Comparison of how common sudoku solving techniques perform when adapted and applied to jigsaw sudokus

WICKMAN TOMAS, ÖHMAN EMIL

Bachelors Thesis
Stockholm, Sweden 2014
Abstract

Sudoku and Jigsaw sudoku are two similar logic puzzles. The goal of the puzzles is to place numbers in a 9*9 grid until certain rules are met. In both Jigsaw sudoku and normal sudoku the numbers 1 to 9 needs to be placed in every row, column and box. The difference between the two puzzles is that in a normal sudoku every box is a symmetric 3*3 square, while in the jigsaw variant the boxes have irregular, although still nine cells big, boxes.

When implementing algorithms for solving the normal sudoku, it normal to use the fact that all the boxes are symmetric. This will not work for the jigsaw sudoku though.

In this report we will investigate the effectiveness of the three most common sudoku solving algorithms Rule Based backtracking, Human-Like solving and Dancing Links and how we arrived at the result that Human-Like solving was, in almost every case, the most effective algorithm for solving both sudoku and jigsaw sudoku.
Sammanfattning

Sudoku och Jigsaw sudoku är två likartade logikspel som går ut på att placera ut tal enligt vissa regler på en spelplan som har 9x9 rutor. I både jigsaw sudoku och vanligt sudoku ska talen ett till nio finnas med i varje kolumn, rad och box exakt en gång. Det som skiljer dem åt är att i sudoku är boxarna symmetriska och 3x3 rutor stora medan i jigsaw sudoku har de varierande form av 9 rutor.

Vid implementation av algoritmer för lösning av sudoku är det enkelt att använda symmetrin av hur de ser ut, medan det för jigsaw sudoku inte fungerar då boxarna ej är symmetriska.

I denna rapport undersöker vi effektiviteterna hos de tre algoritmerna Rule Based backtracking, Human-Like solving och Dancing Link för att visa hur vi kom fram till resultatet att Human-Like solving i nästan alla fall var den effektivare algoritmen både för vanliga sudoku och jigsaw sudoku.
Statement of collaboration

Both have been working on large parts together but we still have divided our responsibilities so that time consuming parts do not need to be done together:

- Implementions: Tomas has been responsible for the study of algorithm implementations that could be applied to jigsaw sudoku and regular sudoku. Implementation research of sudoku and jigsaw sudokus and how the time measurement will be done has been Emil's responsibility.

- Report writing: Emil has been responsible for the introduction and background. The methods first part has Emil been responsible for and the second part has Tomas been responsible for. Tomas has been responsible of the writing of Appendix A and Emil of the making of describing images.

- Remaining parts: The non mentioned parts has been both responsibilities.
Contents

Statement of collaboration . . . . . . . . . . . . . . . . . . . . . . . . . . . v

Contents vi
0.1 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1 Introduction 3
1.1 Purpose . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.2 Problem description . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Background 5
2.1 What is a sudoku . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 What is a jigsaw sudoku? . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3 How will the solving differ? . . . . . . . . . . . . . . . . . . . . . . . 6

3 Methods 9
3.1 Measuring time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
3.2 Comparison setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
3.3 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

4 Results 13
4.1 Hardware used . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
4.2 Average time of sudokus (µs) . . . . . . . . . . . . . . . . . . . . . 13
4.3 Average time of jigsaw sudoku (µs) . . . . . . . . . . . . . . . . . . 14
4.4 Comparison between jigsaw sudoku and normal sudoku . . . . . . 15
4.5 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
4.6 Self-criticism . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
4.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

5 Appendix A 19
5.1 Rule Based Backtracking . . . . . . . . . . . . . . . . . . . . . . . . . 19
5.2 Human-Like Solving . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
5.3 Dancing Links . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

Bibliography 37

vi
0.1. DEFINITIONS

0.1 Definitions

Grid
Consists of the entire puzzle, 9*9 cells.

Nonominos
9 neighbouring cells of the Jigsaw puzzle that contains the numbers 1-9 exactly once.

Box
3*3 cells of a Sudoku game that contains the numbers 1-9 exactly once.

Cell
1*1 section for a number of the puzzle that can contain only one of the numbers 1-9.

Row
A row of 9 cells that contains the numbers 1-9 exactly once.

