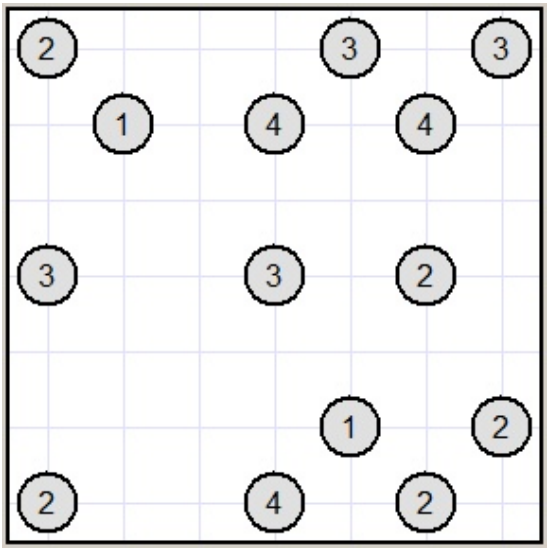
Universiteit Leiden
The Netherlands

Bachelor Informatica & Wiskunde



Generalising, solving, classifying and generating Hashi puzzles

Tiemen Ykema

Supervisors:
Jeannette de Graaf (LIACS) & Floske Spieksma (MI)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/08/2025

Abstract

Hashi, or Bridges, is a logical puzzle. Bridges must be placed to connect the islands in the puzzle. Islands indicate how many bridges should be attached to them, and all islands must be connected together. This offers an intriguing challenge different from grid-based puzzles, which are more common. The main focus is on providing a generalised version of the puzzle, multiple solving techniques, a classification and ways to generate puzzles.

Contents

1 Introduction

Imagine a group of islands close together. People using boats to get around from one island to another. Yet, this is becoming inconvenient with the rising number of travelers between the islands. That's why each island got together teams of builders to build bridges. To build a bridge between two islands, both islands need a team. Each team can only focus on one bridge. You are the one designer. Try to connect all the islands. All teams of builders need to be put to work. If only there was someone to help solve this puzzle.

This bachelor thesis will focus on the Hashi (Hashiwokakero) puzzle genre[1], also known as Bridges or Building bridges. Hashi is an intriguing and purely logical puzzle genre. One with numerous unanswered research questions. Its solutions are not made up of placed symbols, like in a grid-based puzzle genre. Instead, elements of the puzzle must be connected to reach a solution. The Hashi puzzle genre is discussed in many sources. Almost all in the form of a description with unnecessary extra wording. Therefore, this thesis provides its own version. All story is left out with the goal to have concise rules. These rules are also separated in the smallest sentences. This will make it easier to generalise upon.

1.1 Hashi definition

Hashi is a genre of puzzles. These all share the same type of input and identical objective. The input is what is presented to the solver. The objectives are what the solver must accomplish. Any output where all objectives are accomplished is a solution. Solvable puzzles have at least one solution. Uniquely solvable puzzles have exactly one solution. The input of an instance of Hashi is often represented on a rectangular **grid** of any size and contains the following elements:

- Grid intersections may contain *islands*.
- All *islands* contain a *number*.
- *Numbers* range from 1 through 8.

The objective is to place *bridges* such that the following constraints are met.

- Pairs of distinct *islands* share 0, 1, or 2 *bridges*.
- *Bridges* are oriented horizontally or vertically.
- *Bridges* do not cross other *bridges* or *islands*.
- The number of *bridges* attached to each *island* must equal the *island*'s *number*.
- All *islands* and *bridges* must form a single *connected* component.

There is a rule for what a *connected* component is.

- Distinct *islands* sharing at least one bridge are called *connected*.

[1]When mentioning Hashi, it means the whole type of puzzles, not some instance(s) of it. For the latter, Hashi puzzle instance will be used. Hashi and instance might be left out. Those words are implied. An example Hashi puzzle instance is on the titlepage.

- Distinct *islands* being *connected* is a transitive property.
- In a *connected* component, any two distinct *islands* are *connected*.

1.2 Thesis goals

This thesis will address the following key objectives:

- Expand the Hashi puzzle genre with generalisations, allowing more freedom in creating puzzles.
- Develop separate solving techniques that, through combined efforts, solve puzzles.
- Develop a puzzle difficulty classification system based on subsets of needed solving techniques.
- Generate uniquely solvable puzzles.

These key objectives address both theoretical and practical aspects of the Hashi puzzle genre. They will contribute to the broader field of puzzle-solving algorithms. Generalising the puzzle definition to allow for novel variants introduces originality and expands the scope of the classic Hashi puzzle genre. Human solving tricks will form a basis on how the computer might find solutions with a more general solving method. In this way, the research builds upon existing knowledge, while aiming to uncover novel insights into possible solving techniques. Classifying the puzzle difficulty based on these solving techniques can increase understanding of the complexity of puzzles. It also provides a framework for future studies aiming to generate puzzles by difficulty group or scale. Lastly, developing a method to generate uniquely solvable instances ensures practical relevance. It offers tools for use in educational or recreational contexts. Overall, these objectives are relevant, original, and build upon established research, making them a valuable contribution to the field.

1.3 Research questions

The main research question is

How can generalising, solving, classifying, and generating instances of Hashi be realised?

A subdivision into multiple questions is made.

- How can the Hashi definition be generalised to allow variants of Hashi that can be solved with generalised or new methods?
  - Which conditions in the Hashi definition can be relaxed? Are they interesting?
  - Which additions to the Hashi definition introduce more possible restrictions?
  - What consequences do relaxations of and additions to the definition have on solving?
- Which solving techniques can be used for solving Hashi?
  - Which different solving techniques or tricks do humans use on Hashi?
  - Based on which puzzle information can be determined that two distinct islands must necessarily be connected by some number of bridges or cannot be connected?
  - Can any puzzle be solved, considering only local solving techniques, only local and global techniques, or will a method like backtracking be unavoidable?

- (How) can puzzles be classified by difficulty using subsets of solving techniques?
  - How can solving techniques be separated to function on their own towards a (partly) solved puzzle?
  - How powerful is each solving technique?
  - Can logic be found in the difficulty classification of Hashi puzzle sources, or might it be human assigned?
- How can uniquely solvable instances for Hashi be generated?
  - How can solvable instances for Hashi be generated?
  - How and when can it be validated that an instance has a unique solution?
  - (How) can it be ensured that any allowed puzzle might be generated?

1.4 Thesis overview

The Hashi puzzle genre was introduced in Section 1 along with formulating the thesis goals and reasearch questions Some related work will be discussed in Section 2. The main part of this thesis will then consist of four parts. First will be shown how the puzzle genre can be generalised in Section 3. Then how puzzles can be solved using a combination of multiple techniques in Section 4. How different puzzles can be classified will be shown in Section 5. Furthermore, methods of generating puzzles will be discussed in Section 6. Lastly, some experiments are presented in Section 7 and conclusions and future research in Section 8.

This bachelor thesis is authorised by the LIACS and MI departments of Leiden University and supervised by Jeannette de Graaf (LIACS) and Floske Spieksma (MI).

2 Related Work

The Hashi puzzle is a logical puzzle, thus belonging to single-player combinatorial game theory in mathematics. Logical puzzles have long interested researchers due to their potential to enhance critical thinking and problem-solving skills [MEWA90, Olwnd]. A deep understanding of a puzzle's type is essential not only for solving existing puzzles, but also for generating new puzzles, preferably uniquely solvable and interesting. While some logical puzzles like Sudoku and its variants have been extensively covered by previous work, enough opportunity remains to explore other types of puzzles.

There is some previous work on Hashi. This thesis takes a step forward on what was previously accomplished, thereby contributing mutually to the broader field of study. Knowledge used from previous work is limited to the knowledge used to compare techniques against each other. The reason being the use of different techniques, besides human techniques that are common knowledge known by puzzlers. The generalisation also takes a different spin on Hashi that is different from anything that was done before. Previous work on Hashi sometimes contains descriptions of a few human techniques, like a bachelor thesis by T. Morsink and another work by Malik et al. [Mor09, MEP12]. Both of these also include a version of backtracking that is not guaranteed to solve any Hashi puzzle. The first one includes another method where puzzles are transformed to an input of an external form of SAT solver and a method to generate solvable puzzles not always

having unique solutions. Human techniques can also be found on Wikipedia [Wik25]. Another work focuses on mathematical restrictions, performs a branch-cut backtracking algorithm that solves any puzzle, and also generates solvable puzzles not always having unique solutions [CLLV19]. It has also been proven that **Hashi** is NP complete [And09].

Some previous bachelor theses in Leiden are similar to this one in that they solve and generate logical puzzles. There is one from Gerhard van der Knijf on solving and generating puzzles with connectivity constraints in general [vdK21], one from Hanna Straathof on the Kuroshuto puzzle [Str25], one from Casper Jol on the Marupeke puzzle [Jol23], finally, one from Rob Mourits on the Nurimeizu puzzle [Mou22]. There are many more. However, since these are about different puzzle genres, they only serve as examples of how a thesis about puzzles can be useful to the field.

3 Generalising

From the clearly separated rule set in the **Hashi** definition, it is easy to take a critical look at every rule. If they might limit the puzzle in some way, it can be generalised. The next definition will provide an intuitive generalisation for every rule in the classic definition, if there is any.

Generalised Hashi definition

The input of an instance of generalised **Hashi** contains the following elements:

- A **graph** $G = (V, E)$ of any size.
- Two natural numbers, M and k.
- An indicator function $cross : E^2 \to \{0, 1\}$.
- The nodes in set V represent *islands*.
- All *islands* contain a natural *number*.

The objective is to place *bridges* such that the following constraints are met.

- Pairs of distinct islands share up to at most M *bridges*.
- *Bridges* can only be placed on graph edges.
- *Bridges* must not *cross* other *bridges*: $x, y \in E$ cannot both have bridges for $cross(x, y) = 1$.
- The number of *bridges* attached to each island must equal the island's *number*.
- All islands and *bridges* must form k *connected* components.

The defintion of a connected component is as provided in the **Hashi** definition in Section 1.1.

The key correspondences between these definitions include the use of bridges and islands with numbers. It includes the fact that distinct islands sharing at least one bridge are connected and that the number of bridges attached to each island must equal the island's number. Apparently these form the core concept of this puzzle genre.

The key difference is that a general graph is used instead of being limited to a rectangular grid. This works well, because the core concept of this puzzle does not rely on the rectangular grid structure to begin with. The added value of the rectangular grid is in keeping it simple and easy to understand and visualise. While generalising, however, it is not important anymore. This difference introduces several things to address. Firstly, the rectangular concept of orientation is lost. Therefore, the rule forcing horizontal and vertical bridges is replaced by allowing bridges only on graph edges. This leaves full freedom to the puzzle instance maker, since the graph, including its edges, is chosen. Secondly, the planar structure and thus the original meaning of crossing are also lost. The most elegant solution to this is simply explicitly defining which graph edges are or are not crossing instead. This is done with an indicator function

$$cross : E^2 \to \{0, 1\}$$
$$cross(x, y) = \begin{cases} 1 \text{ if } x, y \in E \text{ cannot both have bridges, since those bridges would cross} \\ 0 \text{ if } x, y \in E \text{ can both have bridges, since those bridges would not cross} \end{cases}.$$

The function is part of the puzzle instance, again leaving it completely to the puzzle instance maker. This function is necessary to be able to let classic **Hashi** puzzles also satisfy the rules of generalised **Hashi**. In that case, the function is not up to the puzzle maker, but decided upon the spatial locations of islands. Any two edges $x, y \in E$ with $cross(x, y) = 1$ are called a conflict edge pair, since there is a conflict if both edges would contain a bridge. The edges that form a conflict edge pair with another edge are called conflict edges of that edge. Lastly, the graph structure removes the upper bound of only four other islands to possibly share a bridge with. Before, islands could only share bridges with the closest island to the north, south, east, and west, if there was one. As a result, together with the maximum of 2 bridges shared by a pair of islands, the number of an island was limited to 1 through 8. For generalised **Hashi**, numbers above 8 can naturally be allowed. This was not possible before. Of course,e number is still limited by the graph size and its specific edges. However, the graph size itself may be arbitrarily large and the edges arbitrarily chosen.

