Ranking Based Evaluation of Algorithms That Estimate The Difficulty Level of A Combinatorial Puzzle Instance

Kazuya HARAGUCHI

Department of Information Technology and Electronics, Faculty of Science and Engineering, Ishinomaki Senshu University, Ishinomaki 986-8580

Abstract

We propose a novel framework to evaluate an algorithm that estimates the difficulty level of a combinatorial puzzle instance like Sudoku. The highlight is that we employ the notion of ranking to evaluate such algorithm. In the evaluation framework, we first design a test set of puzzle instances somehow. The algorithm gives numerical scores to these instances so that they represent the difficulty levels. The scores naturally induces a ranking on the test set, from an "easy" instance to a "difficult" one. Besides, we prepare another "true" ranking by aggregating the opinions of human solvers, where reasonable aggregation can be made by Majority Ranking from the social choice theory. Then, we measure the distance between the two rankings; the smaller the distance is, the better the algorithm should be. We describe the advantages of the proposed framework through a case study on BlockSum puzzle.

1　Introduction

Combinatorial puzzles are employed in primary education or employment examination these days, and it is significant to develop such algorithms to generate puzzle instances of various types automatically[1-3]. Further, it should be useful to classify the generated instances by their difficulty levels. However, estimation of difficulty level is not an easy task since difficulty itself is a subjective concept and the difficulty levels that ones feel for a puzzle instance should be different from solver to solver. Even so, if we dare to do this, we may need to evaluate the difficulty estimation algorithms based on human solvers' behavior in solving puzzle instances.

In this paper, we propose to evaluate a difficulty estimation algorithm for a combinatorial puzzle based on ranking. In fact, there have been developed several algorithms that estimate the difficulty level of a given puzzle instance automatically[4-7]. These algorithms deliver a numerical score to the instance based on how it is solved by the assumed computational model; during the solving process, the harder situations the model encounters, the higher the score becomes.

When we evaluate the effectiveness of a difficulty estimation algorithm, we first collect a set of benchmark puzzle instances. We call this benchmark instance set a *test set*. In general, it is not easy to interpret the meaning of the numerical score given by a difficulty estimation algorithm. For example, one instance I is not "twice as difficult as another instance J" since the score of I is the double of the score of J. The score becomes meaningful only for comparison. Therefore, what we should observe is not the actual values of the scores but the ranking on the test set that is induced by the numerical order of the scores. We call this ranking the *score ranking*.

Given a test set, the algorithm to be evaluated induces the score ranking automatically. Then how do we identify whether the score ranking is good or bad? Apart from the score ranking, we prepare a target ranking on the test set somehow, say the *true ranking*. By this, we can measure the distance between the score ranking and the true ranking by such

metrics as Kendall-tau distance. The smaller the distance is, the better the algorithm should be. This is the proposed way of evaluating difficulty estimation algorithms.

The key issue is whether we can have a "reasonable" true ranking or not. Our approach is described as follows; We bring together numerous human solvers and let them solve the instances in the test set. We then aggregate the "opinions" of the solvers into the true ranking by utilizing a technique called *Majority Ranking*[8] from the social choice theory. There are many representation forms of "opinions." Solving time (a numerical value) is not definite, and we should try other possibilities, e.g., rating (a discrete value). No matter which the "opinion" is represented by a numerical value or by a discrete value, Majority Ranking aggregates the "opinions" into a reasonable ranking.

As a case study, we take up BlockSum puzzle. Like Sudoku, BlockSum puzzle asks to fill the cells in an $n \times n$ grid with integers so that the integers form an $n \times n$ Latin square and satisfy an additional condition. Haraguchi et al. developed a difficulty estimation algorithm for BlockSum puzzle recently[1]. The algorithm has several ad-hoc parameters as other algorithms do. We investigate which parameter configuration induces the score ranking nearest to the true ranking. Then we observe that some class of configurations induces score rankings nearer to the true ranking than other class. This tendency does not appear so explicitly as long as we use Pearson correlation coefficient as the criterion.

The paper is organized as follows. We review related works in Section 2. In Section 3, we prepare terminologies on ranking and BlockSum puzzle, along with our difficulty estimation algorithm[1]. In Section 4, we present the proposed framework to evaluate difficulty estimation algorithms. Then we present a case study on BlockSum puzzle in Section 5. Finally, we present concluding remarks in Section 6.

