A New Difficulty Metric For Sudoku

Klemens Kranawetter* and Bernhard Geiger†
Institute for Signal Processing and Speech Communication,
Graz University of Technology, Graz, AT - Austria
(Dated: June 3, 2013)

Abstract. Sudoku games are usually classified with - rather vague - difficulty levels such as "'easy"', "'challenging"', "'devilish"', etc. But what is the underlying metric behind such a classification? The most common way to determine the difficulty of a puzzle was and still is to estimate or count the number of calculations a more or less randomly chosen solving algorithm needs to actually solve the puzzle. Our idea is to link the analysis of Sudoku with information theory, especially with what is called partial information decomposition (PID). The main idea behind our work is to look at the initially given numbers, also called clues, as information sources that provide information about a target variable - the solution of the puzzle. If the puzzle is set properly, the sources provide enough information to determine the solution. The connection to the new difficulty metric is as follows: Any properly set puzzle can obviously be made easier if one just adds clues. From the information theory point of view the adding of clues does not increase the information about the solution or target variable, as the solution is fully specified with less clues and still is fully specified with the additional ones. What has changed is the redundant part of the information about the solution that is provided by the set of clues. With that in mind we intend to define a single-letter number which does not depend on any parameters that measures the redundancy and maps it onto a difficulty scale.

Keywords: Sudoku, Information Theory, Partial Information Decomposition

I. INTRODUCTION

This article can be seen as a summary of the ideas that were developed during a student project at the Institute for Signal Processing and Speech Communication (SPSC) at Graz University of Technology. Projects of this kind are offered for master students in electrical engineering or computer science.

Supervisor DI Bernhard Geiger and student Klemens Kranawetter collaborated on the issue in the winter term of 2012.

The goal of the project was to develop a new difficulty metric for the number game Sudoku. According to [4] 'Sudoku is a puzzle in which $N = n^2$ different symbols (usually digits 1 through N) are to be arranged in an $N \times N$ array such that the arrangement agrees with given clues and meets the puzzle constraints'. The constraints or rules can be formulated as in [1]: 'Fill in the array so that every row, every column, and every $n \times n$ subsquare contains the digits exactly once'.

It seems as if there were two reasons why Sudoku appears in the interest of researchers from various disciplines: On the one hand the puzzle can be modeled mathematically in many different ways. This variability makes the problem accessible from vastly different directions, such as optimization, chaos theory, etc. Our approach can be seen as a connection with the field of information theory. On the other hand it is clear that the puzzle itself is rich of interesting aspects and facets. One example besides the quest for a difficulty metric is the enumeration of possible grids, as it is demonstrated in [2].

What attracted us to the problem was the instinctive assumption that the difficulty of a puzzle could in one or the other way be related to the redundancy in the set of statements that comes with the set of clues, which we wanted to investigate more deeply. The statements made by a single clue, say **C**, would be: 'No other **C** in my subsquare!','No other **C** in my row!', 'No other **C** in my column!', 'No other digit in my cell!'

Enough statements of this kind specify the solution of the puzzle, and it is common knowledge that the statements of 17 clues in a 9×9 Sudoku can suffice. The reason why redundancy appears to be naturally linked to the difficulty of a puzzle is the fact that any puzzle can be made easier if one just adds clues to those that are already given. Through the adding of clues the information about the solution does not increase at all - if the initial puzzle is set properly, the solution is specified completely and uniquely and still is specified completely and uniquely with the additional clues. The number of statements though has risen, which makes it obvious that the redundant part must have grown in one way or the other. In other words one could say that if the initial amount of clues already makes it possible to figure out the solution, additional clues simply reduce the number of steps one needs to take to get to there, but do not increase any kind of information about the solution.

From a theoretical perspective it was our goal to tell the redundant part of the information about the solution from the rest. Separating the different parts in information that is brought by a multivariable system is an ongoing research topic in information theory, hence we

*Electronic address: klemens.kranawetter@student.tugraz.at
†Electronic address: geiger@tugraz.at; URL: `http://www.spsc.tugraz.at`

took a look at some papers that deal with this issue.

