Hashiwokakero

Group 2

Ngày 4 tháng 8 năm 2025

1 Thông tin nhóm

Bảng 1: BẢNG THÔNG TIN THÀNH VIÊN NHÓM

Tên	Task	Complete
Trịnh Đức Thiên	A*, Brute Force, Backtracking	100%
Mai Gia Chung	Pysat, Report	100%
Võ Trung Tín	Testcase, Video, Benchmark	100%

2 Algorithm

2.1 Pysat

- **Hàm khởi tạo `__init__`**:
  - Nhận vào một đối tượng `board` biểu diễn trạng thái ban đầu của bài toán.
  - Tạo một đối tượng `CNFGenerator` dùng để sinh các mệnh đề logic từ `board`.
  - Thiết lập thời gian tối đa cho việc giải là 10 giây.
- **Hàm `solve()`**:
  1. Gọi phương thức `generate_cnf()` từ `CNFGenerator` để tạo danh sách mệnh đề (clauses) và các biến (variables) logic đại diện cho hành động (như việc đặt cầu).
  2. Duyệt qua các mệnh đề và thêm chúng vào một đối tượng `CNF` để chuẩn bị truyền cho SAT solver.
  3. Sử dụng bộ giải `Glucose4` để giải hệ mệnh đề:

- Nếu tìm được nghiệm (`SAT`), ánh xạ các biến logic sang giá trị `True/False`.
- Với mỗi biến có giá trị `True`, xây dựng lại hành động tương ứng trên `solved_board` (ví dụ: thêm cầu giữa hai đảo).

4. Nếu vượt quá thời gian giới hạn, dừng việc giải và trả về `None`.
5. Nếu có lỗi trong quá trình xử lý, in ra thông báo lỗi và trả về `None`.

Luồng hoạt động

- **Biến đổi bài toán thành logic mệnh đề**: Thông qua lớp `CNFGenerator`, bài toán được chuyển sang dạng mệnh đề logic chuẩn CNF.
- **Giải SAT**: Dùng thư viện PySAT và bộ giải Glucose4 để tìm nghiệm thỏa mãn tất cả các ràng buộc.
- **Diễn giải nghiệm SAT**: Các biến logic được ánh xạ ngược trở lại hành động cụ thể trên bài toán ban đầu.

2.2 A*

Tổng quan

Thuật toán **A*** là một thuật toán tìm kiếm tối ưu thường được sử dụng để tìm đường đi tốt nhất trong không gian trạng thái. Trong bối cảnh bài toán Hashiwokakero thuật toán A* được triển khai thông qua lớp `AStarSolver`.

Mục tiêu của giải thuật là tìm một trạng thái bảng (`board`) thoả mãn tất cả ràng buộc: số cầu từng đảo phải đúng, không có cầu cắt nhau, và toàn bộ đảo phải được nối liên thông.

Luồng hoạt động

Giải thuật A* hoạt động theo các bước chính sau:

1. Khởi tạo trạng thái đầu tiên từ bảng.
2. Đưa trạng thái vào **hàng đợi ưu tiên** với trọng số là hàm đánh giá heuristic.
3. Trong mỗi vòng lặp, lấy ra trạng thái tốt nhất, kiểm tra xem đã là lời giải chưa.
4. Nếu chưa, sinh ra các trạng thái kế tiếp bằng cách thử nối cầu hợp lệ, đánh giá heuristic và tiếp tục đưa vào hàng đợi.
5. Dừng lại khi tìm được lời giải hoặc sau một số bước giới hạn để tránh vòng lặp vô hạn.

Cài đặt

1. Khởi tạo (`__init__`) Lưu trữ bảng ban đầu, khởi tạo các tập:

- `open_set`: hàng đợi ưu tiên chứa các trạng thái cần xét.
- `closed_set`: tập hợp các trạng thái đã xét để tránh lặp.
- `counter`, `steps`: đếm số trạng thái và số bước thực hiện.

2. Phương thức `solve()` thực hiện vòng lặp chính của thuật toán:

- Bắt đầu từ trạng thái gốc, tính giá trị heuristic và đưa vào `open_set`.
- Trong mỗi bước, lấy trạng thái tốt nhất, kiểm tra xem đã hoàn thành chưa.
- Nếu chưa, tạo ra các trạng thái kế tiếp bằng cách thử thêm một hoặc hai cầu giữa các cặp đảo hợp lệ.
- Với mỗi trạng thái mới, tính giá trị heuristic và đưa vào hàng đợi nếu chưa từng xét.
- Dừng thuật toán nếu vượt quá số bước tối đa hoặc tìm được lời giải.

3. Hàm đánh giá heuristic (`_heuristic`) Hàm này đánh giá mức độ "tốt"của một trạng thái dựa trên các yếu tố sau:

- **Connection Penalty**: phạt 20 điểm cho mỗi cầu còn thiếu của một đảo.
- **Overconnection Penalty**: phạt 40 điểm cho mỗi cầu thừa của một đảo.
- **Isolation Penalty**: phạt 30 điểm nếu một đảo chưa có kết nối nào.
- **Connectivity Penalty**: phạt 60 điểm cho mỗi thành phần liên thông không nối với phần còn lại.

