Masarykova univerzita
Fakulta informatiky

Difficulty Rating of Sudoku Puzzles: Comparison of Several Techniques

Master Thesis

Martin Křivánek

Brno, 2011

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

In Brno, May 29, 2011
Martin Křivánek

Advisor: Mgr. Radek Pelánek, Ph.D.

Acknowledgement

I would like to thank my advisor Radek Pelánek for his patience, consultations and guidance throughout the whole process, my family for their support, and finally Martin Chmelík for the proofreading.

Abstract

Predicting difficulty of problems is important for good functioning of tutoring and e-learning systems as people enjoy solving problems which are of the right difficulty. Sudoku puzzle is a good candidate for studying this topic – it has simple rules and there is a lot of data on human solving readily available. We provide an overview of algorithmic approaches for solving Sudoku puzzle and try to develop metrics for predicting difficulty for human solvers based on several of them – namely simulated annealing, harmony search and brute force search. These algorithms are also tested for their generalizability on a slightly different problems – predicting order of cell filling during solving Sudoku puzzle and predicting difficulty of a different puzzle, Nurikabe.

Keywords

Human problem solving, difficulty prediction, Sudoku, Nurikabe, constraint satisfaction problems.

Contents

Chapter 1

Introduction

With increasing importance and usage of tutoring and e-learning systems it is becoming more and more important to assess difficulty of problems. These systems can be used for teaching a wide range of subjects, including programming, mathematics, languages and many more. Humans enjoy solving and learning the most when they are served with the problems and tasks of the right difficulty.

When the problems are too easy, solvers become bored, on the other hand, too difficult problem can lead to frustration and refutation of future learning. It is therefore important to be able to predict the difficulty of the problems in advance to be sure to serve solvers with the right problems.

It is not without problems though. Main reason is that humans are different from computers. Not just in the way they look, but more importantly, in the way they think and solve problems.

At first glance it may look as an uneven match. Computers rule great computational power. They are able to go through many different variants and perform complex calculations in a blink.

However, there are problems in which just pure physical power is not enough. Humans, slow as they may seem, have few tricks up theirs sleeve – for example human mind is able to see patterns or divide problem into subproblems with smaller complexity.

The question therefore is how to predict difficulty of the problems for humans by computer.

Sudoku puzzle is a good candidate for studying this problem as it has simple rules and lot of data on human solving is available due to the current popularity of the puzzle and many internet portals which provide opportunity to solve it.

In this work we first define Sudoku puzzle and provide overview of approaches how to solve it algorithmically. Then we describe three different algorithms – simulated annealing, harmony search and brute force search – and evaluation of metrics based on these algorithms for predicting difficulty on extensive set of data on human solving of Sudoku puzzle. Finally we try these algorithms on two different problems – predicting order of cell filling during solving Sudoku puzzle and predicting difficulty of a different puzzle Nurikabe – to test their generalizability.

Chapter 2

Sudoku puzzle

Sudoku is a logic based number placement pencil-and-paper puzzle which gained its popularity in Japan.

First puzzles similar to Sudoku were based on magic squares. Magic square is an $n \times n$ grid filled with an arrangement of distinct integers such that numbers in all rows, columns, and both diagonals sum to the same constant. Puzzles with magic squares with some numbers removed were published already in the end of the 19th century in France.

First modern Sudoku was published in 1979 by *Dell Magazines*. The puzzle was later introduced in Japan by publisher Nikoli in April 1984 with name *Suji wa dokushin ni kagiru* which can be translated as ”the digits must be single”. This was later abbreviated to Sudoku and the puzzle became world-wide famous with many recognized newspaper publishing puzzle daily, and even several live TV shows. World championship in solving Sudoku puzzle and its variants is organized every year.

In this chapter we provide definition of the problem, discuss data on human solving which we have available and provide overview of algorithmic techniques for solving Sudoku puzzle.

2.1 Sudoku Problem

Sudoku problem consists of a given $n^2 \times n^2$ grid divided into $n \times n$ distinct squares partially filled with given numbers and the task is to fill each cell so that the following three criteria are met:

1. Each row of cells contains numbers from 1 to n^2, each exactly once
2. Each column of cells contains numbers from 1 to n^2, each exactly once
3. Each $n \times n$ square block of cells contains numbers from 1 to n^2, each exactly once.

Most common variant is with $n = 3$, therefore 9 cells in each row, column and square block. For future reference, unless we state otherwise, whenever we mention Sudoku puzzle, we mean this variant on a 9×9 grid.

Example of a Sudoku puzzle can be seen in Figure 2.1.

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

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

Figure 2.1: Example of a Sudoku puzzle and its solution.

Generalized variant of Sudoku on $n \times n$ is known to be an NP-complete problem as was proved by T. Yato and T. Seta [14].

2.2 Human Data

The most sure way how to get data about human solving would be to conduct controlled laboratory experiments. Clearly, the main drawback of this approach is the time consuming and demanding nature of it.

But there is already a huge amount of data available due to big and still increasing popularity of online solving of Sudoku puzzle.

Data obtained from web portals may be of a worse quality than from controlled experiments – for example, there is no guarantee that some of the solvers were not distracted during solving, and for sure some were. However, analysis done by R. Pelánek [12] shows that these data are very robust and therefore can be used for evaluation.

We have data from three web portals specializing on Sudoku puzzle: `fed-sudoku.eu`, `sudoku.org.uk` and `czech-sudoku.com`.

Dataset from `fed-sudoku.eu` contains information about 1089 puzzles, all from year 2008. Mean number of solvers is 131 per puzzle. For each puzzle we have the total time needed to complete it by each user. Most users solved many puzzles, therefore data are mostly from experienced and skilled solvers.

Second dataset from `sudoku.org.uk` contains data on puzzles published in years 2006-2009. There was one puzzle per day, in total 1331 puzzles. The mean number of solvers per puzzle is 1307. We have only summary data for each puzzle – numbers of solvers and mean time needed to solve it – no data on individual solvers. There was a significant improvement of human solvers during the first years as they were learning, so to have consistent data we used for evaluation only 731 puzzles from the later period, when the solvers skill become stabilized.

Mean solution time per puzzle on data from `fed-sudoku.eu` is smaller than from `sudoku.org.uk`. Main reason for this difference is probably because the former is used mainly by more experienced solvers and the later by general public and also contains more difficult puzzles.

Last dataset from `czech-sudoku.com` is different from the previous. Amount of puzzles and solvers is smaller, but we have a complete record of each play. Every filling of number and time between fillings are recorded. We have these complete records for about 50 users and 15 puzzles.

