PAPER *Special Section on Foundations of Computer Science*

BlockSum is NP-Complete

Kazuya HARAGUCHI[†a)], ***Member and*** **Hirotaka ONO**[††b)], ***Nonmember***

SUMMARY BlockSum, also known as KeisanBlock in Japanese, is a Latin square filling type puzzle, such as Sudoku. In this paper, we prove that the decision problem whether a given instance of BlockSum has a solution or not is NP-complete.
key words: *NP-completeness, combinatorial puzzle, Latin square,* BlockSum

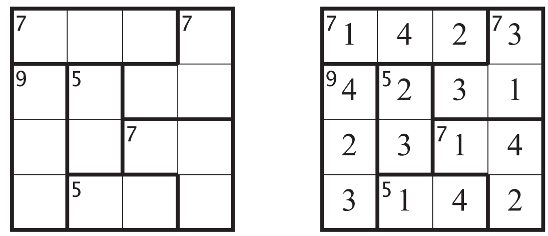
1. Introduction

In this paper, we show that BlockSum puzzle is computationally hard like many other combinatorial puzzles, e.g., Sudoku [1], Tetris [2], PuyoPuyo [3], [4], and classic Nintendo games [5]. Let n denote a natural number. An instance of BlockSum puzzle is given as an $n \times n$ grid of empty cells such that the n^2 cells are partitioned into disjoint subsets and each subset is assigned an integer. Any subset of cells is connected, i.e., it consists of side-adjacent cells. We call a subset of connected cells a *block*. We call the integer assigned to a block the *demand of the block*. A player of BlockSum puzzle is asked to fill all the n^2 cells with integers in $[n] = \{1, 2, \ldots, n\}$ so that the following conditions are satisfied.

Latin square condition: The integers assigned to the cells form an $n \times n$ Latin square, i.e., in each row and in each column, every integer in $[n]$ appears exactly once.

Demand condition: In each block, the sum of the integers assigned to the cells equals to the demand of the block.

We show a BlockSum instance and its solution ($n = 4$) in Fig. 1. In the figure, a block is indicated by a subset of cells surrounded by boldface lines, and its demand is indicated by a small digit. In Japan, BlockSum puzzle is often taken up in various media these days since Tetsuya Miyamoto, who is a successful private tutoring teacher, uses this puzzle for tutoring elementary school students [6], [7]. This motivates us to investigate whether or not BlockSum puzzle is essentially hard as other puzzles in computational sense. For related works on BlockSum puzzle, Haraguchi et al. recently proposed an algorithm that automatically produces BlockSum instances with various difficulty levels [8].



(Instance) (Solution)

Fig. 1 A BlockSum instance and its solution ($n = 4$).

We show the NP-completeness of the decision problem version of BlockSum puzzle. Given a BlockSum instance, the decision problem asks to identify whether it has a solution or not. We refer to this decision problem simply as BlockSum. It is the following theorem that we intend to prove.

Theorem 1: BlockSum is NP-complete even if every block consists of at most 2 cells.

We give the proof of Theorem 1 by means of reduction from Monotone Not-All-Equal 3SAT, a well-known NP-complete problem [9]. Although the computational hardness of a recreational puzzle does not necessarily imply its amusingness, it is a common nature of widely accepted recreational puzzles. In the sense, our result might imply that BlockSum is not only useful for educational purpose but also potentially fascinating for puzzlers. Preparing terminologies, notations and lemmata in Sect. 2, we present the proof in Sect. 3. We give concluding remarks in Sect. 4.

2. Preliminaries

2.1 Latin Square

Suppose an $n \times n$ grid of cells. We refer the cell in the i-th row and in the j-th column to $(i, j) \in [n] \times [n]$. We denote an assignment of integers in $[n]$ to the cells by a function $\varphi : [n] \times [n] \to [n]$. The value $\varphi(i, j)$ represents the integer assigned to cell (i, j). We call φ an $n \times n$ *Latin square*, or a *Latin square of order n*, if, in each row and in each column, each integer in $[n]$ appears exactly once, i.e., $\varphi(i, j) \neq \varphi(i, j')$ for any $i, j, j' \in [n]$ $(j \neq j')$ and $\varphi(i, j) \neq \varphi(i', j)$ for any $i, i', j \in [n]$ $(i \neq i')$. We define the *standard Latin square of*

Manuscript received March 28, 2012.
Manuscript revised July 25, 2012.
†The author is with the Department of Information Technology and Electronics, Faculty of Science and Engineering, Ishinomaki Senshu University, Ishinomaki-shi, 986–8580 Japan.
††The author is with the Department of Economic Engineering, Faculty of Economics, Kyushu University, Fukuoka-shi, 812–8581 Japan.
a) E-mail: kzyhgc@gmail.com
b) E-mail: hirotaka@en.kyushu-u.ac.jp
DOI: 10.1587/transinf.E96.D.481

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

Fig. 2 The standard Latin square of order 4.

order n as follows:

$$\varphi(i,j) = \begin{cases} i-j+1 & \text{if } i \geq j, \\ i-j+1+n & \text{otherwise.} \end{cases}$$

Figure 2 shows the standard Latin square of order 4.

Assume that $n = pq$ holds for two natural numbers p and q. In the proof, we may partition the $n \times n$ grid of cells into p^2 equally sized square subgrids so that each subgrid has $q \times q$ cells. For any $a, b \in [p]$, we denote the subgrid from the cell $(q(a-1)+1, q(b-1)+1)$ to the cell $(q(a-1)+q, q(b-1)+q)$ by $S^{n,p}_{a,b}$. We define $S^{n,p}_{a,b}$ as a set of cells as follows:

$$S^{n,p}_{a,b} = \{(q(a-1)+k, q(b-1)+\ell) \mid q = n/p,\ k, \ell \in [q]\}. \tag{1}$$

Let us introduce a systematic way to construct a Latin square satisfying a certain condition. Let us denote a $p \times p$ Latin square by π. Let us denote a set of p^2 Latin squares of order q by $\Psi = \{\psi_{a,b} : [q] \times [q] \to [q] \mid (a,b) \in [p] \times [p]\}$. We define $\varphi : [n] \times [n] \to [n]$ as the integer assignment that is obtained by pasting $\psi_{a,b} \in \Psi$ to each subgrid $S^{n,p}_{a,b}$ and by adding $q(\pi(a,b)-1)$ to the integers, i.e., for any $a, b \in [p]$ and any $k, \ell \in [q]$,

$$\begin{aligned} &\varphi(q(a-1)+k, q(b-1)+\ell) \\ &\quad = \psi_{a,b}(k,\ell) + q(\pi(a,b)-1). \end{aligned} \tag{2}$$

Proposition 2: The φ given by (2) is an $n \times n$ Latin square.

