Efficiency of the hybrid AC3-tabu search algorithm for solving Sudoku puzzles

JOAKIM LINDBERG, MARTIN STEIER
Efficiency of the hybrid AC3-tabu search algorithm for solving Sudoku puzzles

JOAKIM LINDBERG
MARTIN STEIER

Degree Project in Computer Science, DD143X
Supervisor: Arvind Kumar
Examiner: Örjan Ekeberg

CSC, KTH, May 8, 2015
Abstract

There are many different algorithms for solving Sudoku puzzles, with one of the newer algorithms being the hybrid AC3-tabu search algorithm. Since the algorithm has not been subject of much research, the aim of this thesis is to increase the knowledge of it.

This thesis evaluates the efficiency of the hybrid AC3-tabu search algorithm by analyzing how quickly it solves puzzles compared to two other solving algorithms: one using brute-force search, and one combining human solving techniques with brute-force search. This thesis also investigates if there is a correlation between the number of puzzle clues and the solving time for the hybrid AC3-tabu search algorithm.

The results show that the hybrid AC3-tabu search algorithm is less efficient than the two other algorithms, and that there seems to be a correlation between the number of clues and the solving time for the algorithm. The conclusion is that due to the algorithm’s low efficiency and some of its characteristics, it is not suitable for solving Sudoku puzzles.
# Contents

1 Introduction 1  
1.1 Problem definition ......................................... 2  
1.1.1 Delimitations ........................................ 2  
1.2 Problem statement ........................................ 2  
1.3 Motivation and aim ....................................... 3  
1.4 Terminology .............................................. 3  

2 Background 4  
2.1 Tabu search .............................................. 4  
2.2 AC-3 ...................................................... 6  
2.2.1 CSP .................................................. 6  
2.2.2 Arc consistency ...................................... 6  
2.2.3 Description of the AC-3 algorithm .................. 7  
2.3 Hybrid AC3-tabu search .................................. 10  
2.4 Brute-force search ....................................... 10  
2.4.1 Backtracking ......................................... 10  
2.5 Human solving techniques ............................... 10  
2.5.1 Naked singles ....................................... 11  
2.5.2 Hidden singles ...................................... 11  
2.5.3 Naked pairs ......................................... 11  

3 Method 12  
3.1 Implementation .......................................... 12  
3.1.1 Brute-force search ................................... 12  
3.1.2 Combined solver ..................................... 14  
3.1.3 Hybrid AC3-tabu search ............................... 15  
3.2 Tests ..................................................... 20  
3.2.1 Technical details ...................................... 20  
3.2.2 Data .................................................. 20  
3.2.3 Testing procedure .................................... 21  
3.2.4 Limitations .......................................... 22
4 Results
4.1 Solved puzzles .................................................. 23
4.2 Solving times ..................................................... 24
  4.2.1 Solving time dispersion for the hybrid solver ........... 26

5 Discussion ......................................................... 27
  5.1 Efficiency analysis ............................................. 27
  5.2 Correlation between clues and solving time ............... 28
  5.3 Method criticism .............................................. 28
    5.3.1 Implementation ......................................... 28
    5.3.2 Programming language .................................. 28
    5.3.3 Time limit ............................................... 28
  5.4 Conclusions .................................................. 29
    5.4.1 Correlation between clues and solving time .......... 29
    5.4.2 Efficiency ............................................... 29

Bibliography .......................................................... 30

Appendices .......................................................... 32

A Number of found puzzles ...................................... 32

B Number of solved puzzles ..................................... 33

C Average and median solving times ........................... 34

D Standard deviation for the hybrid solver .................... 35
Chapter 1

Introduction

Sudoku is a logic-based puzzle game first published in 1979 as *Number Place* in a magazine in the United States [1]. In 1984, the game was included in a magazine in Japan as “suuji wa dokushin ni kagiru”, which was later abbreviated to Sudoku [1]. Around 2005, it became internationally popular [2].

The most common version of Sudoku is played on a $9 \times 9$ cell grid divided into $3 \times 3$ cell blocks. Each cell can contain a number between 1 and 9, and the currently possible numbers for a cell are called the cell’s candidates. In a Sudoku puzzle, some of the cells are already filled with clues while the rest are empty. The goal of the game is to solve the puzzle by filling all 81 cells so that each row, column and block contains the numbers 1 to 9. For the puzzle to be a valid Sudoku puzzle it must have a unique solution, and at least 17 clues are needed for this [3].

Figure 1.1(a) illustrates a Sudoku puzzle and Figure 1.1(b) illustrates the corresponding solution.