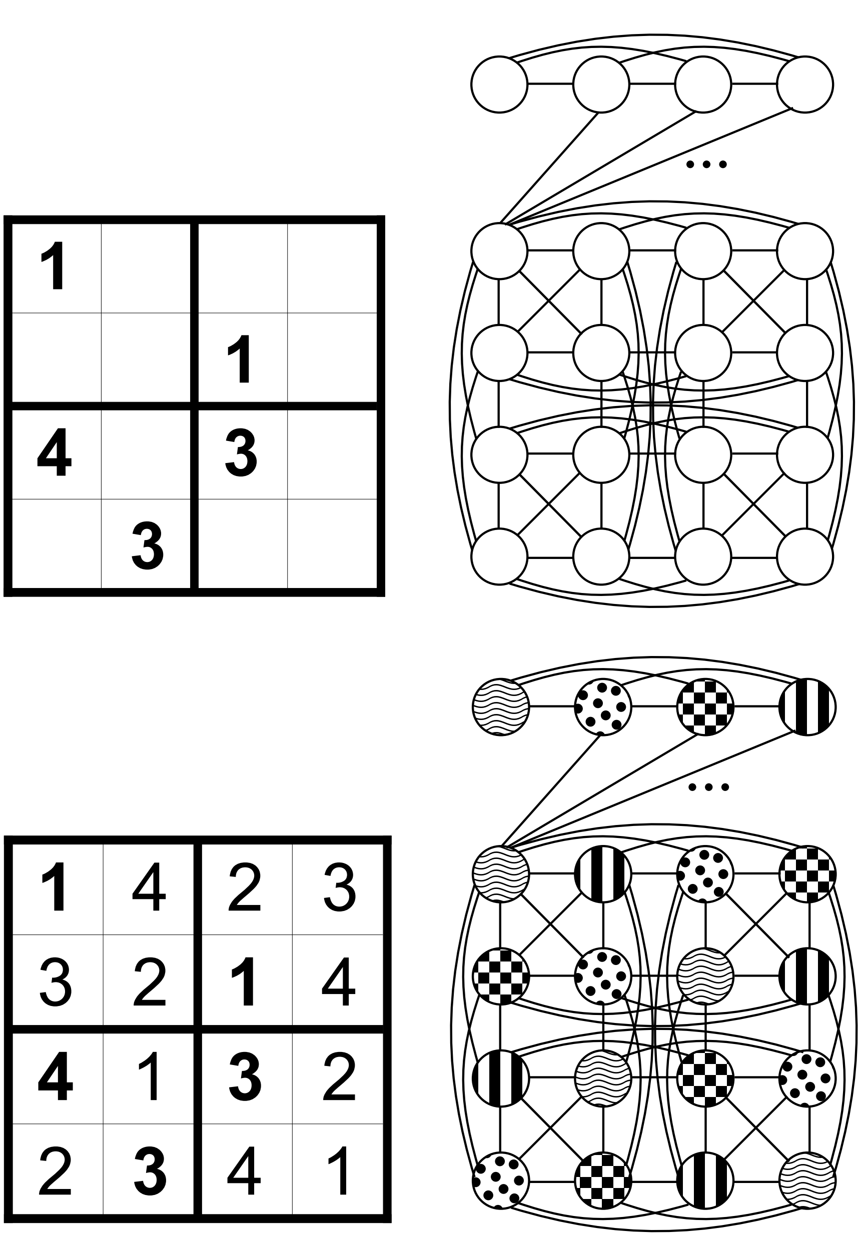
2.3 Overview of Approaches for Solving Sudoku Puzzle

There was already done a research in the field of solving Sudoku puzzle and we attempt to summarize them in this section.

Approaches are mostly adapted from solving Graph Coloring Problem. Aim of Graph Coloring Problem is to find assignments of colors to vertices in a given graph, such that no two adjacent vertices have the same color.

We can transform every Sudoku puzzle to an instance of graph coloring problem by making a vertex of each cell and make edges among all vertices in a same row, column and square block respectively. We need to further add one vertex per each number and connect all these vertices to form a clique. When transforming a Sudoku assignment we connect every cell with a given value to all the vertices in this clique but the one representing the number which was given to the cell. This transformation is depicted in figure 2.2.

We can group all the approaches to several categories – genetic algorithms, local search techniques, and those for solving constraint satisfaction problems.



1			
		1	
4		**3**	
	3		

1	4	2	3
3	2	**1**	4
4	1	**3**	2
2	**3**	4	1

Figure 2.2: Sudoku puzzle of size 4x4 and its solution. Corresponding graph for graph coloring is depicted on the right. Every cell of the puzzle has one corresponding vertex, edges represent contraints – two connected vertices cannot share the same color. One vertex is added for each color and these vertices are all conected to form a clique. Cells which are given in the assignment are connected to all the vertices of this clique but the one representing number which has this cell given.

Genetic Algorithms

Genetic algorithms have in common that there is a population of individuals (partial solutions) and new individuals are created by modification and combining of existing ones.

Swarm Optimization

This approach was described by A. Moraglio and J. Togelius [11]. We create a search space of partial solutions (filled Sudoku grids obeying only the row criterion), and many individuals (around 100) in this space which move around the search space. Each individual can move in two ways:

- Mutation - swap two cells in one row.
- Crossover - two individuals are randomly merged together.

Despite heavy optimization of parameters, this algorithm was unable to solve more than half of Sudoku puzzles it was run on.

Attempt to improve this algorithm was done by T. Mantere and J. Koljonen [9]. There is an addition of a *belief space* – cube $9 \times 9 \times 9$ indicating the "belief" that at a given cell is a given number. This belief space is used when generating new individual.

This variant achieves slightly better results, but still is not able to solve all the puzzles.

Harmony Search

Harmony search is an attempt to simulate the way music players play while playing improvised music and its use to Sudoku was described by Z. W. Geem [2].

We describe Harmony search further and in more detail in its own dedicated section.

Local Search Techniques

While using local search techniques we define a search space of partial solutions – in case of Sudoku usually filled Sudoku grids obeying just one of the criterions, neighborhood of each partial solution – in Sudoku case usually all the partial solutions which we get by swapping two cells in a given unit (row, column, or square area) and fitness function (evaluating how good is particular partial solution).

Algorithm starts in a random point of a search space and move to one of its neighbors. This is repeated until we reach a solution (global minimum of the fitness function).

Algorithms differ in a way we choose the neighbor in which we move in each step.

Tabu Search

Tabu search was described by A. Hertz and D. Werra [4] originally for solving graph coloring problem, but can be used as well for solving Sudoku puzzle.

In each step we move to one of the neighbors with better value of fitness function, chosen randomly. If there is none, we choose randomly any of the neighbors.

In addition we maintain a list of given length of *tabu moves* and algorithm is not allowed to perform a move while the move is in this list. Tabu list is constructed as follows – whenever a number in a cell is changed, we put this cell to a tabu list and forbid changing its value for some time.

This approach is able to solve all the Sudoku problems authors tried it on.

Simulated Annealing

Simulated annealing is inspired by a real process of cooling of melted metal in metallurgy.

During simulated annealing we are allowed to move to a state with higher value of a fitness function with certain probability, which is decreasing over time.

Its use for Sudoku was described by R. Lewis [8]

We describe simulated annealing further and in more detail in its own dedicated section.

Constraints Satisfaction Problem

Sudoku is an example of a constraint satisfaction problem (CSP) as was shown by H. Simonis [13].

Constraint satisfaction problem is defined as a triple of sets $\langle D, X, C\rangle$, where D is a set of variables, X is set of domains – every variable has its finite domain of values – and C set of constraints – every constraint specifies some subset of variables and its relation.

These constraints are of two types – either we use the *all different* constraint, than in case of Sudoku there are 27 constraints, one for each row, column and square block area, or we use only constraints specifying that two cells differ, in that case we have 32 constraints per each row, column and square area, making total of 864 constraints.

Two most common techniques for solving CSP are brute force search using backtracking and constraints propagation. We also describe two other techniques – belief propagation and constraints relaxation.

Brute Force Search

In the brute force search each possible combination of assignment values to the variables is systematically generated and then tested if it satisfies all the constraints.

The search space of this approach is unfeasibly huge, but we can improve it by generating partial assignments, where values are assigned sequentially to the variables. After each added value, we check if any constraint is violated. If yes we stop continuing developing this partial solution and return back. This method is called backtracking.

We describe brute force search with backtracking in its own section.

Constraints Propagation

In constraint propagation we try to find value of a variable by reasoning about constraints. Its application to Sudoku was described by H. Simonis [13] who used quite complicated reasoning techniques. R. Pelánek [12] provides a simple model using just two reasoning techniques.

Every cell has assigned a *candidate set* – a set of values which can be placed into a cell without breaking any of the Sudoku constraints. Then we apply one of the reasoning techniques to reduce the set for some cell. By repeating this steps we incrementally solve the puzzle.

However, application of this reasoning does not always lead to a solution. Therefore it is often combined with backtracking if none of the reasoning techniques can be used to further advance in solving.

This reasoning is close to the way humans solve the puzzle. Several different techniques can be used. The most simple ones are called *Naked single* and *Hidden single*.

Naked single technique can be used for assigning value to a particular cell when all the other numbers are already used in a row, column or square block in which this cell lies.

Hidden single technique is similar – when there is just one cell in a row, column or square block which can contain a particular value as all other placements would violate some rule, this number has to be in that cell.