Proof: We show that φ satisfies the Latin square condition. Clearly, the integers that φ assigns to the $n \times n$ grid are in $[n]$. Let us take any two cells (i, j) and (i, j') that are in the same i-th row. If they are in the same subgrid $S^{n,p}_{a,b}$, they are assigned different values by φ since each subgrid is pasted a Latin square and all the integers in the subgrid are added the equivalent value, that is $q(\pi(a,b)-1)$. Otherwise, i.e., if (i, j) and (i, j') are in different subgrids $S_{a,b}$ and $S_{a,b'}$ respectively, the two cells are assigned different values since φ assigns the integers of disjoint ranges to these subgrids due to $\pi(a,b) \neq \pi(a,b')$. Analogously, any two cells in the same column are assigned different values. □

Lemma 3: For any natural number z, there is a $2^z \times 2^z$ Latin square $\varphi : [2^z] \times [2^z] \to [2^z]$ such that

$$\varphi(i,j) = \begin{cases} 1 & \text{if } i = j, \\ j & \text{if } i \neq j \text{ and } i = 1, \\ i & \text{if } i \neq j \text{ and } j = 1. \end{cases} \tag{3}$$

1	2
2	1

$(z = 1)$

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

$(z = 2)$

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

$(z = 3)$

Fig. 3 Latin squares that satisfy (3).

Proof: Prove by induction. When $z = 1$, we have such a Latin square, as shown in Fig. 3. For a general $z \geq 2$, assume that we have a $2^{z-1} \times 2^{z-1}$ Latin square that satisfies (3). Let us denote it by π_{z-1}. Then it is easy to see that a Latin square constructed by (2) with $p = 2$, $q = 2^{z-1}$, $\pi = \pi_1$, and $\psi_{a,b} = \pi_{z-1}$ for any $(a,b) \in [2] \times [2]$ satisfies (3). This and Proposition 2 show the lemma. (For example, see Fig. 3 for $z = 2$ and 3. We can see that shaded cells surely satisfy (3).) □

2.2 BlockSum

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 set of connected cells. Let us denote a block by $B \subseteq [n] \times [n]$. We call B a *k-block* if $|B| = k$. In particular, we call B an *$(r \times c)$-block* if it is an (rc)-block such that the cells form an $r \times c$ rectangle. Let us denote $\varphi : [n] \times [n] \to [n]$. When a block B is given by $B = \{(i_1, j_1), \ldots, (i_k, j_k)\}$, we denote by $\varphi(B)$ the sequence $(\varphi(i_1, j_1), \ldots, \varphi(i_k, j_k))$ for convenience.

We represent a BlockSum instance by $I_{\mathrm{BS}} = (\mathcal{B}, \sigma)$, where $\mathcal{B}$ denotes a partition of the n^2 cells into blocks and σ denotes a function $\sigma : \mathcal{B} \to [n^2(n+1)/2]$. For any block $B \in \mathcal{B}$, the value $\sigma(B)$ denotes the demand of B. We call σ a *demand function*. Then the decision problem BlockSum is formally defined as follows.

BlockSum

Instance: A BlockSum instance $I_{\mathrm{BS}} = (\mathcal{B}, \sigma)$, where $\mathcal{B}$ is a partition of n^2 cells into blocks and σ is a demand function $\sigma : \mathcal{B} \to [n^2(n+1)/2]$.

Question: Is there an assignment $\varphi : [n] \times [n] \to [n]$ such that φ is a Latin square (the Latin square condition) and $\sum_{(i,j) \in B} \varphi(i,j) = \sigma(B)$ holds for any $B \in \mathcal{B}$ (the Demand condition)?

When the answer is "yes," we say that the instance I_{BS} is

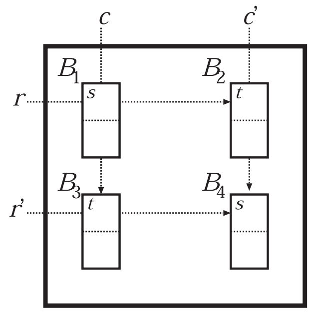


Fig. 4 A BlockSum instance whose demands are exchanged.

solvable and that the assignment φ satisfying the two conditions is a *solution of I_{BS}*.

Here, we show a technique of transferring a BlockSum instance by exchanging demands that will be used in the proof of NP-completeness. Suppose that, in $I_{\rm BS} = (\mathcal{B}, \sigma)$, four 2×1 blocks B_1, B_2, B_3, B_4 are in $\mathcal{B}$ such that they form

$$B_1 = \{(r, c), (r+1, c)\},\ B_2 = \{(r, c'), (r+1, c')\},$$
$$B_3 = \{(r', c), (r'+1, c)\},\ B_4 = \{(r', c'), (r'+1, c')\},$$

and $\sigma(B_1) = \sigma(B_4) = s$ and $\sigma(B_2) = \sigma(B_3) = t$. In Fig. 4, we illustrate how $B_1, \ldots, B_4$ are located in $I_{\rm BS}$. Let us construct another instance by exchanging the demands between B_1 and B_2 and the demands between B_3 and B_4. That is, denoting the constructed instance by $I'_{\rm BS} = (\mathcal{B}, \sigma')$, we define the demand function σ' as follows:

$$\sigma'(B) = \begin{cases} t & \text{if } B = B_1 \text{ or } B_4, \\ s & \text{if } B = B_2 \text{ or } B_3, \\ \sigma(B) & \text{otherwise.} \end{cases}$$

Assume that $I_{\rm BS}$ has a solution $\varphi : [n] \times [n] \to [n]$ such that

$$\varphi(B_1) = \varphi(B_4) = (s_1, s_2),\ \varphi(B_2) = \varphi(B_3) = (t_1, t_2),$$
$$s_1 + s_2 = s,\ t_1 + t_2 = t. \tag{4}$$

Clearly, the following $\varphi' : [n] \times [n] \to [n]$ is a solution of $I'_{\rm BS}$.

$$\varphi'(B_1) = \varphi'(B_4) = (t_1, t_2),\ \varphi'(B_2) = \varphi'(B_3) = (s_1, s_2),$$
$$\varphi'(i, j) = \varphi(i, j) \text{ for any } (i, j) \notin B_1 \cup \cdots \cup B_4. \tag{5}$$

Lemma 4: If φ is a solution of $I_{\rm BS}$ that satisfies (4), the assignment φ' given by (5) is a solution of $I'_{\rm BS}$.

