Incremental Constraints for Integer Linear Programming to Solve Hashiwokakero

Sheng-Yu Cheng[a] and Chang-Biau Yang[b] *

[a]Kaohsiung Municipal Kaohsiung Senior High School, Kaohsiung, Taiwan
[b]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan

Abstract

Based on the integer linear programming (ILP) model for solving the Hashiwokakero puzzle, proposed by Coelho et al. in 2019, we introduce an incremental constraint method. In the original IP model, the constraints for checking the connectivity of the puzzle solution are huge and difficult to implement. In our incremental method, the constraints for checking the connectivity are added gradually. As experimental results show, the execution time required for solving the dataset of Hashiwokakero is reduced with our incremental method.

Keywords: Hashiwokakero, integer linear programming, NP-complete, incremental constraint

1 Introduction

Hashiwokakero, also known as *Hashi* or *bridges*, is a puzzle invented by Nikoli Co., Ltd. in 1990 [8]. Hashi is played on a grid. Some grid points have circles, where each of the circles, called an *island*, has a number inside it. Players have to draw vertical or horizontal lines between islands. These lines are called *bridges*. A simple example is shown in Figure 1.

The connection scheme of the bridges has to satisfy the following rules:

(1) Each bridge starts on an island and ends on another island.

(2) Each bridge is either vertical or horizontal.

(3) Every two bridges do not cross each other.

*Corresponding author. E-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

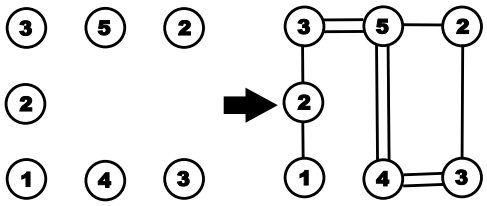

Figure 1: An example of the Hashi puzzle. Here, the total number of connected bridges (vertical or horizontal lines) should be equal to the number inside the island (circle), and these bridges cannot be crossed.

(4) At most two bridges can be connected between a pair of islands.

(5) The number inside the island is equal to the number of bridges connected to the island.

(6) Every island can be reached from any other island.

Specifically, rule (6) is called the *connectivity constraint* in the game. A more complicated puzzle is shown in Figure 2

For the previous studies on Hashi, in 2009, Anderson [1] proved that the Hashi puzzle is an NP-complete problem via reduction from a Hamiltonian circuit problem on a grid graph of unit distance. Moreover, in 2019, Coelho *et al.* [2] proved that the Hashi puzzle without the connectivity constraint is still NP-complete by reducing the problem of reconstructing disjoint sets of orthogonal segments (RDOS) to the modified Hashi puzzle. Note that the RDOS has been proven to be NP-complete [2]. In addition, in 2011, Golan proved that Hashi puzzle can be reduced to the minesweeper puzzle [5].

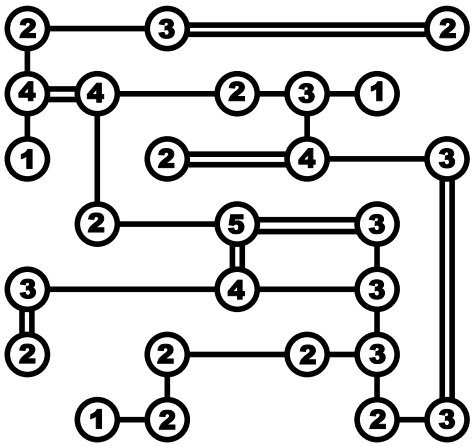

Figure 2: A finished Hashi puzzle with 26 islands on a 7×7 grid.

For solving Hashi puzzle, Malik *et al.* [7] used a heuristic algorithm based on the depth-first search (DFS) to solve it, where the heuristic strategy is according to human player's decisions when playing Hashi. Table 1 summarizes the previous studies of Hashi.

Coelho *et al.*[2] introduced an *integer linear programming* (ILP) model for efficiently solving the Hashi puzzle. Based on their ILP model, we first add a maximization objective and then add connectivity constraints incrementally. The testing data are downloaded from the website of Vidal [9], one of the authors for publishing the article in 2019 [2]. As experimental results show, the execution time required for solving the puzzles is reduced.

The rest of the paper is organized as follows. Section 2 describes the previous research. Section 3 presents our algorithm. Section 4 illustrates the result of computational experiments. The conclusion is given in Section 5.