Many more, complicated, techniques exists but just these two were used in the model.

In every step of the computation one of these techniques is used and applied. If there are more places where this can be done, one is chosen randomly.

In case none of these techniques is applicable, we compute a *refutation score* of all cells. The refutation score stands for number of simple steps (application of simple techniques) which are needed to demonstrate inconsistency of assignment of a particular value to this cell. This is calculated by a randomized sequence of simple steps, therefore refutation score is a randomized variable.

Cell which has lowest refutation score (for all tried Sudoku puzzles there always was at least one cell with a finite refutation score) is easiest to continue solving and number of steps needed for refutation of the value are considered as a difficulty of that single step.

Several metrics for predicting difficulty can be based on this refutation score. *Serate metric* is a maximal difficulty of a used technique, in other words maximal of all minimal refutation scores in each step. *Refutation sum* metric is a mean sum of refutation scores over 30 randomized runs of the model.

Both these metrics achieve a good correlation with mean human solving times – Serate metric has correlation of 0.70 and 0.80 on data from `fed-sudoku.eu` and `sudoku.org.uk`, respectively; Refutation sum metric has correlation 0.68 and 0.83 on these data.

Belief Propagation

This approach was described by T. K. Moon and J. H Gunther [10].

We create a bipartite graph. One partition has 27 vertices and each represents one constraint (of the ″all different″ type), the other one has 81 vertices, each for one cell of the Sudoku grid. Edges are connecting ″cell″ vertices with ″constraint″ vertices if the constraint applies to the cell.

Then the vertices are sending messages to each other:

- ″Constraint″ vertices send a message to all its connected ″cell″ vertices with an information with how high probability this constraint is satisfied if the given cell has a given value (it is a vector of 9 numbers).

- ”Cell” vertices send message to all its connected constraints with information with how high probablity it has a particular value in condition all *other* constraints this cell is connected to are satisfied (it is again a vector of 9 numbers).

Each vertex then update its beliefs.

This approach works best on graph with no cycles. Unfortunately, graph of Sudoku constraints contains quite a lot of cycles. This causes creation of biases and therefore invalid solutions.

Constraint Relaxation

This is a novel technique developed by R. Pelánek [7] inspired by a phase transition which occurs in many constraint satisfaction problems. Phase transition is a name for a steep increase of difficulty when a certain parameter of the problem change from very low, for which it is easy to find a solution, to very high, for which it is easy to prove that there is no solution.

Constraint relaxation technique does not try to solve Sudoku puzzle, instead it provides a direct measure of a given problem difficulty.

Some subset of constraints is removed from the constraints set for Sudoku. With the constraints removed, puzzle can have (and usually has) more solutions.

Hard puzzles have faster growth of number of solutions with removing constraints. This can be explained as that easier puzzles have greater redundancy – therefore even if some of the constraints are removed, there is slower growth of number of new solutions.

Metric for predicting difficulty is based directly on the number of new solutions. It is an average of number of solutions when certain number (in case of the experiments it was 45) of constraints is removed.

This metric has correlation with median time of human solving 0.40 and 0.46 on data from `fed-sudoku.eu` and `sudoku.org.uk`,

We picked up three algorithms, each one from a different category, and tried them on a real data of a human solving.

Chapter 3

Experimental Evaluation of Algorithms on Sudoku

Three different algorithms, every one from one class of approaches – simulated annealing representing local search techniques, harmony search genetic algorithms and brute force search CSP solving approaches – were chosen and tried on a real set of human data.

For each algorithm we describe the general functioning, application of the algorithm to the Sudoku problem and discuss results which the metrics achieved. Every algorithm was run 100 times on both sets of data from `fed-sudoku.eu` and `sudoku.org.uk`.

3.1 Simulated Annealing

Simulated annealing is a randomized technique for improving local optimization algorithms proposed initially by S. Kirkpatrick, C. D. Gellat and M. P. Vecchi [6]. It is motivated by and named after annealing in metallurgy, a technique in which crystalline solid is heated and slowly allowed to cool down until it achieves its most regular possible crystal lattice configuration (i.e. its minimum lattice energy state) and thus is free of crystal defects [3].

In analogy with the real process simulated annealing tries to search for the global minimum of a discrete optimization problem.

Outline of the Algorithm

Algorithm consists of these steps:

1. Select an initial candidate solution ω
2. Select an initial temperature t_0, cooling parameter α, reheat temperature t_{min} and cost function $cost$
3. While cost of our candidate solution is not minimal do
    (a) Choose a new candidate solution ω_{new} from neighborhood of the current candidate solution

(b) If $cost(\omega_{new})$ is less than $cost(\omega)$ or with probability which is based on temperarture t change current candidate solution to the new one: $\omega \leftarrow \omega_{new}$ – this probability allows to get out of a local minimum which would otherwise prevent us in reaching the global minimum – and decrease temperature: $t \leftarrow \alpha t$

(c) If temperature t is lower than t_{min} and we still have not find the optimal solution we need to reheat the whole process and set temperature t back to t_0 – this allows us to "start again" as we can now much freely move to the candidate solutions with higher costs

Application of the Simulated Annealing to the Sudoku Problem

Our application of simulated annealing to Sudoku is based on the approach described by Rhyd Lewis [8].

Candidate solutions

As our candidate solutions we consider completely filled Sudoku grids which obey this two rules – given numbers are unchanged and fixed and in each square block, numbers from 1 to 9 are present.

Thus only the remaining two Sudoku criterions – unique numbers in rows and columns have to be met.

Neighborhood candidate solution

With a given candidate solution, set of neighborhood candidate solutions consists of such solutions in which two numbers within one square block, neither of them given, are swapped.

Therefore, selection of a new candidate solution is done in this way:

1. Choose randomly one square block.

2. In this square, choose randomly two non-given positions.

3. Swap numbers on these positions.

Cost function

Because our candidate solutions always meet the square block criterion, we have to measure just the violation of rows and columns criterions.

In our approach we count for each row and column, how many numbers from 1 to 9 are *not present*. The total sum of these values is the cost of a given candidate solution. You can see an example of computation of cost function in Figure 3.1.

Obviously, an optimal solution to the Sudoku puzzle will have a cost of zero.

1	3	7	2	9	6	2	7	6	3
2	4	8	1	5	3	4	8	9	2
9	5	6	4	7	8	5	3	1	1
3	2	5	1	2	9	5	6	7	2
4	1	9	3	4	6	3	2	9	3
7	8	6	8	5	7	1	8	4	3
6	2	8	6	3	9	8	4	1	2
7	4	5	4	1	7	5	6	3	3
9	3	1	8	2	5	7	9	2	2
2	3	3	3	1	3	2	2	2	**42**

missing numbers

missing numbers

cost

Figure 3.1: Example of computation of *cost* of a particular candidate solution (only square block area criterion of Sudoku is met). We count number of missing numbers from 1 to 9 in each row and column and sum it up.

Temperature and probability

As noted in the outline of the simulated annealing algorithm, in each step we choose one of the neighbors of our current candidate solution, compute its cost and a) if the cost is lower, move to this candidate solution or b) if the cost is higher, move to this candidate solution with probability $e^{-\delta/t}$, where δ is the proposed change in the cost and t is the current temperature.

As we can see, the lower the change of cost, the higher the probability of change is and with decreasing temperarature we have lower chance of changing into a candidate solution with higher cost – system is getting into its local minimum over time.

After the change of the current candidate solution, we lower the actual temperature:

$$t \leftarrow \alpha t$$

where $\alpha \in (0,1)$ is a cooling rate which sets how fast we cool the temperature.

To ensure we do not get stuck in a local minimum which is not an optimal solution, when temperature gets below specified minimal temperature t_{min} and we still have not find an optimal solution, we set temperature back to the initial temperature. This is called a *reheat*.

Example of a run of the simulated annealing algorithm on a Sudoku problem with one reheat can be seen in figure 3.2.