At this point one might ask what properties there are that distuinguish our approach from already existing concepts. On the one hand it is the connection to information theory that is rather new. On the other hand is the clear concept. Older metrics in literature vastly suggest counting the number of steps or time it takes a solving algorithm to actually solve the puzzle. Since algorithms depend on a variety of parameters, the estimated difficulty is often almost as arbitrary as if a human player would have guessed it.

II. A NEW METRIC FOR SUDOKU

We already pointed out that the central idea behind our concept can be described as 'the assumption that the difficulty of a puzzle could somehow be related to the redundancy among the set of statements that is associated with the puzzle'. The term 'set of statements' describes the entirety of expressions that arise when we ask for an exhaustive descritption of the constraints that a possible solution to a given puzzle has to fulfill. Since these constraints evolve from the interplay between rules and clues the elements of the set of statements are all of one of these forms (**C** stands for one single clue):

No other **C** in my subsquare!
No other **C** in my row!
No other **C** in my column!
No other digit in my cell!

Further on we will denote the 'set of statements' as $\mathcal{B}$. As every single clue contributes four elements to $\mathcal{B}$, we get $|\mathcal{B}| = 4L_C$ for every imaginable $N \times N$ Sudoku with L_C givens.

One possible motivation for this idea is the simple fact that any Sudoku puzzle can be made easier through the adding of clues. If we consider a properly set puzzle with L clues represented by $\mathcal{S}_L$ that has one unique solution, we can be sure that any child $\mathcal{S}_{L+k}, k \in \mathbb{N}$ that is derived from $\mathcal{S}_L$ through the adding of k consistent clues has the same unique solution. On the other hand it is clear that $|\mathcal{B}_{\mathcal{S}_{L+k}}| = |\mathcal{B}_{\mathcal{S}_L}| + 4k$, which means that the set of statements has grown but no new information about the solution is provided through the additional clues. In such a situation we could say that we have at least $4k$ redundant statements. Put in other words the more difficult parent and the easier child differ through 4k redundant statements.

At this point it is important to state that puzzles with L clues are not automatically more difficult than others with $L + k$ clues! There are certainly examples where the opposite is the case, but children that are related to their parents through a set of additional clues definitely are easier to solve than their ancestors. If we had a procedure, say D, that would calculate the 'correct' degree of difficulty for a given $\mathcal{S}_L$, we could be sure that

$$D(\mathcal{S}_L) \geq D(\mathcal{S}_{L+k}) \tag{1}$$

would hold for every imaginable child $\mathcal{S}_{L+k}$.

That being said we can try to figure out a way to measure this sinister quantity called redundancy. Our **first** idea at this point is to introduce a graphical representation for $\mathcal{B}$. Therefor we construct a 3D object that contains N layers for an $N \times N$ Sudoku. These layers are used to represent the initially given clues and the area their statements *act on*, e.g. in a 4×4 Sudoku a '1' in the right lower corner would *act on* the right lower subsquare, the first row and the fourth column as it bans other '1s' from settling down in one of these cells. The layers themselves are $N \times N$ grids, where the i^{th} is used to represent those clues whose associated digit is 'i'. The cells in the influenced areas get highlighted with a tag. With that it also becomes possible to respect the fourth statement 'No other digit in my cell!' by tagging the cells above or below the cell of the clue in the other layers. Below we give an example for a 4×4 puzzle. Due to the fact that we use tensors in Matlab to implement this graphical representation, we gave the resulting object the name 'Sudoku Tensor'.