2.3 Monotone Not-All-Equal 3SAT

Let $V = \{v_1, v_2, \ldots, v_N\}$ denote a set of N Boolean variables. For a variable $v \in V$, v is called the *positive literal* and $\bar{v}$ is called the *negative literal*. A *clause* is a set of literals over V. A *truth assignment* for V is denoted by $\tau : V \to$ {True, False}. Given a true assignment τ, if $\tau(v) =$ True, then the positive literal v is true and the negative literal $\bar{v}$ is false. If $\tau(v) =$ False, then v is false and $\bar{v}$ is true. A clause is called *not-all-equal under τ* if the clause has at least one literal that is true under τ and at least one literal that is false under τ. The decision problem Not-All-Equal SAT is defined as follows.

Not-All-Equal **SAT**
Instance: A Not-All-Equal SAT instance $I_{\rm SAT} = (V, C)$, where V is a set $V = \{v_1, v_2, \ldots, v_N\}$ of N Boolean variables, and C is a collection $C = \{C_1, C_2, \ldots, C_M\}$ of M clauses over V.
Question: Is there a truth assignment $\tau : V \to$ {True, False} such that each $C_b \in C$ is not-all-equal under τ?

We abbreviate Not-All-Equal SAT into NAE-SAT. When the answer is "yes," we call $I_{\rm SAT}$ *NAE-satisfiable*. The problem NAE-3SAT is the special case of NAE-SAT such that $|C_b| = 3$ for any $C_b \in C$. The NAE-3SAT is known to be NP-complete [10]. Further, the problem Monotone NAE-3SAT is the special case of NAE-3SAT such that all literals are positive. The Monotone NAE-3SAT is still NP-complete [9].

Lemma 5: NAE-SAT is NP-complete even if we restrict an instance $I_{\rm SAT} = (V, C)$ so that

(i) any variable in V appears as 2 positive literals and as 1 negative literal in C,
(ii) any clause in C has either 2 or 3 literals, and
(iii) any clause of 2 literals has one positive literal and one negative literal, and any clause of 3 literals has only positive literals.

Proof: We construct a polynomial transformation from a Monotone NAE-3SAT instance $J_{\rm SAT} = (W, \mathcal{D})$ to a NAE-SAT instance $I_{\rm SAT} = (V, C)$ so that it satisfies (i), (ii) and (iii). Let us denote $N = |W|$ and $M = |\mathcal{D}|$. Suppose that, in $\mathcal{D}$, a Boolean variable $w_a \in W$ appears as a positive literal k_a times. We take a set of k_a Boolean variables, denoted by $X_a = \{x_{a,1}, x_{a,2}, \ldots, x_{a,k_a}\}$. Taking the set X_a for each Boolean variable $w_a \in W$, we define $V = X_1 \cup \cdots \cup X_N$. We have $|V| = \sum_{a=1}^{N} k_a = 3M$.

We generate a set of clauses over V as follows. For each Boolean variable $w_a \in W$, we replace all k_a positive literals w_a'-s in $\mathcal{D}$ with k_a positive literals $x_{a,1}, x_{a,2}, \ldots, x_{a,k_a}$ respectively. We denote the set of clauses generated in this way by $C^{(3)}$. We have $|C^{(3)}| = |\mathcal{D}| = M$. Each clause in $C^{(3)}$ has exactly 3 literals since each clause in $\mathcal{D}$ does so. For any variable in V, its positive literal appears in $C^{(3)}$ exactly once. We do not have any other literals in $C^{(3)}$.

Next, we take another set of clauses over $V = X_1 \cup \cdots \cup X_N$. The set is denoted by $C^{(2)} = C_1^{(2)} \cup \cdots \cup C_N^{(2)}$, where each $C_a^{(2)}$ is a set of clauses over X_a for the variable $w_a \in W$ as follows:

$$C_a^{(2)} = \left(\bigcup_{b=1}^{k_a - 1} \{\{x_{a,b}, \bar{x}_{a,b+1}\}\}\right) \cup \{\{x_{a,k_a}, \bar{x}_{a,1}\}\}. \tag{6}$$

We have $|C^{(2)}| = \sum_{a=1}^{N} k_a = 3M$. Clearly, each clause in $C^{(2)}$ has exactly 2 literals. For any variable in X_a, its positive and negative literals appear in $C_a^{(2)}$ exactly once respectively.

The NAE-SAT instance $I_{\mathrm{SAT}} = (V, C)$ with $V = X_1 \cup \cdots \cup X_N$ and $C = C^{(2)} \cup C^{(3)}$ satisfies (i), (ii) and (iii) from the discussion so far. We have only to show that the above procedure is a polynomial transformation. Clearly, the construction can be done in a polynomial time in N and M. We claim that J_{SAT} is equivalent to I_{SAT}, i.e., J_{SAT} is NAE-satisfiable if and only if I_{SAT} is NAE-satisfiable. To show the necessity, suppose a truth assignment $\tau : W \to \{\text{True}, \text{False}\}$ such that each clause in $\mathcal{D}$ is not-all-equal. From τ, we construct a truth assignment $\omega : V \to \{\text{True}, \text{False}\}$ as follows; for each variable x in X_a, we let $\omega(x) = \tau(w_a)$. Thus all the variables in the same X_a are assigned the same truth value. From (6), we see that any clause in $C^{(2)} = C_1^{(2)} \cup \cdots \cup C_N^{(2)}$ is not-all-equal under ω. Recall that any clause $C \in C^{(3)}$ has been generated from a certain clause $D \in \mathcal{D}$, by replacing literals over W with ones over V. From the assumption, there is a positive literal $u \in D$ that is true under τ. If u is a positive literal of a variable $w_a \in W$, then $\tau(w_a)$ should be True. In C, u is replaced with a certain literal $x_{a,b}$. Since $\omega(x_{a,b}) = \tau(w_a) = \text{True}$, C has a literal that is true under ω. Similarly, the clause D also has a literal that is false under τ, which guarantees that C has a literal that is false under ω. Hence, C is not-all-equal under ω. We are done with the necessity. The sufficiency can be shown in an analogous way. □

3. Proof of NP-Completeness of BlockSum

Now we are ready to prove Theorem 1, the NP-completeness of BlockSum. It is easy to see that BlockSum is in NP; given an BlockSum instance and an integer assignment $\varphi : [n] \times [n] \to [n]$, we can verify whether or not φ is a solution of the instance in $O(n^2)$ time.