Introducing the graph generalised concepts and meanings. The other generalisations introduce parameters to replace fixed numerals. The maximum of 2 bridges shared by a pair of islands is replaced by M bridges. Instead of forming 1 connected component, k components must be formed. Allowing up to M bridges for a chosen M also removes the upper bound on island numbers, just like the graph structure did. Because there is no upper bound on the number that islands can have anymore, there is no need to specify which range of numbers is allowed, like in the classic **Hashi** definition. However, some subtlety arises with just leaving out the range. While it is clear the numbers are not negative, nothing prevents them from being 0. However, in the original definition, islands with number 0 are indeed excluded. After all, such an island can never be connected to any other island. Therefore, will always form an additional connected component of its own and therefore removing any possible solution. Ways to deal with this issue include the following

- Still banning 0's by definition, like in the original definition.
- Workaround by saying 0's are excluded from the connected component rule.
- Ask for k components by definition. This allows up to $k - 1$ 0's as a natural part of solvable puzzles.

Here, the last one is by far the best option. It is more general than banning 0, it does not use a workaround for a specific number. Furthermore, besides solving the issue, it is a valuable generalisation on its own.

The key takeaway is that while simplicity and some structure is lost, the general definition allows any classic puzzles and many more. Any puzzle satisfying the original definition can be modelled as a puzzle satisfying the general definition. This is exactly what a generalisation should do. For any classic **Hashi** puzzle, a graph can be constructed. It will have edges between islands that were located next to each other without islands in between. Classic **Hashi** uses $M = 2$ and $k = 1$. Together with the structure of this constructed graph, this forces the island's numbers to be between 1 and 8, which they indeed are. The *cross* function can be made such that it aligns with the spatial meaning of crossing, like in the classic **Hashi** definition. Therefore, this is a successful generalisation. Note that for the rest of this thesis, k is assumed to be 1. Solving puzzles where this is not the case will be discussed in Section 8, along with other future research possibilities.

Example

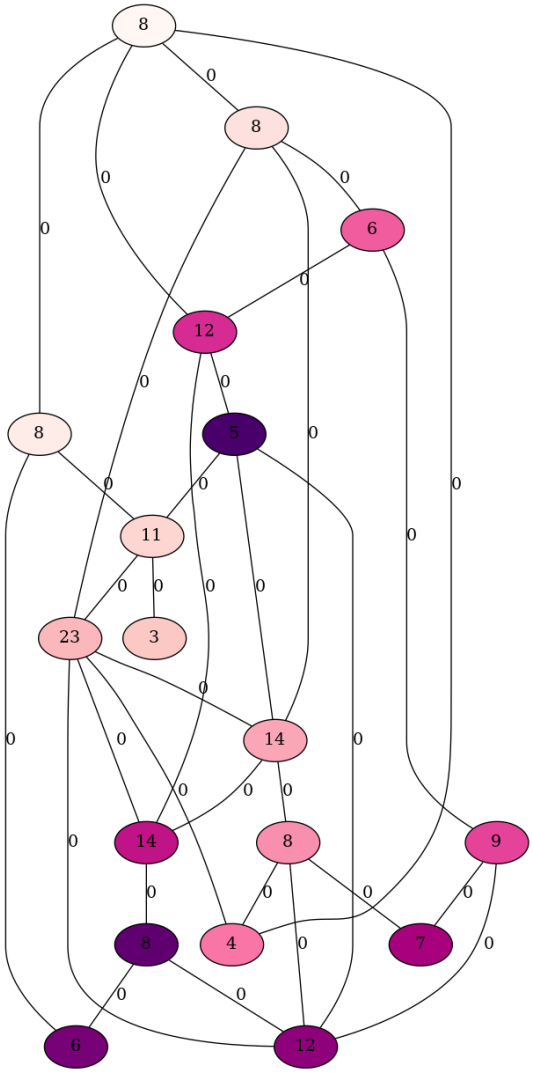


Figure 1: A generalised Hashi puzzle instance with $M = 4$, to show an example of what it might look like.

Figure 1 is a picture output of a puzzle, generated by the second generalised puzzle generator that will be described in Section 6. For this puzzle, the global maximum number of bridges is 4. In ilsand with number 7 in the right bottom corner, with only two edges, shows that edges having 4 bridges need to exist in a solution. This example also shows non-horizontal and non-vertical edges. Because the bridges are placed on these graph edges, instead of purely horizontal and vertical, a node can have more than 4 edges. This can be seen on the middle left, the island with number 23. More edges means that the number can be even higher. It might look like this puzzle has no or multiple solutions if one tries to solve it by hand. However, that is because the conflict edge pairs are not properly shown in this picture. Note that if lines are crossing, that does not mean those edges are crossing. It is just the way the graph was projected on 2D. The actual conflict edge pairs that make this puzzle uniquely solvable are invisible on the picture.

4 Solving

To solve Hashi, a solving method is constructed using multiple solving techniques[2]. All are individually able to deduce where bridges need to be or other information, progressing the puzzle towards a solution. All of them have their own strengths, limitations and scenarios where they deduce something resulting in progress. For Hashi, a distinction exists between solving techniques using local and global information.

Local information is found close around a specific node or edge. Formally, a reference to the edges of a node or nodes sharing an edge is followed no more than $t \in \mathbb{N}$ times before the information is retrieved. For any local technique, t is known in advance, regardless of the puzzle size. Two examples are given. Note that each edge stores its own maximum number of bridges. Let's imagine a local technique only using the number of a node itself and the information gathered from the edges attached to that node. To do this, only $t = 1$ reference is needed to access all required information. Therefore, the technique only using this information is a local technique. Another local technique might need to know all edges unable to have a bridge when some specific edge has gained a bridge. These can be stored as a list of other edges for every edge, only needing $t = 1$ reference so far. However, if from those edges, their nodes and the edges attached to these nodes need to be looked at, that technique would have $t = 3$. The first example relates to the Degree of freedom technique in Section 4.1 and the second example to the simplifying technique Reducing prevented edges in Section 4.2.

Global information includes looking at an unknown number of nodes and edges. For example, a technique that aims to find a whole connected component. The number of references will be dependent on how big the component is, thus on the puzzle size. There is no fixed $t \in \mathbb{N}$ such that the number of references will stay below t regardless of the puzzle size, while executing this technique on one component. Local information can always be accessed in fixed time, no matter how big the puzzle instance is. Global information cannot. This makes techniques using only local information much cheaper.

For this reason, a solving method combining all techniques was constructed in such a way that techniques using local information are always prioritised. Other techniques are only used when local techniques cannot progress the puzzle anymore.

Some explanation is given on what to expect from the solving techniques, with regard to classic versus generalised Hashi. Most of the techniques will work correctly for any classic or generalised puzzle, regardless of how many solutions there are or what value of M or k they might have. Some exceptions are the following. The external human techniques from Section 4.1.1 only work for classic Hashi, since they assume $M = 2$, $k = 1$ and islands only having up to 4 neighbours to share bridges with. These human techniques have no chance of finding multiple solutions if there are more than 1. Next, the local connectivity technique from Section 4.1.2 assumes $k = 1$. This means it will work incorrectly when this technique is still used on a puzzle with $k \neq 1$. The global techniques from Section 4.3 assume $k = 1$ as well. They too rely on the connectivity constraint from the classic Hashi definition. All islands and bridges must form a single connected component. Future research could provide adapted or other techniques in these last two cases. Otherwise these

[2]A solving technique is simply a technique that helps progress while solving, a solving method is a method that will be able to solve puzzles on its own.

techniques could simply not be used when the solver comes across a puzzle with $k \neq 1$. Note that the main focus of the combined solver of this thesis is on solving generalised puzzles with any kind of graph, thus allowing more than 4 neighbouring nodes to share bridges with and alternative definitions of which bridges cross. Allowing any kind of M is an extension that can be supported by the solver relatively naturally when generalising techniques to be independent of exact numbers. As such, it remains close to the thesis objectives and has been focussed on as well. The input parameter k, however, is mainly included out of completeness and as an idea for future research, since it will alter solving techniques quite a bit, not staying close to the research of this thesis.

First some local solving techniques are given. These will include human techniques described in other sources. Some more techniques will be shown to simplify the puzzle. This is done by not looking at solved or conflicting parts anymore. Then the more complex techniques that use global information will be discussed. After that, backtracking will be discussed and finally two theorems on puzzles not having exactly one solution.

4.1 Local solving techniques

A definition and theorem will be provided for a powerful technique to locally deduce where some bridges need to be. The intuition here will be that a specific number of bridges must be placed somewhere, because if they are not, the number of bridges still to be placed is strictly higher than how many can still be placed elsewhere. Another perspective is supposing all possible positions except one have their maximum allowed number of bridges. The leftovers then need to be placed at the one possible position left.

Definition 1 (Degree of freedom technique)**.** *Take any (possibly uniquely solvable) instance of* Hashi *with at least an island i.*

(a) Let n_i be the number assigned to island i.

(b) Let S_i be the collection of islands that can potentially share a bridge with i.

(c) Let M_{ij} be a maximum number of bridges that can be placed between islands i and j, such that we can prove or have proved that no valid solution can have strictly more than M_{ij} bridges shared between i and j. In an implementation the value of M_{ij} will change as the puzzle progresses towards a solution.

(d) Let $C_i := \sum_{j \in S_i} M_{ij}$ be the maximum total number of bridges that can be attached to island i.

Now define the ***degree of freedom*** *as $DoF_i := C_i - n_i$. So the difference between how many bridges there can be and how many there need to be at island i.*

Theorem 1 (Degree of freedom technique)**.** *Any valid solution must have at least $M_{ij} - DoF_i$ bridges shared by islands i and j, for every $j \in S_i$.*

Proof. Let $j \in S$. Suppose islands i and j share strictly less than $M_{ij} - DoF_i$ bridges in a valid solution, say $M_{ij} - DoF_i - k$, $k > 0$. Then we have that

$$n_i - M_{ij} + DoF_i + k = n_i - M_{ij} + C_i - n_i + k = C_i - M_{ij} + k > C_i - M_{ij}.$$

The left hand side is the remaining number of bridges island i needs besides the bridges shared between islands i and j. The right hand side is the maximum total number of bridges where i is one of the islands the bridges connect besides the bridges shared between islands i and j. The fact that this left hand side is strictly greater than the right means that we have more bridges to add then positions where we can add them. This makes it impossible to arrive at a valid solution that has strictly less than $M_{ij} - DoF_i$ bridges shared between islands i and j. □

Intuitively one can say that we place $M_{ij} - DoF_i$ bridges shared between i and j (for every $j \in S$) because if we place as many bridges as we can somewhere besides shared with j, then we still need to place the remaining $M_{ij} - DoF_i$ bridges, which can only be shared between i and j. Algorithmically, the technique works like described in Algorithm 1.

Algorithm 1: Degree of Freedom

```
Let e_ij be the current number of bridges shared between islands i and j;
Pop an island i from the queue;
Progress ← False;
DoF_i ← Σ_{j∈S_i} M_ij − n(i);
for j ∈ S_i do
    if M_ij − DoF_i > e_ij then
        e_ij ← M_ij − DoF_i;
        Progress ← True;
    end
end
Return Progress;
```

4.1.1 Human techniques

The degree of freedom technique was made baed on research on human techniques to solve Hashi puzzles. More specifically, what they lack and how an algorithm can be made that makes as little assumptions as feasible. Here, human techniques understood to be techniques used and known by average puzzlers. These include specific patterns easily spotted by humans. The degree of freedom technique is more general and ecompasses all following useful human techniques, making them no longer needed for solving. Yet, it is still useful to touch upon these human techniques. Comparing them shows that for some specific patterns the degree of freedom technique will yield the same result as one of the human techniques. None of the techniques in this subsection will be proven to be correct since all of the correct techniques are encompassed by the technique in the previous subesection and the one in the following subsection. These human techniques will also be useful for an experiment in Section 7. The solving strength of human techniques from two external sources will be compared against the combined solving method of this thesis. Note that it is hard to get exactly correct how these sources would have implemented what they say, but an attempt was made.

Staying close to the wording and meaning of the previous works, some human techniques are as follows. On Wikipedia some human techniques are described quite clear [Wik25].