Column
A column of 9 cells that contains the numbers 1-9 exactly once.
Chapter 1

Introduction

1.1 Purpose

What makes this subject interesting is that sudoku is a well investigated and known subject, but not similar games that has many similarities. Therefore it is interesting to investigate how the most common sudoku solving algorithms performs on various jigsaw sudokus and try to compare the results with similar difficulties of normal sudokus. We will compare the results on terms of computation time. With the knowledge of how the algorithms compare comes a deeper understanding of which algorithms are the best under different circumstances.

1.2 Problem description

There are many different ways of solving a jigsaw sudoku puzzle both by computers and by humans. This report will investigate the three algorithms Rule Based backtracking, Human-Like solving and Dancing Link.
Chapter 2

Background

2.1 What is a sudoku

Sudoku is a logic-based puzzle with a 9×9 cell grid consisting of non-overlapping boxes of 3×3 cells. When starting to solve the puzzle, it consists of 17-30 cells with numbers between 1-9 prefilled. The more difficult the game is, the less prefilled cells it will have[4]. The goal is to fill each cell of the puzzle with a number between 1-9 according to the following rules.

1. No box is allowed to have multiple cells with the same number.
2. Each row of the grid contains all numbers between 1-9.
3. Each column of the grid contains all numbers between 1-9.
4. Only one number in each cell.

Each puzzle has only one solution [4]. The puzzle got popular in Japan in the mid of the 1980:s [4] and started to grow nationally in Sweden in the spring of year 2005 [4][5].
CHAPTER 2. BACKGROUND

Figure 2.1: A sudoku.

2.2 What is a jigsaw sudoku?

Jigsaw sudoku is a logic based puzzle with a 9*9 grid consisting of non-overlapping irregular shapes of 9 cells with no empty holes in it called nonominos. At start the puzzle consists of various numbers of cells between 1-9 prefilled depending of the difficulty of the game. The goal is to fill each cell of the puzzle with a number between 1-9 according to the following rules.

1. No nonomino is allowed to has multiple cells with the same number.
2. Each row of the grid contains every number between 1-9 exactly once.
3. Each column of the grid contains every number between 1-9 exactly once.
4. Only one number in each cell.

Each puzzle has only one solution.

2.3 How will the solving differ?

In contrast to normal sudoku the jigsaw sudoku does not have symmetrical boxes. This allows for some new problems and some new possibilities when it comes to solving the puzzles. When solving with Dancing Links for example we will have to use different constraints in the matrix in order to solve the jigsaw sudokus in comparison to the normal sudokus.

2.3. HOW WILL THE SOLVING DIFFER?

The human-like solving approach will be able to utilize new rules because of the asymmetric boxes, and the Rule Based backtracking solver will have to take care of which box it is currently in since it cannot easily be calculated by assumed symmetry. Some of these will allow the jigsaw sudoku to be solved faster and some will make the solver slower.

\[\text{Figure 2.2: A jigsaw sudoku}^2.\]

Chapter 3

Methods

The investigation we will do is to test the three algorithms on various difficulties of jigsaw sudokus to find out how they compare to each other in terms of computation time. We will also investigate how the solving time depends on the the number of boxes that the rows a and columns intersect with since this will be the major difference between the two puzzles.

3.1 Measuring time

We start measuring time just before we compute the solutions with the different algorithms and stop the measuring at the end of the program. We chose to use the gettimeofday() function from the time.h [2] library to measure time. We chose to measure in microseconds since we figured milliseconds would be too inexact and nanoseconds would be too exact and introduce too much noise.

All the measurements were performed at the same time and on the same hardware in order to prevent external software to interfere with the results.

3.2 Comparison setup

We got all of our puzzles for both sudoku and jigsaw sudoku from the same place [1]. Because we chose to compare the runtime of the algorithms on sudoku and jigsaw sudoku with regard to the difficulty of the puzzles, we needed to get all puzzles from the same website to ensure that the definition of difficulty did not change. In total we used five of each puzzle type and difficulty, then we took the average runtime to solve these for each algorithm. There were three difficulties in total which were easy, medium and hard.
3.3 Algorithms

Rule Based backtracking

The first algorithm we choose was the Rule Based backtracking way. The algorithm simply goes through each possible number and checks if it’s valid to place it in the next cell. If no number could be placed in the cell, the algorithm backtracks to the previous cell and tries a new number there. This goes on until the final cell has been filled. This algorithm will look almost the same for both types of sudokus.

An example can be seen in Appendix A where we show one step of the backtracking algorithm on a 4x4 sudoku. Even though the dimensions differ the principles for solving a 9x9 sudoku are exactly the same. Keep in mind that the dimensions of the boxes in the example are only 2x2.