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
& & & & & & & & \\
& & & & & & & & \\
& & & & & & & & \\
\hline
3 & & & & & & & & \\
4 & 8 & 9 & & & & & & \\
2 & & & & 4 & 7 & 1 & & \\
1 & 2 & 5 & & & & 8 & & \\
& & 8 & & 7 & 1 & & & \\
5 & & & & & & & & \\
9 & & & & 5 & 4 & & & \\
6 & 1 & & & & & 3 & & \\
5 & & & & & & 7 & & \\
\hline
\end{tabular}
\end{figure}

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
7 & 1 & 6 & 8 & 3 & 5 & 4 & 2 & 9 \\
3 & 4 & 8 & 9 & 1 & 2 & 5 & 6 & 7 \\
2 & 5 & 9 & 4 & 7 & 6 & 1 & 3 & 8 \\
1 & 2 & 5 & 3 & 4 & 7 & 9 & 8 & 6 \\
6 & 3 & 4 & 2 & 8 & 9 & 7 & 1 & 5 \\
9 & 8 & 7 & 5 & 6 & 1 & 3 & 4 & 2 \\
8 & 7 & 2 & 1 & 9 & 3 & 6 & 5 & 4 \\
5 & 6 & 1 & 7 & 2 & 4 & 8 & 9 & 3 \\
4 & 9 & 3 & 6 & 5 & 8 & 2 & 7 & 1 \\
\hline
\end{tabular}
\end{figure}

*Figure 1.1.* A Sudoku puzzle with 25 clues (a) and its corresponding solution (b).
CHAPTER 1. INTRODUCTION

Sudoku puzzles can be solved using a wide variety of techniques. Human solving techniques are mostly based on logic, since other techniques are usually neither practical nor fun to use for humans. However, computers are able to use algorithms that are impractical for human use, such as brute-force search which enumerates all possible solutions until the correct one is found. Other examples are algorithms based on heuristics [4] or on the fact that Sudoku can be modelled as a constraint problem [5]. One such algorithm is the hybrid AC3-tabu search algorithm, which is the main focus of this thesis.

1.1 Problem definition

This thesis analyzes how efficient the hybrid AC3-tabu search algorithm is in solving Sudoku puzzles compared to two other solving algorithms: one using brute-force search and one combining human solving techniques with brute-force search. The analysis consists of comparing how fast the algorithms solve puzzles with different number of clues. Using the results, this thesis also investigates if the number of clues is correlated with the solving time for the hybrid AC3-tabu search algorithm.

1.1.1 Delimitations

Sudoku version

This thesis focuses solely on the Sudoku version using $9 \times 9$ grids and the aforementioned rules.

Human solving techniques

Only three of the simplest and most common human solving techniques are implemented in this thesis. However, many puzzles require more advanced techniques than the implemented ones. Thus, to ensure that the solver using human solving techniques is able to solve all puzzles, the solver switches to the implemented brute-force search when no further progress can be made with human solving techniques.

1.2 Problem statement

- How efficient is the hybrid AC3-tabu search algorithm in solving Sudoku puzzles compared to brute-force search and an algorithm combining human solving techniques with brute-force search?
  - Is the hybrid AC3-tabu search algorithm suitable for solving Sudoku puzzles?
- Is there a correlation between the number of puzzle clues and the solving time for the hybrid AC3-tabu search algorithm?
1.3 Motivation and aim

Hybrid AC3-tabu search is a relatively new algorithm and has not been subject of much research. In fact, only one paper was found regarding the algorithm, namely [4], in which the algorithm is proposed. Therefore, the aims of this thesis are to increase the knowledge about solving Sudoku puzzles with the algorithm, and to determine if the algorithm should be investigated further or if the algorithm’s efficiency is too low to justify further investigation.

1.4 Terminology

**Cell**: Either empty or contains a number between 1 and 9.

**Clue**: A number which is assigned to a cell and cannot be changed.

**Candidate**: A possible number for a certain cell.

**Block**: A collection of 9 cells forming a $3 \times 3$ structure.

**Grid**: A collection of 81 cells forming a $9 \times 9$ structure, or equivalently a collection of 9 blocks forming a $3 \times 3$ structure.

**Puzzle**: A grid with a fixed number of clues leading to a unique solution.

**Solution**: A grid where all cells have been filled with numbers and each of the numbers 1 to 9 occurs exactly once for each row, column and block.

**Candidate elimination**: Since a number in a cell cannot be assigned to another cell in the same row, column or block, the number can be removed (“eliminated”) as a candidate in the other cells.

**The combined solver**: The implemented Sudoku puzzle solver using both human solving techniques and brute-force search.

**The hybrid solver**: The implemented Sudoku puzzle solver using the hybrid AC3-tabu search algorithm.
Chapter 2

Background

This chapter first presents tabu search and AC-3, and then describes how they are combined in the hybrid AC3-tabu search algorithm to solve Sudoku puzzles. After that, brute-force search and human solving techniques are presented.

2.1 Tabu search

Tabu search is a metaheuristic used for solving combinatorial optimization problems, and was first mentioned in a publication by Glover in 1989 [6]. It has been applied to a wide variety of areas such as telecommunications [7], scheduling [8], and vehicle routing [9].

Tabu search begins by generating an initial solution for the problem to be solved. It then uses local search algorithms to improve the solution until it becomes an optimal solution, either local or global. To avoid getting stuck at a local optimal solution in which no improvement can be found, tabu search allows moves that make the current solution worse. However, there is a risk that the search might return to the previous local optimal solution when trying to improve the solution. Tabu search mitigates this risk by using tabu lists. [10]

Tabu lists contain rules called tabus that describe which moves are not allowed. The lists are classified into three types depending on how long a tabu persists in the list: short-term lists, intermediate lists and long-term lists. Short-term lists prevent returning to recently visited solutions, while intermediate and long-term lists guide the search in a desired direction. [10]