- An island showing ‘3’ in a corner, ‘5’ along the outside edge, or ‘7’ anywhere must have at least one bridge radiating from it in each valid direction.
- A ‘4’ in a corner, ‘6’ along the border, or ‘8’ anywhere must have two bridges in each direction.
- This can be generalized as added bridges obstruct routes: a ‘3’ that can only be travelled from vertically must have at least one bridge each for up and down, for example.
- Two islands with number 1 can only share a bridge if this is the complete solution.

For the work by Malik et al. [MEP12], more interpretation is needed as there is little to no description of the techniques and one specific example for each.

- Connect an island with its neighbours when the number of bridges owned by the island is the same as the number of bridges that can be placed.
  - Only an example of a ‘4’ in a corner is given and that there need to be 2 bridges is only mentioned in that specific example. Yet, for experimental purposes, all other such cases have been assumed to be correctly included as well. So for any ‘2’, ‘6’ or ‘8’ with 1, 3 or 6 neighbours, 2 bridges will be placed towards all neighbours.
- Connect an island with its neighbours when there is only one neighbour island near the island with the rest of the weight of one.
- When there is a ‘3’ that can only reach two other islands, and one has only a number 1, one bridge must be placed to that island and 2 bridges to the other island.
- Connect an island with its neighbours if the island has a weight of N and the number of neighbouring islands is N, with $(N-1)$ neighbours have a weight / remaining weight $= 1$, then build a bridge from the island with neighbours who do not have a remaining weight of 1.
  - Here it would be useful to know what exactly they mean with neighbours. If nodes with remaining weight 0 are still considered neighbours or not. This is not specified, and no mention of not looking at already solved islands exists. Therefore, neighbours are assumed to still include ones with remaining weight 0.
  - Also, for the neighbours the word ’remaining’ is specifically added. For the island it just says ’weigth’ seemingly implying the original weigth of the starting puzzle. However, that interpretation seems contradictory to the example which shows only one neighbour with remaining weight 1, while the original weight of the island is 4 and has two bridges already.
  - Regardless of which way to interpret the things above, this technique will place bridges where they should not. Therefore, this technique, as it is written, is wrong. Thus, it is not used in the comparison. The problem lies in the island possibly already sharing bridges with its neighbours which also counts towards neighbours possibly having remaining weight 1.
- If two islands with weight one are connected, then the two islands will become isolated. Therefore, these islands must be connected with other islands instead.

– Nothing is said about these islands possibly being the only two islands of a puzzle.

4.1.2 Local connectivity

All techniques so far but one have not touched upon the connectivity constraint. One human technique that does not use global information does this. This is the technique about not placing a bridge between two islands with number 1. Unless these two islands are the entire puzzle, in which case placing 1 bridge between them is the unique solution. Suppose two islands with number 1 are sharing a bridge. They will both not share any other bridges, because that would violate the definition saying the number of bridges attached to each island must equal the island's number. This means, however, that these two islands will never be able to form a connected component with any other islands. Because that would require at least one bridge to be shared between one of those islands and another island. Therefore, to have a valid solution where two islands with number 1 share a bridge, these two islands must be all islands. Since the definition states all islands and bridges must form a single connected component.

This existing technique is, in essence, another special case. It can be generalised along with the generalisation of the Hashi definition with M. For example, suppose two islands having number 2 share 2 bridges. Just like the previous case, they will not be able to share any other bridges. Therefore, the same argument applies. Thus, only in a puzzle where two islands having number 2 are the only two islands, may they share 2 bridges. Otherwise, they can share a bridge, just not two. In general, when 2 is not a global maximum number of bridges, the following can be said.

Theorem 2. *Two number m islands can share up to $m-1$ bridges, unless they are all islands.*

Proof. Note that if an island with number m shares more than m bridges with another island, it can never be part of a valid solution. The number of bridges attached to each island must equal the island's number, not be more. This applies even if those two islands are all islands.

Suppose a solution exists where two number m islands share m bridges. From the definition, the number of bridges attached to each island must equal the island's number. Therefore, the islands may not share any bridges with any other islands. This means those two islands are connected with each other, but not with any other islands. However, because all islands and bridges must form a single connected component, these two islands must be all islands, since a valid solution was assumed. □

The local connectivity constraint can be neatly tied into the degree of freedom technique. This will use the convenient nature of the degree of freedom technique. Using a specific maximum number of bridges for each edge allows for any other technique to insert a lower maximum. This means all information from the result of Theorem 2 can be used seemlessly by the degree of freedom technique. Any bridge that can be deducted because of the local connectivity constraint, can then also be deducted by the degree of freedom technique. That is, if the values for the maximum number of bridges for specific edges are set in a special way. This can be done by going through all edges only once. For every edge between nodes that share the same number m, the maximum number of bridges can be set to the minimum of the global maximum and $m-1$.

Since specific maximum numbers of bridges are being set at the start already, more known information can be inserted. Any maximum number of bridges must also not be higher than either island's number. Though the coming simplifying techniques will dynamically encompass this as well. So at best doing this at the start will reduce solving time.

The maximum number of bridges are set once at the start by Algorithm 2. This makes sure any information known from the local connectivity constraint is used while executing the degree of freedom technique for the rest of the solve. This makes another separate technique for the local connectivity constraint obsolete.

Algorithm 2: Smartly setting max bridges

for *edges* **do**
  edge specific max bridges = min(global max bridges, first island's number, second island's number $-\mathbb{1}_A$), A = island's numbers are equal;
end

4.2 Simplifying techniques

Note that no solving technique needs to consider parts of the puzzle that necessarily must be solved in a known way. As such, the puzzle can be reduced to only the part that still needs solving. For Hashi, we split off the possible positions for bridges, where the exact number of bridges in any possible solution is known. These edges will remain in a separate list that is not considered for progressing the puzzle. For any node, all edges that have not been put in the separate list are called the active edges of that node. There are two main cases when the number of bridges for an edge can become known. The first is when an edge is known to have at least one bridge. Therefore, any edge that cannot have a bridge at the same time as that edge should contain exactly 0 bridges and never needs to be considered again. The second is when it becomes known that an edge requires another bridge and then it reaches the maximum number of bridges for to that edge. Reaching this maximum might mean that an adjacent island has reached its designated number of bridges, for example. All that island's active bridges can then be put in the separate list and the island will be called satisfied.

The simplifying techniques are described in Algorithm 3, Algorithm 4, and Algorithm 5. These are all local techniques, since they only use local information.

Algorithm 3: Reducing prevented edges

```
/* One or more bridges are about to be placed on edge e; */
if Edge e has 0 bridges yet then
    for edges c that cannot have any bridge when edge e has any bridge do
        for both nodes sharing edge c do
            if edge c in active edges list of this node then
                Remove edge c from the list of active edges and add it to a list of edges that
                 never get any bridge;
            end
        end
    end
end
```

Algorithm 4: Reducing local max bridges

```
/* One or more extra bridges have just been placed on edge e; */
for both nodes sharing edge e do
    Update how many extra bridges this node can support;
    for edge e' in active edges list of this node do
        Local max bridges of edge e' = min(local max bridges of edge e', how many bridges
         edge e' currently has + how many extra bridges this node can support);
    end
end
```

Algorithm 5: Reducing completed edges

```
/* One or more bridges have just been placed on edge e; */
for both nodes sharing edge e do
    for edge e' in active edges list of this node do
        if this edge has reached its maximum number of bridges or either node that shares
         this edge has 0 bridges left to place then
            for both nodes sharing edge e' do
                Remove edge e' from the list of active edges and add it to a list of edges
                 that has reached the exact number of bridges it must have, so that no
                 more bridges need to be considered;
                Lower how many bridges in total this node needs with its active edges
                 (since some of its bridges with a previously active node are now cut out of
                 the active puzzle);
                Perform a check on the node if it is still possible to reach a solution;
            end
        end
    end
end
```

4.3 Global solving techniques

Important to note here is that techniques using global information have a worse complexity than techniques that only use locally available information. This is why, the combined solver uses the local techniques to their full solving potential before using global techniques at all.

In this section, one global technique will mainly be discussed. This technique combines many ideas that were gradually added into one final technique. First, the main idea behind the technique will be explained. Then, each idea that improved upon the main idea will be discussed. Finally, two separate techniques that were not implemented will be described.

The main idea of the technique is based on the definition requiring all islands and bridges to form a single connected component. The technique will only work for generalised puzzles if $k = 1$. For a starting node it will first search the entire component of nodes that are connected with bridges to the starting node. Then, it will draw a conclusion about whether adding specific bridges shared by the starting node and others will close that component. That means that if those specific bridges were to be added the component has no more chance to be connected to any other nodes that are not already in the component. If this is the case for those specific bridges, then it knows those cannot be placed like that. Therefore, at a bridge must be placed differently from those specific bridges. If there is one option for that, a bridge will be placed.

Algorithm 6 and Algorithm 7 will show a simplified version of the implementation for the global technique. The actual code for this is divided among multiple functions and is very large and complex in comparison to that of the degree of freedom technique. Algorithm 6 shows the main part of the technique, and Algorithm 7 the most crucial sub-technique. The sub-technique has one necessary and two optional arguments. A node i, set of nodes to ignore, and a set of edges that conflict with an edge towards i. The first argument is the starting node. The second argument is a set of nodes that will not count as finding a node outside the component that can be connected to the component by placing a bridge. The first time Algorithm 7 is called it is used to ignore the nodes the starting node can share a bridge with, the second time to ignore the starting node. The last argument is used when trying to find out if placing a bridge on some specific edge will make a closed component. It then checks against the conflict edges of that edge in a way that will be described after the algorithms.

Algorithm 6: Global solving technique(s)

```
Let e_ij be the current number of bridges shared between islands i and j;
Progress ← False; /* For making sure backtracking is only used when no other technique
 can make more progress; */
for node i in all nodes except ones who have the same number of bridges attached as their
 number; do
    Bridgeable nodes ← nodes that i CAN SHARE at least one more bridge with;
    /* Check if i is the only node of its current component that can still share a bridge
     with a node outside the component; */
    if Algorithm 7 returns False with arguments i, bridgeable nodes, and an empty set then
        Return False; /* Do not do anything and it will try again for another node; */
    end
    Initialise non isolated nodes with all bridgeable nodes;
    Initialise isolated nodes empty;
    for node j in bridgeable nodes do
        /* Else there is no way to form a closed component; */
        if The number of additional bridges that can be attached to node j ≤ the number of
         additional bridges that can be attached to node i then
            if Algorithm 7 returns 1 with arguments j, the set with only node i, and the
             conflict edges from the edge between node i and j then
                Remove node j from non isolated nodes and add it to isolated nodes;
            end
        end
    end
    Find all subsets of the isolated nodes such that if enough bridges towards the selected
     isolated nodes were placed to satisfy those nodes, the starting node is exactly
     satisfied as well.
    for subsets do
        if there is only one node that can share a bridge with node i, but is not in the
         subset then
            if while making sure there is at least one bridge between that node and i an
             additional bridge is added then
                Return True;
            end
        end
    end
end
Return False;
```

Algorithm 7: Almost closed component searching

```
Called with a node i, set of nodes to ignore, and a set of edges that conflict with an edge
  towards i. The last two are optional and can be empty;
possible outside bridgeable nodes is initialised as an empty set; /* A set of nodes that are
  possibly not in the component, but a bridge can be placed such that they will be; */
visited ← set with just node i in it;
for node j in nodes that i SHARES at least one bridge with do
    if node j not visited then
        if node j not in list to ignore then
            for edge e' in active edges list of node j do
                if e' has 0 bridges and is not in the list of conflict edges then
                    for both nodes sharing edge e do
                        Add it to possible outside bridgeable nodes;
                    end
                end
            end
        end
        add node to visited;
        Recursive call on node j, keeping other arguments the same;
    end
end
/* After the recursive part; */
Remove all visited nodes from the possible outside bridgeable nodes;
Return True if possible outside bridgeable nodes is nonempty and False if empty;
```

The first idea for an improvement on the basic idea is seen in Algorithm 7. This algorithm uses a search similar to Depth First Search. In particular, for each visited node, all nodes that share a bridge with them will be visited. As such, when visiting a node, it is not yet clear if nodes that do not share a bridge with the node will later be visited or not. This means that whenever a node has another node that it can share a bridge with, but does not yet, it is not known whether that other node is in the same component or not. Therefore, it is insufficient to stop searching when such a node is found, but the whole component needs to be searched. The purpose of looking at the component is to see if there are any nodes outside the component that can share a bridge with a node inside the component besides the starting node. At the end of searching the whole component, any nodes in the component will be removed from the set of nodes that might be this kind of node outside the component. Because this idea was utilised in the technique, even if a node besides the starting node still has opportunities to form additional bridges with other nodes inside the component, the technique still recognises that the component can become closed by adding bridges to the starting node.

