Algorithm for Solving Hashiwokakero Puzzle

James French
CS 315 Honors -- Algorithms
Department of Computer Science
University of Kentucky
james.dylan.french@gmail.com

ABSTRACT

In this paper, we describe an algorithm to solve the puzzle game Hashi. Determining if a Hashi puzzle is solvable or not is an NP-Complete problem, thus this program focuses on puzzles which are already known to be solvable. Though very abstract by nature of being a puzzle, the analysis used could apply to the connection of network nodes, etc. given certain restraints

CCS Concepts

• **Theory of computation**→Design and analysis of algorithms→Graph algorithms analysis

Keywords

Hashi; puzzle; graph problem; graph algorithm; NP-Complete

1. INTRODUCTION

Hashiwokakero is a puzzle game where your objective is to connect every node on the board with every other node, i.e. you can follow a path from any starting node to any ending node once the puzzle is complete. Each node is given a value denoted by a number, and each node must have that number of bridges extending from it, in a cardinal direction, to another node, with a maximum of two bridges in each direction. Bridges cannot intersect.

The puzzle is solvable through a series of logical decisions based on the number of neighbor nodes, the value of the nodes, and other factors.

2. Techniques for Solving

Techniques for solving a Hashi puzzle can be split into 3 main categories

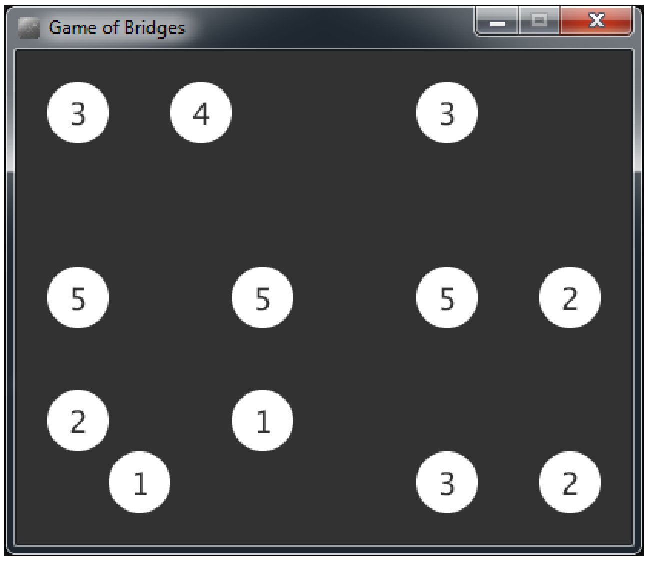
2.1 Forced Connections

These are situations where a connection must be made in a particular, and generally speaking, obvious manner.

Examples would include a node with only a single neighbor, wherein one or two bridges will be made depending on the value of the isolated node; a node where the number of possible remaining moves matches exactly the remaining number of bridges that needs to be connected to the node, wherein all remaining possible bridges would be made.

2.2 Singular Assumptions

There are numerous instances where one or more individual bridges can be build, without knowing how to completely fill the node.



A typical, albeit simple, Hashi puzzle in an uncompleted state

Examples include a 3 in the corner of the board, where you can infer that at least one bridge must extend to each of the neighboring nodes, since two bridges to one node will not reach the 3 bridge requirement. This can be expanded for a 5 on the side of the board with 3 neighbors where fully connecting two neighbors is not sufficient, and a 7 in the middle of the board with four neighbors surrounding it.

This can also be generalized to a situation where a node with one remaining bridge is one of two neighbors to another node that requires two bridges. Connecting to just the node with a remaining value of 1 will not be sufficient, but you cannot say for sure that the connection will be made or not- you can, however, build one bridge to the other neighbor of a higher value, as it is necessary to fulfill the original node's value requirement.

2.3 Island Avoidance

Perhaps the most robust technique, yet the hardest to model in an algorithmic sense, is the analysis of a new bridge creating an island, or a segmented group of nodes whose values have all been fulfilled, but do not connect with all the nodes on the board.

Analyzing this requires the Hashi player to think ahead regarding what would happen *if* a move was made. If an island were to be created by a move, it should be avoided, and if only one other move remains for the given node, a connection made there instead.

One also has to consider multiple moves at a time- if a 2 were sandwiched between two 1's and another node, both the 1's cannot be connected with a bridge, and the player knows at least one bridge must be connected to the other node with a non-1 value.