Each iteration of the tabu search begins by letting the local search find a set of potential new moves, which are then evaluated using the tabu lists. If a move is not considered tabu it is marked as admissible. However, even if a move is considered tabu it might be marked as admissible if it fulfills some aspiration criteria. These criteria override the tabu lists if the lists are preventing a move that leads to a better solution. [10]

After all potential new moves have been evaluated, the best admissible move found is performed. If the resulting solution is worse than the previous solution,
2.1. TABU SEARCH

the previous solution must be a local optimal solution. Thus, to avoid getting stuck at a local optimal solution, tabu search uses the resulting solution even if it is worse than the previous one. [10]

To prevent getting stuck in an infinite loop by constantly returning to the previous solution, the tabu lists are updated after each tabu search iteration. This is done differently depending on which type of tabus are used. For example, the reverse of the move just performed could be added to a short-term tabu list to avoid undoing the previous move for the next few iterations. [10]

The tabu search will continue until a stopping criterion, such as reaching a certain number of local search iterations, has been fulfilled, after which the best found solution is returned. [10]

The tabu search algorithm is illustrated in Figure 2.1, which is an adapted version of Figure 1 and Figure 2 in [10].

![Figure 2.1.](image)

(a) The tabu search algorithm. (b) Selection of the best admissible move.
2.2 AC-3

AC-3 is an algorithm proposed by Mackworth in 1977 for solving a constraint satisfaction problem (CSP) using arc consistency [11]. This section first defines CSP and arc consistency, and then describes the AC-3 algorithm.

2.2.1 CSP

A CSP consists of a set of variables with their corresponding domains, and a set of constraints. To solve a CSP, the variables must be assigned values from their domains so that all constraints are satisfied. [12]

As an intuitive example, the variables in a CSP could be cars, the domains could be the cars’ possible colors, and the constraint could be that no two cars are allowed to have the same color. A solution to the CSP would then be assigning each car with a color that no other car has been assigned with.

If only unary or binary constraints are used, it is possible to model CSPs as graphs, with nodes representing variables and directed arcs between the nodes representing constraints. Using this model, arc consistency can be defined. [11]

2.2.2 Arc consistency

An arc \((x_1, x_2)\) is arc consistent if for every value in \(x_1\)’s domain there exists a value in \(x_2\)’s domain that satisfies the constraint that the arc represents. If the arc is inconsistent, there exists at least one value in \(x_1\)’s domain for which the constraint is not satisfied. [11]

For example, consider two nodes \(x_1\) and \(x_2\), and two arcs \((x_1, x_2)\) and \((x_2, x_1)\) representing the constraints \(x_1 = x_2\) and \(x_2 = x_1\) respectively. The corresponding graph is shown in Figure 2.2(a) and the nodes’ domains are shown in Figure 2.2(b). \((x_1, x_2)\) is arc consistent since every value in \(x_1\)’s domain has at least one value in \(x_2\)’s domain that satisfies the constraint. However, \((x_2, x_1)\) is not arc consistent since there does not exist a value in \(x_1\)’s domain for which the constraint is satisfied for the value 3 in \(x_2\)’s domain.

\[\text{Figure 2.2. (a) The nodes and arcs. (b) The domains of } x_1 \text{ and } x_2. \text{ In (b), arrows connect values for which all constraints are satisfied.}\]

6
2.2. AC-3

2.2.3 Description of the AC-3 algorithm

The AC-3 algorithm solves a CSP by removing node values that create arc inconsistencies until all arcs are consistent or a domain becomes empty. The algorithm is illustrated in Figure 2.3. Note that the requeueing step is necessary since those arcs might have become inconsistent after the removal of node values. [11]

![Figure 2.3. A flowchart describing the AC-3 algorithm.](image-url)
**Example**

As an illustration of how the AC-3 algorithm solves a CSP, consider the following example. The CSP has three variables $x_1, x_2, x_3$ with their corresponding domains \{0, 1\}, \{0, 1\}, \{1, 2\} and two constraints $x_1 = x_2$, $x_2 = x_3$. The graph representation of the CSP is shown in Figure 2.4. Note that two arcs are needed for each constraint due to the use of directed arcs.

![Figure 2.4. The graph representation of the CSP.](image)

Using this graph representation the AC-3 algorithm can solve the CSP, and the process is described in the following list. Figure 2.5 illustrates how the nodes’ domains change in each step.

1. Enqueue the arcs in the (arbitrary) order $(x_1, x_2)$, $(x_2, x_1)$, $(x_2, x_3)$, $(x_3, x_2)$.
2. $(x_1, x_2)$ selected: Since all values in $x_1$’s domain match some value in $x_2$’s domain, the arc is consistent.
3. $(x_2, x_1)$ selected: Since all values in $x_2$’s domain match some value in $x_1$’s domain, the arc is consistent.
4. $(x_2, x_3)$ selected: 1 exists in $x_3$’s domain but 0 does not, so 0 must be removed from $x_2$’s domain. Since a value was removed from $x_2$’s domain, all arcs entering $x_2$ must be requeued and therefore, $(x_1, x_2)$ is requeued. (Since $(x_3, x_2)$ is already in the queue, it is not requeued.)
5. $(x_3, x_2)$ selected: 1 exists in $x_2$’s domain but 2 does not, so 2 must be removed from $x_3$’s domain. Since a value was removed from $x_3$’s domain, all arcs entering $x_3$ must be requeued and therefore, $(x_2, x_3)$ is requeued.
6. $(x_1, x_2)$ selected again: 1 exists in $x_2$’s domain but 0 does not, so 0 must be removed from $x_1$’s domain. Since a value was removed from $x_1$’s domain, all arcs entering $x_1$ must be requeued and therefore, $(x_2, x_1)$ is requeued.
7. $(x_2, x_3)$ selected again: Since all values in $x_2$’s domain match some value in $x_3$’s domain, the arc is consistent.
8. $(x_2, x_1)$ selected again: Since all values in $x_2$’s domain match some value in $x_1$’s domain, the arc is consistent.

