet]. 2008 [cited 2012 Mar 13]. Available from: University of Vaasa, Department of Electrical Engineering and Automation Web site: http://www.stes.fi/step2008/proceedings/step2008proceedings.pdf#page=60

[11] Morrow J. Generating Sudoku Puzzles as an Inverse Problem. [homepage on the Internet]. 2008 [cited 2012 Mar 13]. Available from: University of Washington, Department of Mathematics Web site: http://www.math.washington.edu/ morrow/mcm/team2306.pdf

[12] Royle G. Minimum Sudoku. [homepage on the Internet]. No date [cited 2012 Mar 13]. Available from: The University of Western Australia, Web site: http://www.math.washington.edu/ morrow/mcm/team2306.pdf

[13] Ackley D, Hinton G. A Learning Algorithm for Boltzmann Machines. [homepage on the Internet]. 1985 [cited 2012 Mar 13]. Available from: The University of Western Australia, Web site: http://learning.cs.toronto.edu/ hinton/absps/cogscibm.pdf

[14]

Appendix A

RDF

And here is a figure

Figure A.1. Several statements describing the same resource.

that we refer to here: A.1