

## Course

### 23CLC03 - Cơ sở trí tuệ nhân tạo

## Mục Lục

<b>Course.....</b>	<b>1</b>
<b>Mục Lục.....</b>	<b>1</b>
1. Thông Tin Nhóm.....	2
2. Định nghĩa biến logic (Define Logical Variables).....	2
3. Xây dựng ràng buộc CNF (Formulate CNF Constraints).....	3
4. Khởi tạo CNF tự động (Automate CNF Generation).....	4
5. Sử dụng thư viện PySAT (Solve Using PySAT).....	4
6. Thuật toán A* (Apply A* search algorithm).....	4
7. Thuật toán Brute-force & Backtracking.....	4
<b>8. Đánh giá &amp; tổng kết.....</b>	<b>4</b>
<b>9. Tham khảo.....</b>	<b>4</b>

## 1. Thông Tin Nhóm

**Tên nhóm: Group - 12**

Họ Tên	MSSV	Phân công
Lê Đức Hanh	20127491	Report, xây dựng hàm ràng buộc, khởi tạo CNF tự động, pySAT
Bùi Thế Anh	23127021	A*, brute-force, backtracking, đánh giá hiệu suất
Phạm Hoàng Khánh Đăng	21127588	pySAT, A*, brute-force, đánh giá hiệu suất
Phan Hồng Hà	20127489	Demo, report, định nghĩa biến logic, khởi tạo CNF tự động

## Đánh giá yêu cầu đồ án

Yêu cầu đồ án	Tiến độ
Mô tả logic bài toán	Hoàn thành
Tự động tạo CNF	Hoàn thành
Sử dụng thư viện PySAT để giải bài toán	Hoàn thành
Áp dụng thuật toán A*	Hoàn thành
Áp dụng thuật toán: <ul style="list-style-type: none"><li>• Brute-force</li><li>• Backtracking</li></ul>	Hoàn thành
Thử nghiệm và đánh giá hiệu suất thuật toán	Hoàn thành
Quay video: <ul style="list-style-type: none"><li>• Giới thiệu các thuật toán đã thực hiện.</li><li>• Chạy demo.</li></ul>	Hoàn thành
Báo cáo	Hoàn thành

## 2. Định nghĩa biến logic (Define Logical Variables)

### Biến

Đối với mỗi cặp đảo liền kề A và B, định nghĩa hai biến Boolean:

- $x1_{A,B}$ : tồn tại một cầu đơn nối giữa đảo A và đảo B.
- $x2_{A,B}$ : tồn tại một cầu đôi nối giữa đảo A và đảo B.

### Ràng buộc

- Mỗi cặp đảo chỉ có thể có 0, 1 hoặc 2 cầu nối, không được quá 2 cầu giữa 1 cặp đảo
- Số lượng cầu nối với một đảo phải đúng bằng số được ghi trên đảo đó.
- Các cầu không được phép giao nhau.

### Trường hợp đặc biệt:

- Nếu một đảo chỉ có một đảo liền kề duy nhất theo một hướng chính (bắc, nam, đông hoặc tây), thì bắt buộc phải nối với đảo đó, có thể dùng cầu đơn hoặc cầu đôi tùy vào số được ghi trên đảo.
- Không được nối hai đảo mang số 1 với nhau, vì điều này có thể khiến hai đảo bị tách biệt khỏi nhóm liên thông.
- Hai đảo đều mang số 2 không được đặt cầu đôi giữa chúng, vì sẽ gây ra chia tách bản đồ dẫn đến phạm quy định của trò chơi.
- Nếu một đảo mang số 6 và chỉ có 3 hướng có thể nối cầu, thì cần đặt 2 cầu ở cả 3 hướng đó để đạt tổng là 6. Nếu có 1 đảo liền kề là đảo số 1 thì cần 5 cầu được đặt ở 3 hướng còn lại, vì vậy có thể ngay lập tức đặt 1 cầu ở cả 3 hướng đó.
- Nếu đảo mang số 7, tức là chỉ thiếu 1 cầu trong tổng 8 cầu có thể đặt dẫn đến cần có thể đặt 1 cầu ở cả 4 hướng sau đó mới xác định tiếp.
- Nếu đảo mang số 8, thì lập tức đặt 2 cầu ở tất cả các hướng trong 4 hướng (bắc, nam, đông, tây) nên lập tức có thể được giải.