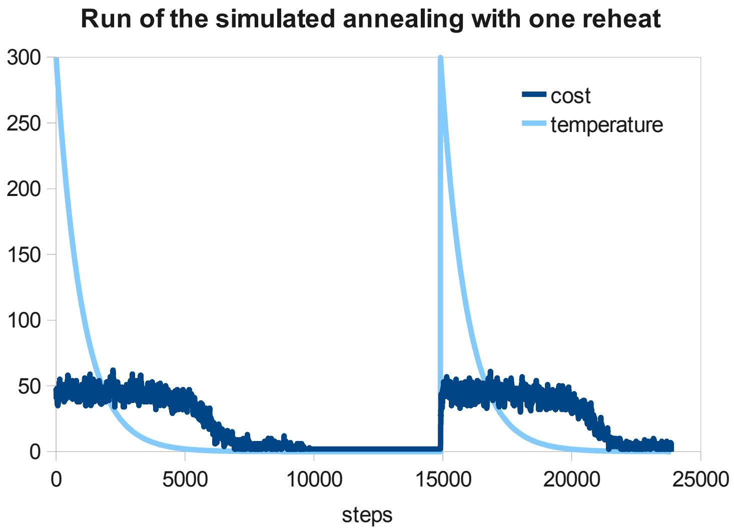


Figure 3.2: Example of one run of the simulated annealing algorithm on a Sudoku puzzle with one reheat. Changing of temperature and cost depending on number of steps is shown.

Evaluation of the Algorithm

Sensitivity to parameters setting

Algorithm has several parameters – initial temperature, cooling rate and minimal temperature for reheat – and choice of them influence the results. Therefore we include discussion of impact of each parameter.

Too high initial temperature just add a constant number of steps to each execution – we are longer just swapping randomly content of cells. On the other hand, too low value can prevent the system to get out of a local minimum. Initial temperature of value 40 seems to be useful – allowing around 80 % of swaps that increase cost.

Too high minimal temperature will not allow the system to cool down enough and to finish by finding an optimal solution. Too low will result again in constant increase of steps for each run as we will wait longer in a local minimum before reheating the system again.

Thus, if both initial and minimal temperature are set a bit on "the safe side", we just get a constant increase of steps but it does not differ in respect to correlation with difficulty for humans.

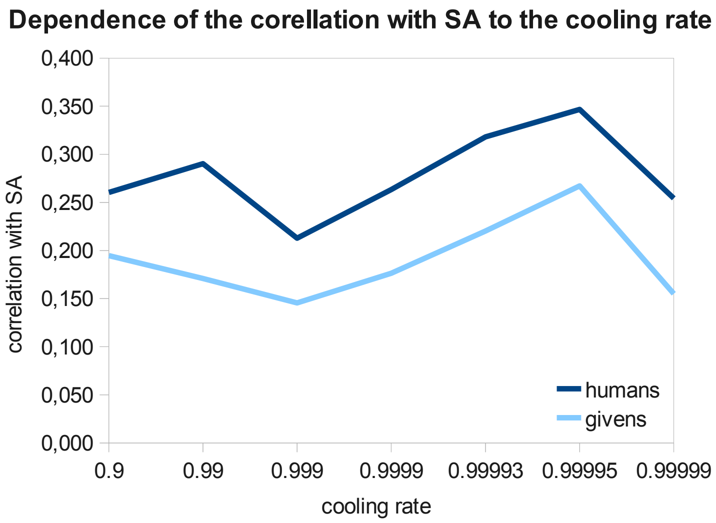


Figure 3.3: Correlation of the simulated annealing algorithm with human median time and number of given cells in an assignment on puzzles from `fed-sudoku.eu` server for different values of a cooling rate. Best results were achieved for the value of cooling rate 0.99995.

Right setting of the cooling rate is the most complicated. Cooling rate determines how fast will the system cool down from a high temperature. Obviously, the lower the cooling rate, the more steps we need to perform. But impact to correlation to median time of human solving is interesting. See Figure 3.3 for the results with different settings. Best results were achieved with slow cooling – cooling rate around 0.99995.

Experimental Results

On each Sudoku puzzle from both data sets we performed 100 runs of simulated annealing algorithm, each time with a different random seed.

Correlation of median of number of steps of simulated annealing algorithm on each puzzle with median time of human solvers is 0.38 for `fed-sudoku.eu` and 0.39 for `sudoku.org.uk` data.

Correlation of median human time and median number of steps for data from `fed-sudoku.eu` can be seen in figure 3.4.

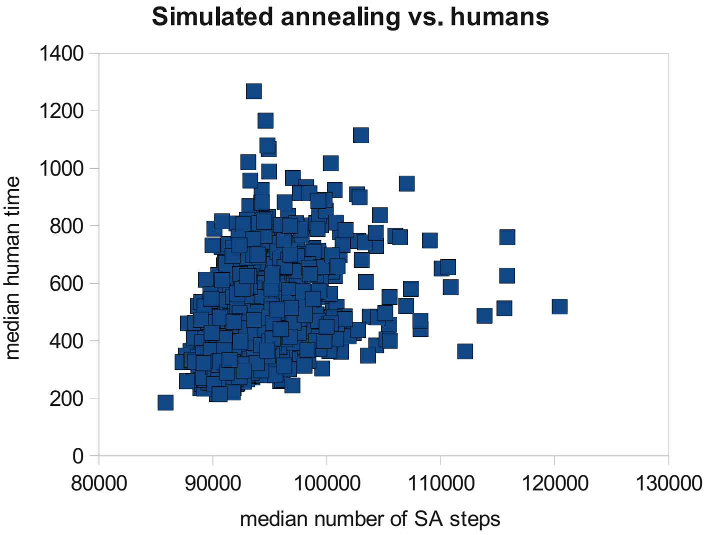


Figure 3.4: Dependency of median time for humans solvers and median number of steps of the simulated annealing algorithm on a particular Sudoku puzzle

3.2 Harmony Search

Harmony search is a musicians' behavior inspired evolutionary algorithm developed by Z. W. Geem, J. H. Kim and G. V. Loganathan [15]. It tries to mimic behavior such as memory consideration, pitch adjustment, and random consideration.

It was applied by the same authors to various optimization problems such as water network design, traffic routing, satellite heat pipe design and music composition.

Outline of the Algorithm

We create several instances of candidate solutions and we repeat following steps:

- Create a new candidate solution - with probability h put to a given cell number from one in the improvisation memory (randomly chosen) and randomly change by "pitch"; with probability $1 - h$ put there a random number.

- Evaluate the new candidate solution by a fitness function, if it is better then the worst current solution, they are switched.

Algorithm terminates when the solution is reached.

Application of the Harmony Search to the Sudoku Problem

Candidate Solution

Candidate solution is a completely filled grid 9×9 with numbers from 1 to 9 and cells which were given in an assignment contain the assigned number. All other cells are however filled with no condition, therefore candidate solution may violate one or more of the Sudoku rules.

Harmony Memory

Candidate solutions are stored in a Harmony Memory. This memory holds certain number of candidate solutions, called Harmony Memory Size (HMS). It is a parallel to how many musicians are playing together.

Execution

In the beginning HMS candidate solutions are randomly generated and stored in harmony memory.

