The most difficult Sudoku puzzles are quickly solved by a straightforward depth-first search algorithm

Armando B. Matos
armandobcm@yahoo.com

LIACC
Artificial Intelligence and Computer Science Laboratory
Universidade do Porto

September 9, 2016

Contents

Abstract

In this experimental work we apply a straightforward depth-first-search (SDFS) algorithm to some of the most difficult Sudoku puzzles. Here, "SFDF" means a very simple DFS algorithm using a fixed cell order and *without any algorithmic enhancements* (like constraint propagation).

The computational experiments are divided in two parts:

– A few difficult or very difficult (under the human point of view) puzzles were solved by the SDFS program; the results are analysed in some detail.
– All the 49 151 minimum (17 clues) puzzles in the file prepared by Gordon Royle (of The University of Western Australia) were solved by the same program. Some statistic information is presented. The puzzle requiring more cell visits (about 1.9×10^9 for the exaustive search) is also analised. The median of the execution time is 0.28 seconds.

Most of the puzzles were quickly solved, usually in less than one second. No puzzle requiring more than 3 minutes (in a relatively old computer) was found.
This experiment suggests: (i) a very simple algorithms quickly solves the most difficult Sudoku puzzles, (ii) the difficulty level (for an human) of a Sudoku puzzle has nothing to do with the CPU execution time of the SDFS algorithm that solves it, (iii) the cell nodes with the largest average branching are located at the top of the tree, while the most visited cells correspond to nodes of the search tree that are roughtly located at half height of the tree.

Note. Although the the title of this report is very probably true ("…the most difficult… are quickly solved"), a complete "proof" requires testing the SDFS algorithm for many other inputs, such as all the symmetries of each of the 49 151 puzzles.

1 Introduction

Sudoku is played by hundreds of millions of people worldwide, therefore there is no need to explain the rules (all relevant information can be found on a simple Google search). In this work we consider only the more usual $9{\times}9$ version. and assume that each puzzle has an unique solution. However, we will often measure the CPU time used to find the first solution *and* the CPU time used in an exaustive search (which would detect further solutions, if they existed).

Sudoku is often considered a challenging puzzle whose solution requires human intelligence as well as some knowledge of techniques and tricks. We will see that that is not the case: there are straightforward computer programs that quickly solve puzzles that, under the human point of view, are extremely difficult.

In this short work we first consider (and discard) a really stupid programming technique known as "generate and test". Then we turn our attention to a more efficient, but perhaps equally stupid (under the human viewpoint), technique known as "depth-first-search", DFS, and apply this technique to a few difficult or very difficult Sudoku puzzles (under the human point of view); the results of this experiment are analised. With the same primitive DFS algorithm we also solve all the 49 151 minimum puzzles prepared by Gordon Royle of the University of Western Australia.

As we will see, intelligence, smart techniques, and complex heuristics can be surpassed by a primitive DFS search.

It was shown in [3] that any Sudoku puzzle having an unique solution must have at least 17 clues[1]. We may expect that puzzles with 17 clues ("minimum number of clues") are difficult.

Definition 1 A Sudoku puzzle is *minimum* if it has exactly 17 clues.

The great majority of the puzzles analised in this work is minimum.

2 A brief note on the technique of "generate and test"

Some algorithms for solving Sudoku puzzles that are really unfeasible. For instance, "generate and test" (GT) is a very stupid algorithm indeed. It consists in generating all the possible ways of filling the free cells of a Sudoku puzzle (ignoring the constraints), considered in some fixed order. For each possibility, check if it is a solution.

To get an idea of the computation time needed to find the solution using an algorithm based on the GT technique, suppose that there are 18 clues, so that

[1]In 2006, when [2] was published, it was not known whether that minimum is 16 or 17.

$9 \times 9 - 18 = 63$ free cells remain. Suppose further that the multiset of clues is $\{1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9\}$. Then we have to fill the 63 free cells with seven 1's, seven 2's, ..., and seven 9's. The number of ways of doing this is

$$\frac{63!}{(7!)^9} \approx 0.945 \times 10^{54}.$$

If our very fast computer generates and tests each possibility in one picosecond (10^{-12}sec)[2], the execution time is about 10^{42} seconds. This is MUCH longer than the age of the universe. I don't call this "feasible".

3 "Straightforward" depth-first search (SDFS) technique

By "straightforward" depth-first search (SDFS) we mean the simple depth-first search *without any improvements.* In particular, (i) *no constraint propagation technique is used* [2, page 84] and (ii) the cell order that defines the search tree is fixed. This very primitive algorithm seems (I tested more than 49 000 minimum Sudoku puzzles) to solve in a short time all Sudoku puzzles.

3.1 General comments

We begin with a few considerations about the hardware and software used.

– The computer used was a somewhat old iMac with a 2.66GHz Intel Core Duo CPU, so the reader should have no difficulties in getting significantly faster execution times.

– The programming language used was C. Every program used, say `prog.c`, was compiled with the usual command `gcc -O3 prog.c`.