Human-Like solving

We based our humanlike solving algorithm on how a human would try to solve the jigsaw sudoku. This means we try to emulate the tactics and moves a human would use to find all the numbers of a sudoku board. A better description of each step can be found in Appendix A.

The objective of the algorithm is to eliminate the numbers in the cells by applying rules until only one number remains in the cell. Then we fill out the cell with the remaining number and continue until all the numbers have been filled out.

The algorithm works by iterating through the steps described in Appendix A, and whenever a step is applied, it jumps back the start of the loop and start searching for a step that can be applied. If no steps could be applied it means we have exhausted the possible steps and we then solve the remaining cells with our backtracking algorithm.

Dancing Links

Dancing Links is a variation of the exact cover algorithm developed by Knuth[3]. The exact cover problem is a problem where, given a sparse matrix, is there a way to choose rows from the matrix so that we have a new matrix where each row has exactly one 1 in each column?

Knuth devised an algorithm called Algorithm X[3] to solve this problem[11]. While Algorithm X does find a solution the algorithm is naive and the problem is NP-Complete. Thus the approach is not very effective.

What Knuth did was that he implemented a fast heuristic to solve this problem using two doubly linked list to represent the sparse matrix. This allowed for fast insertion and removal of the nodes since every node in the linked list held information about which nodes it had previously been connected too. It also removed the need to check every cell in the row to see if a column already
was covered or was going to be covered, thus speeding up the process significantly. The Dancing links algorithm is a constraint algorithm, so we will need to be able to represent the sudokus and the jigsaw sudoku has some form of constraint problem using a sparse matrix.

A more in-depth explanation of how the Dancing Links algorithm can be found in Appendix A.

How will the sparse matrix look for the sudokus?

First we need to identify what the constraints are that we need to have fulfilled for the sudoku to be solved, and in our case the constraints are the rules of the game. For the first rule we will need 81 columns to represent the total number of cells in our standard 9x9 sudoku. For the second rule there will be 9 rows and each row can hold 9 numbers which gives a total of 81 columns. The third rule has 9 columns and each column can hold 9 numbers which gives a total of 81 columns. The fourth rule gives 9 boxes with 9 numbers in each which gives us 81 columns. Thus our sparse matrix will be 324 columns wide. And since we will need one row for each cell and one for each number in the cell, we will get 81*9 rows which will be 729 rows. This makes the total size of the sparse matrix for a standard sudoku 324x729.

How will the sparse matrix differ between jigsaw sudoku and normal sudoku?

Since the the rules are exactly the same for both types of problems the sparse matrix will have exactly the same amount of rows needing to be satisfied. The difference will be in how we present the possible solutions in the boxes. The normal sudoku will have regular patterns in the box constraint columns while the jigsaw sudoku will be irregular in its patterns.
Chapter 4

Results

4.1 Hardware used

Operating System: Ubuntu 12.04 LTS
Memory: 3.8 GiB
Processor: Intel® Core™2 Quad CPU Q9550 @ 2.83GHz × 4
OS type: 64-bit
Date: April 12th, 2014
Location: Computer lab red (röd), kth

4.2 Average time of sudokus (µs)

Dancing Links

<table>
<thead>
<tr>
<th>Sudoku</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>5396</td>
<td>4496</td>
<td>6204</td>
<td>5529</td>
<td>5406</td>
</tr>
<tr>
<td>Medium</td>
<td>6810</td>
<td>4266</td>
<td>6337</td>
<td>5396</td>
<td>5702</td>
</tr>
<tr>
<td>Hard</td>
<td>5029</td>
<td>4643</td>
<td>7090</td>
<td>6536</td>
<td>5824</td>
</tr>
</tbody>
</table>

Rule-Based backtracking

<table>
<thead>
<tr>
<th>Sudoku</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>61</td>
<td>77</td>
<td>860</td>
<td>4441</td>
<td>1359</td>
</tr>
<tr>
<td>Medium</td>
<td>1249</td>
<td>162</td>
<td>86</td>
<td>394</td>
<td>472</td>
</tr>
<tr>
<td>Hard</td>
<td>1414</td>
<td>2570</td>
<td>472</td>
<td>2545</td>
<td>1750</td>
</tr>
</tbody>
</table>
CHAPTER 4. RESULTS