Tổng điểm heuristic là tổng các loại penalty trên. Trạng thái hoàn hảo sẽ có điểm số bằng 0.

4. Đếm thành phần liên thông (`_count_components`) Sử dụng thuật toán DFS (Depth-First Search) để đếm số thành phần liên thông trong đồ thị cầu. Mỗi thành phần tách biệt làm tăng penalty, giúp heuristic hướng thuật toán tới các trạng thái có tính liên thông cao hơn.

5. Sinh trạng thái kế tiếp (`_generate_successors`) Tạo các trạng thái mới bằng cách thử nối một hoặc hai cầu giữa các cặp đảo còn có thể nối. Cầu được thêm theo hướng ngang hoặc dọc tùy vị trí.

6. Hàm hỗ trợ khác

- `_board_hash()`: sinh chuỗi định danh duy nhất cho mỗi trạng thái để sử dụng trong `closed_set`.
- `_copy_board()`: tạo bản sao của bảng game để tạo trạng thái mới mà không ảnh hưởng trạng thái cũ.

2.3 Backtracking

Tổng quan

Lớp `BacktrackingSolver` triển khai giải thuật quay lui nhằm tìm lời giải cho trò chơi Hashiwokakero như sau:

- Mỗi đảo có đúng số cầu yêu cầu.
- Các cầu không cắt nhau.
- Tất cả các đảo phải được nối thành một mạng liên thông.

Thuật toán hoạt động bằng cách thử các cách đặt cầu hợp lệ giữa các đảo theo từng bước và đệ quy kiểm tra xem có thể dẫn tới lời giải hay không. Nếu không, thuật toán sẽ quay lui và thử hướng khác.

Cài đặt

1. Khởi tạo (`__init__`) Lưu trữ bảng ban đầu và tạo một biến `solution` để giữ lời giải khi tìm thấy:

- `self.board`: bảng trò chơi ban đầu.
- `self.solution`: mặc định là `None`, sẽ lưu trạng thái lời giải cuối cùng nếu có.

2. Phương thức giải `solve()` Gọi phương thức quay lui bắt đầu từ trạng thái ban đầu:

- `_backtrack(self.board)`: đệ quy tìm lời giải.
- Nếu tìm thấy lời giải, lưu vào `self.solution` và trả về.

3. Phương thức quay lui `_backtrack()`

1. **Điều kiện dừng**: Nếu trạng thái hiện tại đã là lời giải hoàn chỉnh (kiểm tra bằng `is_solved()`), thì lưu lại và kết thúc chương trình tìm kiếm.
2. **Lựa chọn đảo**: Tìm một đảo chưa có đủ số cầu cần thiết.
3. **Thử kết nối cầu**:

- Duyệt qua tất cả các đảo còn lại để xem có thể kết nối cầu hay không.
- Thử thêm cầu đơn và cầu đôi nếu hợp lệ (theo chiều ngang hoặc dọc).
- Mỗi lần thử sẽ sao chép bảng hiện tại để tránh ảnh hưởng trạng thái gốc.

4. **Đệ quy**: Gọi lại `_backtrack` với bảng mới. Nếu lời giải được tìm thấy, trả về `True`.
5. **Quay lui**: Nếu không có lựa chọn nào dẫn tới lời giải, quay lui về bước trước đó và thử hướng khác.
6. **Tối ưu hóa**: Chỉ xử lý một đảo chưa hoàn thành tại mỗi bước (dùng `break`) nhằm giảm không gian tìm kiếm.

4. Hàm sao chép bảng `_copy_board()` Hàm này tạo bản sao của bảng hiện tại, bao gồm:

- Sao chép `grid` để giữ nguyên trạng thái ô.
- Sao chép danh sách `bridges` để đảm bảo độc lập với các bước thử khác.

Luồng hoạt động

1. Lựa chọn đảo Tìm một đảo chưa đủ cầu để bắt đầu thử các hướng mở rộng, từ đó thu hẹp không gian tìm kiếm và tăng cơ hội tìm ra hướng giải quyết.

2. Kết nối Với mỗi cặp đảo có thể nối, chương trình sẽ thử:

- Thêm một cầu đơn (số lượng = 1).
- Nếu có thể, tiếp tục thêm cầu đôi (số lượng = 2).

Sau mỗi lần thêm cầu, thuật toán đệ quy tiếp tục thử trên trạng thái mới.

3. Đệ quy Khi không thể tiếp tục hoặc không dẫn tới lời giải hợp lệ, chương trình quay trở lại bước trước đó để thử hướng khác

4. Tối ưu hóa Chỉ xử lý một đảo chưa hoàn thành tại một thời điểm (`break` sau vòng lặp), giúp giới hạn chiều sâu đệ quy và tăng tốc độ xử lý với các bảng có nhiều đảo.