2 Related Works

Previous difficulty estimation algorithms are mainly developed for Sudoku. In the previous algorithms, the difficulty level of a given Sudoku instance is estimated by the number of cells that can be fulfilled by a certain constraint propagation scheme[7], the number of fixed cells[4], the size of search tree[5], and the sum of weight values of the techniques that are used to solve the instance[6]. In the papers[4,5,7], the effectiveness of an algorithm is evaluated by how the difficulty scores fit to the ratings that are given by a puzzle designer. Since difficulty level is a subjective concept, the opinion of only one human is not sufficient to represent the true difficulty level. Thus we should take into account the opinions of various solvers from novices to experts to realize the true difficulty level. Then, Palanek[6] collects anonymous solvers on the Internet and evaluates the algorithms by Pearson correlation coefficient between the difficulty scores and the average solving times. However, solving time is not the only item that can represent the true difficulty level. Our proposal is to evaluate difficulty estimation algorithms by means of ranking distance. In the proposed framework, Majority Ranking enables us to construct a reasonable true ranking from data on numerous human solvers' behavior, no matter which the true level is represented by numerical items or by discrete items.

3 Preliminaries

3.1 Ranking

For a natural number n, we denote $[n] = \{1, 2, \ldots, n\}$. Let S denote any set of elements. A *ranking on* S (or a *ranking of the elements in* S) is a function $\pi: S \to [|S|]$. For any $a \in S$, the value $\pi(a)$ represents the rank of a. We assume

any ranking to be *partial*, that is, we may have a tie $\pi(a) = \pi(b)$ for some $a, b \in S$ $(a \neq b)$. For a partial ranking π and any ordered pair $(a, b) \in S \times S$ of elements, we define $\delta_\pi(a, b) = -1$ if $\pi(a) < \pi(b)$, $\delta_\pi(a, b) = 1$ if $\pi(a) > \pi(b)$, and $\delta_\pi(a, b) = 0$ otherwise. Note that $\delta_\pi(a, b) = -\delta_\pi(b, a)$. To measure the closeness between two partial rankings π and π', we introduce a generalization of Kendall-tau distance[9,10]. Denoted by $d(\pi, \pi')$, this metric is defined as follows;

$$d(\pi, \pi') = \frac{1}{2} \sum_{(a,b) \in S \times S} |\delta_\pi(a, b) - \delta_{\pi'}(a, b)|. \quad (1)$$

3.2 Latin Square and BlockSum Puzzle

Let $n \geq 2$ denote a natural number. Given an $n \times n$ grid of cells, we denote a cell in the i-th row and in the j-th column by $(i, j) \in [n]^2$. Let us denote an assignment of integers in $[n]$ to the cells by a function $\varphi: [n]^2 \to [n]$; the value $\varphi(i, j)$ represents the integer assigned to cell (i, j). The function φ is partial in general. We may represent φ by a set of triples $T_\varphi = \{(i, j, \varphi(i, j)) | \varphi(i, j) \text{ is defined}\}$. A function $\phi: [n]^2 \to [n]$ is an extension of φ if $\phi(i, j) = \varphi(i, j)$ whenever $\varphi(i, j)$ is defined. In other words, ϕ is an extension of φ if $T_\phi \supseteq T_\varphi$. We call φ a *partial Latin square* if any two triples (i, j, v) and (i', j', v') in T_φ satisfy at least two of the followings: $i \neq i'$, $j \neq j'$ or $v \neq v'$. We simply call φ a *Latin square* if it is a total function.

An instance of BlockSum puzzle is given as an $n \times n$ grid of cells such that the set of n^2 cells is partitioned into disjoint subsets called *blocks*, each block being assigned an integer in $[n(n+1)/2]$. We assume that each block is connected, i.e., it is made out of side-adjacent cells. Given a BlockSum instance, a solver is asked to fill all the n^2 cells with integers in $[n]$ so that the following two conditions are satisfied.

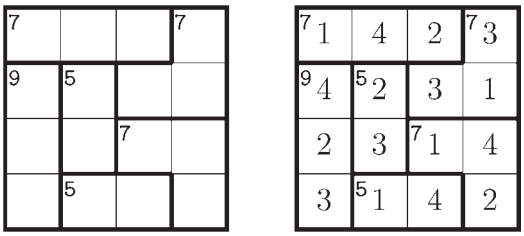

Figure 1: A BlockSum instance and its solution (n=4)

Latin square condition: In each row and in each column, any integer in $[n]$ appears exactly once.

BlockSum condition: In each block, the sum of the integers assigned to the cells equals to the number assigned to the block.

In Figure 1, we show a BlockSum instance for n=4, together with its solution.