Human-Like solving

<table>
<thead>
<tr>
<th>Sudoku</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersections</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>Easy</td>
<td>253</td>
<td>249</td>
<td>359</td>
<td>449</td>
<td>327</td>
</tr>
<tr>
<td>Medium</td>
<td>531</td>
<td>147</td>
<td>443</td>
<td>399</td>
<td>380</td>
</tr>
<tr>
<td>Hard</td>
<td>430</td>
<td>371</td>
<td>322</td>
<td>564</td>
<td>421</td>
</tr>
</tbody>
</table>

4.3 Average time of jigsaw sudoku (µs)

Dancing Links

<table>
<thead>
<tr>
<th>Sudoku</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>5250</td>
<td>4863</td>
<td>5598</td>
<td>5647</td>
<td>5564</td>
</tr>
<tr>
<td>Medium</td>
<td>6600</td>
<td>3793</td>
<td>7060</td>
<td>6465</td>
<td>5979</td>
</tr>
<tr>
<td>Hard</td>
<td>6081</td>
<td>5794</td>
<td>6871</td>
<td>5316</td>
<td>6015</td>
</tr>
</tbody>
</table>

Rule-Based backtracking

<table>
<thead>
<tr>
<th>Jigsaw sudoku</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>154</td>
<td>114</td>
<td>236</td>
<td>126</td>
<td>157</td>
</tr>
<tr>
<td>Medium</td>
<td>16977</td>
<td>741</td>
<td>35056</td>
<td>83140</td>
<td>33987</td>
</tr>
<tr>
<td>Hard</td>
<td>31166</td>
<td>23577</td>
<td>11255</td>
<td>53935</td>
<td>27450</td>
</tr>
</tbody>
</table>

Human-Like solving

<table>
<thead>
<tr>
<th>Jigsaw sudoku</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersections Easy</td>
<td>70</td>
<td>76</td>
<td>68</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>Easy</td>
<td>454</td>
<td>320</td>
<td>491</td>
<td>323</td>
<td>397</td>
</tr>
<tr>
<td>Intersections Medium</td>
<td>80</td>
<td>72</td>
<td>66</td>
<td>62</td>
<td>70</td>
</tr>
<tr>
<td>Medium</td>
<td>974</td>
<td>374</td>
<td>720</td>
<td>486</td>
<td>638</td>
</tr>
<tr>
<td>Intersections Hard</td>
<td>74</td>
<td>70</td>
<td>68</td>
<td>80</td>
<td>73</td>
</tr>
<tr>
<td>Hard</td>
<td>246</td>
<td>430</td>
<td>706</td>
<td>785</td>
<td>541</td>
</tr>
</tbody>
</table>
4.4 Comparison between jigsaw sudoku and normal sudoku

Performance on jigsaw sudoku

Performance on normal sudoku
4.5 Discussion

Dancing Links

The Dancing Links algorithm was the most consistent of all algorithms. However it was not very effective compared to the human like solving. This is because the solving speed of the Dancing Links algorithm is not affected by the difficulty by the sudoku but by the size of the sparse matrix it uses.

Human like solving

This algorithm outperformed every other one except for the easy jigsaw case where the rule base backtracking did better. One interesting thing to note is that the algorithm solved the hard jigsaw sudokus faster then it did solve the medium. This most likely has to do with the fact that the hard sudokus had more box intersections for the rows and columns which allowed for more effective techniques to be used.

Rule Based Backtracking

By far the most inconsistent algorithm in terms of solving speed. This has to do with the fact that the harder the sudokus get, the bigger the chance that it recurses down the wrong branch gets. This causes a lot of backtracking to occur an thus the algorithm takes a long time to finish. This is also why we see results such as hard jigsaw sudoku 3, where it only took about 1125 µs, which is less then a twentieth of any other result it got for hard jigsaws. It most likely takes a path that results in little backtracking and thus a fast solving speed.

4.6 Self-criticism

Although the results tell a lot about how the different algorithms performs on the different kinds of sudokus, they are far from perfectly implemented. There is likely some performance enhancements that can be done to trim time off the algorithm solving speed. The definition of what makes a sudoku difficult is also not very strictly defined. What we did was that we used the same page for all our sudokus[1] so that we would have a consistent definition. However this does not guarantee that the way that the website defines it is the best way. There could also have been a bigger selection of sudokus since we now only test for four of each difficulty. This means the rule based solver could just as well have gotten four easy ones and outperformed the human like solver which might have produced another result completely.
4.7 Conclusion

By looking at the results and comparing the solving time it becomes obvious that the human like solving approach is the fastest solver with few exceptions. The rule based backtracking did outperform it sometimes but this was most likely due to luck with the recursion path. The Dancing Links algorithm did perform consistently but it could never get to the same speed as the human like solver.
Chapter 5

Appendix A

5.1 Rule Based Backtracking

\[\text{http://bitcook.de/wp-content/uploads/2013/10/2x2_sudoku_backtracking1=300x245.png}\]

Figure 5.1: Backtracking procedure, based on image from \(^{1}\).
1. This is the first step we check in the backtracking algorithm. Here we have placed a 1 in cell (2,1). However this breaks 3 and 4. Thus we backtrack to the previous state and try to place another number in the cell.