In each following step a new candidate solution is improvised using some of the tree mechanisms: random selection, memory consideration, and pitch adjustment. Finally, this new candidate solution is evaluated by fitness function.

$$\mathbf{x^{NEW}} = \begin{bmatrix} x_{11}^{NEW} & x_{12}^{NEW} & \cdots & x_{19}^{NEW} \\ x_{21}^{NEW} & x_{22}^{NEW} & \cdots & x_{29}^{NEW} \\ \vdots & \cdots & \cdots & \cdots \\ x_{91}^{NEW} & x_{92}^{NEW} & \cdots & x_{99}^{NEW} \end{bmatrix}$$

Random Selection Value of x_{ij}^{NEW} can be chosen randomly from range $1..9$ with probability of $(1 - HMCR)$, where $HMCR$ is a parameter of the algortihm and the abbreviation stands for harmony memory considering rate.

$$x_{ij}^{NEW} \leftarrow x_{ij},\ x_{ij} \in \{1, 2, ..., 9\},\ w.\,p.(1 - HMCR)$$

Memory Consideration With probability $HMCR$ the new value is chosen from a randomly chosen candidate solution stored in harmony memory.

$$x_{ij}^{NEW} \leftarrow x_{ij},\ x_{ij} \in \{x_{ij}^1, x_{ij}^2, ..., x_{ij}^{HMS}\},\ w.\,p.HMCR$$

Pitch Adjustment When a new value is obtained using memory consideration it can be further adjusted and moved to neighboring values (plus or minus one) with a probability PAR, another parameter of the algorithm. It cannot move over the upper limit (9) or bellow lower limit (1).

$$x_{ij}^{NEW} \leftarrow \begin{cases} x_{ij}^{NEW} + 1,\ w.\,p.\ PAR \times 0.5 \\ x_{ij}^{NEW} - 1,\ w.\,p.\ PAR \times 0.5 \\ x_{ij}^{NEW},\ w.\,p.\ 1 - PAR \end{cases}$$

Fitness Function Newly created candidate solution $\mathbf{x^{NEW}}$ is compared with those stored in harmony memory. For this purpose fitness function is used. It has this form:

$$f(\mathbf{x}) = \sum_{i=1}^{9} \left| \sum_{j=1}^{9} x_{ij} - 45 \right| + \sum_{j=1}^{9} \left| \sum_{i=1}^{9} x_{ij} - 45 \right| + \sum_{k=1}^{9} \left| \sum_{(l,m) \in B_k} x_{lm} - 45 \right|$$

where x_{ij} is a value of cell at row i and column j and B_k is set of coordinates for a square block k.

Because properly filled Sudoku puzzle contains all numbers from 1 to 9 in each row, column and square block, sum of these numbers is always 45. Therefore fitness function of a properly filled Sudoku puzzle is zero.

However if value of fitness function of a particular candidate solution is zero, it does not guarantee that candidate solution is also a properly filled Sudoku puzzle. This will be discussed in the evaluation section.

If value of fitness function of the newly created candidate solution is lower than value of fitness function of the worst candidate solution stored in harmony memory, this worst stored candidate solution is replaced in memory by the newly created one.

Evaluation of the Algorithm

Original authors tried Harmony Search on two Sudoku puzzles. One fairly easy, with more than half of cells filled already in assignment. Second one, called ″hard″ by authors, but of moderate difficulty in comparison with normal Sudoku puzzles.

Algorithm is able to solve the easy one, but when solving the hard one, it finishes in a local minima.

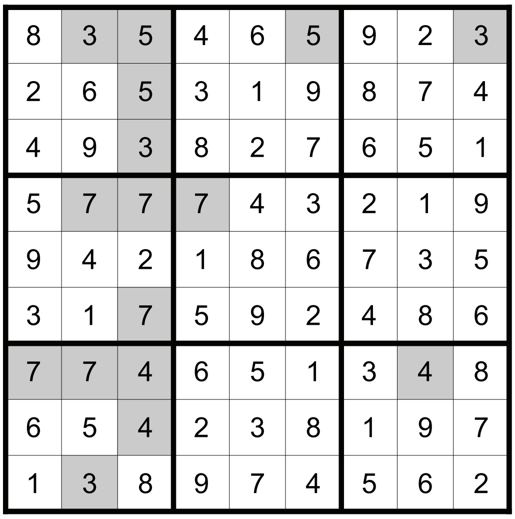
In our experiments, the algorithm was able to solve just very easy puzzles. No puzzle from `fed-sudoku.eu` dataset was solved, even when trying many different combinations of parameters harmony memory size, harmony memory considering rate and probability of pitch.

Algorithm, as presented, has several problems.

First problem is the fitness function. Authors state: ″It should be noted that, although the sum of each row, each column, or each block equals 45, it does not guarantee that the numbers 1 through 9 are used exactly once. However, any violation of the uniqueness affects other row, column, or block which contains the wrong value jointly.″

However, it is still possible that filled Sudoku grid, in which each row, column and block sums to 45, but violate some of the uniqueness rules, exists. Example of such filling is shown in figure 3.5. The simplest such filling is when all cells are filled with number 5 – in that case clearly all rows, columns and square blocks sum up to 45 but it is not a valid Sudoku solution.

Therefore we decided to use another fitness function. We used the same as is used in simulated annealing. It calculates how many numbers from 1 to 9 are missing in each row, column and square block and sums all these



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

Figure 3.5: Example of filled Sudoku grid, in which each row, column and block sums to 45, but nevertheless it is not a valid Sudoku solution. Grey background highlights violation of the Sudoku rules.

numbers together.

Obviously, the proper solution has value of this new fitness function zero and what is more important, it works also the other way around. When value of fitness function of a candidate solution is zero, this candidate solution is a proper solution as it does not violate any of the Sudoku rules.

Nevertheless, even with this new fitness function, computational power of the algorithm did not improve.

We think it may be mostly because algorithm tries to do small improvements by pitch adjustments which are not particularly useful, because in Sudoku puzzle, change even of plus or minus one in one number can make a big difference in a value of the fitness function.

As the algorithm did not solve any puzzles, we tried to find if it is possible to still use it to predict difficulty for humans.

We used a simple metric – the cost of the best candidate solution in a harmony memory after 10 000 algorithm steps. Harmony memory had size 50, harmony memory considering rate was 70% and probability of pitch 10%. We performed 100 runs on all puzzles and took median of this value.

Correlation with median times for humans is 0.18 on `fed-sudoku.eu` data and 0.22 on `sudoku.org.uk`.

Correlation of median human time and median cost of the best candidate solution after 10 000 steps on data from `fed-sudoku.eu` can be seen in figure 3.6.

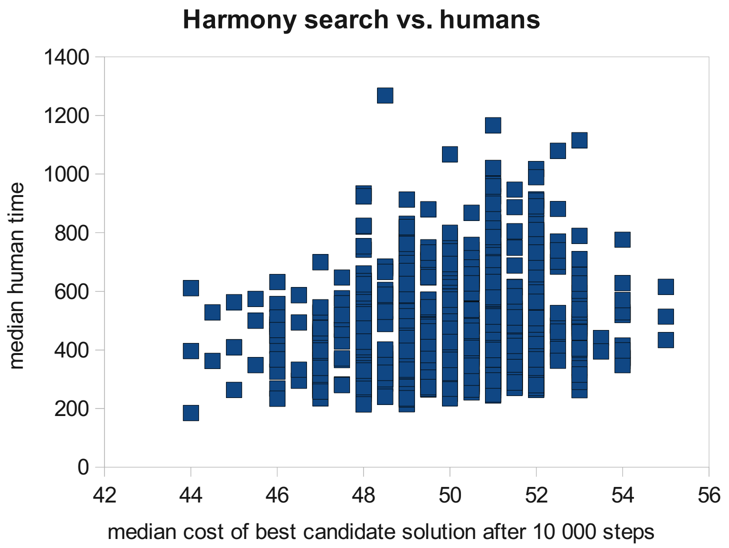


Figure 3.6: Dependency of median time for humans solvers and median cost of the best candidate of the harmony search algorithm.

3.3 Brute Force Search

Brute force search is an algorithm performing an exhaustive enumerating of all possible candidates for solution and checking whether they satisfy all problem conditions. Therefore it is an approach which can be used for a great variety of problems.

Search space even for a seemingly small Sudoku grid consisting of 81 cells is enormous. If we consider all possible grids filled with numbers there would be more than 10^{52} candidate solution which we would need to check. That is more than any computer can compute in a reasonable time.

Therefore, we need to reduce the search space. Two methods are used in order to achieve this. First of them is to cut out branches which cannot lead to any proper solution. As soon as we can recognize them algorithm stops exploring this branch and backtracks.

Another useful way how to find the solution fast is to reorder the search space. The aim is to consider candidate solutions with higher change of being the proper solution first.

Both these methods are problem specific and the speed of the algorithm greatly depends on the choice.