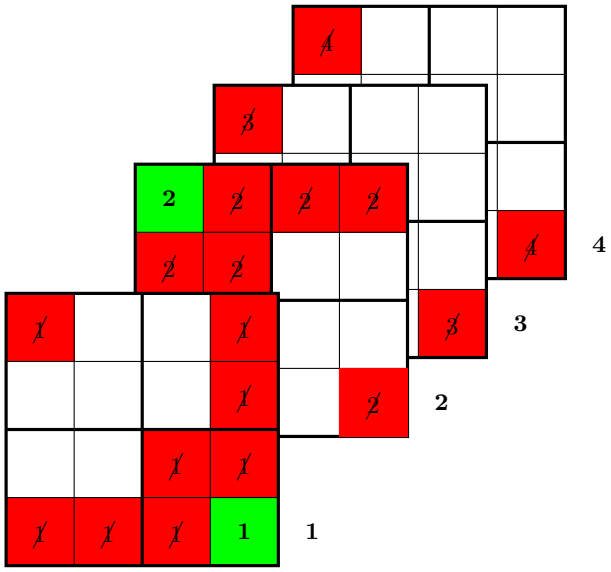


FIG. 1: Sudoku Tensor for two Clues in a 4x4 Sudoku.

The **second** idea shifts the focus of our considerations to those squares in the Sudoku Tensor that are covered by more than one statement and refines the notion of 'redundant statement'. Instead of trying to figure out how many redundant statements as a whole there are, we will develop three routines that are based on those cells that are affected by more than one statment. We are going to refer to these cells as 'redundant cells'. With this change it becomes possible to resolve the information that is given by a set of clues a lot finer. It turns out that the analysis of the structure of the redundant cells opens a big variety of possibilities.

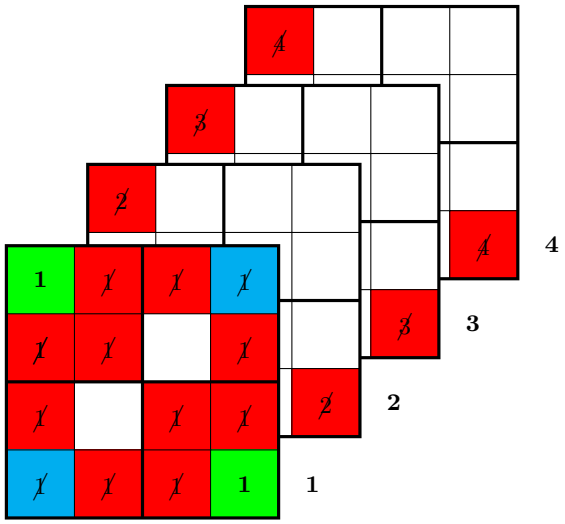


FIG. 2: Sudoku Tensor with redundant (cyan) statements from a 4x4 Sudoku.

With that in mind we can proceed to some definitions that will be helpful for the actual formualtion of the redundancy metrics:

- Whole set of clues: $\mathcal{C}$
- Single Clue: c_l
- Sudoku Tensor: $\mathcal{T}$
- Single Cell: t_{ijk}
- Number of statements on a cell given a single clue c_l or a set $\mathcal{X}$ of clues: $V_{c_l}(t_{ijk})$ or $V_{\mathcal{X}}(t_{ijk})$
- The cardinality of a set $\mathcal{X}$: $|\mathcal{X}|$

$\underline{R_{Average}}$

$$R_{Average} = \frac{\sum_{l=1}^{|\mathcal{C}|} |\{t_{ijk}|V_{c_l}(t_{ijk}) \geq 1 \wedge V_{\mathcal{C}\setminus c_l}(t_{ijk}) \geq 1\}|}{|\{t_{ijk}|V_{\mathcal{C}}(t_{ijk}) \geq 1\}|} \tag{2}$$

Our first metric $R_{Average}$ takes a look at one clue c_l after the other and compares it to the rest $\mathcal{C}\setminus c_l$. With comparing we mean counting the amount of cells that are covered by the statements of a single clue c_l and the statements of the rest of the clues $\mathcal{C}\setminus c_l$. The single results are summed up and then divided through the number of cells that have more than one statement on them considering the whole set $\mathcal{C}$: $|\{t_{ijk}|V_{\mathcal{C}}(t_{ijk}) \geq 1\}|$. The last step is performed to get a result between 0 and 1. The idea behind $R_{Average}$ can be described as the attempt to calculate the average amount of redundancy provided by a single clue.