2 Preliminaries

The ILP model of Coelho *et al.* [2] is described as follows. Let a Hashi puzzle be represented by an undirected graph $G(V, E)$. V denotes the set of vertices (all islands), and E denotes the set of edges (all possible vertical lines or horizontal lines). For each vertex (island, drawn by a circle) $v \in V$, let d_v represent the number of bridges required to connect to v, which is recorded inside a circle for v. δ_v denotes the set of vertices vertically or horizontally adjacent to v. If $u \in \delta_v$, then $e_{uv} \in E$ is a possible bridge connecting u and v.

Let y_{uv} and x_{uv} denote the existence variable and multiplicity variable of edge e_{uv}, respectively. In other words, $y_{uv} = 1$ if and only if e_{uv} is selected as a bridge. If $y_{uv} = 1$, then $x_{uv} \in \{1, 2\}$ indicates the number of lines connecting u and v. Additionally, let Δ be the set of pairs of crossing edges. Coelho *et al.* [2] derived the following equations of ILP from the rules of the Hashi puzzle.

$$\sum_{u \in \delta_v} x_{uv} = d_v, v \in V \tag{1}$$

$$y_{uv} \leq x_{uv} \leq 2y_{uv}, (u, v) \in E \tag{2}$$

$$y_{uv} + y_{st} \leq 1, ((u, v), (s, t)) \in \Delta \tag{3}$$

$$\sum y_{uv} \geq 1, (u, v) \in E, S \subset V, u \in S, v \in V \setminus S \tag{4}$$

$$x_{uv} \in \{0, 1, 2\}, (u, v) \in E \tag{5}$$

$$y_{uv} \in \{0, 1\}, (u, v) \in E \tag{6}$$

Equation1 represents that the number of bridges connected to v is equal to d_v. Equation2 restricts the multiplicity of an edge to be maximum of 2, minimum of 1 if the existence is 1, and to be equal to 0 if the existence is 0. Equation3 forbids the cross of two edges. Equations 5 and 6 tell the possible values of multiplicity and existence, respectively.

Equation 4implements the strong connectivity constraint, which makes every vertex be reachable from another vertex, since the equation itself means every subgraph is connected to its complement. Since it is not practical to enumerate all subgraphs of G in Equation 4, Coelho *et al.* thus introduced the weak connectivity constraint, which hope to force the solution to be at least a spanning tree, given as follows.

$$\sum y_{uv} \geq |V| - 1, (u, v) \in E \tag{7}$$

3 Our Algorithms

There are five variant versions of our algorithm. In the first version, we adopt loosen constraints, excluding the connectivity constraint. In other words, Equations 1, 2, 3, 5 and 6 are included, and do not maximize anything (maximize a constant). In the second version, we add the weak connectivity constraint to the first version, namely including Equations 1, 2, 3, 5, 6 and 7, and do not maximize anything.

Table 1: The previous studies of Hashi puzzles.

Year	Author(s)	Note
2009	Andersson [1]	solving Hashi puzzle with DFS, NP-complete, proved via reduction from unit-distance Hamiltonian circuit
2011	Golan [5]	reducing Hashi to Minsweeper
2012	Malik *et al.* [7]	DFS, heuristics and backtracking
2019	Coelho *et al.* [2]	solving Hashi puzzle with ILP

In the third version, we use the maximization of the edge amount of the solution to replace Equation 7, as follows.

$$maximize \sum y_{uv}, (u, v) \in E \tag{8}$$

In other words, we employ the objective function in Equation 8 to enhance the connectivity constraint. However, the maximization of the connected edges still cannot ensure the connectivity of the solution graph.

In the fourth version, we incrementally add the strong connectivity constraint (a portion of Equation 4) to the third version and solve the model repeatedly. Whenever we get a solution from ILP, we examine the validity by checking whether the solution forms a connected graph. If the solution is not valid, starting from a random vertex, we invoke DFS to find the first connected component. The vertices in the found component form the S in Equation4. Subsequently, we can add a strong connectivity constraint for forcing at least one edge between S and $V \setminus S$ included in the subsequent ILP solver iterations.