– The SDFS search order is shown in Figure 1, page 4. The search tree associated with some fixed Sudoku puzzle and some particular cell order may be very large – sometimes it has many millions of nodes.

– The main function of the SDFS program (not to be confused with the C `main` function) can be seen in Figure 2, page 5. This version corresponds to an exhaustive search, that is, a search for every possible solution.

– The programs and examples we used can be obtained from "2016 – Straightforward depth-first search solves difficult Sudoku puzzles" in `http://www.dcc.fc.up.pt/~acm/`

[2]This is a very fast computer, because in 10^{-12} it has to test if the digit placed in *each* one of the 63 free cells of the grid does not occur in the corresponding row, column and 3×3 square.

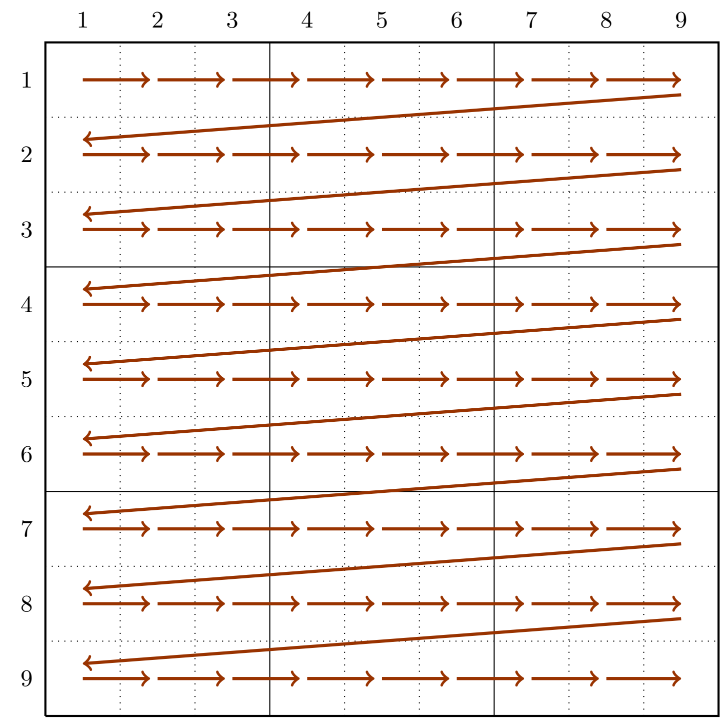

Figure 1. The cell order used to define the depth-first search.

```
void solve(int i,int j){
  int v;
  if(j==9){          // When next cell is in next row:
    j=0;
    i++;
  }

  if(i==9 && j==0){  // When a solution was found:
    display();       // Print the solution, CPU time, branching...
    return;
  }

  nodes++;           // One more node visited
  cnt[i][j]++;       // One more visit to this node

  if(m[i][j]>0){     // When this cell is a clue:
    br[i][j]++;
    solve(i,j+1);
    return;
  }

  for(v=1;v<=9;v++){ // Free cell. Assign all possible values
    m[i][j]=v;
    if(ok(i,j)){     // Test constraints: row, column, 3*3 square
      br[i][j]++;    // One more branch
      solve(i,j+1);
    }
    m[i][j]=0;       // Mark the cell as free
  }
```

Figure 2. The main SDFS function. In this version a call "`solve{0}{0}`" finds all the solutions. A cell containing 0 is currently free.

3.2 Visits and branching

Relatively to a DFS algorithm, we will often mention the “number of visits” to a cell of a Sudoku puzzle.

Definition 2 The *number of visits* that a DFS algorithm makes to some particular cell is the number of call’s made from a tree node corresponding to the previous cell.

In terms of the Prolog box model (due to Lawrence Bird), redo’s are not considered visits of the current cell; however, they are counted as visits of the cell that is called.

We will also use the concept of average branching.

Definition 3 Let n be the number of visits of some specific cell. For $i = 1$, 2,..., n, that is, for each visit to this cell, let b_i be the number of integers that can be assigned to that cell without violating the rules of Sudoku. The *average branching* of that cell is $(b_1 + b_2 + \ldots + b_n)/n$. ⊠

For instance, in the example of Figure 8 (page 14), the top left cell, that is, the cell in which the SDFS starts, has $n = 1$ and average branching of 5 (possible assignments: 2, 3, 4, 7, and 9). If the cell is a clue, its average branching is of course 1.

Properties 1 The following properties are easy to prove.

1. For any Sudoku puzzle P with solution S, any of the following operations T generates a Sudoku puzzle $T(P)$ with solution $T(S)$. We say that the puzzles P and $T(P)$ are mathematically equivalent. The operations are combinations of the following (we transcribe [6]):

   – permutations of the 9 symbols,
   – transposing the matrix (that is, exchanging rows and columns),
   – permuting rows within a single block,
   – permuting columns within a single block,
   – permuting the blocks row-wise,
   – permuting the blocks column-wise,

2. Whenever a Sudoku puzzle has a unique solution, the DFS algorithm visits exactly once both the initial (top left) and the final (bottom right) cells.