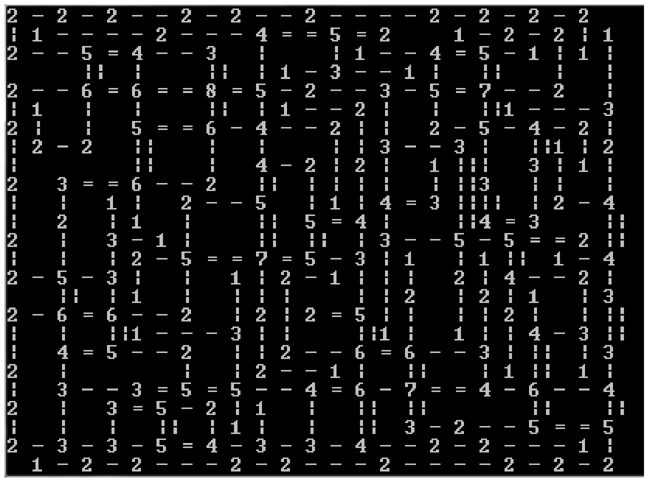
3. Details of Algorithm Representation

3.1 The Board and connections

The board itself is input as a series of numbers or periods indicating nodes with associated values or empty space. This is stored in a standard 2-dimensional vector of integers, with periods being interpreted as having a value of 0.

The network of nodes is stored as a tree, with any nodes which intersect in a cardinal direction being stored as "neighbor" pointers of each other. A series of bools, integers, and other variables are used to keep track of a node's associated bridge count, which directions are full, which potential moves are valid, etc.

From here, whenever a connection is made between two nodes, the space between them is scanned, and replaced with values ranging from -1 to -4, with -1 and -2 corresponding to single and double vertical bridges, and -3 and -4 corresponding to single and double horizontal bridges. Storing everything as an integer allows for easier numerical manipulation, while the distinction between bridge orientation helps to recognize bridge overlap and more easily print the completed board.



```
2 - 2 - 2 - - 2 - 2 - - 2 - - - - 2 - 2 - 2 - 2
¦ 1 - - - - 2 - - - 4 = = 5 = 2     1 - 2 - 2 ¦ 1
2 - - 5 = 4 - - 3   ¦     ¦ 1 - - 4 = 5 - 1 ¦ 1 ¦
      ¦¦  ¦     ¦¦  ¦ 1 - 3 - - 1 ¦   ¦¦    ¦   ¦
2 - - 6 = 6 = = 8 = 5 - 2 - - 3 - 5 = 7 - - 2   ¦
¦ 1   ¦   ¦     ¦¦  ¦ 1 - - 2 ¦   ¦   ¦¦1 - - - 3
2 ¦   ¦   5 = = 6 - 4 - - 2 ¦ ¦   2 - 5 - 4 - 2 ¦
¦ 2 - 2   ¦¦    ¦   ¦     ¦ ¦ 3 - - 3 ¦   ¦¦1 ¦ 2
¦         ¦¦    ¦   4 - 2 ¦ 2 ¦   1 ¦¦¦   3 ¦ 1 ¦
2   3 = = 6 - - 2   ¦¦  ¦ ¦ ¦ ¦   ¦ ¦¦3   ¦ ¦   ¦
¦   ¦   1 ¦   2 - - 5   ¦ 1 ¦ 4 = 3 ¦¦¦¦  ¦ 2 - 4
¦   2   ¦ 1   ¦     ¦¦  5 = 4 ¦     ¦¦4 = 3     ¦¦
2   ¦   3 - 1 ¦     ¦¦  ¦¦  ¦ 3 - - 5 - 5 = = 2 ¦¦
¦   ¦   ¦ 2 - 5 = = 7 = 5 - 3 ¦ 1   ¦ 1 ¦¦  1 - 4
2 - 5 - 3 ¦   ¦   1 ¦ 2 - 1 ¦ ¦ ¦   2 ¦ 4 - - 2 ¦
    ¦¦  ¦ 1   ¦   ¦ ¦ ¦     ¦ ¦ 2   ¦ 2 ¦ 1   ¦ 3
2 - 6 = 6 - - 2   ¦ 2 ¦ 2 = 5 ¦ ¦   ¦ ¦ 2 ¦   ¦ ¦¦
¦   ¦   ¦¦1 - - - 3 ¦ ¦     ¦¦1 ¦   1 ¦ ¦ 4 - 3 ¦¦
¦   4 = 5 - - 2   ¦ ¦ 2 - - 6 = 6 - - 3 ¦ ¦¦  ¦ 3
2   ¦         ¦   ¦ 2 - - 1 ¦   ¦¦    ¦ 1 ¦¦  1 ¦
¦   3 - - 3 = 5 = 5 - - 4 = 6 - 7 = = 4 - 6 - - 4
2   ¦   3 = 5 - 2 ¦ 1   ¦   ¦¦  ¦¦        ¦¦    ¦¦
¦   ¦   ¦   ¦¦  ¦ 1 ¦   ¦   ¦¦  3 - 2 - - 5 = = 5
2 - 3 - 3 - 5 = 4 - 3 - 3 - 4 - - 2 - 2 - - - 1 ¦
  1 - 2 - 2 - - - 2 - 2 - - - 2 - - - - 2 - 2 - 2
```