Another idea is about finding subsets of the isolated nodes. Isolated nodes is a set filled as described in Algorithm 6. For any subset of isolated nodes such that placing enough bridges to satisfy all nodes in the subset will exactly satisfy the starting node as well the following can be said. If these bridges were to be placed, a closed component will form not including all nodes of the puzzle. Therefore, the number of bridges the starting node can still additionally share cannot be divided

over only the nodes of the subset. Therefore, the starting node must share a bridge with at least one other node. If there is only one other node available, then the starting node must share a bridge with that node. This has been done considering an arbitrary subset of the isolated nodes, because it is more powerful than only considering what happens when enough bridges were placed to satisfy one single node and the starting node at the same time. This idea allows the technique to place a bridge in the following example scenario. For example, a 4 that already shares one bridge with some satisfied node and has an edge towards a 1, 2 and 3. If the 4 shares one bridge with the 1 and two with the 2, a closed component is formed without the 3 in it. Therefore, 1 and 2 together form one of the subsets that should be found, so will the 3 on its own.

Finally, there's another idea tied into the global solving technique. It will be called the Special Case technique. That is because this idea came to mind after seeing a puzzle not be solved before this idea. It was then figured out by hand which additional bridge could be placed in that specific scenario. Then the idea was tied into the existing global technique. The idea comes from the following. When another node, besides the starting node, is found that can share a bridge with a node that is definitely outside the component, there is still another way to form a closed component. This has to do with conflict edge pairs, edges that cannot both have a bridge. In the above scenario, placing one bridge can satisfy one of the nodes and prevent the other node from sharing a bridge with a node outside the component at the same time. This results in a closed component. Because of this, Algorithm 7 has the possibility to provide conflict edges. It will consider this scenario as forming a closed component too and act on it just like otherwise.

Besides this case, there are likely many more situations that cannot be solved unless another specific idea is taken into account. Clearly, there can be many variations of global solving techniques. These might be similar, but each might take situations into account that others do not. This is likely because global solving techniques are used to gain information in a lot of differing puzzle situations. This was also ultimately the reason to implement backtracking, rather than continuing to try to improve the solver until it would be able to solve any puzzle. The latter not even sure to be possible.

Other global techniques that were chosen not to be implemented include the following. Firstly, keeping track of all components and all possible edges to connect a component with other components. Then using that at least one of those edges will need a bridge. Secondly, to look if there are edges that conflict with all edges from such a set. This means that edge will have exactly 0 bridges in a valid solution. Implementing these would likely cost a lot of bookkeeping and extra functionality for remembering and effectively using sets of edges where at least one needs at least one bridge. It is also not intuitively clear that this will indeed improve the solving strength.

4.4 Backtracking

Backtracking is a common solving technique. It is known for being able to solve various problems without needing much domain knowledge. When applied correctly it will always arrive at a solution. It will arrive at all solutions if multiple exist. The word backtracking means retracing steps. That is essentially all it does. It chooses randomly or based on a heuristic. When it fails, it undoes the previous choice. However, a drawback of its simplicity is that it might take much longer to solve problems. Another is that implementing backtracking does not make one learn much about the problem itself. For these reasons, a solver was made that avoids backtracking until necessary.

However, it is still implemented for the property that it is known it can solve any Hashi puzzle. This achieves the goal of having created a program that will solve any instance of Hashi. Backtracking solves any instance by eventually having tried all combinations of bridgeplacements unless they could not lead to a valid solution.

Doing backtracking on a Hashi puzzle goes like this. One bridge is placed that is not guaranteed to be part of a solution. Before doing this, the 'state' of the puzzle instance is stored on a stack. After placing the bridge, the non-backtracking solving techniques take over again. Then a solution will be reached, an error will occur or another bridge needs to be placed that is not guaranteed to be part of a solution. That last part is where we can distinguish how many steps - bridge placements - of backtracking are used. A more detailed description is given in Algorithm 8.

The bridge to be placed is chosen by a heuristic. This could be any heuristic, even random choosing. However, for this thesis, one specific heuristic is used. It orders all edges according to some layered criteria, then takes the first option. First, any edges with no bridges yet are given priority over all edges that have bridges already. This is because going from zero to nonzero bridges is more impactful then from some to more, because of conflict edges. Second, each edge will get a score

$$\sqrt{p} + \sqrt{q}$$

where p is the number of active edges attached to one node and q of the other node. The lowest score is taken. This means the edge will belong to nodes that have as little possible other edges to add a bridge to. This is because the chance to place a correct bridge is favorable and whenever the bridge is incorrect there are fewer other positions left to place a bridge. If there was just one alternative, that bridge can be placed regardless of backtracking.

Algorithm 8: Backtracking

```
First use any non-backtracking technique until none are useful anymore;
while puzzle not solved do
    Enable backtracking mode to handle errors differently;
    Save dynamic puzzle instance state in a stack;
    Use heuristic to select 'best' edge for increasing its bridges by 1 only considering edges
      where it is possible;
    Save chosen edge in another stack;
    Increase chosen edge's bridges by 1;
    while any non-backtracking technique is still useful and thus being used do
        if error occurs and in backtracking mode then
            Set state back to the previously saved state, popped from the stack;
            Pop previously chosen edge from the other stack;
            if that stack is empty then
                Disable backtracking mode;
            end
            Mark the edge as known 0 bridges;
        end
    end
end
```

4.5 0 or multiple solutions

Something of note is that none of the above solving techniques assume a puzzle to be uniquely solvable or even solvable. This is a desirable, since finding a solution will not make that known to be the only correct solution otherwise. Instead, checks able to prove the puzzle has 0 solutions in some scenarios have even been implemented. These include a check at the very beginning that sums up all island's numbers.

Theorem 3. *If the sum of all island's numbers is odd in a puzzle, there are 0 solutions.*

Proof. Suppose the sum of all island's numbers is odd. From the Hashi definition, the number of bridges attached to each island must equal the island's number. Therefore, the sum of all island's numbers is the total number of attachments of bridges to islands. The definition also says bridges are shared by pairs of distinct islands. So any bridge is attached to two islands. Therefore, the sum of all island's numbers is twice the number of bridges needed to satisfy all island's numbers. This means there are 0 solutions, since the sum is not even while it is twice the number of bridges needed for any solution. □

Another check determines for a node if it has more than zero active edges or does not need to share any additional bridges. This check is done immediately whenever the additional bridges an island need or its active edges decreases. This will shortcut the solving process as early as possible.

Theorem 4. *If a puzzle has any node with zero active edges that needs to share nonzero additional bridges to equal its number, there are 0 solutions.*

Proof. Suppose a node exists with zero active edges, but needs to share nonzero additional bridges. Then, there are no other nodes available to share those nonzero bridges with. Therefore, that node will never satisfy its number. This means there are 0 solutions, since from the Hashi definition, the number of bridges attached to each island must equal the island's number. □

Multiple solutions are a different matter. While most solving techniques are not able to find all or any solutions if there exist multiple, backtracking will. In fact, backtracking is a solving method on its own, since it will find all solutions even without any of the other techniques.

Example

An example of a difficult Hashi puzzle taken from previous work on Hashi by Coelho et al.[CLLV19]. It requires the global connectivity constraint and is shown in Figure 2.

Figure 2: Example of a difficult Hashi puzzle by Coelho et al. requiring the global connectivity constraint.

An example on how this can be solved using the solving techniques described in Section 4 is shown in Figure 3.

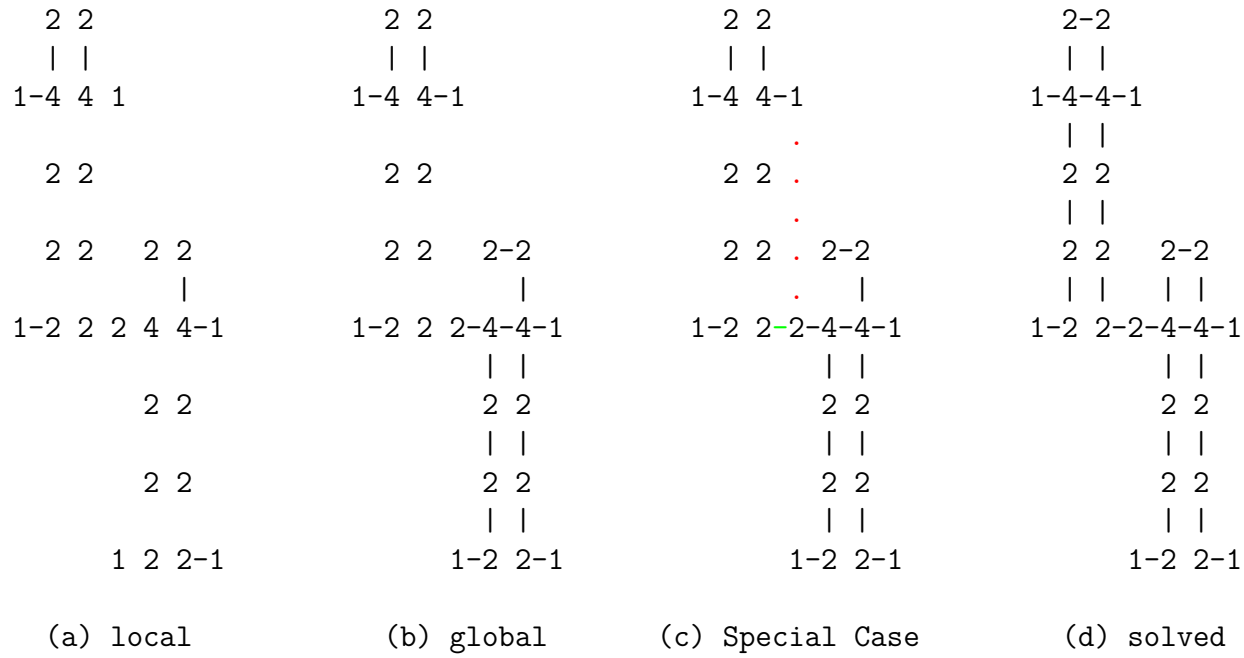


Figure 3: Solving the puzzle using different stages of the solver.

Part (a) shows the puzzle instance including all bridges that can be placed with the local solving techniques and simplifying techniques. Lot's of easy puzzles can be completely solved with local solving techniques and simplifying techniques. Even without the local connectivity constraint. Yet, in this puzzle, it only places 25% of its bridges. In fact, all these are placed using only the local connectivity constraint. Without that 0 bridges can be deduced. All of this shows the difficulty of this puzzle.

Part (b) shows the state global solving techniques could take the puzzle before the Special Case technique was implemented.

Part (c) shows a specific situation that the global solving techniques were not able to solve before. However, after looking at this scenario the Special Case technique was found which could also solve this situation. In Figure 3(c) the red dotted line is the edge that will satisfy the node below and prevent the only other node from sharing a bridge with a node outside the component. That only other node is the 2 on the right of the red dotted line. Instead of the vertical red dotted line, which cannot be included in a valid solution, the horizontal green bridge needs to be placed. This shows how hard it would be to be able to solve all puzzles without backtracking. It seems like lots of similar, but slightly different scenarios will need to be considered. It might very well be possible to deduce how the puzzle can be solved without backtracking for a scenario. Yet, the problem is that the more puzzles a solver can solve, the more specific its added techniques must be to solve even more puzzles. This also makes it hard to show a solver can solve all puzzles. A solve might perform well on hundreds of puzzles, but there might very well still be another puzzle that it will not solve.

Part (d) shows the solved puzzle after applying the solver after the specially added solving technique has done its work in advancing the puzzle.