3. Using the order shown in Figure 1, page 4, the average branching of the following 21 cells is at most 1: bottom row cells, rightmost column cells, bottom-right block cells. ⊠

In the following section we analyse in some detail a few Sudoku puzzles, while in Section 4 (page 19) we describe the use of the same SDFS algorithm to solve all the 49 151 minimum Sudoku puzzles in the file prepared by Gordon Royle (University of Western Australia) [5, 6].

The main conclusion of this work is perhaps the following:

> Every Sudoku puzzle we tested was quickly solved by the SDFS algorithm. We have not yet found a puzzle whose solution requires more than 3 minutes.

3.3 A few examples analysed in detail

We selected a few difficult or very difficult puzzles and tried to solve them using a straightforward DFS technique, see for instance [4, 7].

3.3.1 Example A: a “very difficult” puzzle, 24 clues

This puzzle[3] is represented in Figure 3, page 9. It is considered very difficult.

The SDFS algorithm found the first solution in 7 milliseconds, while the exhaustive search took only 8 milliseconds!

Example A

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

```
1. CPU time:              0.007 seconds
2. Number of visits:             74 210
3. Total CPU time:        0.008 seconds
4. Total no. of visits:          81 420
```

Figure 3. A “very difficult” Sudoku puzzle is solved almost instantaneously. Lines `1`, `2`, `3`, and `4`, denote respectively: the CPU time used to find the *first* solution, the number of visits to the nodes of the search tree during that search, the CPU time used to find *all* the solutions (there is only one), and the total number of visits to the nodes of the search tree during the exaustive search.

[3] Alastair Chisholm 2008, `www.indigopuzzles.com`, Problem number 6907.

3.3.2 Example B: a “difficult” minimum puzzle

See[4] Figure 4, page 10.

The first solution was found in 0.17 seconds, while the exhaustive search took slightly more (0.23 seconds). This “difficult” puzzle took more time to solve than the “very difficult” puzzle of Example A (Figure 3, page 9).

Example B

				4				9
		2		1				
5							7	3
	9							
		4				1		
			5		7			
		1		2				
					3		8	5

```
1. CPU time:            0.173 seconds
2. Number of visits:        1 857 828
3. Total CPU time:      0.227 seconds
4. Total no. of visits:     2 443 465
```

Figure 4. This Sudoku puzzle is minimum. The meaning of the lines below the grid is explained in Figure 3 (page 9).

[4]Alastair Chisholm 2008, `www.indigopuzzles.com`.

3.3.3 Example B with the order of the rows reversed

If we reverse the order of the rows (9, 8. . . , 1) in Figure 4 (page 10), we get the puzzle in Figure 5 (page 11).

The first 14 cells of the SDFS order (see Figure 1, page 4) do not contain any clue and so that we may perhaps expect a larger branching at the top of the search tree. This may justify the large increase in execution time: it is now about 8 seconds for both algorithms (first solution and exhaustive search), while in the initial example it was about 0.2 seconds!

Example B-inv

					3		8	5
		1		2				
			5		7			
		4				1		
	9							
5							7	3
		2		1				
				4				9

```
1. CPU time:              7.847 seconds
2. Number of visits:         88 178 562
3. Total CPU time:        7.851 seconds
4. Total no. of visits:      88 217 462
```

Figure 5. The puzzle of Figure 4, page 10 with the rows in reverse order. This example is from a Wikipedia entry with the name "Sudoku puzzle hard for brute force", see [8].

Note. In [9] a "brute force" `C++` program was used to solve this example in 21 seconds. "Our" SDFS, in a somewhat old computer, took less than 8 seconds. ⊠

For an human reversing of the order of the rows does not change the apparent difficulty of the problem, but the execution time of the SDFS algorithm can change dramatically.

We could think that a puzzle with no clues in the first 2 rows (18 instead of 14 empty cells) would be still more difficult. However, no such puzzle (with an unique solution) can exist because, for any solution, the swap of those 2 rows generates a different solution, see Properties 1, page 6.

The cells with the largest average branching (see Definition 3 in page 6) are displayed in Figure 6, page 12. They are either at the top of the search tree or in the third column of the puzzle.