$\underline{R_{Bipart}}$

To be able to formulate our next metric mathematically, we have to make clear what we understand under a 'bipartition of a set'. A bipartition is a seperation of a given set $\mathcal{X}$ into two subsets $\mathcal{A}$ and $\mathcal{B}$ that cover the whole $\mathcal{X}$, but do not have any elements in common, i.e. $\mathcal{A} \cap \mathcal{B} = \emptyset$. For R_{Bipart} we take a look at the whole set of bipartitions $\mathcal{P} = \{\{A_i, B_i\}|A_i \cup B_i = \mathcal{C}\}$, figure out which combination $\{A_i, B_i\}$ provides the largest number of cells that are covered by both A_i and B_i and norm the result through $|\{t_{ijk}|V_{\mathcal{C}}(t_{ijk}) \geq 1\}|$.

$$R_{Bipart} = \frac{\max(|\{t_{ijk}|V_{A_i}(t_{ijk}) \geq 1 \wedge V_{B_i}(t_{ijk}) \geq 1\}|)}{|\{t_{ijk}|V_{\mathcal{C}}(t_{ijk}) \geq 1\}|} \tag{3}$$

$$\text{where} \quad \{A_i, B_i\} \in \mathcal{P} \tag{4}$$

There are a couple of ideas that lay beneath this approach. First we thought about the redundancy lattice from [3]. We wanted to set up an analysis structure comparable to this lattice, but soon found out that on the one hand the lattice grows incredibly fast with the number of clues $|\mathcal{C}|$, so that it gets impossible to calculate all the values for the nodes of the lattice, and on the other hand we always intended to get a single letter number that represents the metric. To get the single letter number we thought about taking the maximum from all the values inside the lattice. As it is clear that this maximum lays somewhere in the subset of bipartitions we soon decided to reduce our analysis to $\mathcal{P}$.

Secondly we wanted to know the following: If we had to choose one subset $\mathcal{X} \subseteq \mathcal{C}$ such that - if we dismissed this subset $\mathcal{C}$ and made the remaining clues $\mathcal{C}_2 = \mathcal{C} \setminus \mathcal{X}$ the new set of clues - the difference between the number of cells that are assigned to $\mathcal{X}$, $\{t_{ijk}|V_{\mathcal{X}}(t_{ijk}) \geq 1\}$, and the number of cells that are covered by $\mathcal{X}$ but not by $\mathcal{C}_2$ gets maximal:

$$|\{t_{ijk}|V_{\mathcal{X}}(t_{ijk}) \geq 1\}| - |\{t_{ijk}|V_{\mathcal{X}}(t_{ijk}) \geq 1 \wedge V_{\mathcal{C}_2}(t_{ijk}) = 0\}| \longrightarrow \max$$

$\underline{R_{\geq 2}}$

$R_{\geq 2}$ answers how many redundant cells there when we take into account the whole set of clues $\mathcal{C}$:

$$R_{\geq 2} = \frac{|\{t_{ijk}|V_{\mathcal{C}}(t_{ijk}) \geq 2\}|}{|\{t_{ijk}|V_{\mathcal{C}}(t_{ijk}) \geq 1\}|} \tag{5}$$

Again the result is mapped into $[0, 1]$ by a division through the number of cells that are covered by at least one statement.

III. RESULTS

Implementation

We already pointed out that the graphical object we call **Sudoku Tensor** is handled as a tensor in Matlab. For an $N \times N$ puzzle we get an $N \times N \times N$ tensor, where the i^{th} layer stands for for the i^{th} digit. The layers themselves are $N \times N$ matrices, that contain ones and zeros. Ones denote that a digit i would not be in contradiction with any of the represented statements and could be filled in, zeros denote the opposite: at least one statement forbids that a symbol equal to the very digit gets filled in. Since every given also 'shines' on the other layers through the statement 'No other digit in my cell', we also assign zeros to these kind of cells.