5 Classifying

Classification of puzzle instances is a way to distinguish puzzles with different properties. Some classifications relate to an estimation of difficulty of the puzzle instance, but not all. For example, instances of generalised Hashi can be grouped by their global maximum number of bridges. Each class then has instances where this number is set to a specific integer. For example, all classic instances belong in the class where the global maximum number of bridges is 2, a rule appearing within the classic definition. There are also classifications related to the size of the puzzle. The number of nodes and number of edges. They might have some correlation with difficulty, but size does not determine difficulty in itself.

There is also the total number of bridges that must be placed. This is one of the most interesting classifications, if combined with the maximum number of bridges times the number of edges. Because this will determine how many bridges need to be placed in how many possible spots for bridges. These parameters will influence the total different bridge configurations without taking the rules into account. This in turn might be closely correlated to the difficulty. This is extra interesting since these parameters are known instantly before attempting to solve the puzzle. More on this in the experiment in Section 7.2. Other parameters suspected of having correlation with difficulty can be accessed after solving. These include the ratios of bridges placed before using global techniques and before using backtracking and also the number of solve steps.

A classifier was made that attaches classification parameters and there values to puzzles before, during and after solving. To easily show these and compare to others, some extra parameters are introduced. These do not have to do with the puzzle itself, but how to present the rest of the data. There is also filter parameter that stores a set of classification parameters which will make only those show up. This is nice to gather certain statistics without having to find where they are

between lots of others. However, this cannot be considered a classification of itself. Then there are parameters to identify puzzles read from files. The folder it was read from, the name of the file and a puzzle number, which is the identifying part of the name.

A list of classification parameters and short explanations is shown in Table 1.

max bridges	Global maximum number of bridges for any islands-pair
number of nodes	How many islands the instance has
number of edges	How many island-pairs can a priori share bridges
edges to nodes ratio	Number of edges divided by number of nodes
total bridges needed	Total number of bridges that needs to be placed in a solution
bridges to nodes ratio	Total bridges divided by number of nodes
bridges to edges ratio	Total bridges divided by number of edges, average number of bridges on edges
number of conflict edge pairs	Number of pairs of edges that cannot have a bridge at the same time
trivially connected	True if there is a unique solution before using global connectivity techniques
ratio bridges placed	Placed bridges divided by total bridges - First: ratio bridges placed before using global techniques - Then: ratio bridges placed before using backtracking - Only if not solved: ratio bridges placed
bridges to be placed	Only if not solved: bridges remaining to be placed
total backtracking steps	How many times a bridge has been placed based on the backtracking heuristic
backtracking depth	Max number of backtracking-placed-bridges the puzzle relies on at a time
solve steps	Number of steps done (nodes inspected) to solve
number of used edges	How many edges have at least 1 bridge
number of unused edges	How many edges have 0 bridges
used edges to edges ratio	Only if solved: number of edges with nonzero bridges divided by number of edges
bridges to used edges ratio	Only if solved: average number of bridges on used edges
solution is tree	True if the solution is a tree and False if it contains a cycle
folder	Only with puzzles read from files: the name of the folder the puzzle comes from
name	Only with puzzles read from files: the name of the puzzle
nr	Only with puzzles read from files: the identifying number of the puzzle

Table 1: These are implemented classifications.

5.1 Separating puzzles in difficulties

The most interesting classification for Hashi puzzles seems to be some not having any additional solution when all islands and bridges do not need to form 1 single connected component anymore. Meaning that even without that rule, all islands and bridges form 1 single connected component automatically after solving. These puzzles thus never need a solving technique that uses that connectivity constraint while solving. In general, this seems to make those puzzles significantly easier to solve. In fact, they might all be solvable with just the Degree of freedom technique and the Simplifying techniques. However, this is not known. Further classification methods relating to difficulty are based on which solving techniques are needed in order for the puzzle to be completely solved. The ratio of bridges that is placed using only a specific set of techniques can also be recorded. How far the puzzle can be progressed without using the most powerful technique used to solve it can thus be considered by using this ratio.

An example of concrete difficulty classes using these ideas are provided in Table 2.

Level -1	Already solved puzzles where the solution requires 0 bridges.
Level 0	Solvable with just placing M bridges shared by all distinct island pairs that may share bridges.
Level 1	Solvable with just the degree of freedom technique and reducing prevented and completed edges, by always finding an island with 0 degrees of freedom until solved. The addition of the technique using the local connectivity constraint not included.
Level 2	Solvable with just the degree of freedom technique and reducing prevented and completed edges. The addition of the technique using the local connectivity constraint not included.
Level 3	Solvable with just the local solving techniques, except the addition of the technique using the local connectivity constraint.
Level 4	Solvable with just the local solving techniques.
Level 5	Solvable with just the local and global solving techniques, except the Special Case idea, and only considering subsets of the isolated nodes set of size 1, and not using the idea that differentiates nodes able to share an additional bridge to a node inside the component from nodes able to share an additional bridge to a node outside the component.
Level 6	Solvable with just the local and global solving techniques, except the Special Case idea, and only considering subsets of the isolated nodes set of size 1.
Level 7	Solvable with just the local and global solving techniques, except the Special Case idea
Level 8	Solvable with just the local and global solving techniques.
Level 9	Solvable with backtracking.

Table 2: An example concrete difficulty classes with required solving techniques and the connectivity constraint as a basis. For this example, it does not matter whether these levels are actually interesting to distinguish.

From the integer level, that belongs to the method that can completely solve the puzzle, the ratio of bridges that could be placed only using the solving method of the previous level can be subtracted from this level to create a continuous difficulty scale instead. For example, a puzzle that is solvable with backtracking, but only 10 out of 40 bridges can be placed using just the local and global solving techniques, will be classified as a level 8.75 puzzle.

5.2 Assigning islands in 4 color classes

In Hashi, in general, not all islands can share bridges with all other islands. Some edge cases do exist in the generalised definition. For classic Hashi, however, something a little more interesting can be said. Islands of a classic Hashi can be put into classes, which will have some property.

Theorem 5. *It is possible to define a way to color any classic Hashi puzzle using at most 4 colors c_0, c_1, c_2, c_3, such that no islands colored c_0 can share bridges with islands colored c_3 and no islands colored c_1 can share bridges with islands colored c_2. And such that any solvable classic Hashi, whose islands are not on one horizontal or vertical line, are colored with at least 3 colors.*

Proof. Because classic **Hashi** is played on a grid, each island is associated with a row and column. Rows can be numbered from left to right and columns from top to bottom. Now we have odd and even rows, odd and even columns. Assign c_0 to any island associated with an odd row and odd column, c_1 to any island associated with an odd row and even columns, c_2 to any island associated with an even row and odd column, c_3 to any island associated with an even row and even column. In classic **Hashi**, bridges can only be shared between islands that are in the same row or same column, since the definition states bridges are oriented horizontally or vertically. Now it is almost trivial that no islands colored c_0 can share bridges with islands colored c_3. These islands cannot be in the same row or column. By construction, c_0 is associated with an odd row and odd column, but c_3 with an even row and even column. No row or column can be odd and even so they are distinct. A similar argument applies for colors c_1 and c_2. Because c_1 is associated with an odd row and even column, which is the opposite from c_2, which is associated with an even row and odd column.

Any solvable classic **Hashi**, whose islands are not on one horizontal or vertical line, must have at least two separate rows and two separate columns. Additionally, one of the rows must have at least two islands, because of solvability. Suppose none of the rows have at least two islands. Then, no horizontal bridges will be possible, so all islands must be connected together with vertical bridges. This means all islands must be on one vertical line. Therefore, none of the rows having at least two islands is impossible. The row with at least two islands will provide the existence of 2 of the 3 colors. The other distinct row will provide the 3rd distinct color. Note that empty rows or columns might need to be removed before coloring to ensure at least two rows and two columns have different parity. □

Example

An example puzzle from a Denksport puzzle book [Den] has been solved and colored like described before.

Figure 4: Denksport puzzlebook example. Coloring with 3 colors and only one red island.

In Figure 4, no purple islands can share a bridge with yelow islands, following the proof from earlier. Since there is only one red island and all islands need to be connected, this red islands will need to

share a bridge with a purple and a yellow island. This is the only way for purple and yellow islands to become part of a single connected component. The red island's number is 2, thus must share exactly one bridge with the yellow 6 and one with the purple 3 below. Sharing a bridge with the purple 1 above will result in no other purple islands becoming connected to the red island. This, because they cannot share a bridge with the purple 1 or red island anymore, since they already have enough bridges attached, and cannot share bridges with yellow islands. Thus, no solution can be reached. Furthermore, because the red islands shares a bridge with only one purple and one yellow island, all purple islands must form a single connected component if the bridge shared with the red island is left out. The same for the yellow islands. This essentially splits the puzzle in two different parts, except for the fact they still have conflict edges with each other. Intuitively it can be seen as an hourglass where the red island is the small hole in the middle where all sand must go through to reach below. All these observations can be quickly made with a trained eye, from just the almost trivial coloring. This is a nice way of showing how interesting, but also useful it can be.

6 Generating

There exist numerous ways to generate puzzles. The main differences between them is what kind of puzzles they are able to generate. For example, following the classic Hashi definition, a generator could simply generate a rectangular grid of random size where each cell contains a random number ranging from 1 through 8 or nothing. This generator would thus be able to generate any instance of classic Hashi. However, many of these will not be solvable. And even if they are, they might have more than 1 solution. An ideal generator would provide only puzzles that have exactly 1 solution, but also be able to provide ány puzzle that has exactly 1 solution. This ideal, however, is very hard to achieve for some puzzle genres, which is why an appropriate balance needs to be found.

First, a solvable puzzle generator will be introduced. It can in theory generate every possible solvable generalised puzzle. However, it will not consistently create uniquely solvable puzzles. Unless, in addition, it checks by solving, throws away puzzles with multiple solutions and tries again. After this generator, some ideas on smarter generators will be discussed. Finally a method is proposed to turn puzzles with multiple solutions into uniquely solvable puzzles. This should make the previous generator able to generate all and only uniquely solvable Hashi puzzles. Though it might not be very efficient, since it does require one or more iterations of solving while generating.

6.1 Solvable puzzle generator

For the solvable puzzle generator there is no focus on making puzzles uniquely solvable. It will just create a puzzle knowing it will be solvable. It does so by creating a solution first, including where all bridges should be placed. Then assigning numbers to islands appropriately and removing all bridges afterwards. There are some parameters that can be chosen. If not chosen, they are taken from random distributions. The parameters are

- M: Max number of bridges for any edge
- Number of islands
- Number of used edges
- Number of bridges in total
- Number of unused edges
- Number of conflict pairs

All parameters are non-negative integers. In addition, some of these relate to each other and need to be in certain bounds:

- When islands with nonzero number exist, bridges must be allowed to exist at all:

$$M > 0$$

- There should be enough used edges to connect all islands and not more used edges then a used edge between every node pair:

$$islands - 1 <= used\_edges <= islands * (islands - 1)/2$$

- There should be enough bridges to use all used edges and not more then the maximum number of bridges on all used edges:

$$used\_edges <= bridges <= used\_edges * M$$

- The maximum conflict pairs would every unused edge with every unused edge plus every unused edge with every used edge:

$$conflict\_pairs <= unused\_edges * (unused\_edges - 1)/2 + unused\_edges * used\_edges$$

The generator works as follows. If there are zero or one islands, the trivial puzzle with that number of islands is returned. Otherwise, starting from one island, islands will be added along with an edge shared by an existing island. These edges will start out with one bridge. The existing island to share the edge with is randomly determined, but weighted on how many attached edges islands already have. The number of islands added will be the same as the corresponding parameter since this is easily tunable and some there needs to be some number of islands. Since this procedure will create a connected tree, it will automatically be a solution provided that the island's numbers match. However, not all puzzles are supposed to have a tree solution. Therefore, more edges will be added between existing islands, starting out with one bridge. However, not all puzzles are supposed to have only edges with exactly one bridge. First, edges with multiple bridges will be created by selecting edges that do not have the maximum number of bridges and increasing the number of bridges. This is done until the desired number of bridges is reached. Second, unused edges are added between nodes that did not have a used edge yet. These will have zero bridges on them. After that, conflict pairs will be selected. One of the edges can be any edge. The other is always an unused edge, to not interfere in the possibility of the guaranteed solution. Now these selected edges will never be able to both contain more than zero bridges. Lastly, appropriate numbers are assigned to each node and all bridges are removed from all edges.