The queue is now empty and the CSP has thus been solved, with the only solution being $x_1 = 1$, $x_2 = 1$ and $x_3 = 1$. 
Figure 2.5. The status of the nodes’ domains for each step in the example. Arrows connect values for which all constraints are satisfied. An asterisk next to an arc means that the arc has been requeued in this step.
2.3 Hybrid AC3-tabu search

Hybrid AC3-tabu search is an algorithm for solving Sudoku puzzles proposed by Soto et al. in 2013. The algorithm treats the puzzles as minimization problems where the aim is to minimize the number of empty cells, and uses tabu search for the minimization. Since Sudoku puzzles can be modeled as CSPs, the AC-3 algorithm is used for candidate elimination to improve the efficiency of the tabu search by limiting the search space, both before the first tabu search iteration and in each iteration. [4]

2.4 Brute-force search

Brute-force search is a general problem solving technique that enumerates all possible solutions until the correct one is found. Since there often exists a large number of possible solutions, pure brute-force search is only used for solving problems if no other approach is feasible. [13]

A Sudoku puzzle can have up to 64 empty cells (since at least 17 clues are needed), which means that there can be up to $9^{64} \approx 1.179 \cdot 10^{64}$ possible solutions. However, since there only are about $6.671 \cdot 10^{21}$ valid solutions [14], there is no need to enumerate all possible solutions, which is why a backtracking algorithm can be used.

2.4.1 Backtracking

A backtracking algorithm builds a solution step-by-step and checks after each step that it is still possible to reach a valid solution. If not, it undoes the last step (“backtracks”) and tries to take a different step forward [13]. If no other step forward is possible, it continues to backtrack until such a step is possible. This process continues until either a valid solution is found, or no new steps forward are possible in which case there are no valid solutions.

Since the algorithm backtracks when it detects that a valid solution can no longer be reached, it eliminates the need for trying many invalid solutions [13]. This limits the search space which usually makes the backtracking algorithm more efficient than a pure brute-force search.

2.5 Human solving techniques

There is a wide variety of human solving techniques based on candidate elimination. These techniques vary in difficulty, with the simplest ones considering one row, column or block at a time while the more advanced ones consider multiple rows, columns or blocks at once. This section presents three of the simplest human solving techniques: naked singles, hidden singles and naked pairs. [15]
2.5. HUMAN SOLVING TECHNIQUES

2.5.1 Naked singles

If a cell has exactly one candidate, the candidate must be the correct number. Thus, the candidate can be assigned to the cell, and it can also be removed as a candidate from all other cells in the same row, column or block. [15]

2.5.2 Hidden singles

If only one cell in a row has a certain number as a candidate, the candidate must be the correct number. Thus, the candidate can be assigned to the cell and all other candidates for the cell can be removed. The assigned candidate can also be removed as a candidate from all other cells in the same row. The same situation applies for columns and blocks. [15]

2.5.3 Naked pairs

If two cells in a row contain exactly two candidates and these candidates are the same for the two cells, the candidates must be the correct numbers for the two cells. Thus, the candidates can be removed from all other cells in the same row. The same situation applies for columns and blocks. [15]
Chapter 3

Method

To compare the efficiency of the solvers, it was not possible to perform a comparison using existing data, since the only found data regarding the hybrid solver consisted of solving data for only 8 puzzles [4], which was not considered enough to allow for a fair comparison.

Another approach would be to analyze and compare the time complexities of the algorithms. However, since the hybrid solver uses heuristics and randomness, its time complexity might not give an accurate reflection of its performance in practice.

Therefore, the most appropriate method to compare the efficiency of the solvers was to solve Sudoku puzzles with all three solvers and compare the solving times. The comparison was enabled by implementing all solvers as runnable programs and measuring the execution times.

This chapter first describes how the solvers were implemented, and then how the tests were performed.

3.1 Implementation

3.1.1 Brute-force search

The brute-force search was implemented with backtracking to improve the efficiency. The algorithm is illustrated in Figure 3.1.
3.1. IMPLEMENTATION

Figure 3.1. A flowchart describing the brute-force search algorithm.
### 3.1.2 Combined solver

The combined solver has three phases:

1. Assign all cells with all possible numbers (1 to 9) as candidates. In addition, perform candidate elimination with the initial clues.
2. Use the human solving techniques as much as possible for candidate elimination, and assign numbers to cells if possible.
3. Perform a modified version of the brute-force search. The difference between the modified version and the version in Section 3.1.1 is that the modified version takes advantage of the candidate elimination performed in Phase 2.

The algorithm is illustrated in Figure 3.2, with the three columns representing the three phases.