2.4 Brute Force

`BruteForceSolver` thử tất cả các tổ hợp cầu nối có thể. Giải thuật hoạt động dựa trên ý tưởng duyệt qua toàn bộ không gian tìm kiếm các tổ hợp cầu giữa các đảo.

Với ý tưởng trên, ta được nguyên lý hoạt động của thuật toán như sau:

- Sinh ra danh sách các cặp đảo có thể kết nối với nhau, tức là các cặp đảo nằm trên cùng một hàng hoặc một cột (theo đúng quy tắc của trò chơi Hashiwokakero).
- Với mỗi cặp đảo, thuật toán xét ba trường hợp có thể xảy ra: không nối cầu, nối một cầu, hoặc nối hai cầu
- Duyệt toàn bộ các tổ hợp số lượng cầu nối giữa các cặp đảo nói trên bằng cách sử dụng hàm `product` từ thư viện itertools, tạo ra toàn bộ không gian tìm kiếm có kích thước 3^n với n là số lượng cặp đảo tiềm năng.
- Trả về bảng đầu tiên thỏa mãn điều kiện hoàn thành bài toán.

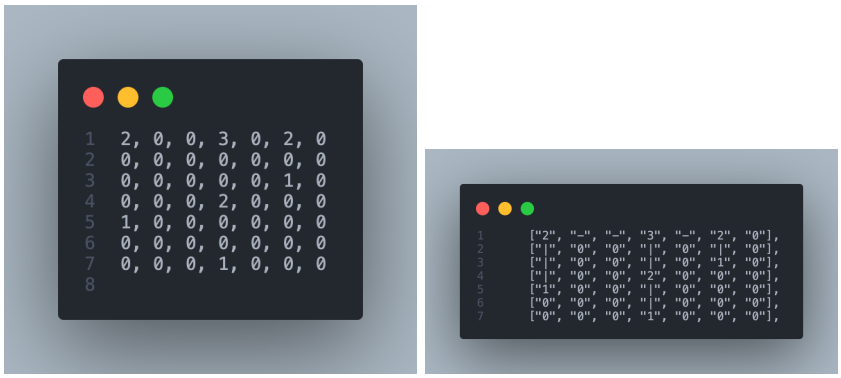
Các thành phần chính của thuật toán

- Hàm khởi tạo `init`: nhận vào một đối tượng board đại diện cho trạng thái ban đầu của trò chơi.
- Hàm `solve`: là phương thức chính thực hiện quá trình thử và kiểm tra tất cả các tổ hợp cầu nối.
- Hàm `generatepotentialbridges`: sinh ra tất cả các cặp đảo tiềm năng có thể nối với nhau, dựa trên điều kiện nằm trên cùng hàng hoặc cột.
- Hàm `copyboard`: tạo một bản sao mới của bảng hiện tại để thử nghiệm cầu nối mà không làm thay đổi bảng gốc.

3 Test case

Tổng quan về các test case

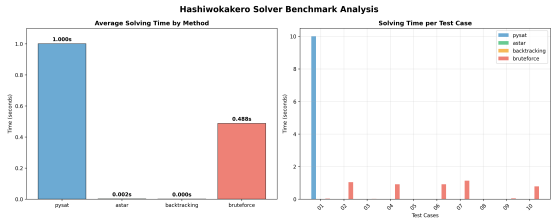
Các bộ test được sử dụng có kích thước đa dạng, từ nhỏ (7x7) đến lớn (13x13), đại diện cho các bài toán Hashiwokakero với độ phức tạp tăng dần. Mỗi tệp `input` mô tả trạng thái ban đầu của bảng chơi, trong đó các số nguyên biểu thị số lượng cầu cần kết nối tại mỗi đảo. Tương ứng, tệp `output` là lời giải hoàn chỉnh, thể hiện các cầu nối hợp lệ giữa các đảo: ký hiệu "`-`" đại diện cho cầu ngang và "`|`" cho cầu dọc.



Hình 1: Ví dụ về một test case: Trạng thái ban đầu (trái) và lời giải tương ứng (phải)

Tất cả lời giải đều tuân thủ nghiêm ngặt các quy tắc của trò chơi và đảm bảo tính đúng đắn về mặt logic.

Benchmark



Hình 2: So sánh hiệu năng giữa các thuật toán giải Hashiwokakero

Biểu đồ trên minh họa hiệu năng giải đố của các thuật toán được triển khai. Qua biểu đồ, có thể đánh giá tương đối khả năng của từng thuật toán trong việc xử lý các bài toán có độ khó khác nhau. Tuy nhiên, do kết quả còn phụ thuộc nhiều vào quá trình sinh dạng chuẩn CNF và đặc điểm cấu trúc từng test case, nên hiệu năng của mỗi thuật toán có thể biến đổi đáng kể giữa các trường hợp.