For the fifth version, since it is inefficient to add merely one constraint (connectivity between one component and its complement) in each iteration, we design a more efficient way to add our constraints. Equation 4 implies that every subgraph of the Hashi game should construct a strong connectivity constraint. Therefore, we add a strong connectivity constraint for every single component in the fractional solution. In other words, we add several constraints in one iteration. This can effectively reduce the iteration count. Section 4 gives the details.

Table 2 shows the construction of our algorithm. Algorithm 1 presents the pseudo-code for versions 3, 4 and 5.

Table 2: Construction of our algorithm.

version	adopted equations	objective function
1	(1)(2)(3)(5)(6)	maximize a constant
2	(1)(2)(3)(5)(6)(7)	maximize a constant
3	(1)(2)(3)(5)(6)	maximize (8)
4	(1)(2)(3)(5)(6) and incrementally adding (4) by one constraint per iteration	maximize (8)
4	(1)(2)(3)(5)(6) and incrementally adding (4) by several constraints per iteration	maximize (8)

Algorithm 1 An incremental algorithm for solving Hashi

Input: A Hashi game G
Output: A valid answer of G

```
Let ILP be an integer linear programming model
Add constraints (1)(2)(3)(5)(6) to ILP
Set objective to maximize ∑ y_uv (u, v) ∈ E
while True do
    solve the ILP model
    H ← the answer of the ILP model
    if H is connected then
        return H
    else
        x ← a random vertex in H
        S ← DFS on H to find the connected component starting from x
        Add a new strong connectivity constraint with the form in Equation 4
```

4 Experimental Results

In this section, we compare the execution time of the five versions of our algorithm and the experiment results of Coelho *et al.* [2]. The five

versions of our algorithm were implemented using Google Colab [6] with Python. Google Colab is a free cloud-based platform, equipped with COIN-OR cbc [3], a non-profit open source ILP solving package. In Google Colab, we use pyomo [4] to connect our algorithm and COIN-OR cbc package. The Google Colab's computer is equipped with Debian GNU/Linux 10 (buster), a CPU clocked at 2.20 GHz(Intel Xeon), and 16 GB of memory.

We use the same dataset [9] as the technical report [2] used. Each version of our algorithm performs the experiments on the whole dataset. A whole dataset comprises $|V| \in \{100, 200, 300, 400\}$. Each category includes 4 levels of connectivity $\alpha \in \{0\%, 5\%, 10\%, 15\%, \}$, as well as 3 levels of double-edge $\beta \in \{0.25, 0.5, 0.75\}$. Each selected difficulty $(|V|, \alpha, \beta)$ contains 30 distinct cases. The execution time is calculated as the average of the time for the 30 distinct cases.

Table 3 shows our experiment results, displaying the correct rate and average time. We discover that as the number of islands goes up, the level of connectivity goes down and the level of double-edge goes up, the Hashi puzzle gets harder. This observation aligns with the experiment results reported by Coelho *et al.* [2]. The last column presents the experiment results provided by Coelho *et al.* [2].

Since we are unable to reconstruct the solving algorithm in [2], the experimental results in Table 3 are just a copy from their technical report. Their algorithm was implemented with Visual Basic.

Since the dataset with $(|V|, \alpha, \beta) = (400, 0\%, 0, 75)$ is the hardest among all datasets, we analyze it more precisely. Figure 3 shows the number of strong connectivity incrementally added into the ILP model in version 4 and version 5. Note that the total length of the bars is 30, since the dataset with $(400, 0\%, 0, 75)$ consists of 30 test cases.

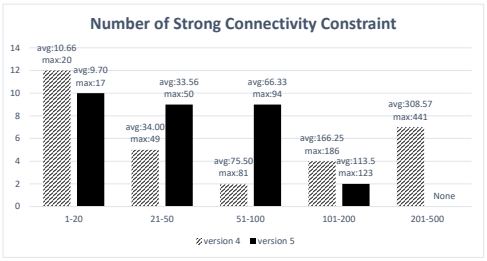

Figure 3: The bar graph for the numbers of strong connectivity constraints added in the test cases with $(|V|, \alpha, \beta) = (400, 0\%, 0.75)$ of algorithm version 4 and version 5.

5 Conclusion

In this paper, we enhance the ILP model for solving Hashi puzzles by adding a maximization function and incremental connectivity constraints. Though the required time for solving some cases of Hashi puzzle still remains high, our approach results in a faster average speed compared to the scenario without a maximization function.