We reduce NAE-SAT to BlockSum. In the reduction, we establish a polynomial transformation from any NAE-SAT instance satisfying (i), (ii) and (iii) of Lemma 5 to a BlockSum instance such that every block consists of at most 2 cells. Let us denote an NAE-SAT instance satisfying (i), (ii) and (iii) of Lemma 5 by $I_{\mathrm{SAT}} = (V, C)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of N Boolean variables and $C = \{C_1, C_2, \ldots, C_M\}$ is the set of M clauses over V.

We construct a BlockSum instance $I_{\mathrm{BS}} = (\mathcal{B}, \sigma)$ on a $(10MN_0) \times (10MN_0)$ grid of cells, where $N_0 = 2^{\lceil \log_2(N+1) \rceil}$. Since we have

$$N_0 = 2^{\lceil \log_2(N+1) \rceil} \leq 2^{\log_2(N+1)+1} = 2(N+1),$$

the number of cells in the grid is at most $(20M(N+1))^2$, a polynomial in M and N. Let us denote $n = 10MN_0$. The $n \times n$ grid is partitioned into N_0^2 equally sized square subgrids so that each subgrid has $10M \times 10M$ cells. Recall that the notation $S_{a,b}^{n,N_0}$ represents a subgrid defined by (1). In the sequel, we let $S_{a,b}$ represent $S_{a,b}^{n,N_0}$ for convenience if no confusion arises. Any block in $\mathcal{B}$ is contained in one subgrid, or equivalently, no block crosses more than one subgrid. Each subgrid consists of 1-blocks and 2-blocks. Let the caligraphic

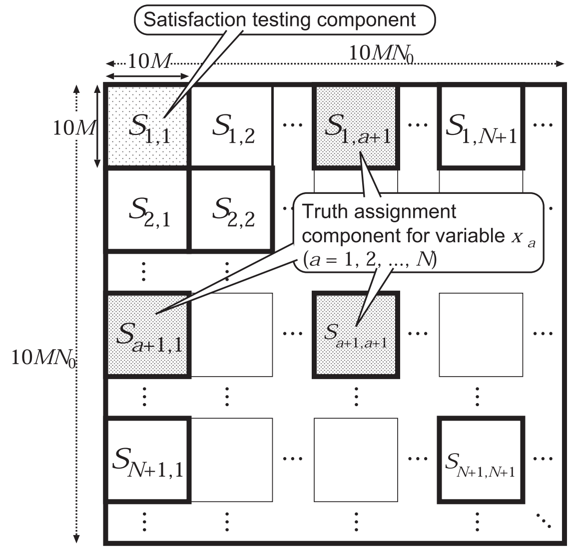


Fig. 5 Overview of the $n \times n$ grid of the BlockSum instance to be constructed ($n = 10MN_0$).

$\mathcal{S}_{a,b}$ denote the partition of a subgrid $S_{a,b}$ into blocks. We take the partition $\mathcal{B}$ as the union of the partitions of the subgrids:

$$\mathcal{B} = \bigcup_{(a,b) \in [N_0] \times [N_0]} \mathcal{S}_{a,b}.$$

For the demand function σ, we first set σ so that the instance $I_{\mathrm{BS}} = (\mathcal{B}, \sigma)$ surely has a solution, regardless of the satisfiability of I_{SAT}. Then we will exchange the demands of some blocks by means of Lemma 4 so that the resulting instance has a solution if and only if I_{SAT} is NAE-satisfiable. To guarantee that I_{BS} has a solution, we construct an $n \times n$ Latin square somehow, denoted by φ, and set the demand $\sigma(B)$ of each block $B \in \mathcal{B}$ as follows:

$$\sigma(B) = \sum_{(i,j) \in B} \varphi(i, j). \tag{7}$$

Clearly, the BlockSum instance $(\mathcal{B}, \sigma)$ has φ as a solution. For φ, we take such an $n \times n$ Latin square that can be given by the pasting and adding procedure of Proposition 2; since $N_0 = 2^{\lceil \log_2(N+1) \rceil}$, there is an $N_0 \times N_0$ Latin square that satisfies (3) of Lemma 3, denoted by π. We can construct such π in $O(N^2)$ time. Then the $n \times n$ Latin square φ is constructed by (2), with π and $10M \times 10M$ Latin squares $\psi_{a,b}$'-s that are appropriately chosen for each subgrid $S_{a,b}$. The point is that we can choose the Latin square $\psi_{a,b}$ for the subgrid $S_{a,b}$ independently from other subgrids.

Now we explain how we define the partition $\mathcal{S}_{a,b}$ and the Latin square $\psi_{a,b}$ for each subgrid $S_{a,b}$ that determines the demands of the blocks along with π. We illustrate the roles of the subgrids in Fig. 5. We use $S_{1,1}$ as the satisfaction testing component. In other words, we embed the blocks into $S_{1,1}$ that correspond to the M clauses. For any $a \in [N]$, we use $S_{1,a+1}$, $S_{a+1,1}$ and $S_{a+1,a+1}$ as the truth assignment components for the variable v_a. These subgrids are

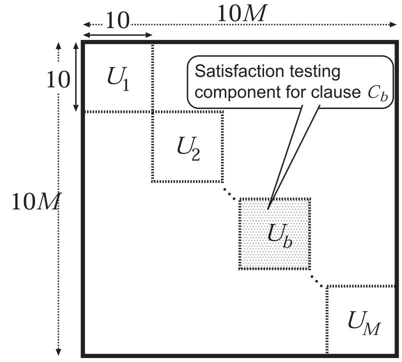


Fig. 6 Overview of the partition of $S_{1,1}$ that we use as the satisfaction testing component.

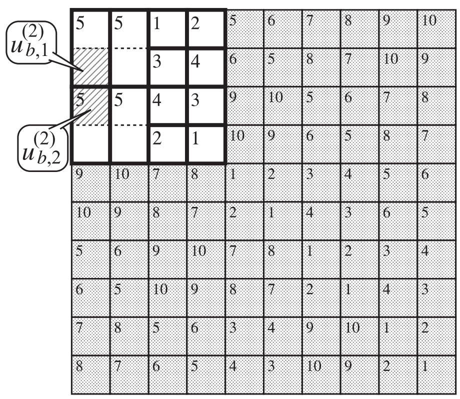


Fig. 7 Overview of U_b when the clause C_b has 2 literals.

contained in our grid properly since we have

$$N+1 = 2^{\log_2(N+1)} \leq 2^{\lceil \log_2(N+1) \rceil} = N_0.$$

We use the other N_0^2-1-3N subgrids for garbage collection.