Let us give a mathematical formulation of BlockSum puzzle. Two cells (i, j) and (i', j') are *adjacent* if $|i - i'| + |j - j'| = 1$. The adjacency defines the connectivity of cells. A block is a subset of connected cells. We represent a BlockSum instance by $I = (P, \sigma)$, where P denotes a partition of the n^2 cells into blocks and σ denotes a function $\sigma: P \to [n(n+1)/2]$. For any block $B \in P$, $\sigma(B)$ denotes the integer assigned to B. We call B a *unit block* if $|B| = 1$. Two blocks B and B' are adjacent if there exist $(i, j) \in B$ and $(i', j') \in B'$ that are adjacent. We can regard the rule of BlockSum puzzle as follows; given an instance $I = (P, \sigma)$, the solver is asked to find out such a function $\varphi: [n]^2 \to [n]$ that is a Latin square (Latin square condition) and that satisfies $\sum_{(i,j) \in B} \varphi(i, j) = \sigma(B)$ for any $B \in P$ (BlockSum condition). We call φ a *solution of* I if φ satisfies the above two conditions. We call φ a *partial solution of* I if there exists an extension of φ that is a solution of I.

3.3 A Difficulty Estimation Algorithm for BlockSum Puzzle

We describe our algorithm[(1)] that estimates the difficulty level of a BlockSum instance. The algorithm retains a collection R of *inference rules*. Starting with the partial solution φ with $T_\varphi=\emptyset$, the algorithm attempts to solve the given instance $I=(P,\sigma)$ by applying the inference rules in R repeatedly. An inference rule states that, if I and φ satisfy a certain condition, we can specify an integer v that should be assigned to a certain empty cell (i, j). In this case, we can update the partial solution $T_\varphi \leftarrow T_\varphi \cup \{(i, j, v)\}$. Later we introduce examples of inference rules.

Assume that I has been solved by the way described above. Then the algorithm estimates the difficulty level of I by computing the score as follows; That I is solvable is guaranteed by a sequence $s=((i_1, j_1, r_1), (i_2, j_2, r_2), \ldots, (i_{n^2}, j_{n^2}, r_{n^2}))$, which represents that the instance is solved by filling cells from (i_1, j_1) to (i_{n^2}, j_{n^2}) by applying inference rules r_1 to r_{n^2}, respectively. Each inference rule r in R is given a parameter value called *weight*. The weight represents the sophistication level of the inference rule; the larger the weight is, the more sophisticated the rule should be. We denote the weight of r by $w(r)$. Then we define the *score of I with respect to s*, denoted by $\rho_s(I)$, as the sum of the weights of $r_1, r_2, \ldots, r_{n^2}$ in the sequence s;

$$\rho_s(I)=w(r_1)+w(r_2)+\cdots+w(r_{n^2}). \quad (2)$$

There may exist plural sequences that guarantee the solvability of I. Among these, we use such s that is obtained as follows; When we try to solve the instance, we need to decide (i_k, j_k, r_k) for $k=1, 2, \ldots, n^2$. If there are plural candidates for (i_k, j_k, r_k), we choose the (i_k, j_k, r_k) that attains the minimum weight $w(r_k)$ since we would like to model the behavior of human solvers who may prefer simple rules that have small weights. We denote the score of I with respect to such obtained s by $\rho(I)=\rho_s(I)$, and simply call $\rho(I)$ the *score of I*. We expect that, the larger $\rho(I)$ is, the more difficult it should be. In the rest of the paper, we focus on the instances that can be solved by the way described above. Hence any instance is given a numerical score by the algorithm.

Before closing this section, we describe the 10 inference rules that are used in the case study of Section 5. Among these, 9 rules are already explained in the paper[(1)], where 4 rules are derived from Latin square condition and the other 5 rules are derived from BlockSum condition. In what follows, the instance is denoted by $I=(P,\sigma)$, a partial solution of I is denoted by φ, the empty cell that an inference rule fills with an integer is denoted by (i, j) and the integer assigned to (i, j) is denoted by v.

The 4 inference rules from Latin square condition We denote these 4 rules by r_1^{LS}, r_2^{LS}, r_3^{LS} and r_4^{LS}. The r_1^{LS} states that, if $n-1$ distinct integers are already assigned in the $n-1$ cells of the i-th row except (i, j), or the $n-1$ cells in the j-th column except (i, j), assign the remaining integer to (i, j). The r_2^{LS} is a generalization of r_1^{LS} as follows; Let C denote the set of cells in the i-th row and in the j-th column except (i, j). Thus we have $|C|=2(n-1)$. The r_2^{LS} states that, if $n-1$ distinct integers appear in C, assign the remaining integer to (i, j). The r_3^{LS} states that, if an integer v is already assigned to certain $n-1$ cells out of the i-th row and out of the j-th column, assign v to (i, j). The r_4^{LS} is a generalization of r_3^{LS}, but we omit the details since they are not significant in the subsequent discussion.

