

Evaluation: Hashi-Solver (Group M)

Noel Huibers, Laura Schröder, Florian Würmseer

22.11.2023

1 Introduction

In the context of Hashi puzzles, also known as Bridges, we aim to evaluate the scalability of our encoding method as grid sizes increase. This assessment will measure the performance of our solver and explore ways to make the encoding more efficient. Our objective is to ensure that our solver can efficiently handle larger grids while maintaining accuracy. The following sections will present our analysis of the solver's performance using Criterion and the steps we've taken to optimize its efficiency.

2 Dataset

The benchmarks and measurements were designed to evaluate the solver's performance across a range of puzzles with varying grid sizes and complexities. The following test files were employed, each characterized by a specific grid size and layout of the puzzle:

- **test1.txt** - Featuring a grid size of 5x5. It includes a straightforward puzzle layout with minimal islands and bridges, ideal for assessing the solver's efficiency in handling simpler, less demanding puzzles.
- **test8.txt** - This file presents an intermediate level of complexity with a 9x9 grid. It introduces a more challenging puzzle layout, including a greater number of islands and more intricate bridge connections.
- **test20.txt** - Featuring a grid size of 15x15. A puzzle layout with a dense arrangement of islands and bridges. It tests the solver's capability to efficiently process and solve large-scale, intricate puzzles that demand significant computational power and advanced problem-solving strategies.

Each test file progressively increases in complexity and grid size, providing a comprehensive spectrum for evaluating the solver's performance. This diverse dataset aims to examine the solver's scalability, efficiency, and robustness in handling puzzles ranging from simple to highly complex. The goal is to identify strengths, limitations, and potential areas for optimization in the solver's design and implementation.

3 Performance Measurement: Solver Modes

A comparative analysis of the `encode_mode`, `solve_mode`, and `esr_mode` functions was conducted. The benchmark was performed multiple times to ensure reliability of the results.

Test Case	Average Execution Time (ns)	Performance Change
encode/test1	448.07	+0.26% to +0.50%
encode/test8	454.19	-0.36% to +1.72%
encode/test20	451.30	-0.43% to +1.16%
solve/test1	458.95	-3.69% to -0.85%
solve/test8	461.68	-1.41% to +0.02%
solve/test20	462.67	-0.85% to +0.47%
esr/test1	462.34	-1.69% to +0.08%
esr/test8	458.21	-0.61% to +0.51%
esr/test20	450.55	-0.16% to +0.96%

Table 1: Benchmark Results

The benchmark results provide insights into the performance of various backend functions under different test conditions. The following observations can be made:

- **Consistency in Execution Time:** The average execution times for the `encode` and `esr` functions across different test files are relatively consistent, indicating stable performance across different input sizes.
- **Performance Changes:** Notable performance changes are observed in the `solve` function, especially for `test1`, where the execution time shows a significant decrease. This might suggest optimizations or efficiencies in the algorithm that become more apparent with certain types of input.
- **Impact of Test File Size:** Larger test files (e.g., `test20`) do not show a significant increase in execution time, which is indicative of efficient algorithm scalability with larger input sizes.
- **Outliers:** The presence of outliers in several test cases, particularly in `test8` and `test20` for the `esr` function, could indicate potential inconsistencies in performance under certain conditions. These outliers might warrant further investigation to understand their cause.
- **Overall Stability:** The majority of the performance changes are within a small margin, suggesting overall stability in the function implementations. However, the variance in the `encode` function for `test8` and `esr` function for `test8` and `test20` could be areas to explore for potential optimizations.

In conclusion, the benchmark results show a generally stable and efficient performance across the tested functions, with some areas of potential optimization and a need for further investigation into the causes of outliers.

4 Performance Measurement: Helper-Functions

Function	Test1 (μ s)	Test8 (μ s)	Test20 (μ s)
parse_input	862.78	615.68	1843.81
generate	268.67	875.07	1596.31
generate_dimacs	113.71	148.44	241.44
solve	16.26	12.22	15.49
write_solution	97.07	76.26	75.93
reconstruct_puzzle	81.50	168.97	323.24

Table 2: Average Execution Times of Functions

These results, obtained through multiple executions, provide valuable insights into the performance characteristics of different functions within our solver. The key observations from the analysis are as follows:

- The `parse_input` function shows a notable increase in execution time with the increasing complexity and size of the test cases, indicating its sensitivity to the input size.
- In contrast, the `solve` function maintains relatively consistent and low execution times across all test cases, suggesting its efficiency is less affected by the complexity of the puzzle.
- The `generate` and `generate_dimacs` functions exhibit a moderate increase in execution times with larger puzzles, which aligns with their roles in puzzle generation and DIMACS format conversion.
- The `write_solution` and `reconstruct_puzzle` functions demonstrate a balanced performance across different test cases, although a slight increase in time is noticeable in more complex puzzles for `reconstruct_puzzle`.

These findings highlight the areas where optimization efforts could be focused, particularly in the `parse_input` and `generate` functions for larger puzzles. Additionally, the overall efficiency and scalability of the `solve` function under varying conditions are noteworthy. In conclusion, the benchmarking exercise has successfully identified performance trends and potential bottlenecks, setting the stage for targeted improvements in the solver's performance.

5 Optimization Opportunities

Based on the previous results, potential areas for performance optimization are identified. Recommendations for improving the efficiency of the functions are provided.

TODO benches and tests after refactoring + performance improvements