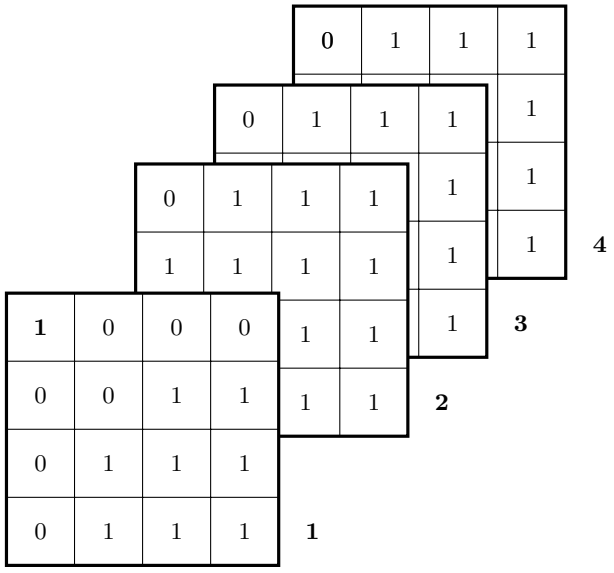


FIG. 3: Matrix Representation of the Sudoku Tensor for a 4×4 puzzle with just one clue, a 1 in the upper left corner.

For $\mathbf{R_{Average}}$ we create two tensors, one for the device under test c_l, the current clue, and one for the rest $\mathcal{C} \setminus c_l$ and check how many elements with entry equal to zero in both tensors there are. Based on that the final result is achieved by a repetition of this procedure for all clues and the calculation of the average as it described in the previous section.

$\mathbf{R_{t \geq 2}}$ is even easier than that, since we only need to count the amount of cells with more than one statement covering them. Therefor we set up a second tensor which we use to count how often a zero is assigned to a cell. With this additional information the whole procedure collapses to a simple $\overset{?}{\geq} 2$ question.

$\mathbf{R_{Bipart}}$ is definitely the trickiest one. Our first attempt that worked well for 4×4 Sudokus was to primitively go from one bipartition $\{A_i, B_i\}$ to the next and check what the 'value of the cost function' $J = |\{t_{ijk} | V_{A_i}(t_{ijk}) \geq 1 \wedge V_{B_i}(t_{ijk}) \geq 1\}|$ was. Unfortunately, this strategy fails when it is applied to 9×9 Sudokus, because the amount of bipartitions grows with $\mathcal{O}(2^{\mathcal{C}-1})$. It turns out though that the result for 4×4 Sudokus, where we were able to carry out the calculations, always lays in those subsets of $\mathcal{P}$, where $|A_i| \approx |B_i| \approx \frac{|\mathcal{C}|}{2}$. This fact is not at all surprising since bipartitons with subsets that contain a small amount of clues will never provide the maximum. As soon as we discovered that, we thought about solving our problem through an optimization algorithm that seeks through those bipartitions that contain subsets with cardinality $|\mathcal{X}| = \text{ceil}(\frac{\mathcal{C}}{2}) \pm \delta$, where $\delta \in [0, 1, 2, 3]$ turned out to be a good choice. The actual computation of $J = |\{t_{ijk} | V_{A_i}(t_{ijk}) \geq 1 \wedge V_{B_i}(t_{ijk}) \geq 1\}|$ is executed with two tensors for the two partitions, exactly the way we did it for $R_{Average}$.