![Figure 3.2. A flowchart describing the combined solver algorithm.](image-url)
3.1. IMPLEMENTATION

3.1.3 Hybrid AC3-tabu search

Candidate selection

The candidate selection has three phases:

1. Try selecting a candidate for an empty cell that does not break any tabu if added to the grid. The selection is done using the roulette wheel selection described below.
2. If Phase 1 was unsuccessful, try selecting a filled cell whose number can be removed without breaking any tabu.
3. If Phase 2 was unsuccessful, select a filled cell whose number can be removed, ignoring the tabu lists.

The algorithm is illustrated in Figure 3.3, with the three columns representing the three phases.

Roulette wheel selection

The roulette wheel selection in Phase 1 selects one of the empty cells randomly but has a higher probability of selecting a cell with fewer candidates. This is done by assigning weights to all cells in the grid and letting the weights decide the probability for the cells to be selected.

Let \( c_i \) be the number of candidates for cell number \( i \), where \( 1 \leq i \leq 81 \). The weights \( w_i \) are calculated as

\[
w_i = \begin{cases} 
  1/c_i & \text{if } 2 \leq c_i \leq 9 \\
  0 & \text{otherwise}
\end{cases}
\]

Note that cells with only one candidate have already been assigned that candidate, and should thus have a weight of zero.

Then, the probability \( p_i \) of selecting cell \( i \) is calculated as

\[
p_i = \frac{w_i}{\sum_{j=1}^{81} w_j}
\]
Figure 3.3. A flowchart describing the candidate selection algorithm.
3.1. IMPLEMENTATION

AC-3

To enable use of the AC-3 algorithm in Section 2.2.3 for candidate elimination on a Sudoku puzzle, the puzzle was modeled as a CSP and then modeled as a graph according to the method in Section 2.2.1. In the graph, nodes represent cells and the nodes’ domains represent the cells’ candidates. The arcs are used for eliminating candidates in the following way.

Every node has arcs going to all other nodes in the same row, column or block. This way, the arc \( (x_1, x_2) \) between cells \( x_1 \) and \( x_2 \) is consistent as long as for every candidate in \( x_1 \) there exists at least one candidate in \( x_2 \) with a different number. Note that this means that if two or more candidates exist in \( x_2 \), the arc is consistent.

In addition to performing the candidate elimination, the implemented AC-3 algorithm fills in the number for cells that only have one candidate left.

Tabu search

The implemented tabu search uses all three tabu list types. The short-term list is of limited size and prevents undoing the last few moves, while the intermediate and long-term lists are of unlimited size and prevent returning to unsolvable grids.

Unsolvable grids are found by using the AC-3 algorithm to detect if assigning the selected candidate to the cell leads to a grid where at least one cell has no candidate left. If so, the selected candidate is added to the intermediate tabu list and another candidate is selected. If all candidates for all cells lead to an unsolvable grid the algorithm is forced to clear a cell, after which the intermediate list is cleared and the current grid is added to the long-term tabu list.

The stopping criteria used are that all cells in the grid have been filled or that a certain number of iterations have been performed.

The tabu search algorithm is illustrated in Figure 3.4.
Figure 3.4. A flowchart describing the tabu search algorithm.
3.1. IMPLEMENTATION

The hybrid solver

While implementing the hybrid solver, it was discovered that the solving times for puzzles with few clues were several orders of magnitude higher than the solving times for the brute-force solver and the combined solver. After investigating the cause, it was found that if the tabu search guessed incorrect numbers for the first few cells, it sometimes had to perform a large number of local search iterations simply to find out that the first guesses were incorrect. However, if it guessed correct numbers for the first few cells, the solver usually reached the correct solution in few iterations.

Therefore, to increase the performance of the tabu search, the solver was revised to restart the tabu search after a fixed number of iterations by resetting the grid and clearing the short-term and intermediate tabu lists. The long-term tabu list is retained since the grids in the long-term list can never lead to a solution.

The revised hybrid solver is illustrated in Figure 3.5.

![Flowchart of the hybrid solver](image_url)

Figure 3.5. A flowchart describing the hybrid solver.
CHAPTER 3. METHOD

3.2 Tests

3.2.1 Technical details

All tests were run on an Intel NUC DC3217BY. The following configuration was used:

**CPU:** Intel Core i3-3217U, dual-core, 1.8 GHz
**RAM:** Kingston KVR16S11/8, DDR3, 8 GB
**Storage:** Crucial M500, SSD, 120 GB
**OS:** Lubuntu 14.10, 64-bit
**Programming language:** Java 8
**Compiler:** javac, 64-bit, build 1.8.0_40
**JVM:** HotSpot, 64-bit, build 25.40_b25 (mixed mode)

To minimize the interference on the tests, a clean install of the operating system was made and no applications other than the compiler and the JVM were installed. Furthermore, the graphical shell was disabled and the tests were run from a command line shell.

3.2.2 Data

The puzzles used for testing were found on the Internet. The biggest finds were 49,151 puzzles with 17 clues [16], five sets with 25, 30, 35, 40 and 45 clue puzzles containing 10,000 puzzles each [17] and 10,000 + 50,000 puzzles with a varying number of clues between 19 and 32 [18, 19]. Many smaller puzzle sets were also found from various web sites. After removing duplicates, the total number of found puzzles was 168,337.

Figure 3.6 illustrates the number of found puzzles for each number of clues. A table containing the exact numbers can be found in Appendix A.