Outline of the Algorithm

The Algorithm is recursively trying to fill values into cells. The non-fixed cells are empty at the beginning of the algorithm.

Algorithm works in steps. In each step we suggest a value to fill in the next cell:

- If we do not have yet assigned the next cell in which to try filling we choose randomly one of those which currently have the least number of available numbers to fill into. Then we add this cell into our ordered list of cells.
- Check if suggested value is allowed in the cell (it does not violate any of the Sudoku rules). If not, we backtrack.
- If value does not violate any rule, we write it into the cell. If we filled whole grid, we have found a solution and algorithm terminates. If we still have part of the grid unfilled, we run recursively this algorithm on the next cell suggesting all the values from 1 to 9.
- If we have not find a solution, we empty the cell again and backtrack.

We measure the length of execution of this algorithm in number of these steps performed.

Both heuristic – choosing the next cell we try to fill – and cutting of branches – checking if newly added number does not violate any Sudoku rules – have to be used in order to achieve reasonably fast execution. If we choose the order randomly or check validity of the solution when the whole board is filled, computation times are enormous. It takes several hundred millions of steps to find a solution.

Choosing the cell which currently has the lowest number of possible values which can be filled in without violating any of the Sudoku rules greatly reduces the computation time sometimes to just a few hundred steps. Also it mimics the behavior of humans when solving the Sudoku puzzle.

Evaluation of the Algorithm

As expected, this straightforward approach does not yield too good results.

On a data from fed-sudoku.eu the correlation coefficient between median of number steps of brute-force algorithm from 100 runs was 0.16, on a data from sudoku.org.uk slightly higher – 0.25.

Even the change of median for minimum, maximum, or spread does not yield any better results.

Correlation of median human time and median number of steps for data from sudoku.org.uk can be seen in figure 3.7.

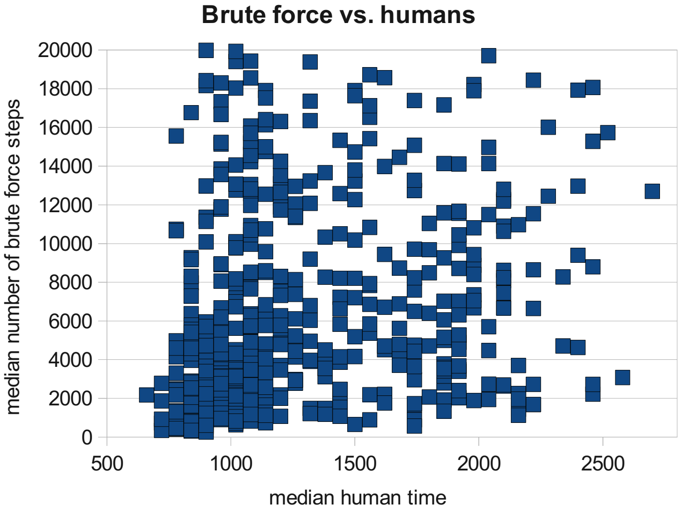


Figure 3.7: Dependency of median time for humans solvers and median number of steps of the brute force algorithm on a particular Sudoku puzzle

3.4 Comparison of the Metrics

In a table 3.1 you can see a comparison of correlation coefficients among different difficulty rating techniques.

As can be seen, best results are achieved with a model mimicking human behavior (both serate metric and refutation sum are based on constraint propagation). Simulated annealing and constraint relaxation achieve worse results and even if based on completely different approaches, perform similarly.

It seems that genetic algorithms in general are not suitable for this kind of problems as neither harmony search nor swarm optimization are able

	fed	org
brute force	0.16	0.25
harmony search	0.18	0.22
simulated annealing	0.38	0.39
constraint relaxation	0.40	0.46
serate metric	0.70	0.80
refutation sum	0.68	0.83

Table 3.1: Correlation coefficients among difficulty rating metrics and human median time in an ascending order.

to solve the puzzle. Also metric based on the best partial solution found after certain number of steps does not give a good prediction of a problem difficulty.

Brute force search, as expected, is not a good predictor either.

Chapter 4

Generalizability of the Algorithms

As the primary goal is not just to predict difficulty of Sudoku puzzles, but to predict difficulty of problems for humans in general, we try all the algorithms on a different problems to test their general applicability.

The two chosen problems are closely related. First of them is to predict order in which cells of a particular Sudoku puzzle are filled during solving them by a human solver. The other one is predicting difficulty of a different pen-and-paper puzzle Nurikabe.

4.1 Predicting Order of Cell Filling

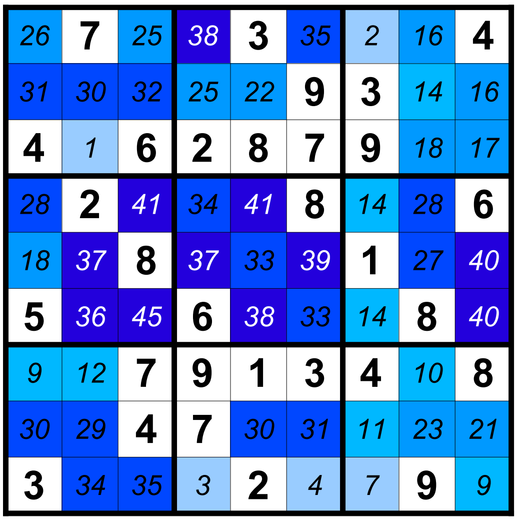
Even though it may seem strange at first, apart from predicting difficulty of a particular Sudoku puzzle, it is useful to be able to predict order in which individual cells will be filled.

This finds use mostly in online Sudoku tournaments, in which organizers want to prevent solvers from cheating – solving the puzzle by a computer solver. In that case cells are filled in an unusual way. Even if the solvers try to fill the cells in a random order, so it is difficult to spot by eye, it is still different from the order, in which the cells would be filled if solved properly.

Of course, there is more than one way how to solve particular puzzle. But still, some cells are likely to be filled sooner and some later. Example of this order of filling can be seen in figure 4.1.

As Harmony Search is not able to solve our Sudoku puzzles, we did not use it for this experiment. Therefore we tried only simulated annealing and brute force search.

In each run we make an order of cells according to a metric depending on an algorithm. For each algorithm, we performed 100 runs on 10 different Sudoku puzzles from web portal `czech-sudoku.eu` for which we have a detailed data about human solving efforts including logs from every solving with order of filled cells.



26	7	*25*	*38*	3	*35*	*2*	*16*	4
31	*30*	*32*	*25*	*22*	9	3	*14*	*16*
4	*1*	6	2	8	7	9	*18*	*17*
28	2	*41*	*34*	*41*	8	*14*	*28*	6
18	*37*	8	*37*	*33*	*39*	1	*27*	*40*
5	*36*	*45*	6	*38*	*33*	*14*	8	*40*
9	*12*	7	9	1	3	4	*10*	8
30	*29*	4	7	*30*	*31*	*11*	*23*	*21*
3	*34*	*35*	*3*	2	*4*	*7*	9	*9*

Figure 4.1: Order in which human solvers filled cells in one particular Sudoku puzzle. The lighter blue the field is, the sooner it was filled. Number indicates median order from all the steps in which this cell was filled by human solvers. White fields are the original assignment of the puzzle.

We take median of order of these algorithm dependent metrics for each cell from these 100 runs and compared with order in which human solvers filled the cells.

Simulated Annealing

We try to predict order in which cells will be completed by human solvers by remembering the last step certain cell was swapped. At the end of the run of the simulated annealing algorithm the cells are ordered by this last swap.

Correlation between the median of orders predicted by simulated annealing and median of human order can be seen in table 4.1.

01	02	03	04	05	06	07	08	09	10
0.58	0.43	0.27	0.42	0.58	0.32	0.72	0.50	0.69	0.71

Table 4.1: Correlation coefficients between median of simulated annealing predicted order of cell completion against median human order. The puzzles are ordered from the easiest to the hardest

As can be seen from the table, better results are achieved on harder puzzles. This is mostly because on easier puzzles there are more orders how to fill the cells, while on the harder ones, there is less space for differentiation. This difference is illustrated in figure 4.2.