The 5 inference rules from BlockSum condition We denote these 5 rules by r_1^{BS}, r_2^{BS}, r_3^{BS}, r_4^{BS} and r_5^{BS}. Let B denote the block which (i, j) belongs to. The r_1^{BS} states that, if B is a unit block, then assign $\sigma(B)$ to (i, j). The r_2^{BS} is a generalization

of r_1^{BS} and states that, if integers are already assigned to all the cells in B except (i, j), then assign $\sigma(B)$ minus the sum of these integers to (i, j). Observe that the sum of n integers in one row (resp., in one column) should be $1+2+\cdots+n=n(n+1)/2$. We can specify v if we can tell the sum of the integers in all the cells in the i-th row (resp., in the j-th column) except (i, j) due to some reason. In such a case, we can assign $n(n+1)/2$ minus the sum to (i, j). For example, the cell (1, 1) in the left of Figure 1 is assigned $(1+2+3+4)-9=1$. The r_3^{BS}, r_4^{BS} and r_5^{BS} are based on this idea, but we omit the details.

The remaining inference rule is special in the sense that it assumes the list of candidate integers for each empty cell.

The special inference rule We denote the special inference rule by r^*. Let us denote an empty cell by $(i, j) \in [n]^2$. For each integer $v' \in [n]$, take a partial solution $T_\varphi \cup \{(i, j, v')\}$ and suppose updating it by using the 9 inference rules above. During the process, if we encounter an assignment that violates Latin square condition or BlockSum condition, then we can exclude v' from the candidates for (i, j). The r^* states that, if all the $n-1$ integers except v are excluded in this way, we can assign v to (i, j). (Similarly, r^* assigns v to (i, j) if v cannot be assigned to (i', j) for any $i' \in [n] (i' \neq i)$, or if v cannot be assigned to (i, j') for any $j' \in [n] (j' \neq j)$.)

4 Evaluation Framework

In this section, we explain how to evaluate a difficulty estimation algorithm. First, we design a set of benchmark instances, which we call a test set. Given a test set, the algorithm induces a ranking of the instances by the numerical order of the scores. More precisely, let us denote the test set by $I = \{I_1, I_2, \ldots, I_p\}$, where each I_i $(i \in [p])$ denotes an instance. Then if $\rho(I_1) \leq \rho(I_2) \leq \cdots \leq \rho(I_p)$, we have a ranking $\pi(I_1) = 1$, $\pi(I_2) = 2$, ..., $\pi(I_p) = p$. We call a ranking on the test set induced in this way a score ranking. Apart from the score ranking, we construct the "true" difficulty ranking on the test set. This is called the true ranking. We compare the score ranking with the true ranking by the generalized Kendall-tau distance in (1). The smaller the distance is, the better the algorithm should be.

We do not discuss how to generate a test set. Assuming that the test set is given, we explain how to construct the true ranking on it through observation on human solvers. In Section 4.1, we describe how to sample the data on human solvers' behavior. In Section 4.2, we present the technique Majority Ranking[8] that computes a ranking on the test set from the sampled data. We use this computed ranking as the true ranking.

4.1 How to Sample Data on Human Solvers' Behavior

To sample the data, we let human solvers solve the instances of the given test set. Prior to this, we need to determine the following matters:

(i) the human solvers whom we ask to solve the instances,

(ii) the environment in which the solvers solve the instances, and

(iii) the items we should observe when the solvers solve the instances.

One matter should be determined by taking others into account since they are related to each other. We show two example schemes as follows.

(a) For (i), we may employ real human

solvers. For example, if we are school teachers, we can collect students as solvers. In this case, we can utilize handouts to let the solvers solve the instances for (ii). More precisely, we print handouts on which the instances are written like the left of Figure 1. Then the solvers write the solutions on the handouts. For (iii), the item can be the number of cells to which correct integers are not assigned, i.e., the cells to which incorrect integers are assigned or that are left empty. An instance should be harder if many solvers cannot give correct integers to the n^2 cells. We can also sample the data on such items as solving time and ratings by asking each solver to record the values of these items.

(b) We can collect anonymous solvers on the Internet for (i). In this case, we need to set up such a computer environment for (ii) that admits the solvers to solve the instances and that can store the data on their behavior. For (iii), we can sample the data on more detailed items automatically than (a), e.g., which integer is assigned to which cell in which time.

A main difference between (a) and (b) lies in the possible number of human solvers. Thousands of anonymous solvers are collected from the Internet in Palanek's paper[6], whereas it is hard to collect such a large number of real solvers. However, (a) is not necessarily inferior to (b). The virtue of (a) is that we can control the solvers. More precisely, we can make the solvers work on the puzzle intensively in (a), whereas it is harder to do so in (b). Then we can be more confident on the data sampled by (a) even though the number of solvers is small.

4.2 Majority Ranking

Focusing on a certain item of the sampled data, we obtain a profile like Table 1. In this profile, the item is the rating given by a human solver that takes easy, normal, or hard. We explain Majority Ranking that computes a difficulty ranking on the test set from such a profile.