(1) Satisfaction Testing Components

We use the $10M \times 10M$ subgrid $S_{1,1}$ as the satisfaction testing component. Let us partition $S_{1,1}$ further into M^2 subgrids so that each subgrid has 10×10 cells. We focus on the M subgrids on the diagonal. We denote the M subgrids by $U_1, U_2, \ldots, U_M$. Formally, each U_b is defined as $U_b = S_{b,b}^{10M,M}$ ($b \in [M]$). We illustrate how $U_1, U_2, \ldots, U_M$ are located in Fig. 6. A subgrid U_b corresponds to the clause $C_b \in C$. When C_b has 2 literals (resp., 3 literals), we take the partition of U_b and the demands of the blocks as shown in Fig. 7 (resp., Fig. 8). The cell of a 1-block should be filled with its demand, and we cannot assign the demand value to any other cell in the same row or in the same column. For example, in Fig. 7, there are 1-blocks whose demands are from 5 to 10 in the 1st to 4th rows and in the 1st to 4th columns. Thus we cannot assign 5 to 10 to the upper-left 4×4 subgrid, and thus need to assign 1 to 4 there. We do not take up the gray 1-blocks any more. We associate each literal in C_b with a certain cell in U_b, as shown in the figures. When $|C_b| = 2$, the clause has one positive literal and one negative literal from Lemma 5 (iii). We denote the cell for the positive (resp., negative) literal by $u_{b,1}^{(2)}$ (resp., $u_{b,2}^{(2)}$). When $|C_b| = 3$, the clause has three positive literals, and we denote the cells for these literals by $u_{b,1}^{(3)}$, $u_{b,2}^{(3)}$ and $u_{b,3}^{(3)}$.

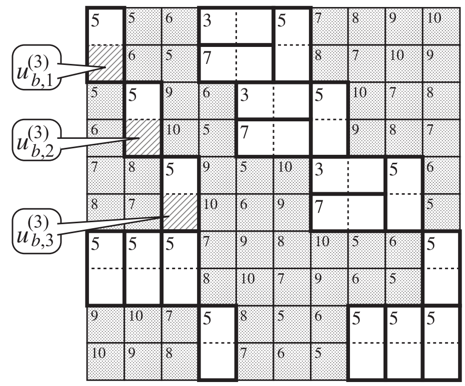


Fig. 8 Overview of U_b when the clause C_b has 3 literals.

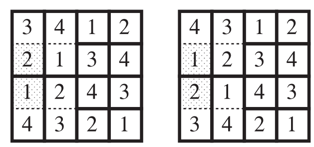


Fig. 9 The 2 possible configurations of integers to the 2-blocks in U_b ($|C_b| = 2$).

When $|C_b| = 2$, the 2-blocks in the subgrid U_b admits only 2 configurations of integers, as shown in Fig. 9. When $|C_b| = 3$, the 2-blocks in the subgrid U_b admits $6 \times 2^4 = 96$ configurations, as shown in Fig. 10; the cells from the 1st row to the 6th row admit only 6 configurations. We can exchange the integers assigned to two out of the four (2×1)-blocks in the 7th and 8th rows. For example, the assignment of $(1,4)$ and $(4,1)$ to the two (2×1)-blocks can be flipped into $(4,1)$ and $(1,4)$. This is also true for the four (2×1)-blocks in the 9th and 10th rows. Let us emphasize that, whether $|C_b| = 2$ or 3, the cells for the literals are assigned either 1 or 2, and they are not-all-equal.

We let every cell out of $U_1, \ldots, U_M$ form a 1-block. In any feasible configuration of integers to $U_1, \ldots, U_M$, the integers from 1 to 10 are assigned in all rows and in all columns of $S_{1,1}$. We readily see that we can assign integers from 11 to $10M$ to the cells out of $U_1, \ldots, U_M$ so that the Latin square condition is satisfied. We use the assigned integers as the demands of the 1-blocks. Then, the configuration of the integers to these cells is unique.

(2) Truth Assignment Components

For each $a \in [N]$, we use the subgrids $S_{1,a+1}$, $S_{a+1,1}$ and $S_{a+1,a+1}$ as the truth assignment components for the variable $v_a \in V$. From Lemma 5 (i), v_a appears as 2 positive literals and as 1 negative literal in C. These 3 literals are associated with some 3 cells in $S_{1,1}$. We denote the 2 cells for the 2 positive literals by $u_{a,1}$ and $u_{a,2}$, and the cell for the negative literal by $u_{a,3}$. As shown in Figs. 7 and 8, these cells belong to $(2{\times}1)$-blocks. We denote the three $(2{\times}1)$-blocks to which

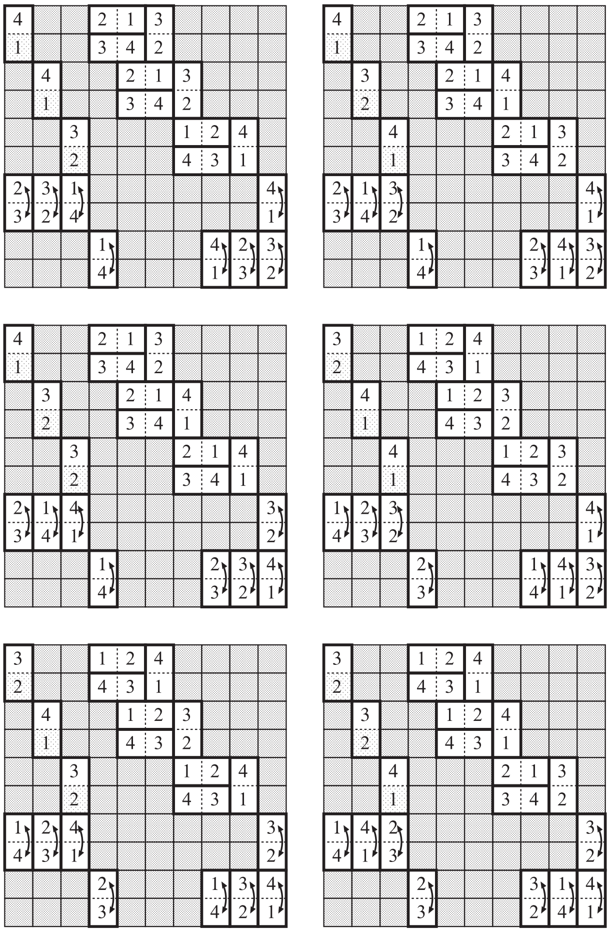


Fig. 10 The $6 \times 2^4 = 96$ possible configurations of integers to the 2-blocks in U_b ($|C_b| = 3$).