Interesting to note is that every solvable puzzle can be generated by this generator. Firstly, the parameters are all random across their entire domain, even when they can be arbitrarily high. So for any valid solvable puzzle, its M, the number of islands, used edges, bridges, unused edges and conflict pairs can appear as the selected parameters by the generator. This generator perfectly respects these parameters. Though another solution with different used and unused edges might exist. Secondly, all steps, like adding islands, edges and assigning bridges are random without restrictions. Such that for any valid solvable puzzle one could select some order of the islands and imagine how the generator could have added them in that order, then added edges until all edges of the puzzle were added and so on.

6.2 Uniquely solvable generators

Smarter generators need to provide a uniquely solvable puzzle without the need to check for exactly 1 solution after generating. They need to inherently create only uniquely solvable puzzles. This will remove the need to discard generated puzzles with more solutions. There are at least three clearly different ways to approach this. The first is to use a solvable generator like before, but instead of simply checking for uniqueness, perform some additional transforming steps on the puzzle after which it is confirmed to be uniquely solvable. The second is to use a method to make a bigger uniquely solvable puzzle from a smaller uniquely solvable puzzle, thus building up the puzzle being uniquely solvable along the way, stopping when it has the desired size. The third is to keep track of what part of the puzzle's solution is already set in stone based on how far the puzzle has been created. As such, this generator can keep going while never introducing something that makes the puzzle have no solution and keep going until the puzzle has one solution.

These all have their advantages and disadvantages. They are noted down in Table 3.

Method	Advantages	Disadvantages
1	If a solvable generator is already available, there is less work left to do. This will be able to generate all uniquely solvable puzzles, if it could generate all solvable puzzles.	The transforming steps must form a finite algorithm that always stops, preferably always resulting in a desired output. this can be challenging or inefficient. A solver might be needed to make sure the transforming steps succeeded, or even before every next step. The bigger the size of the puzzle, the more transformation might be needed.
2	This procedure can be stopped at any time resulting in a valid puzzle. Has potential to prove the resulting puzzle has a unique solution theoretically, bypassing any need of a solver during or after generating.	Finding ways to make a bigger uniquely solvable puzzle from a smaller one takes a very deep understanding of what makes these kinds of puzzles uniquely solvable. It is one thing to find a way to do it, but many different ways to add to a puzzle while keeping it uniquely solvable are need to not have a very limited amount of variation in puzzles. This method will very likely not be able to generate all uniquely solvable puzzles, because of the method of construction and because not all uniquely solvable puzzles need to contain a uniquely solvable sub-puzzle.
3	This seems to have potential to be made more efficient then the other two methods.	This might not be able to generate all uniquely solvable puzzles, because of the method of construction. This requires an absolute solver, that will know immediately when adding something contradicts the solution as build up so far. It might at some point not be feasible to add more to the puzzle to reduce the solutions to 1 solution, if everything that can be added is added.

Table 3: Advantages and disadvantages of different ways to try to make uniquely solvable puzzles.

The first method was chosen to create a uniquely solvable generator. Two different transformations have been used after one another. They are described in the next subsections.

6.2.1 Reducing solutions with conflict edges

One way to see a puzzle with multiple solution can be that it does not have enough constraints. Luckily, in generalised Hashi there is a useful constraint the puzzle maker can add at will: conflict edges. These are edges that cannot both have a bridge. Those bridges would cross and that it not allowed. Adding a conflict edge eliminates any solution using both those bridges. If edges are chosen smartly, it is sometimes possible to remove some solutions, while keeping at least one solution valid.

Firstly, this does require solutions to be found. Not all possible solutions need to be found, but at least two. These are found by calling the solver. Which finds up to all solutions, because of the use of backtracking and saving any found solution. Once a number of solutions is found they are each represented as a list of 0's and 1's. The order is a constant order the edges of the puzzle have. The 0's mean 0 bridges and the 1's mean 1 or more bridges. Now two edges are found such that among all solutions as many as possible have a 1 in the list for that edge, but at least one has a 0 for one of the edges. The edges also should not be a conflict edge pair already. Those two edges will be made a new conflict edge pair. All solutions that had a 1 in the list for both edges disappear. At least one solution will remain, since there was at least a 0. This is because in that solution only one of the edges in the conflict edge pairs is used, which is allowed. This whole process is repeated until only one solutions remains or no suitable edges can be found anymore. The example of this section will include performing this transformation.

There exist cases where it is impossible to transform the puzzle to be uniquely solvable by adding more conflict edges. This is because the technique with adding conflict edges relies on making it impossible for two edges to both contain nonzero bridges. However, because of that, it can only distinguish between zero and nonzero bridges. Suppose a Hashi puzzle has multiple solutions, but all of them use the same edges. That is, if an edge has 0 bridges in one solution, it has 0 bridges in all solutions. The same for nonzero bridges. In this scenario it does not matter what conflict edge pair is added, it will always leave all possibilities or cut out all of them. This is because the same edges must contain bridges in each solution.

This makes it interesting to find more solutions before performing a transformation iteration. After all, the lower the number of solutions searched for, the higher the chances that all those solutions share the same used edges and that this transformations cannot be applied immediately. In that happens more solutions can be searched. Though it must be possible to continue searching from where the solver happened to stop or some solutions need to be found again. Finding more solutions also allows for selecting edges that having a higher chance of eliminating more solutions. This is because the number of solutions that had a 1 in the list for both edges can be higher if the number of found solutions is higher. And these solutions will disappear for sure. There is simply more information to select on. However, each iteration will take longer. And it is not guaranteed that this time will be compensated by more solutions disappearing per iteration. Lots of unknown solutions might also be eliminated after seeing only two, which in that case never have to be found. Clearly, a balance should be found if one wishes to make the generator more time efficient.

While this is a technique designed specifically for generalised Hashi, the idea can be applied with the classic structure intact as well. However, it will be more limited. In generalised Hashi one can assign any two edges to conflict with each other. In Classic Hashi, conflict edges are directly

dependent on spatial locations of islands. Therefore, spatial locations must be changed to introduce conflict edges.

6.2.2 Reducing solutions by lowering numbers

Luckily, there is still another technique to reduce the puzzle to have exactly one solution. This involves lowering the number of two islands that share different numbers of bridges different solutions. More specifically, find an edge that has a bridges in some solution and b bridges in another for some preferably low $0 < a < b \leq M$. For classic Hashi this means $a = 1$ and $b = 2$. These bridges are shared by two islands. How much to lower the two islands numbers then depends on whether or not they are both part of the same cycle of nodes, each node sharing at least one bridge with the next. If they are, lower their numbers by the number of bridges from the solution with the higher number of bridges, so b. If they are not, lower the two islands number's by a. This might sound arbitrary, but these have a clear reason.

After transforming, a new valid solution is guaranteed to exist in both cases. For the cycle case, this is the previous solution with b bridges shared by the two islands, but with b less bridges. This will indeed be a valid solution. Firstly, the two islands whose numbers were lowered by b also have b less bridges attached. Secondly, all numbers of bridges attached to other islands remain equal to the numbers of those islands. Thirdly, all islands and bridges that were connected are still connected in this new solution. For the no cycle case, the new valid solution is also the previous solution with b bridges shared by the two islands, but now with a less bridges. This will be a valid solution for the same reasons. However, nodes still being connected is not because some nodes formed a cycle, but because lowering the b bridges by a still leaves $b - a > 0$ bridges. Therefore, the two nodes remain connected. After transforming, the number of valid solutions should be less in both cases. The other previous solution, with a bridges shared by the two islands, cannot be changed in the same way to get a solution for the transformed puzzle. In the cycle case, simply because there are not enough bridges shared by the two islands to take away. In the non cycle case, because taking away a bridges takes away all bridges shared by those nodes, thus introduces another separate connected component. Because of this, for both two islands, $b - a$ bridges shared with another island need to be taken away. This results in that island needing $b - a$ additional bridges. However, that is either not possible or compensating islands in a chain will result in the same solution as the guaranteed solution. Therefore, lowering numbers like this should reduce the number of solutions even further. This process seems repeatable until all cycles causing multiple solutions are no longer cycles in the transformed puzzle solution. Also, until all edges not part of a cycle have only one valid number of bridges for a solution left. Therefore, with all of the above, a uniquely solvable puzzle generator is realised.

Note that this generator does not completely respect the parameters anymore. More specifically it does not respect the number of used and unused edges because the solution with these might be reduced and another solution can use a different number of edges. It does not respect the number of bridges because it lowers numbers on islands. It can also add additional conflict pairs. However, all uniquely solvable puzzles can still be generated. Since all uniquely solvable puzzles are a subset of all solvable puzzles. All solvable puzzles can be generated by the solvable puzzle generator from Section 6.1. And this generator first generates a puzzle with that one and only modifies it if it is not already uniquely solvable.

Example

To illustrate the effect of the solution reducing techniques, a small puzzle is provided in Figure 5.

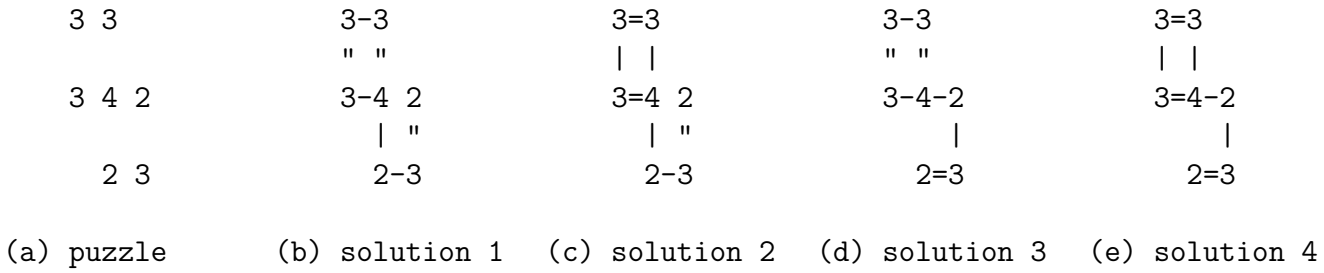


Figure 5: A small puzzle and its 4 solutions. How to transform it to have 1 solution?.

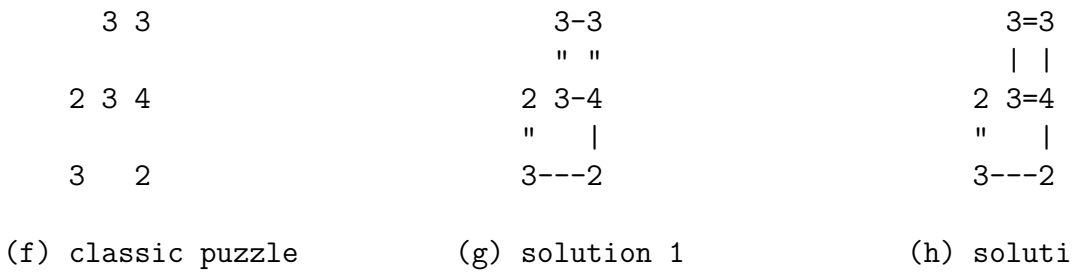
By reading edges left to right, top to bottom, this puzzles 4 solutions can be represented as

$$(1, 2, 2, 1, 0, 1, 2, 1)$$
$$(2, 1, 1, 2, 0, 1, 2, 1)$$
$$(1, 2, 2, 1, 1, 0, 1, 2)$$
$$(2, 1, 1, 2, 1, 0, 1, 2).$$

Though for adding the conflict edges any 2 might as well be a 1, like

$$(1, 1, 1, 1, 0, 1, 1, 1)$$
$$(1, 1, 1, 1, 0, 1, 1, 1)$$
$$(1, 1, 1, 1, 1, 0, 1, 1)$$
$$(1, 1, 1, 1, 1, 0, 1, 1).$$

Now it can easily be seen why adding conflict edges alone will not result in a uniquely solvable puzzle in this example. Regardless, half the solutions can be eliminated. Let's add a conflict edge pair for the 4th and 5th edge. Since the 4th edge has 1 bridge in all solutions and the 5th edge has a 0 in the first half of the solutions. This means the two horizontal edges in the middle of the puzzle can no longer both contain a bridge. Therefore, solution 3 and 4 in Figure 5 are no longer valid. In this example, it is not hard to apply a similar transformation that also preserves this puzzle as a classic Hashi puzzle. Instead of taking the possibility of a bridge on the 5th edge away by adding a conflict edge pair, the 2 and 3 on the right can be flipped to the left hand side of the puzzle to achieve the same. The new puzzle is and its solutions are shown in Figure 6.