Let us denote the number of the instances in the test set by p and the number of the human solvers by q. Let us denote the test set by $I=\{I_1, I_2, \ldots, I_p\}$ and the set of the human solvers by $H=\{h_1, h_2, \ldots, h_q\}$. Each instance $I_i \in I$ is given a *grade* by each human solver $h_j \in H$. The grade is an element of a totally ordered set D that is given. We denote the total order on D by $\leq_D$. Higher grades are used to represent that an instance is harder. For example, if we focus on a numerical item such as solving time, D is the set of real numbers and $\leq_D$ is the usual arithmetic order. If we focus on a discrete item such as rating, D is the set of discrete ordered values, e.g., $D=\{\text{easy, normal, hard}\}$. In this case, we may take easy $\leq_D$ normal $\leq_D$ hard.

Then we have a $p \times q$ matrix $A=(a_{i,j})$ such that $a_{i,j}$ indicates the grade which I_i is given by h_j. Given a matrix A, Majority Ranking computes a ranking on I by the following procedure.

1. For each instance I_i ($i \in [p]$), sort the q grades $a_{i,1}, a_{i,2}, \ldots, a_{i,q}$ in the non-decreasing order of $\leq_D$. Let us denote the sorted sequence by $g'_i=(a'_{i,1}, a'_{i,2}, \ldots, a'_{i,q})$. The grades satisfy $a'_{i,1} \leq_D a'_{i,2} \leq_D \cdots \leq_D a'_{i,q}$.

2. For each instance I_i, construct a sequence $g^*_i=(a^*_{i,1}, a^*_{i,2}, \ldots, a^*_{i,q})$ by taking $a^*_{i,j}$ ($j \in [q]$) as follows.

$$a^*_{i,j}=\begin{cases} a'_{i,\lceil \frac{q}{2} \rceil} & \text{if } j=1, \\ a'_{i,\lceil \frac{q}{2} \rceil+\lfloor \frac{j}{2} \rfloor} & \text{if } j \geq 2 \\ & \text{and } q \equiv j \pmod 2, \\ a'_{i,\lceil \frac{q}{2} \rceil-\lfloor \frac{j}{2} \rfloor} & \text{if } j \geq 2 \\ & \text{and } q \not\equiv j \pmod 2. \end{cases}$$

One sees that $a^*_{i,1}$ is the median of all the q grades, $a_{i,1}, \ldots, a_{i,q}$, given to the instance I_i. To be more general, $a^*_{i,j}$ is the median of $\{a_{i,1}, \ldots, a_{i,q}\} \setminus \{a^*_{i,1}, \ldots, a^*_{i,j-1}\}$.

3. We have p sequences $g^*_1, g^*_2, \ldots, g^*_p$, each of

Table 1: Difficulty ratings of 5 instances I_1, ..., I_5 given by 7 human solvers h_1, ..., h_7

	h_1	h_2	h_3	h_4	h_5	h_6	h_7
I_1	normal	easy	hard	hard	normal	hard	easy
I_2	hard	easy	normal	easy	normal	easy	normal
I_3	hard	normal	easy	normal	hard	hard	normal
I_4	hard	normal	normal	easy	hard	normal	normal
I_5	easy	normal	normal	easy	normal	normal	hard

Table 2: Application of Majority Ranking to the profile of Table 1

	$a'_{i,1}$	$a'_{i,2}$	$a'_{i,3}$	$a'_{i,4}$	$a'_{i,5}$	$a'_{i,6}$	$a'_{i,7}$
g'_1	easy	easy	normal	normal	hard	hard	hard
g'_2	easy	easy	easy	normal	normal	normal	hard
g'_3	easy	normal	normal	normal	hard	hard	hard
g'_4	easy	normal	normal	normal	normal	hard	hard
g'_5	easy	easy	normal	normal	normal	normal	hard

	$a^*_{i,1}$	$a^*_{i,2}$	$a^*_{i,3}$	$a^*_{i,4}$	$a^*_{i,5}$	$a^*_{i,6}$	$a^*_{i,7}$
g^*_1	normal	normal	hard	easy	hard	easy	hard
g^*_2	normal	easy	normal	easy	normal	easy	hard
g^*_3	normal	normal	hard	normal	hard	easy	hard
g^*_4	normal	normal	normal	normal	hard	easy	hard
g^*_5	normal	normal	normal	easy	normal	easy	hard

which corresponds to one instance. Using the total order $\leq_D$ on the grade set D, we define the lexicographic order $\leqslant_D$ on the sequences; we write $g^*_i \leqslant_D g^*_{i'}$ if there exists $j \in [q]$ such that $a^*_{i,1}=a^*_{i',1}$, $a^*_{i,2}=a^*_{i',2}$, ..., $a^*_{i,j-1}=a^*_{i',j-1}$, and $a^*_{i,j} \leq_D a^*_{i',j}$. Induce the ranking of the p instances from the lexicographic order of the p sequences.