$u_{a,1}$, $u_{a,2}$ and $u_{a,3}$ belong by $A_{a,1}$, $A_{a,2}$ and $A_{a,3}$, respectively. For convenience, let us write:

$$A_{a,1} = \{(r_1, c_1), (r_1 + 1, c_1)\},$$
$$A_{a,2} = \{(r_2, c_2), (r_2 + 1, c_2)\},$$
$$A_{a,3} = \{(r_3, c_3), (r_3 + 1, c_3)\},$$

which means $u_{a,1} = (r_1 + 1, c_1)$, $u_{a,2} = (r_2 + 1, c_2)$ and $u_{a,3} = (r_3, c_3)$.

In Fig. 11, we show an overview of $S_{1,a+1}$, $S_{a+1,1}$ and $S_{a+1,a+1}$. The subgrid $S_{1,a+1}$ has three (2×1)-blocks. All the other blocks in $S_{1,a+1}$ are 1-blocks. We denote the three (2×1)-blocks by $D_{a,1}$, $D_{a,2}$ and $D_{a,3}$, where we define

$$D_{a,1} = \{(r_1, 10Ma + 1), (r_1 + 1, 10Ma + 1)\},$$
$$D_{a,2} = \{(r_2, 10Ma + 2), (r_2 + 1, 10Ma + 2)\},$$
$$D_{a,3} = \{(r_3, 10Ma + 3), (r_3 + 1, 10Ma + 3)\}.$$

Thus, these 3 blocks are in the same rows as $A_{a,1}$, $A_{a,2}$ and $A_{a,3}$, respectively. We construct $\psi_{1,a+1}$, the $(10M) \times (10M)$ Latin square that gives the solution to $S_{1,a+1}$ along with π, by permuting the columns of the standard Latin square of order $10M$ so that, in the 1st, 2nd and 3rd columns, 1 is in the r_1-th, r_2-th and r_3-th rows respectively. One can readily see that

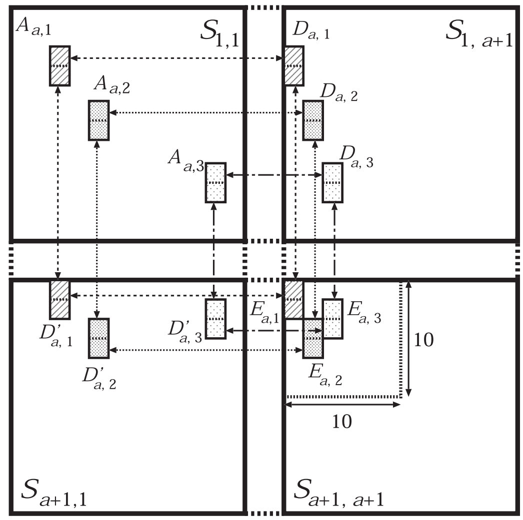


Fig. 11 Overview of the partitions of $S_{a+1,1}$, $S_{a+1,a+1}$ and $S_{1,a+1}$ that we use as the truth assignment components for the variable v_a.

this permutation is always feasible. With π and $\psi_{1,a+1}$, the demands of all the blocks in $S_{1,a+1}$ are determined by (7). In particular, the demands of $D_{a,1}$, $D_{a,2}$ and $D_{a,3}$ becomes:

$$\sigma(D_{a,1}) = \sigma(D_{a,2}) = \sigma(D_{a,3}) = (10Ma + 1) + (10Ma + 2) = 20Ma + 3. \quad (8)$$

Note that the configuration of integers to the subgrid $S_{1,a+1}$ is unique; the 1-blocks need to be assigned their demands. The 2-block $D_{a,t}$ ($t = 1, 2, 3$) must be assigned $(10Ma + 1, 10Ma + 2)$ since, in the r_t-th row (resp., in the $(r_t + 1)$-th row), all the integers from 1 to n except $10Ma + 1$ (resp., $10Ma + 2$) need to be assigned to other cells other than $D_{a,t}$.

We set the partition and the demands of the subgrid $S_{a+1,1}$ in the similar way. The subgrid $S_{a+1,1}$ has three (2×1)-blocks, and all the other blocks are 1-blocks. We denote the three (2×1)-blocks by $D'_{a,1}$, $D'_{a,2}$ and $D'_{a,3}$, where we define

$$D'_{a,1} = \{(10Ma + 1, c_1), (10Ma + 2, c_1)\},$$
$$D'_{a,2} = \{(10Ma + 3, c_2), (10Ma + 4, c_2)\},$$
$$D'_{a,3} = \{(10Ma + 2, c_3), (10Ma + 3, c_3)\}.$$

Thus, these 3 blocks are in the same columns as $A_{a,1}$, $A_{a,2}$ and $A_{a,3}$, respectively. We construct $\psi_{a+1,1}$ by permuting the columns of the standard Latin square of order $10M$ so that, in the c_1-th, c_2-th and c_3-th columns, 1 is in the 1st, 3rd and 2nd rows respectively. With π and $\psi_{a+1,1}$, the demands of all the blocks in $S_{a+1,1}$ are determined by (7). In particular, we have

$$\sigma(D'_{a,1}) = \sigma(D'_{a,2}) = \sigma(D'_{a,3}) = 20Ma + 3. \quad (9)$$

Note that the configuration of integers to the subgrid $S_{a+1,1}$ is unique.

Finally, we explain the partition and the demands for the subgrid $S_{a+1,a+1}$. The key part of $S_{a+1,a+1}$ is the upper-left 10×10 subgrid. We show the 10×10 subgrid in Fig. 12.

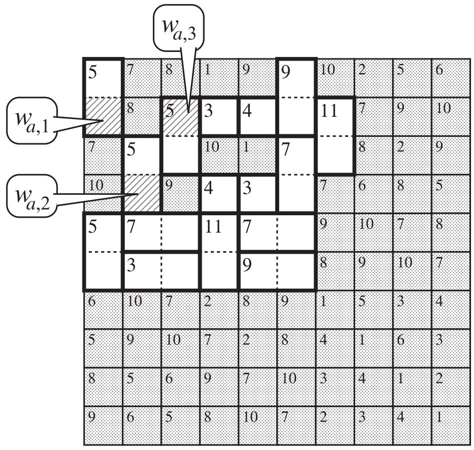


Fig. 12 The upper-left 10×10 subgrid in $S_{a+1,a+1}$.

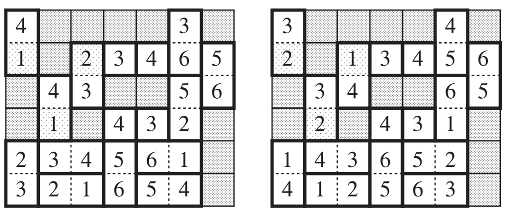


Fig. 13 The 2 possible configurations of integers to the 2-blocks in the upper-left 10×10 subgrid of $S_{a+1,a+1}$.