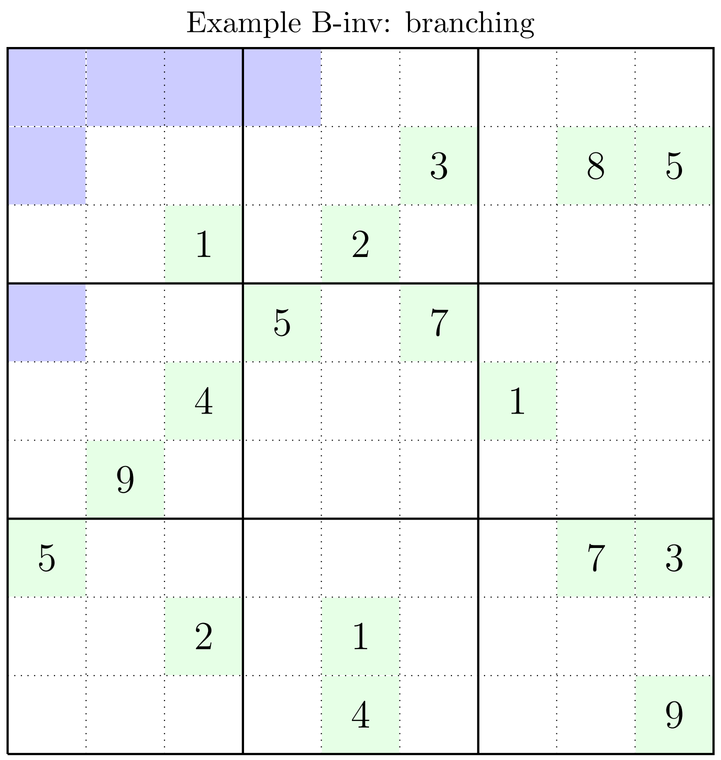

Figure 6. Example from [8]: largest branchings in the search for all solutions. Cells with average branching larger than 3 are coloured blue. The cell at the top of the search tree (top left cell in the figure) has the average branching equal to 7.

3.3.4 Example C: from [3], minimum puzzle

See Figure 7, page 13.

Except for Example A (page 9) all the examples used in this work, including this one, have the minimum number of clues.

Although this problem is probably difficult for humans, it was solved (find the first solution) by the SDFS program in less than 0.2 seconds. The exhaustive search took about 0.3 seconds.

Example C

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

```
1. CPU time:              0.174 seconds
2. Number of visits:          1 811 816
3. Total CPU time:        0.313 seconds
4. Total no. of visits:       3 229 226
```

Figure 7. An example of a minimum puzzle from [3].

3.3.5 Example D: a minimum puzzle from [2]

See Figure 8, page 14. See also Figure 9 (page 15) and Figure 10 (page 16).

This is another minimum puzzle. The first solution was found in less than 0.2 seconds. However, the exhaustive search took about 21 seconds.

Example D

			8		1			
							4	3
5								
				7		8		
						1		
	2			3				
6							7	5
		3	4					
			2			6		

```
1. CPU time:               1.456 seconds
2. Number of visits:          15 050 935
3. Total CPU time:        20.854 seconds
4. Total no. of visits:      215 473 266
```

Figure 8. An example of a minimum puzzle from [2].

Figure 9 (page 15) gives an idea of the average branching for an exhaustive search. Cells with larger branching seem to be at the upper part of the search tree.

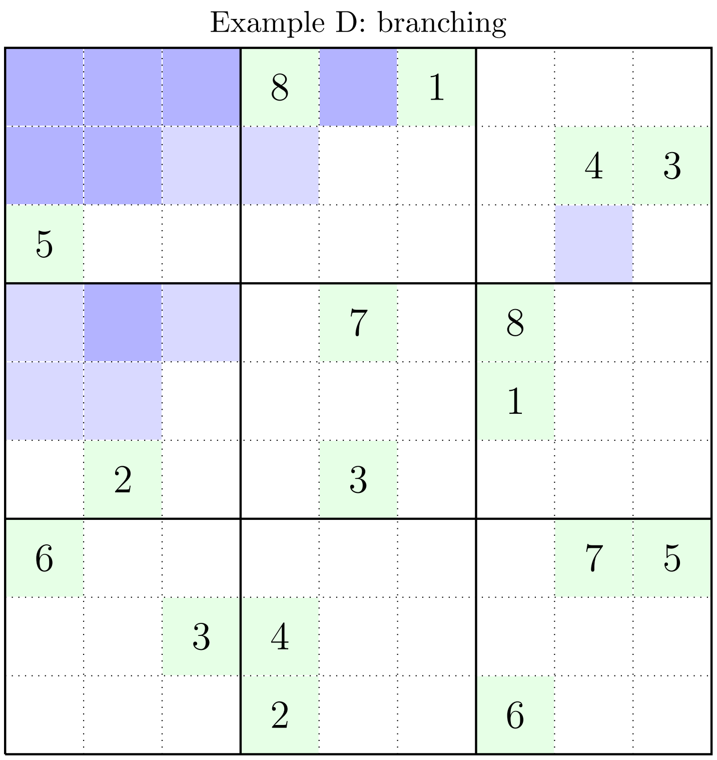

Figure 9. Example D, Figure 8 (page 14). Search for all solutions (not only for the first one): cells with average branching in [2, 3) are coloured light blue. Cells with average branching 3.0 or more are coloured darker blue. Larger branching occurs at the beginning of the search.

Figure 10 (page 16) shows the cells that are visited more often during the (exhaustive) SDFS. They seem to concentrate at half height of the search tree.