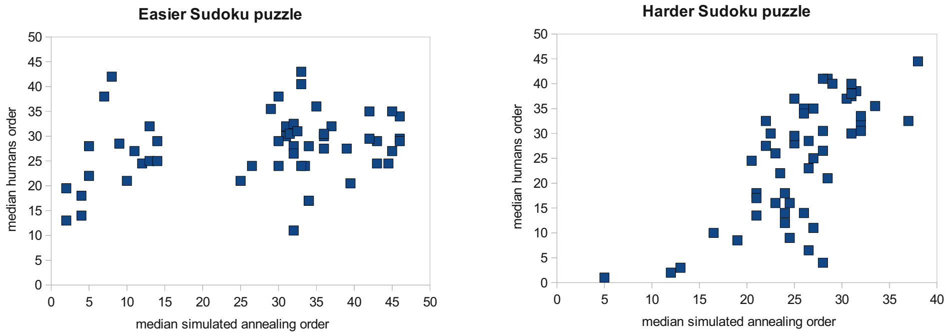


Figure 4.2: Order in which simulated annealing predicted filling of cells for easier and harder Sudoku puzzle.

Brute Force Search

We take the order in which cells were explored by the backtracking algorithm (starting from those with least number of possible values that can be filled into them).

In table 4.2 can be seen the results for all the Sudoku puzzles.

01	02	03	04	05	06	07	08	09	10
0.33	0.18	0.14	-0.11	0.22	0.01	0.39	-0.23	0.50	0.38

Table 4.2: Correlation coefficients for order of cell completion against human order. The puzzles are ordered from the easiest to the hardest

It can be seen that this straightforward metric is not a good predictor of order of cell completion and does not yield any useful results. Correlation of brute force prediction and median of human solving is shown on two examples – puzzle 08 and 09 – in figure 4.3.

4.2 Aplicability to Nurikabe

Nurikabe puzzle, another example of many popular pencil-and-paper puzzles originating in Japan, is a suitable candidate. Nurikabe puzzle is not on the first sight too different from Sudoku – both are played on a rectangular

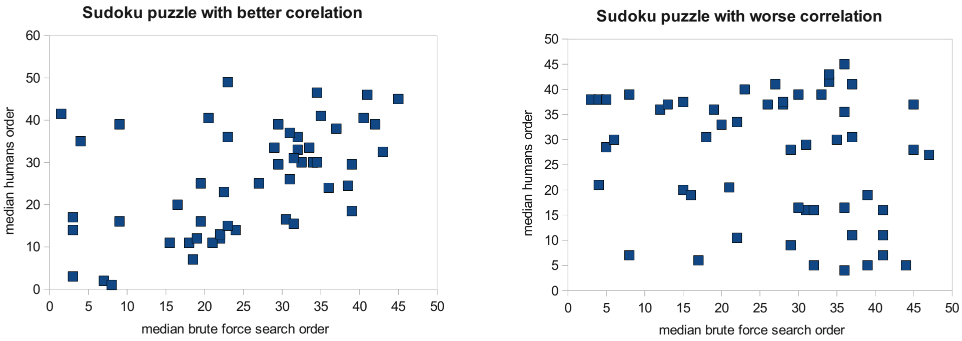


Figure 4.3: Order in which brute force search predicted filling of cells – example of better (puzzle 09) and worse (puzzle 08) correlation

grid with some cells prefilled with natural numbers. They however differ in their structure. Sudoku has just local constraints, imposing some rules to a certain subset of the grid. On the contrary, in Nurikabe there are some global constraints, telling us something about the whole puzzle.

We first describe Nurikabe puzzle, than look in what modifications have to be made to the algorithms and finally sum up with experimental results.

Nurikabe Puzzle

Nurikabe puzzle was first published in the Japanese puzzle magazine Nikoli at March 1991. Its name means *an invisible wall* in Japanese.

The puzzle is played on a rectangular grid with some cells labeled with natural numbers. The goal of the puzzle is to fill cells with black and white color such that they follow certain set of rules. For easier understanding, we will call white cells *islands* and black cells *river*.

These rules have to be fulfilled:

- a cell containing number has to be white (is part of an island).
- a natural number in a cell defines a number of connected white cells – connection is made only horizontally or vertically, diagonal cells are not connected. Every area of white cells contains only one natural number in it and is surrounded by black cells. In other words number defines the size of an island and island has to be surrounded by the river.
- All black cells are linked to one connected area. There is just one river.

- There cannot be any 2-by-2 blocks of black cells. These black blocks are called *pools* and there cannot be any pools in the river.

Example of a Nurikabu puzzle and its solution can be seen in figure 4.4.

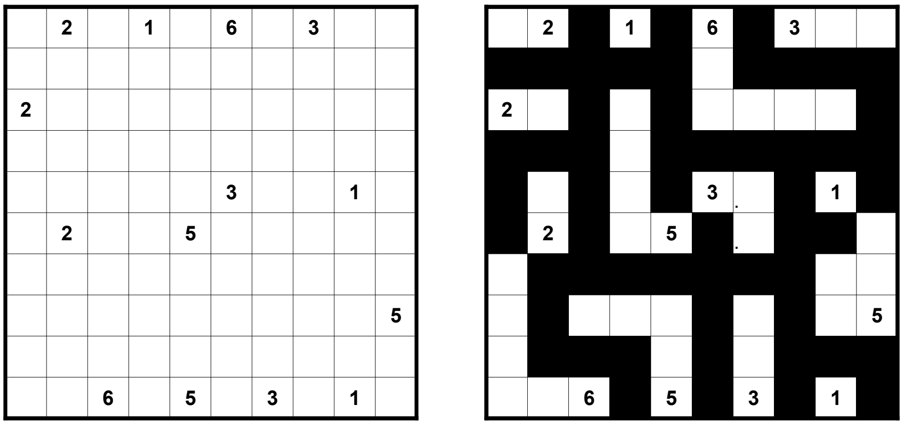


Figure 4.4: Example of a Nurikabe puzzle. On the left is the assignment of the puzzle, each number represents size of one island, on the right filled grid. White cells are "islands", black ones form a "river".

Nurikabe puzzle is also known to be NP-complete as was proved by M. Holzer, A. Klein and M. Kutrib [5] showing that Boolean circuits can be transformed to Nurikabe puzzles.

There was already done research of predicting difficulty of a particular Nurikabe puzzle for humans by P. Babinčák [1]. However, it was concentrated solely on a static properties of the assignment – such as number of islands or area taken by islands – or different dynamic metrics using solver simulating human solving by applying the same techniques as humans do.

Results obtained in this work are included in the experimental section for comparison.

Modifications of Algorithms

All three algorithms described in the previous chapters – simulated annealing, brute force search and harmony search – were modificated to solve Nurikabe puzzle.

Here are described just the necessary changes, principles of the algorithms remain the same.

Simulated Annealing

As a candidate solution we take all Nurikabe grids filled with the right number of black and white cells.

Neighboring solutions are those which are created by swapping the positions of one black and one white cell.

Most complicated and crucial is the construction of the cost function. It is based directly on Nurikabe rules and consists of this four parts:

- Penalty for wrong size of the islands – for each number we count size of the island in which this number lies and difference in size of this island and the number is added to the cost.
- Penalty for connected numbers – for every pair of numbers connected by white cells we add 1 to the cost.
- Penalty for disconnected river – for every connected component over one we add 1 to the cost.
- Penalty for pools – for every 2-by-2 block of black cells we add 1 to the cost.

As can be seen, solution to the puzzle has zero cost and if any candidate solution has zero cost it is a solution.

Harmony Search

Variable parts in a description of a harmony search are the set of candidate solutions and fitness function.

Candidate solutions are all possible colorings by black and white of Nurikabe grid given in an assignment. Fitness function is the same as cost function for simulated annealing.

In every step of harmony search algorithm we create new candidate solutions by random selection or memory consideration with/without pitch in the same way as for Sudoku puzzle.