We focus on the three (2×1)-blocks, denoted by $E_{a,1}$, $E_{a,2}$ and $E_{a,3}$, which are defined as:

$$E_{a,1} = \{(10Ma + 1, 10Ma + 1), (10Ma + 2, 10Ma + 1)\},$$
$$E_{a,2} = \{(10Ma + 3, 10Ma + 2), (10Ma + 4, 10Ma + 2)\},$$
$$E_{a,3} = \{(10Ma + 2, 10Ma + 3), (10Ma + 3, 10Ma + 3)\}.$$

Recall that the Boolean variable $v_a \in V$ appears as 2 positive literals and as 1 negative literal in C. We associate the 2 positive literals with $w_{a,1} = (10Ma + 2, 10Ma + 1)$ and $w_{a,2} = (10Ma + 4, 10Ma + 2)$, the lower cells of $E_{a,1}$ and $E_{a,2}$ respectively. We also associate the 1 negative literal with $w_{a,3} = (10Ma + 2, 10Ma + 3)$ that is the upper cell of $E_{a,3}$. These are represented as shaded cells in Fig. 12. The 2-blocks in the 10×10 subgrid admits only 2 configurations of integers, as shown in Fig. 13. The point is that $w_{a,1}$, $w_{a,2}$ and $w_{a,3}$ are assigned either 1 or 2 in any configuration, and that $w_{a,1}$ and $w_{a,2}$ are assigned the same integer, while $w_{a,3}$ is assigned the different integer from $w_{a,1}$ and $w_{a,2}$. We let every cell out of the upper-left 10×10 subgrid form a 1-block. We readily see that we can assign integers from 1 to $10M$ to the $(10M)^2 - 10^2$ cells so that the assignment satisfies the Latin square condition whichever configuration is employed in the upper-left 10×10 subgrid. We use the assigned integers as the demands of the $((10M)^2 - 10^2)$ 1-blocks.

(3) Garbage Collection Components

Let us denote any $10M \times 10M$ subgrid that is not mentioned above by $S_{a,b}$. We let the partition be the set of $(10M)^2$ 1-blocks, and the standard Latin square of order $10M$ be $\psi_{a,b}$. Thus the configuration of integers to $S_{a,b}$ is unique.

Finally, we transfer I_{BS} to another instance $I'_{\text{BS}} = (\mathcal{B}, \sigma')$ by exchanging the demands of some blocks by means of Lemma 4, while we do not change the partition $\mathcal{B}$. Again, see Fig. 11. For each Boolean variable $v_a \in V$ and $t = 1, 2, 3$, $D_{a,t}$ in $S_{1,a+1}$ is in the same rows as $A_{a,t}$ in $S_{1,1}$ and $D'_{a,t}$ in $S_{a+1,1}$ is in the same column as $A_{a,t}$. The $E_{a,t}$ in $S_{a+1,a+1}$ is in the same column as $D_{a,t}$ and in the same rows as $D'_{a,t}$. Also we have $\sigma(A_{a,t}) = \sigma(E_{a,t}) = 5$ and $\sigma(D_{a,t}) = \sigma(D'_{a,t}) = 20Ma + 3$, where the latter is from (8) and (9). Then we define the demand function $\sigma' : \mathcal{B} \to [n^2(n + 1)/2]$ as follows; for any $a = 1, 2, \ldots, N$ and $t = 1, 2, 3$, we define:

$$\sigma'(A_{a,t}) = \sigma'(E_{a,t}) = 20Ma + 3,$$
$$\sigma'(D_{a,t}) = \sigma'(D'_{a,t}) = 5,$$

and $\sigma'(B) = \sigma(B)$ for any other block B in $\mathcal{B}$.

We have finished explaining the transformation from I_{SAT} to I'_{BS}. Note that every block in I'_{BS} has at most 2 cells. We can readily see that the transformation time is bounded by a polynomial in M and N. Next we show that I'_{BS} has a solution if and only I_{SAT} is NAE-satisfiable.

Suppose that I_{SAT} is NAE-satisfiable. Let $\tau : V \to \{\text{True}, \text{False}\}$ denote a truth assignment such that each clause $C_b \in C$ is not-all-equal. From τ, we first construct a solution φ of I_{BS} and then transform it into a solution of I'_{BS} by means of Lemma 4. For I_{BS}, the configuration to the cells out of $S_{1,1}, S_{2,2}, \ldots, S_{N+1,N+1}$ is unique. Also, for $S_{1,1}$, the configuration to the cells out of $U_1, \ldots, U_M$ is unique, and for $S_{a+1,a+1}$ ($a \in [N]$), the configuration to the cells out of the upper-left 10×10 subgrid is unique. In U_b ($b \in [M]$) that corresponds to the clause C_b, there are 2 or 3 cells with which the literals in C_b are associated. Assign 2 (resp., 1) to the cell if the literal is true (resp., false) under τ. Since C_b is not-all-equal, the assigned integers are not-all-equal. Therefore, when $|C_b| = 2$ (resp., 3), we can employ one of the configurations shown in Fig. 9 (resp., Fig. 10). In the upper-left 10×10 subgrid of $S_{a+1,a+1}$ ($a \in [N]$), assign $(2, 2, 1)$ (resp., $(1, 1, 2)$) to the cells $w_{a,1}$, $w_{a,2}$ and $w_{a,3}$ if $\tau(v_a) = \text{True}$ (resp., False), which leads to one of the configurations shown in Fig. 13. We easily see that, for $t = 1, 2, 3$, $\varphi(A_{a,t}) = \varphi(E_{a,t})$ and $\varphi(D_{a,t}) = \varphi(D'_{a,t})$ hold. Therefore, this φ can be transformed into a solution of φ' by means of Lemma 4.

Conversely, suppose that I'_{BS} is solvable. Let φ' denote any solution of I'_{BS}. For any $a = 1, 2, \ldots, N$, let us see the (2×1)-blocks $D'_{a,1}$, $D'_{a,2}$ and $D'_{a,3}$ in the subgrid $S_{a+1,1}$ whose demands by σ' are 5. Let $w'_{a,1}$ and $w'_{a,2}$ denote the lower cells of the $D'_{a,1}$ and $D'_{a,2}$ respectively and $w'_{a,3}$ denote the upper cell of $D'_{a,3}$.

Lemma 6: We have $\varphi'(w'_{a,t}) \in \{1, 2\}$ for $t = 1, 2, 3$ and $\varphi'(w'_{a,1}) = \varphi'(w'_{a,2}) \neq \varphi'(w'_{a,3})$.