(f) classic puzzle (g) solution 1 (h) solution 2

Figure 6: A small transformed puzzle and its 2 solutions. How to transform it to have 1 solution?.

To avoid confusion, let's continue with a transformation that is identical performed on the original puzzle with added conflict edge pair or performed on the transformed classic puzzle. As such, select the 1st edge. It has 1 bridge in solution 1 and 2 bridges in solution 2. Thereby selecting the two 3's at the top of the puzzle. Since they are part of a cycle, their number will be lowered by $b = 2$. This will give the puzzles and their one solution as shown in Figure 7.

(i) puzzle (j) solution (k) classic puzzle (l) solution

Figure 7: Two small transformed puzzles and their only solution. Subfigure (i) shows a conflict edge pair in red, and (j) shows the resulting edge where no bridge is allowed in red.

7 Experiments

For the experiments, three solving methods will be compared. The combined solver described in Section 4 and the solving methods from Wikipedia and the paper by Malik et al., discussed in Subsection 4.1.1. Furthermore, puzzles from two different puzzle database websites are used. The Janko website [Jana] and the Puzzle-bridges website [Has]. Naturally, these are classic Hashi puzzles only. The websites include categorisation of puzzles by size. From the Janko website, puzzles from the category 9x9 are used. From the Puzzle-bridges website, puzzles from the categories 10x10easy and 7x7dense are used. Dense likely referring to more islands fitting on a smaller grid. Apart from these puzzles, a variety of other puzzles is used. In the order they will appear in Table 7 and Table 11 these are the following. From the Janko website, three of the few puzzles not marked as puzzles with replayable logical solutions. From the Puzzle-bridges website, a puzzle from both the 25x25easy and 15x15dense categories. Two puzzles from a physical Denkspot puzzle book [Den] Five puzzles from the websites that were used as early testing puzzles or shared by others. The puzzle from Coelho et al. [CLLV19] that was shown in Figure 2.

It is important to note that within Hashi puzzle instances there is an enormous number of factors in play. Every puzzle is different and might have specific characteristics that have an effect on results. It is possible to focus on puzzles with roughly the same size or 'difficulty', measured by the websites. However, it is not possible to guarantee puzzles will behave roughly similarly, even if they are from the same size or difficulty category. Because of this, it is very hard to conclude results come from specific parameters or characteristics. For example, some puzzles can be solved without ever using the connectivity constraint. This can have huge consequences on other factors that are experimented on. If the experiments should focus on only 1 factor and keep the rest roughly the same, these would need to be split out from other puzzles. However, it is not feasible to identify all of such factors and characteristics. Nor is it feasible to try to eliminate all of these while testing only a specific factor. This seems to be something in the nature of this kind of puzzle collection.

7.1 Solver versus existing human techniques

To get an idea about solving strength for solvers, the ratio of bridges that can be placed using the different solvers is calculated. This ratio will be 1 if all required bridges were indeed placed. The total number of bridges needed to solve the entire puzzle is also shown. For the thesis solver, two ratios are given. The left column is the ratio using only local techniques. The right column is the ratio when global techniques were used, but not backtracking. That column represents the solving strength of the smart part of the thesis solver. Backtracking always gets the ratio to 1, since the backtracking part of this solver solves all Hashi puzzles. This is done by repeatedly guessing from the point where the other techniques got stuck. The number of steps the thesis solver uses to solve the entire puzzle, including backtracking, is also shown. The bottom row shows the mean values. This is done for different ranges of puzzles to avoid a narrow view. Therefore, four groups of results are shown in Table 4, Table 5, Table 6, and Table 7. The last one consists of a wide variety of puzzles.

Janko 9x9 puzzles		Wikipedia solver	Malik et al. solver	Thesis solver		
nr	total bridges	bridges ratio all their techniques		bridges ratio LT	bridges ratio LT+GT	solve steps LT+GT+B
1	34	0.324	0.088	1	1	43
10	30	0.067	0.033	0.6	1	102
14	27	0.259	0.0	0.852	1	80
16	27	0.222	0.0	1	1	40
2	33	0.485	0.03	1	1	44
361	40	0.4	0.325	1	1	50
4	32	0.25	0.0	0.594	1	157
5	39	0.308	0.205	1	1	49
8	30	0.467	0.4	1	1	45
9	39	0.256	0.0	1	1	68
mean	33.1	0.304	0.108	0.905	1.0	67.8

Table 4: Solving strength for 9x9 puzzles by Janko. Measured by ratios of bridges different solvers can deduce. Separate columns exist for using local techniques (LT), global techniques (GT) and backtracking (B). After backtracking, the ratio is always 1. The total number of bridges and number of solve steps for the thesis solver are included.

10x10easy puzzles		Wikipedia solver	Malik et al. solver	Thesis solver		
nr	total bridges	bridges ratio All their techniques		bridges ratio LT	bridges ratio LT+GT	solve steps LT+GT+B
1916045	25	0.28	0.2	1	1	32
2647497	23	0.565	0.261	1	1	26
3334419	25	0.76	0.76	1	1	23
3345352	19	0.789	0.842	1	1	16
3589471	25	0.64	0.56	1	1	24
498530	27	0.889	0.778	1	1	18
6620332	30	0.767	0.367	1	1	24
731665	23	0.696	0.391	1	1	25
9143614	28	0.679	0.429	1	1	27
9561556	26	0.769	0.5	1	1	17
mean	25.1	0.683	0.509	1.0	1.0	23.2

Table 5: Solving strength for 10x10easy puzzles by Puzzle-bridges. Measured by ratios of bridges different solvers can deduce. Separate columns exist for using local techniques (LT), global techniques (GT) and backtracking (B). After backtracking, the ratio is always 1. The total number of bridges and number of solve steps for the thesis solver are included.

7x7dense puzzles		Wikipedia solver	Malik et al. solver	Thesis solver		
nr	total bridges	bridges ratio All their techniques		bridges ratio LT	bridges ratio LT+GT	solve steps LT+GT+B
1605088	18	0.222	0.056	1	1	19
1830710	19	0.316	0.316	1	1	21
274144	19	0.316	0.105	1	1	28
27953	20	0.4	0.25	0.65	1	58
3060643	19	0.316	0.053	1	1	20
4734744	18	0.222	0.0	1	1	28
5456883	17	0.118	0.118	1	1	26
5516729	21	0.143	0.095	0.571	1	84
703627	23	0.522	0.261	1	1	34
907518	18	0.278	0.111	1	1	22
mean	19.2	0.285	0.136	0.922	1.0	34.0

Table 6: Solving strength for 7x7dense puzzles by Puzzle-bridges. Measured by ratios of bridges different solvers can deduce. Separate columns exist for using local techniques (LT), global techniques (GT) and backtracking (B). After backtracking, the ratio is always 1. The total number of bridges and number of solve steps for the thesis solver are included.

Various puzzles		Wikipedia solver	Malik et al. solver	Thesis solver		
nr	total bridges	bridges ratio all their techniques		bridges ratio LT	bridges ratio LT+GT	solve steps LT+GT+B
352	45	0.044	0	0.156	0.178	587
524	49	0.041	0.265	0.245	0.367	446
886	176	0.17	0.125	0.938	1	845
5000159	106	0.66	0	1	1	127
4550114	96	0.302	0.135	1	1	132
1	49	0.469	0.082	1	1	59
2	52	0.654	0.346	1	1	53
bigger	58	0.672	0.517	1	1	70
biggest	317	0.363	0.151	0.994	1	735
coelho	28	0.0	0.071	0.25	0.571***	2583
example	34	0.529	0.294	1	1	49
X	124	0.121	0.008	0.379	0.96	1569
Y	120	0.458	0.192	0.95	1	216

Table 7: Solving strength for various puzzles. Measured by ratios of bridges different solvers can deduce. Separate columns exist for using local techniques (LT), global techniques (GT) and backtracking (B). After backtracking, the ratio is always 1. The total number of bridges and number of solve steps for the thesis solver are included.

These results make it quite clear that more bridges can be placed when using the solver this thesis proposes compared to the solver ideas by the two external sources. It can be seen that neither the

Wikipedia or Malik et al. solver always outperforms the other. This is expected, since part of their techniques are closely related, but they both contain ideas in techniques the other does not take into account.

Looking at the solve steps column, it seems all puzzles are solved in reasonable number of iterations of looking at a node.

It seems like when the thesis solver needs to rely on the global techniques or even backtracking, the bridges ratio for the external solvers is also lower. The solve steps are higher. Together suggesting that those puzzles are inherently more difficult, regardless of what kind of solver is used. This experiment also vouches for the thought that which solving techniques are needed to solve puzzles and how far other solving techniques come is a useful measure of difficulty of a puzzle.

Some other observations can be made from this data that were not necessarily part of the original experiment intention. When puzzles need the global techniques and also when they need backtracking, it can take many more solve steps. Puzzles that are solvable by local techniques only use fewer solve steps. There seems to be a correlation between the total bridges needing to be placed and the solve steps. Note that this can also be a correlation between the number of nodes and solve steps and that the total bridges needed correlates to the number of nodes for classic Hashi. Either way, this is expected, since the more bridges need to be placed or more nodes there are, the more often one would expect to look at a node to try and place bridges. This needs to be done until all bridges are placed and all nodes are satisfied after all.

7.2 Mean number of bridges per edge

For this experiment only the thesis solver is used. The purpose of this experiment is to find a classification that is known ahead of solving and has a relation with puzzle difficulty, solving techniques needed, or solving steps, if any exist. The most promising contender seemed the mean number of bridges per edge. This is the total number of bridges divided by the number of edges. The reasoning for choosing this contender is as follows. For classic Hashi, the maximum number of bridges per edge is always 2. Therefore, the total number of bridges that could fit on a puzzle with $|E|$ edges is $2|E|$. Therefore, the closer the total bridges in the solution is to $2|E|$ or 0, the lower the number of possible configurations of bridge placements. Meaning the closer the mean number of bridges per edge is to 0 or 2, thus farther from 1, the lower the number of configurations. Note that if this experiment had differing maximum number of bridges per edge, the mean number of bridges per edge could be divided by it to get values between 0 and 1 instead.

The idea here is that having less configurations of which one is the solution will make it easier to decide where the bridges should be in the solution. Thus, needing lower solve steps. Something else that might be of influence is the number of conflict edge pairs. Any bridge placement configuration that used edges that are in conflict cannot be a solution, still ignoring the numbers in the islands. Therefore, more conflict edge pairs might mean the puzzle can be solved in fewer solve steps. Of course, this is only an idea. The results of this experiment are shown in Table 8, Table 9, Table 10 and Table 11. The tables show the total bridges in the solution, the number of edges, the mean number of bridges per edge and the number of conflict edge pairs. These are known before solving. Furthermore, the tables again show the ratio of deduced bridges for different parts of the thesis solver and number of solve steps. These are recorded during and after solving.

Janko 9x9 puzzles					bridges ratio		steps
nr	total bridges	edges	mean b/e	conflicts	LT	LT+GT	LT+GT+B
1	34	34	1.0	21	1	1	40
10	30	32	0.938	6	0.6	1	118
14	27	29	0.931	14	0.852	1	67
16	27	26	1.038	6	1	1	39
2	33	32	1.031	8	1	1	36
361	40	40	1.0	16	1	1	44
4	32	32	1.0	11	0.594	1	122
5	39	39	1.0	15	1	1	50
8	30	28	1.071	10	1	1	44
9	39	38	1.026	10	1	1	72
mean	33.1	33.0	1.004	11.7	0.905	1.0	63.2

Table 8: Comparison of classifications before and after solving for 9x9 puzzles by Janko. Before solving, the mean number of bridges per edge (total bridges divided by number of edges) and the number of conflict edge pairs are known. After solving, the ratio of deduced bridges and number of solve steps are recorded. Separate columns exist for using local techniques (LT), global techniques (GT), and backtracking (B). The mean bridges per edge for Janko 9x9 puzzles seems suspiciously close to 1.0 for these puzzles. This could have been intended by the puzzle makers.