![Figure 3.6. The number of found puzzles for each number of clues.](image-url)
3.2. TESTS

Each puzzle was stored as an 81-character text string representing the numbers in the 81 cells of the grid. The first nine numbers represented the first row, the next nine numbers represented the second row, and so on. An empty cell was represented using the number zero.

3.2.3 Testing procedure

The testing procedure was the same for all three solvers. First, all puzzle strings were read from file and stored in memory. Then, for each string, the following procedure was performed:

1. Convert the string to a $9 \times 9$ matrix representing the grid.
2. Let the solver attempt to solve the puzzle.
3. Verify that the solution is correct.
4. Save the following information regarding the solving attempt:
   - Number of clues in the puzzle
   - If the puzzle was solved or not
   - Which solver was tested
   - The solving time

Since the solving time for a certain puzzle can vary for each solving attempt, the whole testing procedure was performed 3 times for each solver and the average of the calculated values were used in the results.

Time measurement

Since the testing procedure should be the same regardless of which solver is tested to allow for a fair comparison, only the solving attempt should be measured. Thus, the time was measured using calls to the built-in method `System.nanoTime()` just before and after Step 2 in the list above. However, to correctly measure time, some functionality in the used JVM (HotSpot) had to be considered.

When HotSpot executes a program for the first time, it uses an interpreter to execute the program code and analyzes how frequently each part of the code is executed. Since interpreted code executes slower than compiled code, HotSpot compiles the code parts that are most frequently used to speed up future execution of those parts. [20]

Therefore, to ensure that the code was compiled before Step 2 was performed, a large number of puzzles were solved, without saving the results, before the real tests were performed. To force HotSpot to compile all parts of the code, puzzles that required execution of all parts of the code to be solved were chosen. 15 000 puzzles with 25 clues proved to be sufficient for this purpose.
3.2.4 Limitations

Ignored number of clues

While testing, it was found that the solving time for some of the puzzles greatly deviated from the average. Thus, if only a small number of puzzles were to be tested, a few deviating solving times could affect the average solving time drastically. To minimize this problem, an attempt was made to find as many puzzles as possible. Despite this, there were less than 100 found puzzles for certain numbers of clues, and these numbers of clues were therefore ignored in the results. These numbers of clues were 19, 20, 31-34, 36-39 and 41-44.

Time limit

Even after the improvements made in Section 3.1.3 to the hybrid solver, the solving times were too long to test all puzzles within reasonable time. For example, for a sample of 250 puzzles with 17 clues, the average solving time per puzzle was 39.3 seconds. If that time is representative for all the 49151 puzzles with 17 clues, it would take over 22 days to solve them. Therefore, a time limit of 1 second per puzzle and solving attempt was used. This enabled solving attempts for all found puzzles in less than two days using any of the three solvers.
Chapter 4

Results

In the figures presented in this chapter, the values for the ignored number of clues (see Section 3.2.4) were interpolated. The data from which the figures were created can be found in Appendix B, Appendix C and Appendix D.

4.1 Solved puzzles

The combined solver had the highest percentage of solved puzzles within 1 second, the brute-force solver had the second highest percentage and the hybrid solver had the lowest percentage. This applies to all number of clues (except when two or three solvers manage to solve all of the puzzles).

The percentage of solved puzzles increased for all solvers as the number of clues increased. The only exception to this was that the combined solver managed to solve a higher percentage of 17-clue puzzles than 18-clue puzzles.

Figure 4.1 illustrates the percentage of solved puzzles for the three solvers. All three solvers managed to solve all puzzles with 29 clues or more, so puzzles with more than 29 clues were excluded from the figure.
4.2 Solving times

For puzzles with 17 to 30 clues, the combined solver had the shortest average and median solving times, the brute-force solver had the second shortest times and the hybrid solver had the longest times. For puzzles with 35 clues or more, the brute-force solver had the shortest solving times, the combined solver had the second shortest times and the hybrid solver had the longest times.

For the hybrid solver, the average and median solving times decreased as the number of clues increased, with two exceptions: the average solving time increased between 17 and 18 clues and the median solving time increased between 25 and 26 clues.

Figure 4.2(a) illustrates the average solving times of the solved puzzles for all three solvers, and Figure 4.2(b) illustrates the median solving times. Since the solving times differ with several orders of magnitude between the solvers, both figures use a logarithmic scale for the $y$-axis to improve clarity.
4.2. SOLVING TIMES

Figure 4.2. The average (a) and median (b) solving times in microseconds. Note that the y-axes use a base-10 logarithmic scale.
4.2.1 Solving time dispersion for the hybrid solver

The solving time dispersion for the hybrid solver was measured using the standard deviation. Figure 4.3 illustrates the standard deviation expressed as a percentage of the average solving time for each number of clues.

Figure 4.3. The standard deviation expressed as a percentage of the average solving time for the hybrid solver.
Chapter 5

Discussion

5.1 Efficiency analysis

For any number of clues below 22, the hybrid solver does not even manage to solve 50% of the puzzles within 1 second. For this reason, we consider the solving times for puzzles with below 22 clues to be non-representative for the hybrid solver. This does not affect the results however, since those solving times are still longer than the solving times for the brute-force solver and the combined solver.

One possible explanation for the long solving times is that we had to implement parts of the hybrid solver using our own ideas, which is discussed in Section 5.3.1.