We use the output of this procedure as the true ranking. Basically, Majority Ranking evaluates an instance by means of the median of the grades that are given to the instance. The meaning of using the median is described as follows; An instance should be at least as hard as the median grade since a half of the solvers give higher or equal grades. Similarly, the instance should be at least as easy as the median grade since a half of the solvers give lower or equal grades. The computed ranking satisfies several reasonable properties, but we omit the details due to space limitation.

We apply Majority Ranking to the profile of Table 1. We show the sequences g'_1, ..., g'_5 and g^*_1, ..., g^*_5 in Table 2. Let us denote the obtained true ranking by π^*. Then we have $\pi^*(I_2)=1$, $\pi^*(I_5)=2$, $\pi^*(I_4)=3$, $\pi^*(I_1)=4$ and $\pi^*(I_3)=5$. The true ranking regards I_2 as the easiest and I_3 as the hardest.

5 A Case Study on Block-Sum Puzzle

In this section, we report a case study on the difficulty estimation algorithm for BlockSum puzzle, which was presented in Section 3.2. The algorithm has weights of the inference rules as adjustable parameters. If we take different weight configurations, an instance is given different scores (see (2)), and the resulting score ranking may vary. We observe which parameter configuration of the algorithm induces the score ranking nearest to the true ranking.

5.1 Experimental Settings

To generate the test set, we produce 9 instances ($n=5$) by the generation algorithm that was

proposed by the paper[(1)]. This algorithm generates such an instance (P, σ) that can be solved by applying the inference rules of a given collection R. The algorithm first chooses an $n \times n$ Latin square φ randomly for the solution of the generated instance. Then, once P is determined, σ is automatically determined by the chosen Latin square, that is, the algorithm sets $\sigma(B) = \sum_{(i,j) \in B} \varphi(i, j)$ for any block $B \in P$. Starting with the partition such that all blocks are unit blocks (i.e., $P = \{(i, j) | (i, j)) \in [n]^2\}$), the algorithm repeats merging two adjacent blocks into one block as long as the instance (P, σ) can be solved by means of the inference rules in R. We produce 9 instances by taking R as the set of 10 inference rules in Section 3.3.

Computing the score of each instance, we obtain the score ranking on the test set. The difficulty estimation algorithm has weights of the 10 inference rules as adjustable parameters. One weight configuration assumes an ordering of the inference rules on sophistication level that is represented by a weight value. We consider two classes of weight configurations, denoted by w^A and w^B, that assume the following orderings;

$$\begin{aligned} &w^A(r_1^{\mathrm{LS}}) \le w^A(r_2^{\mathrm{LS}}) \le w^A(r_1^{\mathrm{BS}}) \le w^A(r_2^{\mathrm{BS}}) \\ &\le w^A(r_3^{\mathrm{LS}}) \le w^A(r_4^{\mathrm{LS}}) \le w^A(r_3^{\mathrm{BS}}) \\ &\le w^A(r_4^{\mathrm{BS}}) \le w^A(r_5^{\mathrm{BS}}) \le w^A(r^*), \\ &w^B(r_1^{\mathrm{BS}}) \le w^B(r_2^{\mathrm{BS}}) \le w^B(r_1^{\mathrm{LS}}) \le w^B(r_2^{\mathrm{LS}}) \\ &\le w^B(r_3^{\mathrm{LS}}) \le w^B(r_4^{\mathrm{LS}}) \le w^B(r_3^{\mathrm{BS}}) \\ &\le w^B(r_4^{\mathrm{BS}}) \le w^B(r_5^{\mathrm{BS}}) \le w^B(r^*), \end{aligned}$$

where the only difference is the relative order between $\{r_1^{\mathrm{LS}}, r_2^{\mathrm{LS}}\}$ and $\{r_1^{\mathrm{BS}}, r_2^{\mathrm{BS}}\}$, as indicated by the underlines. Recall that these 4 inference rules are the simplest among the 10 inference rules. We introduce w^A and w^B to investigate which of $\{r_1^{\mathrm{LS}}, r_2^{\mathrm{LS}}\}$ and $\{r_1^{\mathrm{BS}}, r_2^{\mathrm{BS}}\}$ should be given larger weights. Changing the relative difference between weight values, we try 4 concrete configurations for each class. The configurations are denoted by w_1^A, ..., w_4^A and w_1^B ..., w_4^B, and are defined as follows.