10x10easy puzzles					bridges ratio		steps
nr	total bridges	edges	mean b/e	conflicts	LT	LT+GT	LT+GT+B
1916045	25	22	1.136	3	1	1	41
2647497	23	19	1.211	0	1	1	29
3334419	25	19	1.316	1	1	1	17
3345352	19	15	1.267	3	1	1	14
3589471	25	19	1.316	2	1	1	24
498530	27	17	1.588	3	1	1	19
6620332	30	23	1.304	5	1	1	31
731665	23	18	1.278	4	1	1	23
9143614	28	22	1.273	2	1	1	37
9561556	26	18	1.444	1	1	1	22
mean	25.1	19.2	1.313	2.4	1.0	1.0	25.7

Table 9: Comparison of classifications before and after solving for 10x10easy puzzles by Puzzlebridges. Before solving, the mean number of bridges per edge (total bridges divided by number of edges) and the number of conflict edge pairs are known. After solving, the ratio of deduced bridges and number of solve steps are recorded. Separate columns exist for using local techniques (LT), global techniques (GT), and backtracking (B).

7x7dense puzzles					bridges ratio		steps
nr	total bridges	edges	mean b/e	conflicts	LT	LT+GT	LT+GT+B
1605088	18	20	0.9	4	1	1	21
1830710	19	21	0.905	6	1	1	23
274144	19	20	0.95	6	1	1	27
27953	20	21	0.952	6	0.65	1	40
3060643	19	20	0.95	7	1	1	22
4734744	18	21	0.857	8	1	1	27
5456883	17	20	0.85	5	1	1	24
5516729	21	20	1.05	3	0.571	1	79
703627	23	21	1.095	6	1	1	25
907518	18	20	0.9	7	1	1	24
mean	19.2	20.4	0.941	5.8	0.922	1.0	31.2

Table 10: Comparison of classifications before and after solving for 7x7dense puzzles by Puzzlebridges. Before solving, the mean number of bridges per edge (total bridges divided by number of edges) and the number of conflict edge pairs are known. After solving, the ratio of deduced bridges and number of solve steps are recorded. Separate columns exist for using local techniques (LT), global techniques (GT), and backtracking (B).

Various puzzles					bridges ratio		steps
nr	total bridges	edges	mean b/e	conflicts	LT	LT+GT	LT+GT+B
352	45	54	0.833	18	0.156	0.178	393
524	49	58	0.845	24	0.245	0.367	725
886	176	204	0.863	123	0.938	1	1056
5000159	106	89	1.191	61	1	1	133
4550114	96	105	0.914	56	1	1	147
1	49	46	1.065	15	1	1	72
2	52	48	1.083	25	1	1	51
bigger	58	51	1.137	19	1	1	64
biggest	317	334	0.949	407	0.994	1	664
coelho	28	39	0.718	1	0.25	0.643	334
example	34	32	1.062	9	1	1	41
X	124	148	0.838	93	0.379	0.96	1588
Y	120	110	1.091	62	0.95	1	209

Table 11: Comparison of classifications before and after solving for a variety of puzzles. Before solving, the mean number of bridges per edge (total bridges divided by number of edges) and the number of conflict edge pairs are known. After solving, the ratio of deduced bridges and number of solve steps are recorded. Separate columns exist for using local techniques (LT), global techniques (GT), and backtracking (B).

Looking at all the data, there seems to be no direct correlation between the ratio of the total bridges of the solution divided by the number of edges with the solve steps or bridges ratio. This

does not mean that this ratio does not have any influence on the difficulty. Because of the many characteristics of the puzzles at play, the influence of this ratio might as well be overshadowed by other influences. This does mean it sadly should not be used as a way of gauging how difficult a puzzle is before solving. Thus, no way of classifying the difficulty of a puzzle with only data from before solving the puzzle was found with this experiment.

Another interesting observation is about the number of conflict edge pairs for each puzzle category. Conflict edge pairs are interesting since they say something about the spatial locations of islands in relation to other islands for classic Hashi. It seems different categories were made in such a way that the category will have similar numbers of conflict edge pairs, but different compared to other categories. Initially there was an idea that the number of conflict edge pairs would be higher for 10x10easy puzzles compared to 7x7dense puzzles. This was because the 10x10easy puzzles leave much more space in between islands, while 7x7puzzles have islands closely packed together. In classic Hashi, this means the number of conflicting edges for a 'longer' edge can in theory be higher compared to edges between islands closely packed together. However, the data shows that this idea turns out to be the other way around. The 7x7puzzles have a higher number of conflict edge pairs instead. Apparently, the 10x10puzzles are not made in such a way that utilises the space between islands for possible perpendicular bridges. Upon investigating some individual 10x10easy puzzles, it seems the extra space is instead used in a way that islands avoid each other more, thus creating less possibilities where bridges could be crossing each other. This makes sense for puzzles labelled as easy, since giving islands more space in this way might reduce the number of possibilities for islands to divide their bridges to other islands.

8 Conclusions and Further Research

Hashi is a very interesting type of puzzle. It involves ideas most other puzzles do not have. Such as placing connections instead of symbols. This thesis has shown that the potential of the core idea of this puzzle genre is not limited to the classic puzzle definition, but can be extended by the provided generalisation. While solving puzzles without backtracking can come a long way, it seems backtracking will still be required, if only as a way to be able to prove a solver can indeed solve any uniquely solvable puzzle. The most interesting classification for Hashi puzzles seems to be some not having any additional solution when all islands and bridges do not need to form 1 single connected component anymore. Meaning that even without that rule, all islands and bridges form 1 single connected component automatically after solving. Different kinds of level or scale systems based on required solving techniques and the progress without them can be used to classify difficulty of puzzles. While generating puzzles it is important to consider what kind of puzzles can be generated. There are different approaches of generating puzzles, each having their own advantages and disadvantages. It is indeed possible to create a generator that can generate any uniquely solvable puzzle and only generates puzzles that are uniquely solvable.

There are still numerous things to consider for future research on this puzzle or its introduced generalised variant. The generalisation could use some more work. Especially in visualising the generalised puzzles. Methods for visualising a graph are already there, but visualising the conflict edges might be able to be done more naturally. Perhaps something with colors.

Building upon the generalised variant, research could also be done into methods that solve puz-

zles where all islands must form k connected components for $k \neq 1$. Only the local connectivity technique and global techniques will need to be changed. Note that once k components are known, these techniques can still be used on individual components. In fact, they can be used on any group of nodes known to belong to one closed component as soon as it is formed. When $k-1$ closed components are known, the rest of the puzzle may be treated as an individual component as well. A possible simplifying technique will be to reduce k by 1 each time a closed component is split from the rest of the puzzle. The split of component will become a puzzle with $k = 1$. There could also be techniques that prevent 2 closed components not including all islands from forming simultaneously when $k = 2$ for some part split off the puzzle.

Another angle would be to go further into puzzles that do not have a unique solution. Possible ways to find out a puzzle has no valid solution have been discussed. However, for finding multiple solution, only backtracking has a chance so far. Techniques could be devised dedicated specifically into finding multiple solutions without using backtracking. For example, there might be techniques able to identify parts of the puzzle that can be solved in two or more ways regardless of what happens around that part. Another technique could split a puzzle in two and choose to play a different number of bridges somewhere in each copy, when it is likely that a solution exists for both choices of bridges.

Building on the introduced classification, it can be compared to classifications of different Hashi puzzle providers, looking for similarities and differences with their difficulty gradings. Here, puzzle providers might be websites, but also physical puzzle books. It can also be researched from the perspective of their classifications, trying to find grounds that classification could be based on. Or how likely it is they used human solving time. Furthermore, research can be done into trying to generate puzzles that will fall into a desired classification. For some parameters like the number of islands this can be easier than parameters specifying some kind of difficulty.

Since the proposed uniquely solvable puzzle generator is unable to generate a puzzle known to have a classic representation, future research can continue on applying similar transformations that work well for classic Hashi. The generator also uses a parameter to decide how many solutions it should try to find before doing a transformation. As explained, a balance will need to be found so future research could perform experiments with the generator and tune this parameter. More research can also be done into other ways of generating only uniquely solvable puzzles. The proposed different kinds of methods and their advantages and disadvantages could be used as a basis. As well as the described qualities the most desirable generator should have. Research can be done into what makes a Hashi puzzle have a unique solution. Also, in checking this property quicker then solving the entire puzzle.

Finally, research could be done in creating more variants of Hashi and into solving, classifying and generating those new puzzles. For example, an existing variation of Hashi that could be looked into is Hashi^2 [Janb]. In this variation, islands have letters instead of numbers. Each letter represents an unknown number that has to be found out in the middle of solving the puzzle. Other possible variations include introducing a second way to connect islands, like rivers, that can cross bridges, but not other rivers. These could use up the same number in islands or get a separate number. Another one would be allowing bridges to be shared by more than two distinct islands. Then, one bridge might be seen as a platform connecting 5 islands together. Many more even crazy variants could be created and some might actually work well.

References

[And09] Daniel Andersson. *Hashiwokakero is NP-Complete*. PhD thesis, Aarhus University, 2009. pages 101–103. https://cs.au.dk/fileadmin/site_files/cs/PhD/PhD_Dissertations__pdf/koda_thesis.pdf#page=111.

[CLLV19] Leandro C. Coelho, Gilbert Laporte, Arinei Lindbeck, and Thibaut Vidal. Benchmark instances and branch-and-cut algorithm for the hashiwokakero puzzle. *arXiv*, 2019. https://arxiv.org/abs/1905.00973.

[Den] Denksport. Logisch 2-4* vakantieboek. Denksport, Retrieved August 19, 2025. https://www.denksport.com/puzzel/logisch/logisch-2-4-vakantieboek.

[Has] Hashi. Retrieved July 22, 2025. https://www.puzzle-bridges.com/.

[Jana] Otto Janko. Hashiwokakero. Janko, Retrieved July 22, 2025. https://www.janko.at/Raetsel/Hashi/index.htm.

[Janb] Otto Janko. Hashiwokakero². Janko, Retrieved June 30, 2025. https://www.janko.at/Raetsel/Hashi-2/index.htm.

[Jol23] Casper Jol. How to solve and generate marupeke puzzles. Bachelor's thesis, LIACS, Leiden University, 2023. https://theses.liacs.nl/pdf/2022-2023-JolC.pdf.

[MEP12] 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*, 1(1):61–68, 2012. https://media.neliti.com/media/publications/58361-EN-solving-hashiwokakero-puzzle-game-with-h.pdf.

[MEWA90] Roger D. Walker Mark E. Wilson and William A. Anderson. Using logic puzzles for critical thinking. *NACTA Journal*, 34:50–52, 1990. https://www.jstor.org/stable/43766595.

[Mor09] Timo Morsink. Hashiwokakero. Bachelor's thesis, LIACS, Leiden University, 2009. https://liacs.leidenuniv.nl/assets/Bachelorscripties/2009-11TimoMorsink.pdf.

[Mou22] Rob Mourits. Solving and generating the nurimeizu puzzle. Bachelor's thesis, LIACS, Leiden University, 2022. https://theses.liacs.nl/pdf/2021-2022-MouritsM.pdf.

[Olwnd] Shaimaa Olwan. How puzzles and brain teasers make kids smarter, n.d. Learning mole, Retrieved February 21, 2025. https://learningmole.com/how-puzzles-and-brain-teasers-make-kids-smarter/.

[Str25] Hanna Straathof. Solving and generating kuroshuto puzzles. Bachelor's thesis, LIACS, Leiden University, 2025. https://theses.liacs.nl/pdf/2024-2025-StraathofHHanna.pdf.

[vdK21] Gerhard van der Knijf. Solving and generating puzzles with a connectivity constraint. Bachelor's thesis, Radboud University, 2021. https:

//www.cs.ru.nl/bachelors-theses/2021/Gerhard_van_der_Knijff___1006946_
__Solving_and_generating_puzzles_with_a_connectivity_constraint.pdf.

[Wik25] Wikipedia. Hashiwokakero, 2025. Wikipedia, Retrieved July 16, 2025. https://en.
wikipedia.org/wiki/Hashiwokakero.