However, the hybrid solver is over 30 times slower than the second slowest solver for some number of clues. Therefore, we do not believe it is possible to improve the hybrid solver enough to make it more efficient than the other solvers. We base this belief on two problems we have identified with using tabu search to solve Sudoku puzzles.

The first problem is that tabu search uses heuristics, which means that it cannot guarantee that the optimal solution for a problem is found. For a Sudoku puzzle however, only the optimal solution — a completely filled grid — is sufficient. This means that near-optimal solutions that the tabu search might produce, such as a grid containing just one empty cell, are not good enough to qualify as solutions to a puzzle. Thus, the tabu search has to continue searching until the optimal solution is found, which defeats the purpose of a heuristic.

The second problem is the large search space for a Sudoku puzzle. While searching for the optimal solution, the tabu search has to remember previously visited solutions that cannot lead to the optimal solution, as it could otherwise get stuck in a loop. This, in combination with the large search space, means that the tabu lists have to be large. As the lists get filled with more and more tabus, it becomes increasingly inefficient to check them when performing candidate selection, which slows down the search.

These two problems indicate that tabu search, and thus also the hybrid solver, is unsuitable for solving Sudoku puzzles.
5.2 Correlation between clues and solving time

As seen in the results, there seems to be a correlation between the number of clues and solving time for the hybrid solver. However, it should be noted that solving time dispersion was rather high at over 100% of the average solving time for certain number of clues. Also, since the hybrid solver selects candidates using roulette wheel selection, the solving time for a specific puzzle can vary drastically between two solving attempts depending on which candidates are selected.

5.3 Method criticism

5.3.1 Implementation

For the implementation of the hybrid AC3-tabu search algorithm, an attempt was made to stay as close as possible to the implementation in [4]. However, not all parts of the pseudocode were explained, which meant that we had to implement some parts using our own ideas.

For example, there are not enough details regarding the candidate selection to allow a direct implementation of it. In [4], the use of a roulette wheel selection is mentioned but its implementation is not described further. Therefore, there is a possibility that our implementation is less efficient than the implementation in [4].

5.3.2 Programming language

Although other programming languages such as C++ might produce faster running code than Java, only the relative performance between the different solvers is of interest in this thesis since the goal is to compare their efficiency. However, it is possible that Java does not allow for an efficient implementation of the hybrid solver and that it could be implemented to run efficiently in other languages. This was not investigated in this thesis.

5.3.3 Time limit

As mentioned in Section 3.2.4, the solving time was limited to 1 second per puzzle and solving attempt. The consequence of this was that the percentage of solved puzzles decreased for puzzles with few clues. This mainly affected the hybrid solver and to some extent the brute-force solver, while it had a minimal effect on the combined solver.

Theoretically, it is possible that the hybrid solver would manage to solve all puzzles if it was given a small amount of additional time. However, this would also increase the solving times. Since the hybrid solver is already slower than both the brute-force solver and the combined solver, increased solving times would only increase the difference between the solvers.
5.4. CONCLUSIONS

5.4 Conclusions

5.4.1 Correlation between clues and solving time

There seems to be a correlation between the number of clues and the solving time for the hybrid AC3-tabu search algorithm, with an increase in the number of clues leading to a decrease in solving time.

5.4.2 Efficiency

The hybrid AC3-tabu search algorithm is less efficient in solving Sudoku puzzles than the brute-force solver and the combined solver, both in the percentage of solved puzzles and the solving times. Also, because of the discussed problems with tabu search, we do not believe it is possible to improve the efficiency of the hybrid solver to such a degree that it can outperform the other two solvers. Considering all this, we conclude that the hybrid AC3-tabu search is not suitable for solving Sudoku puzzles. Therefore, we see no reason to further investigate the algorithm for such use.
Bibliography


Appendix A

Number of found puzzles

<table>
<thead>
<tr>
<th># clues</th>
<th># puzzles</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>49151</td>
</tr>
<tr>
<td>18</td>
<td>243</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>21</td>
<td>1462</td>
</tr>
<tr>
<td>22</td>
<td>7278</td>
</tr>
<tr>
<td>23</td>
<td>8749</td>
</tr>
<tr>
<td>24</td>
<td>17009</td>
</tr>
<tr>
<td>25</td>
<td>30149</td>
</tr>
<tr>
<td>26</td>
<td>9710</td>
</tr>
<tr>
<td>27</td>
<td>2451</td>
</tr>
<tr>
<td>28</td>
<td>1840</td>
</tr>
<tr>
<td>29</td>
<td>126</td>
</tr>
<tr>
<td>30</td>
<td>10101</td>
</tr>
<tr>
<td>31</td>
<td>5</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>2</td>
</tr>
<tr>
<td>35</td>
<td>10001</td>
</tr>
<tr>
<td>36</td>
<td>2</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>10001</td>
</tr>
<tr>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td>10001</td>
</tr>
</tbody>
</table>
Appendix B

Number of solved puzzles

Numbers of clues for which less than 100 puzzles were found have been omitted from the table.

