Name: Linh Bui

Ltb2128

Project Proposal: Solving the 2048 Game Using Haskell

(2048-solver)

1. Introduction

The 2048 game is a single-player puzzle game where the player combines tiles on a 4x4 grid with the goal of achieving a tile with the value 2048 (at least). It provides a challenge for exploring algorithmic approaches and functional programming techniques, particularly using Haskell. The generalized version of the 2048 game has been proven to be NP-complete, as shown in the research paper by Abdelkader, Acharya, and Dasler (2015). The gameplay of 2048 can be represented as a series of transformations and state updates, tasks that Haskell's immutable data structures and recursive functions can efficiently handle.

This project aims to build a Haskell-based solver for the 2048 game. The solver will use a combination of search algorithms, heuristic evaluations, and functional programming principles to find optimal moves.

2. Problem Statement

The 2048 game has a vast state space due to the possible configurations of the tiles on the board. Designing a solver involves:

- Efficiently representing the game state.
- Generating possible moves and valid tile transitions.
- Evaluating board states to decide the most promising moves.

The project will explore techniques to model these challenges functionally using Haskell, leveraging its type system, immutability, multithreading and recursion.

3. Objectives

The primary objectives of this project are:

1. To design and implement a functional game-solving algorithm using Haskell's functional programming constructs.
    a. The algorithm should consistently achieve a tile of at least 2048 in every game, ensuring the solver can reliably meet this minimum benchmark across different game scenarios.
2. To enhance the solver's performance by integrating parallel processing and multithreading techniques, utilizing Haskell's capabilities for efficient computation and faster decision-making.
3. To compare the solver's effectiveness with existing approaches and identify the strengths and weaknesses of using Haskell

3. Algorithm & Parallelism

We would use expectimax for this project along with parallelism. Expectimax differs from Minimax in that it considers the expected value of random events (e.g., adding a new tile). We can define two types of nodes:

- **Max Nodes**: Represent the player’s moves.
- **Chance Nodes**: Represent the random tile (2 or 4) that is added to the board.

In the 2048 game, there are two types of actions:

- **Player Moves**: The player can swipe left, right, up, or down.
- **Random Tile Generation**: After each player’s move, a new tile (2 or 4) is randomly added to an empty space on the board.

Expectimax Algorithm:

- **Max Node** (Player's Turn):
  - Evaluate each possible move (left, right, up, down).
  - Recursively compute the expected score for each move by examining the resulting board states.
  - Select the move with the highest expected score.
- **Chance Node** (Random Event):
  - Generate all possible board states resulting from adding a new tile (2 or 4).
  - Calculate the expected score as the **weighted average** of all possible board states, considering the probability of each tile (e.g., 90% chance of 2, 10% chance of 4).

Parallelism - we plan to utilize parallelism as followed:

parMap rpar: Applies the Expectimax function in parallel to each possible move or random tile placement.

Max Nodes: For the player’s turn, the algorithm evaluates all possible moves concurrently and selects the move with the highest score.

Chance Nodes: For random tile generation, the algorithm evaluates each possible outcome in parallel and computes the expected score.

The heuristic that would be tested are:

- The more open squares there are, the better
- Having larger value on edges is better
- Having non-monotonic rows and columns is better
- The number of potential merges (adjacent equal values) in addition to open spaces (possible)

4. Expected Outcomes

We expect to achieve:

- 100% of achieving 2048 at least every run (possibly scaling to 8192)
- Multithreaded usage should be able to speed up the program by at least 4 times the sequential solver (with the current machine)
- Insights and analysis into the effectiveness of functional programming for game-solving tasks.

8. Conclusion

This project offers an opportunity to explore the application of functional programming to a popular puzzle game. Through this project, we aim to show the potential of Haskell in solving complex algorithmic problems and contribute to the broader exploration of functional approaches in AI.

Citation:

Abdelkader, A., Acharya, A., & Dasler, P. (2015). *2048 is NP-complete*. Retrieved from https://www.cs.umd.edu/~akader/files/CGYRF15_2048.pdf.

https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048