When output, these numbers are interpreted as standard ASCII text using the - , = symbols for horizontal bridges and | , || for vertical.

4. Hashi Solving Techniques Applied Algorithmically

To solve the puzzle, the nodes are iterated over in sequence, sweeping across the board, until a valid move can no longer be found. If a puzzle potentially has multiple valid solutions, the algorithm will find the first one and end. It also operates on the assumption that the puzzle is valid and thus solvable, though will simply fail to connect all the nodes if this is not the case.

4.1 Forced Connections

Of course the simplest of methods to implement algorithmically, the program simply checks the number of possible moves and compares the result to the nodes value, making as many connections as necessary.

4.2 Singular Assumptions

This technique is more difficult to generalize and requires a wider variety of state checks to ensure all potential situations are discovered. This includes seeing if a node's remaining value is 1 less than the number of remaining possible moves, meaning each neighbor must have at least one bridge connecting it; seeing if a node has two valid directions for moves and if one of them fails to fully satisfy its value condition, meaning the neighbor with the greater value has at least one bridge connecting it; and, in this particular algorithm's case, a hard-coded table checking for the latter condition as generalized for more than two neighbors, which proved a difficult scenario to generally model.

4.3 Island Avoidance

Arguably the most difficult of the primary techniques to model, avoiding islands in the algorithm is approached by fully adding a potential bridge, checking to see if an island was made, and then removing the bridge.

An island is checked for by performing a depth-first search on the tree of nodes, visiting each neighbor node and seeing if any of them still have a remaining valid move that can be performed. If they do, the operation is stopped and the move is considered safe, though not necessarily correct.

This is not the most robust solution for two reasons, though it is also generalized to two bridges being built in a single direction (can a 2 have two bridges connecting to another 2?).

The technique used only eliminates moves that are definitely invalid, instead relying on a combination of the other techniques to solve the puzzle. The "safe" moves cannot be deemed as correct unless they become the only remaining valid move for the node.

Decisions cannot cascade throughout the given section of the tree. If one bridge were to make a "chain" of bridges that moved through multiple nodes and finally at the end forced an island to be created, this method would not detect it. While it will not erroneously create the island, it's entirely possible that some Hashi puzzles rely on this technique of multi-stage forward thinking, which would be difficult to model in an algorithm.

5. Added Value

The structure of a Hashi board somewhat resembles the wider distribution of the internet through large-scale fiber-optic cabling, i.e. not a LAN. Certain hub areas receive much higher traffic volumes, much like the building at 60 Hudson Street in New York, while outliers like island nations, perhaps most notably Australia, will typically have a single large bundle of fiber optic lines running underwater to connect back to mainland countries.

Modeling this large-scale network as a puzzle game makes little sense, but some of the same principles, and thus analysis techniques, still apply- every node must be connected to each other, those with a larger value (bandwidth) have a greater number of connections passing through them, and you wouldn't ever waste cable by crossing them, unless it was at a hub of some sort (another node).

As for the algorithm itself, from what I can gather the Hashi puzzle is not popular enough to have gathered a large volume of attempts at an algorithmic approach. The one paper referenced utilized a much different technique to my own that took an approach much closer to a breadth-first search, by developing a large number of potential connections and then either proving their impossibility or checking for errors after the fact, whereas my approach was very piecemeal and non-aggressive by comparison, only looking at moves guaranteed to work and using a limited approach regarding future prediction.

6. REFERENCES

[1] Shi-Jim Ten, Shih-Yuan Chiu, Cheng-Wei Chou, Tsan-Cheng Su. Hashi Solver. DOI= http://www1.rdoffice.ndhu.edu.tw/exchange/abroad/abroad99/31_paper.pdf