2. We now try to place a 2 in cell (2,1) which will break rule 4 and thus trigger backtracking to the previous state.

3. Here we try to place a 3 in cell (2,1) which we find does not break any of the rules and thus is a valid move.

4. The four is not visited here since we go down the branch in number 3. Would all the branches from step 3 turn out to be faulty we will continue to step 4.

5. Since we filled a number in (2,1) we now move on to (2, 2) and try to place a 1 there which as it turns out breaks rule 4.

6. So we go back to the previous state (3) in which we now try to place a 2 instead in (2, 2). This breaks rule 3.

7. We backtrack again and try to place a 3 in (2,2). This breaks rule 2 and triggers a backtrack back to step 3.

8. Here we place a 4 in (2,2) which follows all the rules.

This continues until either we fill out the last cell or there is no steps left to try. If we managed to fill all the cells we are done, else the sudoku does not have a solution.
5.2. HUMAN-LIKE SOLVING

The very first thing we do is to fill out the sudoku board with all the possible candidates, see fig 5.2. In the beginning, before applying any steps, every cell that doesn’t have a clue has every number as its possible candidates.

1. Check if there is a cell with only one solution

This rule iterates through all the cells and checks if there is a cell with only one candidate for it. If so is the case then fill that cell out with the number. If the last cell has been filled, then we’re done. See figure 5.3.

2. Remove impossible candidates

After we have filled out the new cells with numbers there might be candidates in squares that aren’t valid anymore. This is done by applying the jigsaw sudoku rules which are as follows:

a) Every number has to show up exactly once in the nonomino.

b) Every number has to show up exactly once per row.

[http://www.sudokuwiki.org/jigsaw.htm]
c) Every number has to show up exactly once per column. Thus if there’s candidates to be removed, we do so and jump back to step one, else continue to step three.

3. Search for unique candidate numbers

In this step the algorithm iterates over all the nonominoes, rows and columns to see if there’s a candidate number that occurs only once in any of them. If that’s the case, fill those out and jump back to step one, or if no single numbers are found, continue to step four. See figure 5.4.

4. Find conjugate pairs

Since there was no unique numbers in step three, we check if the rows, columns and nonominoes if there’s two cells that has exactly two identical candidate numbers. If that’s the case, we know that these two numbers has to be in these two cells, meaning we can remove all other candidate numbers on the same row, column or nonomino that these cells occur in. If no such occurrence was found, continue to step five, else go back to step one. See figure 5.5.

5. Pair removal

In this step we examine each nonomino to see if there’s any two or three of the same candidate numbers aligned on the same row or column. If these are the only possible placements of the candidate then we know that there cannot be any other of the same candidate aligned with these numbers on the row or the column, thus those numbers can be removed. See figure 5.6.

6. Intersection removal

The intersection removal step consists of two sub-steps which are the following:

a) Pair or triple in a box:

b) Box Line Reduction

The first move is to fill out the board with all the possible candidate numbers which can be seen in fig. 5.7-5.8.

7. Double pairs [7]

Because the nonominoes looks so different from each other we might sometimes find that by comparing two rows, or two columns in the same way as in step five. If we’re lucky, we might be able to deduce that there’s only two possible rows to place the number in and thus be able to remove two rows of the candidate number in a nonomino. See figure 5.9.
8. Double boxes [8]

Like the box reduction technique we here check if there’s a row or column passing through the with the modification that instead of only checking one row or column we check two at a time to see if we might be able to remove candidate numbers from multiple boxes at a time. See figure 5.10.


Here we separate an equal amounts of nonominoes and an columns or rows. Then we draw a line through those columns an look which of the nonomino cells sticks out of the enclosed column/rows. Those cells must have the same values. See figure 5.11.

10. Check if we are done

If we have reached this far, either we are done or we need to use the backtracking approach.