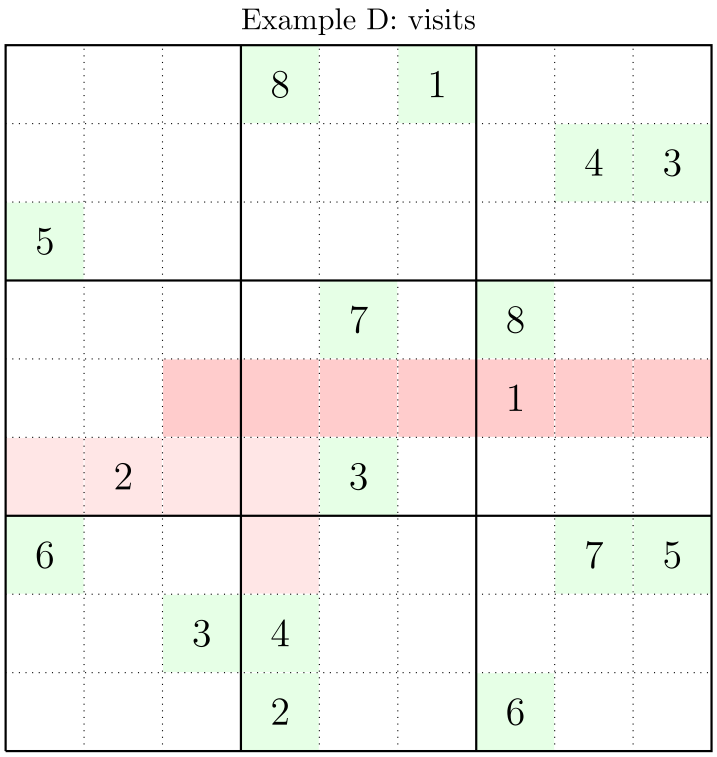

Figure 10. Example D, Figure 8 (page 14). Search for all solutions: cells with more than 5 million visits are red; Cells with more than 10 million visits are darker red.

3.3.6 Example E: "only for the sharpest minds", 21 clues

SDFS seems to solve every Sudoku puzzle, no matter how difficult, in a few seconds at most. Here we present a Sudoku puzzle devised by Arto Inkala. Transcribing "the Telegraph" (July 8, 2016):

> World's hardest Sudoku: can you crack it? Readers who spend hours grappling in vain with the Telegraph's daily Sudoku puzzles should look away now. [See Figure 11, page18]
> The Everest of numerical games was devised by Arto Inkala,
> [`http://www.sudokuwiki.org/Arto_Inkala_Sudoku`]
> a Finnish mathematician, and is specifically designed to be unsolvable to all but the sharpest minds.

The straightforward DFS algorithm solved[5] this very very difficult puzzle in 6 milliseconds! An exhaustive search (looking for more solutions) took 0.2 seconds.

3.3.7 Other difficult puzzles.

Several other difficult Sudoku puzzles were considered. The SDFS algorithm solved most of them in less than 1 second.

For instance, I tested the following "Extreme Unsolveable" puzzles:

```
www.sudokuwiki.org/Weekly_Sudoku.asp?puz=28
First solution: 0.337 seconds,  exaustive search: 0.575 seconds;
```

and

```
www.sudokuwiki.org/Weekly_Sudoku.asp?puz=49
First solution: 0.128 seconds,  exaustive search: 0.404 seconds.
```

⊠

[5]When I noticed that the computer printed the answer almost instantaneously, I thought that all this was perhaps some kind of joke... until I tried to solve the puzzle myself and look at Inkala's "Sudoku page".

Example E: Devised by Arto Inkala
(in “The Telegraph”, “The Sun”, and “Metro”)

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

```
1. CPU time:              0.006 seconds
2. Number of visits:             72 069
3. Total CPU time:        0.234 seconds
4. Total no. of visits:       3 031 696
```

Figure 11. A Sudoku puzzle published in The Telegraph, The Sun, and Metro. It was devised by Arto Inkala. (21 clues). This puzzle is “unsolvable to all but the sharpest minds”, but the SDFS solved it almost instantaneously.

4 SDFS solves all 49151 minimum puzzles (Gordon Royle file)

We first describe the solution of all the 49 151 minimum puzzles (list prepared by Gorden Royle) and then analise in some detail the "most difficult" one (for the SDFS algorithm), that is, the puzzle that took more time to be solved.

4.1 Solving all the 49151 minimum Sudoku puzzles

In this section we describe the use of the same SDFS algorithm (see Figure 2, page 5: "pure" DFS, no algorithmic improvements) to solve all the 49 151 Sudoku puzzles prepared by Gordon Royle (University of Western Australia), [5, 6]. All these puzzles are minimum and have a unique solution. Moreover they are "mathematically inequivalent in that that no two of them can be translated to each other by" the operations[6] mentioned in Properties 1, page 6.

Figure 12 gives some idea of the execution times.