In the future, we will aim to design a more precise description of connectivity constraint for solving Hashi puzzle in the ILP model. It may lead to a faster speed for solving Hashi.

References

[1] D. Andersson, "Hashiwokakero is NP-complete," *Information Processing Letters*, Vol. 109, No. 19, p. 1145–1146, 2009.

[2] L. C. Coelho, G. Laporte, A. Lindbeck, and T. Vidal, "Benchmark instances and branch-and-cut algorithm for the Hashiwokakero puzzle," tech. rep., arXiv, 2019.

[3] J. Forrest, T. Ralphs, S. Vigerske, H. G. Santos, J. Forrest, L. Hafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, Jan-Willem, rlougee, jpgoncal, S. Brito, h-i gassmann, Cristina, M. Saltzman, tosttost, B. Pitrus, F. Matsushima, and to st, "coin-or/Cbc: Release releases/2.10.11," Oct. 2023. `https://github.com/coin-or/Cbc`.

[4] J. Forrest, T. Ralphs, S. Vigerske, H. G. Santos, J. Forrest, L. Hafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, Jan-Willem, rlougee, jpgoncal, S. Brito, h-i gassmann, Cristina, M. Saltzman, tosttost, B. Pitrus, F. Matsushima, and to st, "coin-or/pyomo," 2024. `https://pypi.org/project/Pyomo/`.

[5] S. Golan, "Minesweeper on graphs," *Applied Mathematics and Computation*, Vol. 217, No. 14, pp. 6616–6623, 2011.

[6] Google LLC (Limited Liability Company), "Google colaboratory," 2024. `https://colab.research.google.com/`.

Table 3: Performance of our algorithm

Dataset			Correct Rate (%)					Average Time(s)					
$\|V\|$	β	α	ver1	ver2	ver3	ver4	ver5	ver1	ver2	ver3	ver4	ver5	Coelho *et al.*
100	0.25	0%	33	53	100	100	100	0.43	0.49	0.45	0.51	0.73	0.05
		5%	60	60	83	100	100	0.53	0.67	0.65	0.55	0.72	0.04
		10%	66	66	90	100	100	0.51	0.63	0.65	0.51	0.69	0.04
		15%	63	66	100	100	100	0.50	0.63	0.64	0.51	0.68	0.03
	0.50	0%	10	6	53	100	100	0.53	0.66	0.66	0.62	0.88	0.11
		5%	3	10	66	100	100	0.53	0.65	0.65	0.96	0.80	0.10
		10%	23	20	83	100	100	0.54	0.67	0.68	0.54	0.83	0.11
		15%	26	16	83	100	100	0.53	0.65	0.66	0.58	0.77	0.09
	0.75	0%	0	10	43	100	100	0.56	0.69	0.69	0.64	1.08	0.22
		5%	0	6	46	100	100	0.57	0.70	0.70	0.63	1.06	0.39
		10%	0	3	76	100	100	0.60	0.72	0.73	0.77	0.76	0.39
		15%	0	0	76	100	100	0.56	0.69	0.71	0.63	0.82	0.36
200	0.25	0%	10	26	90	100	100	0.61	0.73	0.73	0.61	0.80	0.16
		5%	40	36	86	100	100	0.61	0.75	0.75	1.05	0.79	0.15
		10%	46	53	93	100	100	0.68	0.82	0.74	0.66	0.81	0.13
		15%	63	70	96	100	100	0.60	0.73	0.75	0.66	0.78	0.13
	0.50	0%	0	0	30	100	100	0.69	0.70	0.81	0.80	1.23	0.52
		5%	0	0	46	100	100	0.73	0.87	0.90	0.93	1.66	0.53
		10%	3	10	50	100	100	0.71	0.85	0.83	0.81	1.27	0.49
		15%	3	0	80	100	100	0.73	0.87	0.87	0.85	1.14	0.44
	0.75	0%	0	0	10	100	100	0.94	1.04	1.03	2.01	3.40	2.13
		5%	0	0	43	100	100	0.88	0.98	1.00	1.66	2.55	2.87
		10%	0	0	43	100	100	1.00	1.04	1.02	1.47	2.75	3.15
		15%	0	10	73	100	100	1.13	1.13	1.04	1.12	1.21	2.91
300	0.25	0%	10	23	66	100	100	0.71	0.84	0.84	0.75	1.08	0.43
		5%	16	23	80	100	100	0.72	0.85	0.85	0.77	0.97	0.38
		10%	16	33	90	100	100	0.72	0.85	0.85	0.72	0.94	0.40
		15%	40	0	86	100	100	0.74	0.87	0.87	0.76	0.97	0.35
	0.50	0%	0	0	20	100	100	0.98	1.13	1.21	1.79	3.60	2.34
		5%	0	0	43	100	100	1.11	1.14	1.16	1.63	2.59	1.86
		10%	0	0	33	100	100	1.01	1.13	1.20	1.43	2.27	1.89
		15%	0	0	63	100	100	1.30	1.39	1.23	1.48	3.01	1.69
	0.75	0%	0	0	3	100	100	1.55	1.65	1.78	47.25	17.36	15.38
		5%	0	0	13	100	100	1.94	2.01	1.99	7.07	12.04	19.35
		10%	0	0	20	100	100	2.40	2.31	1.91	3.70	6.77	18.14
		15%	0	0	46	100	100	2.45	2.44	2.02	3.10	4.80	13.66
400	0.25	0%	6	0	83	100	100	0.82	0.95	0.95	0.90	1.12	0.92
		5%	10	6	83	100	100	0.85	0.99	1.01	0.92	1.17	0.85
		10%	3	16	96	100	100	0.84	0.95	0.98	0.89	1.05	0.73
		15%	3	33	93	100	100	0.88	1.02	1.03	0.88	1.12	0.80
	0.50	0%	0	0	20	100	100	1.4	1.66	1.61	4.98	6.28	6.89
		5%	0	0	20	100	100	1.58	1.73	1.61	2.58	4.93	5.71
		10%	3	3	43	100	100	1.62	1.59	1.49	2.65	3.55	6.55
		15%	0	0	46	100	100	2.63	2.91	3.21	3.89	6.63	7.05
	0.75	0%	0	0	3	100	100	3.57	3.54	3.70	273.68	66.04	76.65
		5%	0	0	20	100	100	4.32	3.87	3.99	75.36	30.13	152.18
		10%	0	0	16	100	100	5.24	5.28	3.31	52.78	28.78	100.68
		15%	0	0	33	100	100	4.10	4.10	3.06	34.02	7.74	84.32