- For w_1^A, we define $w_1^A(r_1^{\mathrm{LS}}) = w_1^A(r_2^{\mathrm{LS}}) = 1/5$, $w_1^A(r_1^{\mathrm{BS}}) = w_1^A(r_2^{\mathrm{BS}}) = 2/5$, $w_1^A(r_3^{\mathrm{LS}}) = w_1^A(r_4^{\mathrm{LS}}) = 3/5$, $w_1^A(r_3^{\mathrm{BS}}) = w_1^A(r_4^{\mathrm{BS}}) = w_1^A(r_5^{\mathrm{BS}}) = 4/5$, and $w_1^A(r^*) = 5/5 = 1$. For w_1^B, we use the same weights as w_1^A except $w_1^B(r_1^{\mathrm{BS}}) = w_1^B(r_2^{\mathrm{BS}}) = 1/5$ and $w_1^B(r_1^{\mathrm{LS}}) = w_1^B(r_2^{\mathrm{LS}}) = 2/5$.

- Let $u^A(r)$ (resp., $u^B(r)$) denote the order of inference rule r in the configuration class w^A (resp., w^B); i.e., $u^A(r_1^{\mathrm{LS}}) = 1$, $u^A(r_2^{\mathrm{LS}}) = 2$, ..., $u^A(r^*) = 10$, $u^B(r_1^{\mathrm{BS}}) = 1$, ..., $u^B(r^*) = 10$. For w_2^A and w_2^B, we define $w_2^A(r) = u^A(r)/10$ and $w_2^B(r) = u^B(r)/10$ for any inference rule r.

- For w_3^A and w_3^B, we define $w_3^A(r) = 2^{u^A(r)-10}$ and $w_3^B(r) = 2^{u^B(r)-10}$.

- For w_4^A and w_4^B, we define $w_4^A(r) = 3^{u^A(r)-10}$ and $w_4^B(r) = 3^{u^B(r)-10}$.

Once we fix a parameter configuration, the algorithm computes the score of an instance. The score of an instance I for a parameter configuration $w_k^A (k = 1, ..., 4)$ (resp., w_k^B) is denoted by $\rho_k^A(I)$ (resp., $\rho_k^B(I)$). We denote the score ranking on the test set induced from the order of ρ_k^A by π_k^A (resp., ρ_k^B by π_k^B).

We describe how we construct the true ranking on the test set. We employ the scheme (a) in Section 4.1 to sample the data on human solvers' behavior. We collect 18 students in the author's institution. After explaining the rule of BlockSum puzzle, we let each solver solve the 9 instances one by one and record the solving time and the difficulty rating that the solver supposes for each instance. We limit the solving time to 600 seconds in order to prevent the solver from taking too much time; if the solver cannot solve an instance in 600 seconds, he must go on to the next instance. When the solver finishes solving an instance or cannot

Table 3: Generalized Kendall-tau rank correlation coefficients between score rankings and true rankings

	Incorrect or empty cells		Solving time		Rating	
	$\kappa(\pi_k^A, \pi^C)$	$\kappa(\pi_k^B, \pi^C)$	$\kappa(\pi_k^A, \pi^T)$	$\kappa(\pi_k^B, \pi^T)$	$\kappa(\pi_k^A, \pi^R)$	$\kappa(\pi_k^B, \pi^R)$
$k=1$	.7500	.7222	.6667	.6667	.7222	.7222
2	.7361	.7361	.6528	**.6806**	.7083	**.7361**
3	.7500	**.7778**	.6667	**.6944**	.7222	**.7500**
4	.7222	**.7500**	.6944	**.7222**	.6944	**.7222**

solve it within 600 seconds, he should choose the rating from 3 discrete ordered values, easy, normal and hard. Then we construct the true ranking for each of the 3 items, i.e., the number of cells to which the solver gives incorrect integers or that are left empty, solving time and rating, which we denote by π^C, π^T, and π^R, respectively. For π^T, we set the domain of the grade to a set of discrete ordered values, $D = \{0, 1, ..., 599\} \cup \{\text{over } 599\}$, with the total order $0 \leq_D 1 \leq_D \cdots \leq_D 599 \leq_D$" over 599" since we limit the solving time to 600 seconds.

It seems preferable that we have solvers of various levels. To realize expert solvers, we gave handouts for practice to a half of the 18 students one week before the above test and let them work on practice. The handouts do not contain the instances of the test set. We confirmed that the other half of the solvers did not have great experience in solving BlockSum instances as of the test. For each instance of the test set, those who had been given practice handouts tended to give more correct integers than those who had not been given.