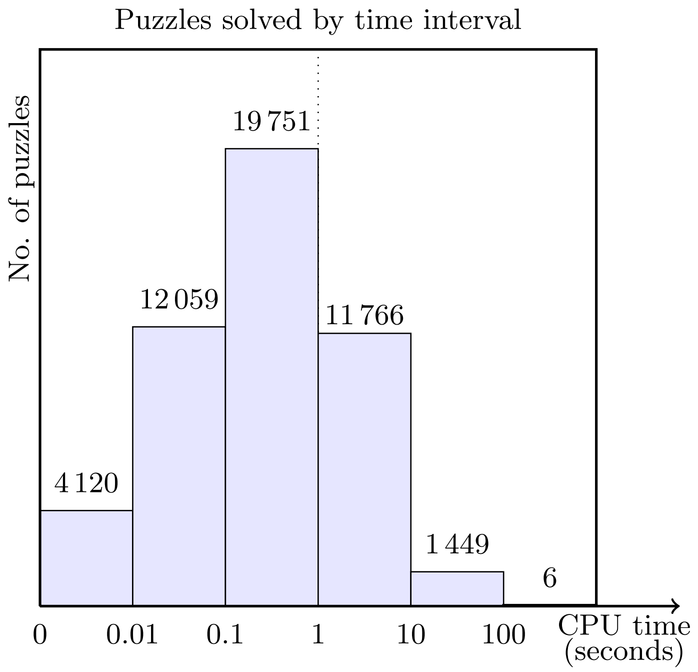

Figure 12. 49 151 minimum Sudoku puzzles: they were all solved by the SDFS algorithm ("first solution" version). More than 73% of the puzzles were solved in less than 1 second (those to the left of the dotted line). For other statistics see Figure 13, page 20.

The SDFS algorithm was used only to search for the first solution.

[6]The application of such operations may of course change the CPU time (and the number of visits).

Some statistics corresponding to the SDFS of the 49 151 minimum Sudoku puzzles are shown in Figure 13, page 20. The *median* (see for instance [1]) of $\{a_1, a_2, \ldots, a_n\}$ is defined here as $a_{(n+1)/2}$ if n is odd and as $(a_{n/2} + a_{1+n/2})/2$ if n is even. Obviously, the "number of visits" column is computer independent.

	Number of visits	CPU time
Median	2 688 418	0.280 seconds
Average	14 983 449	1.490 seconds
Largest	1 553 023 932	162.583 seconds

Figure 13. Statistics corresponding to the solution of 49 151 minimum Sudoku puzzles (by Gordon Royle, The University of Western Australia) by the SDFS algorithm using the "first solution" version.

4.2 “The most difficult” puzzle

Consider the 49 151 Sudoku puzzles in the list [6]. The puzzle with the longest CPU time, about 2 minutes and 40 seconds, is shown in Figure 14, page 21.

The most difficult Sudoku

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

```
1. CPU time:           162.583 seconds
2. Number of visits:     1 553 023 932
3. Total CPU time:     194.891 seconds
4. Total no. of visits: 1 884 424 814
```

Figure 14. “The most difficult puzzle” found. Here, the “difficulty” is measured by the CPU time used by the SDFS algorithm to find the first solution.

Figure 15 (page 22) gives an idea of the cells with more visits (lighter and darker red) and of cells with larger average branching (see Definition 3 in page 6).

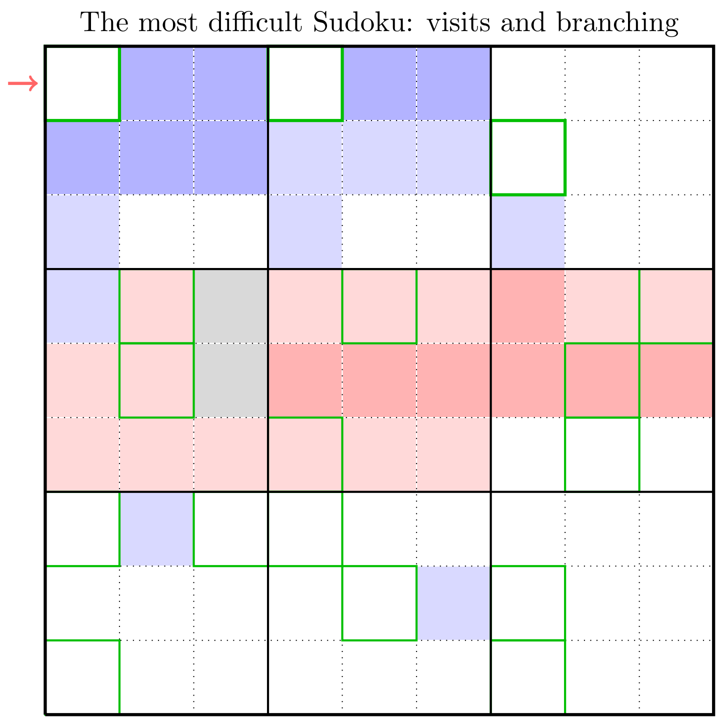

Figure 15. “The most difficult” puzzle: largest number of visits and largest average branching. Here we consider an exaustive tree search by the SDFS algorithm. Clue cells are indicated by a green border (the branching of a clue cell is 1).