Proof: Since the demand $\sigma'(D'_{a,t})$ is 5, $\varphi'(w'_{a,t})$ can be either of 1, 2, 3 or 4. However, it cannot be 3 or 4 since, in the same row as $w'_{a,t}$, there are 1-blocks in the subgrid $S_{a+1,a+1}$

whose demands are 3 and 4. (See Figs. 11 and 12.) The latter can be confirmed easily. □

Lemma 7: For any clause C_b ($b \in [M]$) with 2 literals, suppose that the positive literal is from the Boolean variable v_{a_1} and the negative literal is from v_{a_2}. Let us denote by D'_{a_1,t_1} (resp., D'_{a_2,t_2}) the (2×1)-block in $S_{a_1+1,1}$ (resp., $S_{a_2+1,1}$) that is in the same column as the cells $u^{(2)}_{b,1}$ and $u^{(2)}_{b,2}$ of U_b. Then we have $\varphi'(w'_{a_1,t_1}) \neq \varphi'(w'_{a_2,t_2})$.

Proof: Since D'_{a_1,t_1} and D'_{a_2,t_2} are in the same column (see Figs. 7 and 11), $\varphi'(w'_{a_1,t_1})$ and $\varphi'(w'_{a_2,t_2})$ should not be equal from the Latin square condition. □

Lemma 8: For any clause C_b ($b \in [M]$) with 3 literals, suppose that the positive literals are from the Boolean variables v_{a_1}, v_{a_2} and v_{a_3}. Let us denote by D'_{a_1,t_1} (resp., D'_{a_2,t_2} and D'_{a_3,t_3}) the (2×1)-block in $S_{a_1+1,1}$ (resp., $S_{a_2+1,1}$ and $S_{a_3+1,1}$) that is in the same column as the cell $u^{(3)}_{b,1}$ (resp., $u^{(3)}_{b,2}$ and $u^{(3)}_{b,3}$) of U_b. Then, $\varphi'(w'_{a_1,t_1})$, $\varphi'(w'_{a_2,t_2})$ and $\varphi'(w'_{a_3,t_3})$ are not all equal.

Proof: See Figs. 8 and 11. The values $\varphi'(w'_{a_1,t_1})$, $\varphi'(w'_{a_2,t_2})$ and $\varphi'(w'_{a_3,t_3})$ should be not all equal due to the three (2×1)-blocks that are in the 7th to 8th rows and in the 1st to 3rd columns in Fig. 8. □

From a solution φ' of I'_{BS}, we construct a truth assignment $\tau : V \to \{\textsc{True}, \textsc{False}\}$ as follows; for each Boolean variable $v_a \in V$, when $\varphi'(w'_{a,1}) = 2$ (resp., 1), let $\tau(v_a) \leftarrow \textsc{True}$ (resp., $\textsc{False}$). From Lemma 6, the φ' assigns 2 (resp., 1) to the 2 cells $w'_{a,1}$ and $w'_{a,2}$ and 1 (resp., 2) to the cell $w'_{a,3}$. It is regarded that the 2 positive literals are true (resp., false) and the negative literal is false (resp., true) under τ. On the other hand, from Lemmata 7 and 8, φ' assigns not all equal integers to $w'_{a,t}$'-s that belong to the same columns as U_b in $S_{1,1}$. This means that the clause C_b is not-all-equal under τ.

4. Discussion and Concluding Remarks

We showed that the decision problem version of BlockSum puzzle is NP-complete even if every block has size at most 2.

Although we focus on the existence of a solution of a given BlockSum instance, it is actually easy to generate an instance of BlockSum that has at least one solution; we can generate such an instance from a pair of an $n\times n$ Latin square and a partition of the $n \times n$ grid into blocks [8]. Thus, from the viewpoint of puzzle instance generation, it is important to consider the complexity of deciding whether a BlockSum instance has a solution other than the expected solution. This type of problem (and the complexity) are studied in terms of ASP-completeness, which does not require a polynomial time reduction from an NP-complete problem but requires an ASP-reduction from an ASP-complete problem [1]. Thus, it is an interesting and important open problem to decide whether BlockSum is ASP-complete or not.

References

[1] T. Yato and T. Seta, "Complexity and completeness of finding another solution and its application to puzzles," IEICE Trans. Fundamentals, vol.E86-A, no.5, pp.1052–1060, May 2003.
[2] E.D. Demaine, S. Hohenberger, and D. Liben-Nowell, "Tetris is hard, even to approximate," CoRR, vol.cs.CC/0210020, 2002. (http://arxiv.org/abs/cs.CC/0210020).
[3] H. Muta, "PUYOPUYO is NP-complete (in Japanese)," IEICE Technical Report, COMP2005-14, http://ci.nii.ac.jp/naid/10016436795/, 2005.
[4] T. Matsukane and Y. Takenaga, "NP-completeness of maximum chain problem on generalized PUYOPUYO (in Japanese)," IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J89-D, no.3, pp.405–413, March 2006.
[5] G. Aloupis, E.D. Demaine, and A. Guo, "Classic Nintendo games are (NP-)hard," arXiv.org, http://arxiv.org/abs/1203.1895v1, 2012.
[6] T. Miyamoto, Kyouiku Pazuru (in Japanese), Discover, 2004.
[7] T. Miyamoto, Kyouikuron (The Art of Teaching without Teaching) (in Japanese), Discover, 2004.
[8] K. Haraguchi, Y. Abe, and A. Maruoka, "How to produce BlockSum instances with various levels of difficulty," J. Information Processing, vol.20, no.3, pp.727–737, 2012.
[9] T.J. Schaefer, "The complexity of satisfiability problems," Proc. 10th Annual ACM Symposium on Theory of Computing, pp.216–226, Association for Computing Machinery, 1978.
[10] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, ch. Appendix A9, p.259, W.H. Freeman & Company, 1979.

Kazuya Haraguchi received B.E., Master of Informatics, and Doctor of Informatics from Kyoto University, in 2001, 2003 and 2007, respectively. He is currently with the Department of Information Technology and Electronics, Faculty of Science and Engineering, Ishinomaki Senshu University. His research interest includes algorithms, combinatorial optimization, and their application to artificial intelligence and operations research.

Hirotaka Ono received his B.E, M.E. and Doctor of Informatics degrees from Kyoto University in 1997, 1999 and 2002 respectively. He is currently an associate professor of Department of Economic Engineering of Kyushu University. His research interests include combinatorial optimization, logical analysis of data and distributed algorithms. He is a member of the Information Processing Society of Japan and the Operation Research Society of Japan.