5.2 Results

We compare weight configurations w_k^A and w_k^B with the same $k (k=1, ..., 4)$, to observe which one yields a score ranking nearer to the true rankings. We show the result in Table 3. In Table 3, the indicated values are *generalized Kendall-tau rank correlation coefficients*. Let π and π' denote partial rankings. Denoted by $\kappa(\pi, \pi')$, the generalized Kendall-tau rank correlation coefficient is defined as $\kappa(\pi, \pi') = 1 - d(\pi, \pi')/(|S|(|S|-1))$, where S denotes the set of elements to be ranked. The coefficient takes a value from -1 to 1. The smaller the distance $d(\pi, \pi')$ is, the larger the coefficient $\kappa(\pi, \pi')$ is. It is interesting to see that, for all the true rankings, the class w^B tends to yield a nearer score ranking than the class w^A. Besides, among the $3 \times 4 = 12$ entries, w^B is better than w^A in 8 entries (which are indicated by boldface), w^B ties w^A in 3 entries (which are indicated by underlines), and w^B is worse than w^A in only 1 entry. The observation suggests that $\{r_1^{\mathrm{LS}}, r_2^{\mathrm{LS}}\}$ from Latin square condition should be given larger weights than $\{r_1^{\mathrm{BS}}, r_2^{\mathrm{BS}}\}$ from BlockSum condition. Let us note that the true rankings are not relatively far from each other; $\kappa(\pi^C, \pi^T) = .9167$, $\kappa(\pi^C, \pi^R) = .9722$ and $\kappa(\pi^T, \pi^R) = .9444$.

For comparison, in Table 4, we show Pearson correlation coefficients[11] between difficulty score values and medians of item values that are observed over the 18 solvers. We see advantages of our evaluation framework over the one based on Pearson correlation coefficient; Since Pearson correlation coefficient does not support discrete values in principle, we cannot use rating to represent the true difficulty levels. For π^T, it appears hard to adapt Pearson correlation coefficient to the time limit setting. We regard its median value as 600 when it is "over 599," and thus the coefficient is not rigorous. By definition, Pearson correlation coefficient is less stable against observed item values than Kendall-tau rank correlation coefficient. Table 4 contains a slight difference of coefficients (e.g., .7367 and .7364 for $k=3$ in the number of incorrect integers), and it is not easy to decide

Table 4: Pearson correlation coefficients between actual score values and medians of observed item values

	Incorrect or empty cells		Solving time	
	ρ_k^A	ρ_k^B	ρ_k^A	ρ_k^B
$k=1$	7215	.6935	.6460	**.6560**
2	.7180	.6945	.6534	**.6539**
3	.7367	.7364	.6628	.6626
4	.7366	.7366	.6650	.6650

whether the difference is significant. Thus, it is hard to draw a conclusion from this table. Based on these, we assert that difficulty estimation algorithms should be evaluated by means of ranking distance rather than Pearson correlation coefficient.

6 Concluding Remarks

In this paper, we proposed to utilize the notion of ranking to evaluate the effectiveness of a difficulty estimation algorithm for combinatorial puzzle. We compared weight configurations of a single algorithm in the case study although the proposed framework is originally intended for comparison of variant algorithms. For future work, we need to apply the proposed framework to larger data that include more human solvers and instances. We should also try other puzzle generation algorithms in order to get some feedback.

References

(1) Haraguchi K, Abe Y, Maruoka A (2012) How to produce BlockSum instances with various levels of difficulty. Journal of Information Processing 20-3: 727-737

(2) Haraguchi K (2013) The number of inequality signs in the design of Futoshiki puzzle. Journal of Information Processing 21-1 (to appear)

(3) Haraguchi K, Maruoka A (unpublished manuscript) Puzzle instance design problem in a generalization of "Eight Blocks to Madness."

(4) Lewis R (2007) Metaheuristics can solve sudoku puzzles. Journal of Heuristics 13: 387-401

(5) Ono S, Miyamoto R, Nakayama S, Mizuno K (2009) Difficulty estimation of number place puzzle and its problem generation support. Proc. ICCAS-SICE: 4542-4547

(6) Pelánek R (2011) Difficulty rating of Sudoku puzzles by a computational model. Proc. Florida Artificial Intelligence Research Society Conference: http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS11/paper/view/2517

(7) Simonis H (2005) Sudoku as a constraint problem. Proc. 4th International Workshop of Modelling and Reformulating Constraint Satisfaction Problems: 13-27

(8) Balinski M, Lavaki R (2010) Majority Judgment. MIT Press.

(9) Fagin R, Kumar R, Mahdian M, Sivakumar D, Vee E (2006) Comparing partial rankings. SIAM Journal on Discrete Mathematics 20-3: 628-648

(10) Kemeny JG, Snell JL (1973) Mathematical Models in the Social Sciences (2nd edition). MIT Press.

(11) Edwards AL (1976) An Introduction to Linear Regression and Correlation. W.H. Freeman.