- 🟦 Average branching ≥ 3.
- 🟦 Average branching in $[2, 3)$.
- 🟥 Number of visits $\geq 10^7$.
- 🟥 Number of visits in $[10^6, 10^7)$.
- ⬛ Average branching in $[2, 3)$ AND number of visits in $[10^6, 10^7)$.

Recall that the order of the cells that characterises the DFS is left to right, top to bottom (starting cell pointed by the red arrow; see Figure 1, page 4).

The bar graph in Figure 16 (page 24) shows the number of visits to each cell of the grid when the SDFS algorithm makes an exaustive search.

Note in particular that (i) Cells with numbers 0, 72, 73,. . . , 80 are visited exactly once (see also Properties 1, page 6), (ii) the number of visits to a clue cell equals the number of visits to the next cell; for instance, the 8 in the top row (see Figure 14, page 21) corresponds to the cell number 3; cell number 4 has the same number of visits.

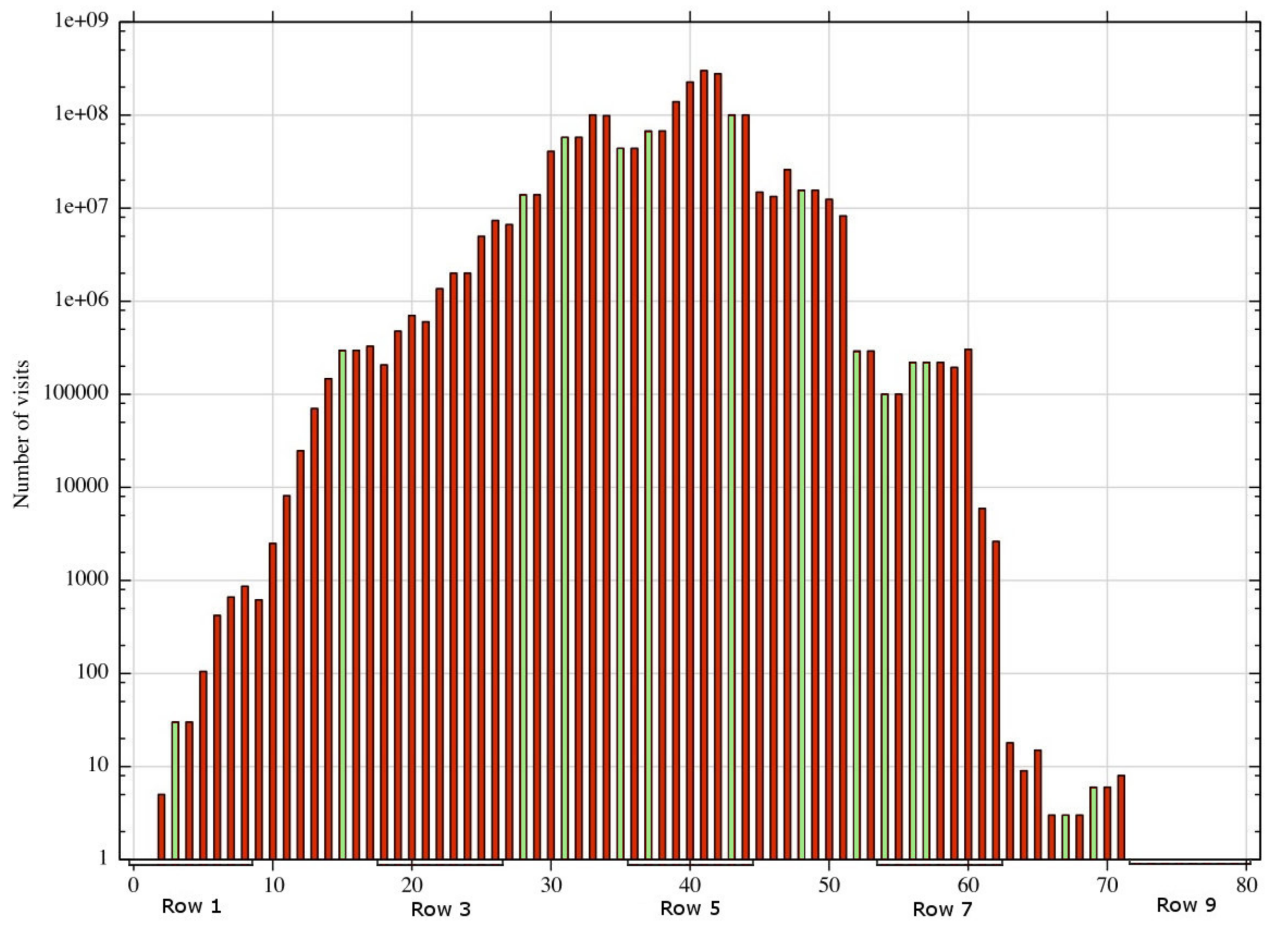

Figure 16. “The most difficult” Sudoku puzzle (see also Figure 13, page 20): exaustive search of the SDFS algorithm. The horizontal numbers, 0 to 80, denote the grid cells with the cell order of Figure 1 (page 4). The height of each bar is the number of visits to each grid cell (that the vertical scale is logarithmic). Green bars correspond to clues (when there is more than 1 visit).