Brute Force search

Even though there are just two colors that can be assigned to each cell, straightforward trying of all possible colorings of the grid does not lead to a solution in any feasible time. Even any use of several heuristics and other optimization did not help.

Approach that turned out to be useful is to expand all the islands in sequence. We choose an island and then generate all possible shapes of this island. Then we check if none of these conditions occurred in the partially filled grid:

- any two numbers are connected by island
- any island is smaller than should be and cannot grow any further
- any two parts of river are separated and cannot be connected
- pool is created

If not, we try to expand next island. Otherwise, algorithm backtracks. When all the islands are expanded, all the remaining undecided cells are turned to river cells and algorithm check if all Nurikabe rules are met and therefore we have found a solution.

The number of steps greatly depends on the order in which we expand islands. The fastest way is to do it in an ascending order of the sizes of the islands. Therefore we start with the islands of size 1, continue with the ones of size 2, etc. Among the islands of the same size we choose order randomly. In this way the search space is smaller as shape of the small islands (particularly the ones of size 1) has less variations and coupled with the fact, that these small islands are surrounded with river, it greatly reduces area available for expanding of the greater islands.

Experimental Results

We have data on human solving on 30 Nurikabe puzzles of various difficulty. Every puzzle was solved by 30 - 60 solvers. Data were collected from system Tutor – `tutor.fi.muni.cz`.

Simulated annealing generally solves Nurikabe puzzle quite fast and we tried several settings of cooling rate. Bruteforce search with described heuristics takes considerably longer. For both these algorithms we considered as a metric number of steps necessary to find solution.

Harmony search was not able to solve single puzzle unless trivial – it always get stuck in a local minima. Therefore, as was the case for Sudoku as well, we took as a metric lowest cost of a partial solution stored in a harmony memory after certain amount of steps (10 000 to be precise). Example of such a partial solution can be seen in figure 4.5.

All algorithms run 100 times on every puzzle and we took median of achieved values.

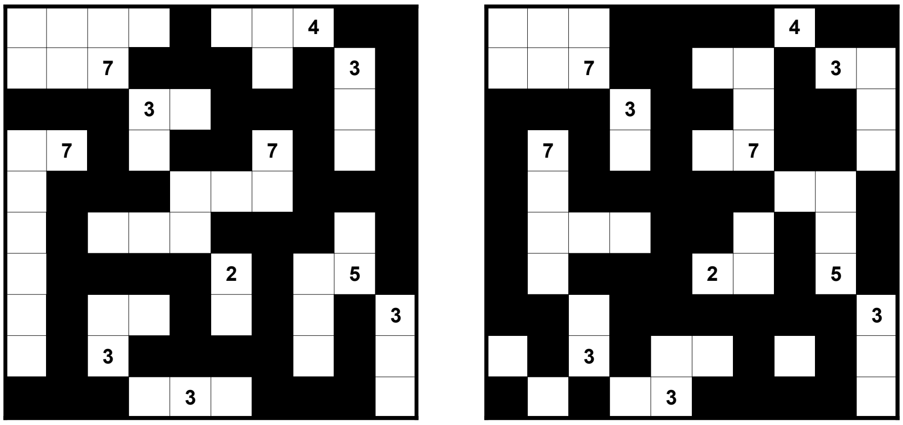


Figure 4.5: Solved Nurikabe puzzle on the left and the best partial solution which was obtained after 10 000 steps of Harmony search algorithm on the right.

Resulting correlations and their comparison to static metrics can be seen in table 4.3.

dynamic metrics	**correlation**
simulated annealing 0.99	0.03
simulated annealing 0.999	0.09
simulated annealing 0.9999	0.02
harmony search	-0.25
brute force search	0.64
static metrics	
number of islands	-0.51
area of islands	0.69
distance from island to island	0.35
number of cells which can belong to multiple islands	0.74

Table 4.3: Correlation coefficients among difficulty rating metrics and human median time for Nurikabe puzzle. Static metrics were tested by P. Babinčák [1]

Simulated annealing has no correlation with human solving time whatsoever and this is true for all tried values of a cooling rate.

Interestingly, lowest cost of a partial solution after certain number of steps of Harmony search has a negative correlation to human solving times.

This means that puzzles, that take longer to solve, end up with a better partial solution. Difficulty of the tough puzzles may be caused directly by this – they offer a likeable way to be solved which however turns out to be only local minima.

Brute force search has quite high correlation with human solving. This can be probably explained by the fact, that whenever there are many islands of small size, search space is considerably smaller than for islands of a bigger size. And therefore this reflects the static property of number and size of the islands.

All this metrics are considerably worse than simple static metrics such as number of islands or area of islands – with the exception of brute force search but this reflects static properties of the assignment as was already discussed. Best static metric is a number of cells which can be possibly part of multiple islands – lies in several circles surrounding cells with numbers with radius of the number.

Given this fact, it seems there is not much use of these algorithms for predicting difficulty as static properties can achieve the same results and are simpler and faster to check.

Chapter 5

Conclusions

We describe a Sudoku puzzle and provide an overview of algorithmic approaches. Three algorithms – simulated annealing, harmony search and brute force search – were tried for predicting difficulty for human solvers on an excessive set of data on human solving collected on internet web servers dedicated to Sudoku puzzle. Results show that best results can be achieved with a model based on a human solving approach using reasoning about constraints. Harmony search and brute force search do not give good results.

To test general usability of the algorithms, they were also tested on two different problems – predicting order of cell filling during Sudoku solving and predicting difficulty of a different puzzle, Nurikabe. Simulated annealing is able to predict with high correlation order of cell filling, on the other hand, it does not provide any correlation with difficulty of Nurikabe. Brute force search is the opposite – not good for predicting order but good for Nurikabe (even though it is still worse than the best of metrics based on static properties of an assignment).

Harmony search and genetic algorithms in general do not seem to be useful on this particular type of problems.

Bibliography

[1] P. Babinčák. Analýza lidí při řešení nurikabe. *Master Thesis, Faculty of Informatics, Masaryk University, Brno*, 2010.

[2] Z.W. Geem. Harmony search algorithm for solving Sudoku. In *Proceedings of the 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems: Part I*, pages 371–378. Springer-Verlag, 2007.

[3] D. Henderson, S. Jacobson, and A. Johnson. The theory and practice of simulated annealing. *Handbook of metaheuristics*, pages 287–319, 2003.

[4] A. Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[5] M. Holzer, A. Klein, and M. Kutrib. On the np-completeness of the nurikabe pencil puzzle and variants thereof. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, pages 77–89.

[6] S. Kirkpatrick, CD Gelatt Jr, MP Vecchi, and A. McCoy. Optimization by Simulated Annealing. *Science*, 220(4598):671–679, 1983.

[7] M. Křivánek and R. Pelánek. Difficulty rating of sudoku puzzles using constraint propagation, simulated annealing, and constraint relaxation. In *IJCAI, submitted*, 2011.

[8] R. Lewis. Metaheuristics can solve sudoku puzzles. *Journal of heuristics*, 13(4):387–401, 2007.

[9] T. Mantere and J. Koljonen. Sudoku solving with cultural swarms. *AI and Machine Consciousness*, 2008.

[10] T.K. Moon and J.H. Gunther. Multiple constraint satisfaction by belief propagation: An example using sudoku. In *Adaptive and Learning Systems, 2006 IEEE Mountain Workshop on*, pages 122–126. IEEE, 2006.

[11] A. Moraglio and J. Togelius. Geometric particle swarm optimization for the sudoku puzzle. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 118–125. ACM, 2007.

[12] R. Pelánek. Difficulty rating of sudoku puzzles by a computational model. In *Twenty-Fourth International FLAIRS Conference*, 2011.

[13] H. Simonis. Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*, pages 13–27. Citeseer, 2005.

[14] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

[15] G. V. Loganathan Z. W. Geem, J. H. Kim. A New Heuristic Optimization Algorithm: Harmony Search. *Simulation*, 76(2):60–68, 2001.