11. Backtracking

If we have reached this point in the algorithm that means there’s no more move to make. To circumvent this problem we simply solve the last remaining cells by applying the backtracking algorithm on the sudoku. Since the remaining cells are usually very few the backtracking has minimal effect on the performance of the algorithm.
Figure 5.3: Last candidate, based on image from ¹.

Figure 5.4: Last in a row, based on image from ¹.

Figure 5.5: Conjugate pairs, based on image from ¹.

¹http://www.sudokuwiki.org
5.2. HUMAN-LIKE SOLVING

Figure 5.6: Pair removal, based on image from 1.

Figure 5.7: Pair in a box, based on image from 1.
Figure 5.8: Box reduction, based on image from 1.
Figure 5.9: Double pointing pairs, based on image from 1.
Figure 5.10: Double boxes, based on image from 1.
Figure 5.11: Law of leftovers, based on image from \(^1\).
5.3 Dancing Links

Algorithm X

1. Pick an unsatisfied constraint

   Start by finding a column that has not yet been covered

2. Pick a row that satisfies that constraint

   Then choose a row that will cover that column

3. Add the row to the solution set

   Since we have chosen a row to be part of our new solution matrix we add it to the set of rows in the solution

4. Delete all rows that satisfy the constraints satisfied by the chosen row

   Since we are only allowed one occurrence in each row we delete all rows that shares columns with the added row

5. 5-9. Repeat previous steps...

6. ...

7. ...

8. ...

9. ...

10. No rows left

Now we have reached a point where all the constraints are not satisfied but we have no rows left to add that satisfy that constraint. This will trigger a backtrack to the last step and try to find a new solution from there. This is called uncovering a column.
5.3. DANCING LINKS

1. Pick an unsatisfied constraint

2. Pick a row satisfying that constraint

Figure 5.12: Picking unsatisfied constraint, based on image from 1.

3. Add the row to the solution set

4. Delete all row that satisfy any of the constraints satisfied by the chosen row

Figure 5.13: Adding a row to the solution set, based on image from 1.

1 http://www.sudokuwiki.org
CHAPTER 5. APPENDIX A

6. Pick a row satisfying that constraint

7. Add the row to the solution set

8. Delete all row that satisfy any of the constraints satisfied by the chosen row

9. Pick an unsatisfied constraint

10. No rows! Backtrack to last choice

Figure 5.14: Picking unsatisfied constraints, based on image from 1.

Figure 5.15: Adding a row to the solution set, based on image from 1.

Figure 5.16: Backtracking, based on image from 1.
Dancing Links

The Dancing Links first need to construct the doubly linked list that it’s going to operate on. This can be seen in fig 5.3 where the top nodes represent the covered columns and the number inside the boxes is the number of possible candidates they have for covering them.

The extra node to the left of the top nodes is the header node. This is there to keep track of when the matrix has been covered since that will mean it points to itself.

The boxes without numbers represents the candidates in the matrix for covering columns, and the lines going to and from them is the pointers in the doubly linked list.

\[2\]


Figure 5.17: A Dancing Links matrix based on a figure from 2.
The Dancing Links basically consists of the following operations on the doubly linked list.

**Cover**

As can be seen in fig 5.4

Just like in the Algorithm X a column is covered by adding a row to the solution set and then removing all the conflicting nodes on a row. This is done by re-adjusting the pointers in the double linked list so that the top row now points to the next available node in the matrix. The red arrows represent the changed pointer. Notice that the nodes still retain pointers to the removed nodes.

Figure 5.18: Covering a column, based on a figure from ³.

5.3. DANCING LINKS

Figure 5.19: Uncovering a column.\(^4\)

**Uncover**

As can be seen in fig 5.5

When uncovering a column with Dancing Link you just undo what you did in the cover step in the exact reverse order. This can easily be done since nodes retained a pointer to their old nodes which makes it so that all you have to do is re-connect them.

\(^4\)http://garethrees.org/2007/06/10/Zendoku-generation/dancing-links-2.png
Bibliography

   collected 2014-04-10

   collected 2014-03-26

[3] Dancing Links, Donald E. Knuth, Stanford University, 15 November 2000,
   arXiv:cs/0011047,
   http://arxiv.org/abs/cs/0011047

    Nationalencyklopedin,
    collected 2014-03-05.

    Nationalencyklopedin,
    collected 2014-03-05.

    collected 2014-04-01

    collected 2014-04-01

    collected 2014-04-05

    collected 2014-04-05

    collected 2014-04-05

    collected 2014-04-05