To loose some words on the optimization algorithm: We start with a pre-optimization, that chooses 1000 bipartitions $\{A_i, B_i\}$ randomly and takes the best combination as a starting point for the actual optimization. The actual optimization steps from the current $\{A_i, B_i\}$ to the next $\{A_{i+1}, B_{i+1}\}$ by exchanging the first clue c_{A1} from A_i with the clue c_{Bi} from B_i that improves J the most if there is a clue that improves J at all, then from $\{A_{i+1}, B_{i+1}\}$ to $\{A_{i+2}, B_{i+2}\}$ by exchanging the second clue and so on. If the algorithm has reached the last clue in A_j, it starts again from the beginning. Above all that there is a loop that changes the sizes $|A_i|$ and $|B_i|$ via δ.

```
1: procedure BiPartMax(C)
2:     for δ ∈ [0,1,2,3] do
3:         J_1 = 0
4:         for i=1:1000 do
5:             create random {A_i, B_i} with
6:             |A_i| = ceil(C/2) − δ and |B_i| = ceil(C/2) − δ
7:             if J({A_i, B_i}) ≥ J_1 then
8:                 save J_1 = J({A_i, B_i})
9:             end if
10:        end for
11:        J_0 = 0
12:        i = 0
13:        while J_{i+1} ≥ J_i do        ▷ As long as we improve
14:            J_i = J_{i+1}
15:            for j = 1:|A_i| do
16:                for k = 1:|B_i| do A_i(j) ⇔= B_i(k)
17:                    if J({A_i, B_i}) ≥ J_i then
18:                        save J_{i+1} = J({A_i, B_i})
19:                        Remember the best k*!
20:                    end if A_i(j) ⇔= B_i(k)
21:                    if k == |B_i| then A_i(j) ⇔= B_i(k*)
22:                    end if
23:                end for
24:            end for
25:        end while
26:        J_δ = J_{i+1}
27:    end for
28: end procedure
```

Experiments

The implementations were tested with 3000 9×9 puzzles that were created with Wangs puzzle generation code (see [5]). In this generator it is possible to vary the diffi-

culty of the created puzzle through a parameter $n \in [0, 1]$, where $n = 0$ indicates a puzzle of the easiest level and $n = 1$ the hardest level possible. This dependency might be a bit confusing, since our metrics measur the redundancy of the puzzles and also return a single letter number $R \in [0, 1]$. The interpretation is different though, $R = 1$ indicates maximum redundancy and hence the easieast possible level, $R = 0$, corresponds to a minimum of redundancy and a high difficulty.

