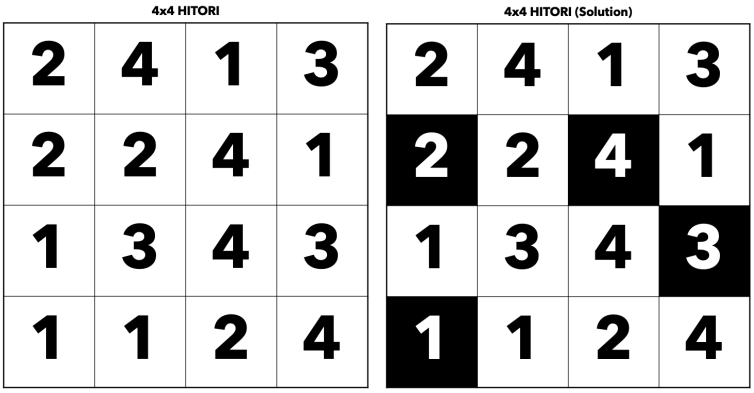
Hitori Project Proposal

Peter Yao (pby2101) Ava Hajratwala (ash2261)

Introduction to Hitori

Hitori is a single-player logic puzzle game that is NP-complete. The initial setup of the puzzle is a grid of numbers, where the player has to shade in specific cells to satisfy three rules: (1) No row or column may contain the same unshaded number more than once (2) shaded cells cannot lie adjacent to other shaded cells, although diagonals are allowed, and (3) all unshaded cells in the solved puzzles must be orthogonal to one another (there can be no unshaded "islands"). Here is an example of an unsolved puzzle and the solved solution[1]:



(a) Starting Board State (b) Solved Board State

Figure 1: Hitori Boards

Known Implementations in Haskell

The known algorithms for solving this puzzle involve a brute force backtracking algorithm, similar to many Sudoku implementations. Some pruning can help speed up the performance so that depth-first-search does not have to explore so many nodes. For each node, the algorithm has to perform steps to check that all the rules from above are satisfied. More importantly, we will also need a function that verifies rule (3) is satisfied. We can implement this check using the "number of islands" problem with early termination.

There are only a few existing implementations of Hitori solvers written in Haskell. Specifically, we plan to adapt code from this library as a starting point for our code.[2] We can make the code slightly more efficient by adding additional logic for tree pruning using existing game-solving strategies, such as those mentioned on this website.[3]

Improvements with Parallelization

Because Hitori has arbitrarily large board sizes, we can scale the size of this problem without simply having the program solve thousands of boards. Additionally, two tree traversals are involved, with the larger tree containing all the board states and the smaller tree containing shaded and unshaded cells. We see two areas where parallelization could help speed up the implementation.

[1] Source: *https://baileyspuzzles.com/how-to-play-hitori/*
[2] Github: *https://github.com/nrpeterson/fxhitori/tree/master*
[3] Conceptis Puzzles: *https://www.conceptispuzzles.com/index.aspx?uri=puzzle/hitori/techniques*

Possible Approaches to this Project

One approach to this project is to follow Simon Marlow's general approach in his Sudoku solver. Because this puzzle uses a backtracking DFS algorithm similar to the Sudoku solver, we can apply his techniques to our implementation.

An alternative approach is a variation of puzzle generation. Most puzzle databases that we can find contain puzzles of up to 25x25 cells. Although puzzle generation can be done trivially (shade cells that don't create islands and number each cell as a different number), it could be an interesting challenge to determine, given a grid and some valid shading of cells, what is the smallest range of numbers necessary to create a solvable puzzle. Here, our algorithm would be more similar to a k-coloring graph problem. However, we are still looking for any existing implementations of this.

Concerns

Like sudoku, solvable Hitori boards on the internet seem to have a "maximum complexity" because there isn't a massive database for large boards. Additionally, some Hitori solvers claim to be able to solve most 25x25 boards in under a second.[4] So far, we have only been able to find boards of this size or smaller, meaning that if we want to work with larger boards, we'll need to generate solvable boards programmatically, which could get complicated. If we only have small boards, we have concerns about overhead from parallelism on such a small timescale. However, we won't need large puzzles if we choose to go the Marlowe route (multiple puzzles at once).

[4] Source: *https://hitori-solver.appspot.com/*