Reverse the row order

If we reverse the row order of "the most difficult" puzzle, the execution time is much shorter. In fact, as shown in the table below, the answer is almost instantaneous!

	Original row order	Reversed row order
CPU time (first solution)	162.6 seconds	0.001 seconds
Visits (first solution)	1 553 023 932	7 744
CPU time (exaustive search)	194.9 seconds	0.010 seconds
Visits (exaustive search)	1 884 424 814	76 181

Compare with the puzzles in Figures 4 (page 10) and 5 (page 11).

More generally, the transformations mentioned in the footnote of page **??** result in other puzzles whose computation times (and number of visits to the search tree) are often drastically different.

Partitioning the Royle set of puzzles

Let us mention a study related to this report which is not described here. Based on Properties 1 (page 6), we define a set of invariant characteristics of a Sudoku puzzle.

By "invariant" we mean that the corresponding value does not change when any of the operations (described in Properties 1, item 1) is applied. As an example, an invariant characteristics is $\langle r_0, r_1, \ldots, r_9 \rangle$, where r_i is the *number* of 3×3 blocks with i clues. For Example A (Figure 3, page 3) this value is $\langle 0, 0, 4, 4, 1, 0, 0, 0, 0, 0 \rangle$.

A set of invariant characteristics correspond to a partition of the Royle set of puzzles [6]. In order to answer the question "how much can a set of invariant characteristics discriminate the Royle set?", we analysed the size of the individual sets of the partition.

5 Tentative conclusions...

We used the straightforward depth-first search (SDFS) to try to solve difficult Sudoku puzzles[7] (The only conclusion that really surprised me was 1).

1. Most of the examples were solved by a straightforward DFS algorithm in less than one second. I was unable to find a puzzle that was not solved in a relatively short time.
2. A puzzle can have the minimum number of clues and be easy (fast) to solve. That is the case of Example C, see Figure 7, page 13.
3. In difficult puzzles, that is, where the computation time is higher, the branching degree is often larger for the cells at top of the search tree. On the other hand, the cells with most "visits" often correspond to nodes at "middle-height" of the search tree[8]. See Figures 15 (page 22) and 16 (page 24). This statement is somewhat vague and needs further experiment and, also, a theoretical justification.
4. The total number of cell visits is roughly proportional to the execution time. A typical proportionality constant is about 10 million to 30 million visits per second.
5. Some fixed, but arbitrary, sequence of cells is initially defined in order to fully characterise the search tree. We used the sequence shown in Figure 1, page 4. The execution time may depend critically on that sequence: compare for instance Example 3.3.2 (page 10) with Example 3.3.3 (page 11).
6. Let t_1 be the computation time used to find the first solution and let t_{ex} be the computation time of an exhaustive search. There are puzzles with

[7]The reader is invited to send me errors and comments regarding this work!

[8]This kind of behaviour is expected, because at the top of the search tree there are few nodes (not to be confused with the puzzle cells), while at the bottom of the tree, that is, at the lower rows, there are few nodes that satisfy the Sudoku constraints (of course, every node of the search tree is visited at most once).

$t_1 \approx t_{\text{ex}}$ (see Figure 5, page 11) and puzzles with $t_1 \ll t_{\text{ex}}$ (see Figure 8, page 14).

7. The level of difficulty (of a Sudoku puzzle) for an human is unrelated to the SDFS execution time. But, perhaps, this should be expected. It seems that an "intelligent being" develops methods and uses facts that allow him to solve the problem with little effort, almost without any trial-and-error. By contrast, the rather primitive SDFS algorithm solves the puzzle using a simple technique that is essentially based on backtracking. Thus, an intelligent method is very different from – almost the opposite of – the method used in SDFS!

References

[1] H. D. Brunk, Mathematical Statistics (2nd Ed.), Blaisdell Publishing, 1965.

[2] Jean-Paul Delahaye, “The science behind sudoku”, Scientific American, June 2006.

[3] Gary McGuire, Bastian Tugemann, Gilles Civario, There is no 16-Clue Sudoku: solving the sudoku minimum number of clues problem, `http://arxiv.org/abs/1201.0749`

[4] Donald E., Knuth, The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms, Addison Wesley Longman Publishing Co. 1997.

[5] Gordon Royle, The University of Western Australia, Minimum Sudoku, `http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php`, July 2016.

[6] Gordon Royle, The University of Western Australia, Minimum Sudoku file, `http://staffhome.ecm.uwa.edu.au/~00013890/sudoku17`, July 2016.

[7] Robert Sedgewick and Kelvin Wayne, Algorithms (4th Ed.), Addison-Wesley, 2011.

[8] `https://en.wikipedia.org/wiki/File:Sudoku_puzzle_hard_for_brute_force.jpg`

[9] `https://github.com/Mathilde94/Sudoku`