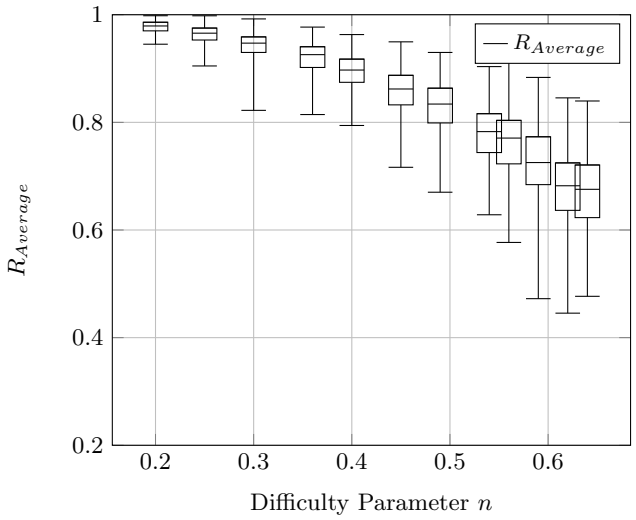


FIG. 4: Results for $R_{Average}$

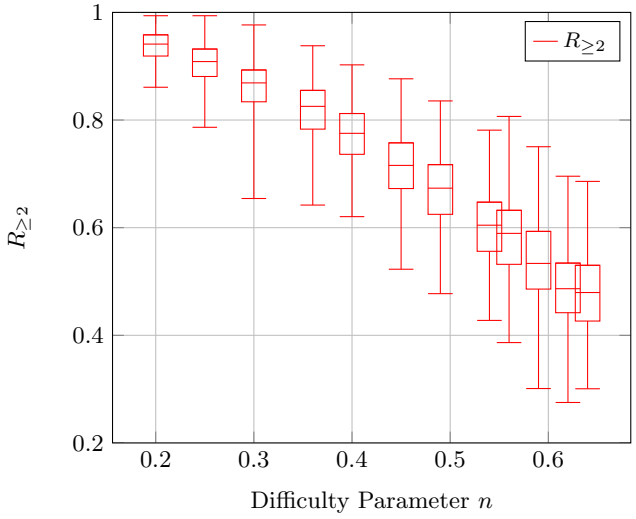


FIG. 5: Results for $R_{\geq 2}$

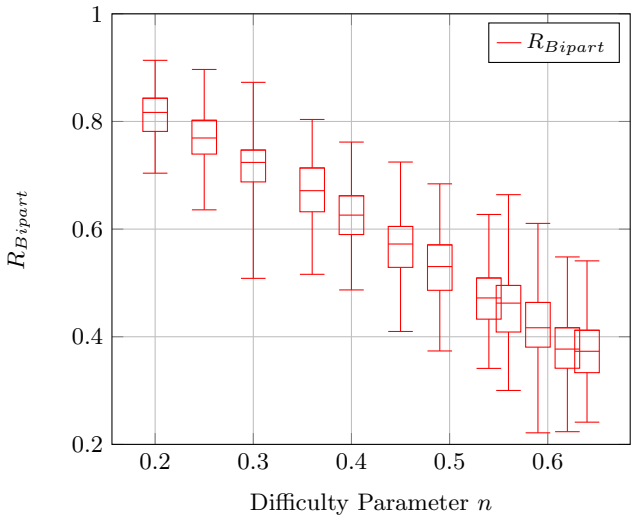
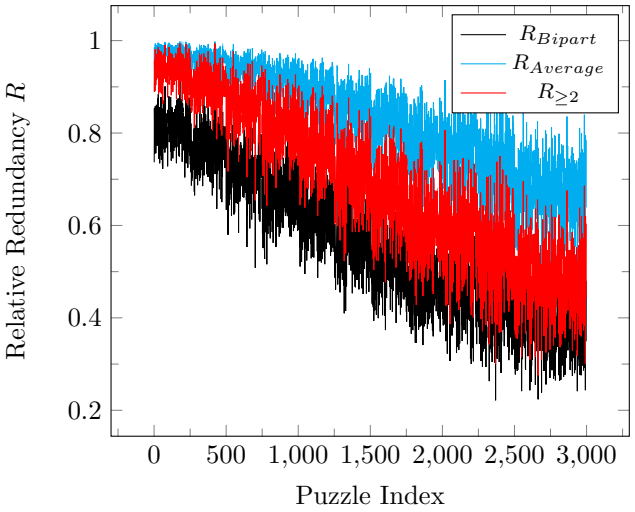


FIG. 6: Difficulty Metric R_{Bipart}



IV. DISCUSSION

Three difficulty metrics for the number game Sudoku were formulated. They are all based on the idea that redundancy within the set of statements that is produced by the initially given clues is somehow related to the difficulty of the puzzle. Each of the metrics shows that easier puzzles are related to a high redundancy, whereas difficult examples are characterized by a smaller amount of redundant statements. It is nice to see that the theoretical ideas seem to be confirmed through the large number of tests we ran through. One property of many other concepts that attempt to measure the difficulty of Sudoku puzzles is that they depend on the behavior of a solving algorithm that is applied to the puzzle. Our approach is independent from any kind of solving algorithm, it simply analysis the structure of the puzzle. Considering the obtained results one could get the impression that our metrics deliver the same results as Wangs generator; it

also has to check the difficulty of the puzzle in one way or the other. A closer look though reveals that our results seem to correct the promised difficulty levels from time to time - puzzles that are labeled difficult have high redundancies, while puzzle labeled easy have low redundancies.

---

[1] Bartlett, Andrew ; Langville, Amy: *An Integer Programming Model for the Sudoku Problem.* College of Charleston, 2006

[2] Felgenhauer, Bertram ; Jarvis, Frazer: *Enumerating possible Sudoku grids.* TU Dresden - University of Sheffield, 2005

[3] Griffith, Virgil ; Koch, Christof: *Quantifiying synergistic mutual information.* Caltech, 2012

[4] Gunther, Jake ; Moon, Todd: *Entropy Minimization for Solving Sudoku.* IEEE, 2012

[5] Wang, Hongtao: *Solve and Create SUDOKU Puzzles for Different Levels.* [Online] available at http://www.mathworks.com/matlabcentral/fileexchange/authors/27052, 2007