<table>
<thead>
<tr>
<th># clues</th>
<th>Combined</th>
<th>Brute-force</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># solved</td>
<td>% solved</td>
<td># solved</td>
</tr>
<tr>
<td>17</td>
<td>48837</td>
<td>99.36%</td>
<td>32924</td>
</tr>
<tr>
<td>18</td>
<td>237</td>
<td>97.53%</td>
<td>223</td>
</tr>
<tr>
<td>21</td>
<td>1433</td>
<td>97.99%</td>
<td>1421</td>
</tr>
<tr>
<td>22</td>
<td>7259</td>
<td>99.73%</td>
<td>7232</td>
</tr>
<tr>
<td>23</td>
<td>8749</td>
<td>100%</td>
<td>8745</td>
</tr>
<tr>
<td>24</td>
<td>17009</td>
<td>100%</td>
<td>17008</td>
</tr>
<tr>
<td>25</td>
<td>30149</td>
<td>100%</td>
<td>30148</td>
</tr>
<tr>
<td>26</td>
<td>9710</td>
<td>100%</td>
<td>9710</td>
</tr>
<tr>
<td>27</td>
<td>2451</td>
<td>100%</td>
<td>2451</td>
</tr>
<tr>
<td>28</td>
<td>1840</td>
<td>100%</td>
<td>1840</td>
</tr>
<tr>
<td>29</td>
<td>126</td>
<td>100%</td>
<td>126</td>
</tr>
<tr>
<td>30</td>
<td>10101</td>
<td>100%</td>
<td>10101</td>
</tr>
<tr>
<td>35</td>
<td>10001</td>
<td>100%</td>
<td>10001</td>
</tr>
<tr>
<td>40</td>
<td>10001</td>
<td>100%</td>
<td>10001</td>
</tr>
<tr>
<td>45</td>
<td>10001</td>
<td>100%</td>
<td>10001</td>
</tr>
</tbody>
</table>
Appendix C

Average and median solving times

The times in the table are in microseconds. Numbers of clues for which less than 100 puzzles were found have been omitted from the table.

<table>
<thead>
<tr>
<th># clues</th>
<th>Combined</th>
<th>Brute-force</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Median</td>
<td>Average</td>
</tr>
<tr>
<td>17</td>
<td>11 570</td>
<td>258</td>
<td>248 451</td>
</tr>
<tr>
<td>18</td>
<td>110 998</td>
<td>38 492</td>
<td>168 628</td>
</tr>
<tr>
<td>21</td>
<td>86 871</td>
<td>48 348</td>
<td>123 492</td>
</tr>
<tr>
<td>22</td>
<td>45 862</td>
<td>15 610</td>
<td>74 047</td>
</tr>
<tr>
<td>23</td>
<td>3 806</td>
<td>795</td>
<td>20 965</td>
</tr>
<tr>
<td>24</td>
<td>1 841</td>
<td>600</td>
<td>11 403</td>
</tr>
<tr>
<td>25</td>
<td>849</td>
<td>325</td>
<td>7 885</td>
</tr>
<tr>
<td>26</td>
<td>976</td>
<td>435</td>
<td>4 750</td>
</tr>
<tr>
<td>27</td>
<td>754</td>
<td>387</td>
<td>3 280</td>
</tr>
<tr>
<td>28</td>
<td>813</td>
<td>448</td>
<td>2 323</td>
</tr>
<tr>
<td>29</td>
<td>640</td>
<td>343</td>
<td>1 534</td>
</tr>
<tr>
<td>30</td>
<td>142</td>
<td>123</td>
<td>685</td>
</tr>
<tr>
<td>35</td>
<td>121</td>
<td>110</td>
<td>117</td>
</tr>
<tr>
<td>40</td>
<td>116</td>
<td>107</td>
<td>35</td>
</tr>
<tr>
<td>45</td>
<td>110</td>
<td>104</td>
<td>14</td>
</tr>
</tbody>
</table>
Appendix D

Standard deviation for the hybrid solver

Numbers of clues for which less than 100 puzzles were found have been omitted from the table.

<table>
<thead>
<tr>
<th># clues</th>
<th>SD (µs)</th>
<th>SD/Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>291369</td>
<td>61.09%</td>
</tr>
<tr>
<td>18</td>
<td>290325</td>
<td>58.84%</td>
</tr>
<tr>
<td>21</td>
<td>289457</td>
<td>70.09%</td>
</tr>
<tr>
<td>22</td>
<td>280950</td>
<td>79.20%</td>
</tr>
<tr>
<td>23</td>
<td>241311</td>
<td>100.46%</td>
</tr>
<tr>
<td>24</td>
<td>206577</td>
<td>113.52%</td>
</tr>
<tr>
<td>25</td>
<td>150435</td>
<td>143.49%</td>
</tr>
<tr>
<td>26</td>
<td>137947</td>
<td>136.81%</td>
</tr>
<tr>
<td>27</td>
<td>104703</td>
<td>140.24%</td>
</tr>
<tr>
<td>28</td>
<td>92299</td>
<td>143.21%</td>
</tr>
<tr>
<td>29</td>
<td>83113</td>
<td>147.49%</td>
</tr>
<tr>
<td>30</td>
<td>13096</td>
<td>198.46%</td>
</tr>
<tr>
<td>35</td>
<td>1866</td>
<td>104.43%</td>
</tr>
<tr>
<td>40</td>
<td>374</td>
<td>33.01%</td>
</tr>
<tr>
<td>45</td>
<td>240</td>
<td>29.41%</td>
</tr>
</tbody>
</table>