[7] R. F. Malik, R. Efendi, and E. A. Pratiwi, "Solving Hashiwokakero puzzle game with hashi solving techniques and depth first search," *Bulletin of Electrical Engineering and Informatics*, Vol. 1, No. 1, p. 61–68, 2012.

[8] Nikoli Co., Ltd., "Puzzle cyclopedia," 2004. ISBN 4-89072-406-0.

[9] T. Vidat, "test data of article benchmark instances and branch-and-cut algorithm for the Hashiwokakero puzzle." website, 2019. `https://w1.cirrelt.ca/~vidalt/en/research-data.html`.

A Appendix: Pyomo Code for ILP

In this appendix, we present the pyomo code to build the ILP model for solving Hashi in Algorithm 2.

Algorithm 2 The pyomo code to build the ILP model for solving Hashi

```
self.model = ConcreteModel()
self.model.vars = Var([f"E{i}" for i in range(self.e)] + [f"T{j}" for j in range(self.e)], domain = NonNegativeIntegers)
self.model.objective = Objective(expr = sum(self.model.vars[f"E{i}"] for i in range(self.e)), sense = maximize)
self.model.constraints = ConstraintList()
# Amount Condition
for i do in range(self.e):
    self.model.constraints.add(self.model.vars[f"T{i}"] <= self.model.vars[f"E{i}"])
    self.model.constraints.add(self.model.vars[f"E{i}"] <= 2 * self.model.vars[f"T{i}"])
# Sum of Island Condition
for i do in range(1, self.d1 + 1):
    for j do in range(1, self.d2 + 1):
        if self.vertex_map[i][j] != None then
            v = self.vertex_map[i][j]
            expr = 0
            for k do in range(4):
                if v.edges[k] != None then
                    expr += self.model.vars[f"E{v.edges[k].idx}"]
            self.model.constraints.add(expr = expr == v.num)
# Crossing Condition
for i do in range(self.cross):
    self.model.constraints.add(expr = self.model.vars[f"T{self.cross_pair[i][0]}"] + self.model.vars[f"T{self.cross_pair[i][1]}"] <= 1)
```