## 3. Xây dựng ràng buộc CNF (Formulate CNF Constraints)

### Loại trừ lẫn nhau (Mutual Exclusion)

Đảm bảo 2 biến không cùng đúng tại 1 thời điểm

$$\neg x1_{A,B} \vee \neg x2_{A,B}$$

### Ràng buộc tổng cầu nối tại đảo (Island Degree Constraints)

Với mỗi đảo A có số yêu cầu là  $n$ , tổng số cầu nối đến A phải bằng đúng  $n$ . Gọi  $\text{Adj}(A)$  là tập các đảo liền kề với A. Ví dụ, nếu A nối với B, C, D thì:

$$\sum x1_{A,X} + 2 \cdot \sum x2_{A,X} = n$$

với  $X \in \text{Adj}(A)$

### Ràng buộc không có cầu giao nhau (No crossing bridges)

Với mỗi cặp cầu ngang (A,B) và cầu dọc (C,D) cắt nhau trên lưới, ta thêm các mệnh đề sau để ngăn việc cả hai cùng tồn tại:

$$\neg x1\_A,B \vee \neg x1\_C,D$$

$$\neg x1\_A,B \vee \neg x2\_C,D$$

$$\neg x2\_A,B \vee \neg x1\_C,D$$

$$\neg x2\_A,B \vee \neg x2\_C,D$$

### Trường hợp đặc biệt

- Không nối hai đảo có số là 1 với nhau: đánh dấu cả biến cầu đơn và cầu đôi giữa chúng là phủ định ( $\neg$ ).
- Không đặt cầu đôi giữa hai đảo có số là 2: đánh dấu biến cầu đôi là phủ định ( $\neg$ ).
- Các trường hợp đặc biệt còn lại: chỉ cần tuân theo logic mô tả ở mỗi trường hợp và đánh dấu các biến tương ứng là đúng (positive) hoặc sai (negative) cho phù hợp.

## 4. Khởi tạo CNF tự động (Automate CNF Generation)

Trong phần này, ta xây dựng hệ thống để **tự động sinh các mệnh đề dưới dạng chuẩn CNF (Conjunctive Normal Form)** dựa trên cấu trúc bản đồ đầu vào của bài toán Hashiwokakero.

### Ý tưởng chính

Đầu vào là một ma trận kích thước  $N \times M$  chứa các số nguyên đại diện cho các đảo cùng với số lượng cầu phải nối vào từng đảo. Từ dữ liệu này, ta tự động sinh ra:

- Danh sách các đảo (vị trí và giá trị).
- Tập các cầu hợp lệ giữa các đảo (theo chiều ngang và dọc, không bị chắn).
- Các cặp cầu cắt nhau (để ràng buộc không cho cùng tồn tại).
- Và cuối cùng là **tập mệnh đề CNF mô tả toàn bộ ràng buộc của bài toán**.

Tất cả các bước trên đều được thực hiện tự động qua code.

```

120 def generate(grid):
121     islands = find_islands(grid)
122     bridges = find_bridges(grid, islands)
123     crossing_bridges = find_crossing_bridges(bridges, islands)
124
125     hashi = HashiCNFManager(grid, islands, bridges, crossing_bridges)
126
127     return hashi.extract_cnf(), hashi.get_statistics(), hashi
128

```

Cụ thể:

- **find\_islands(grid)** → phân tích lưới đầu vào, tìm tọa độ & giá trị các đảo.
- **find\_bridges(grid, islands)** → tìm tất cả cầu có thể nối giữa các đảo.
- **find\_crossing\_bridges(...)** → phát hiện các cầu cắt nhau.
- **HashiCNFManager(...)** → khởi tạo biến logic và xây dựng ràng buộc CNF.
- **hashi.extract\_cnf()** → trích xuất danh sách mệnh đề CNF.
- **hashi.get\_statistics()** → thống kê thông tin như số biến, số mệnh đề, số đảo...
- Trả về các đối tượng để giải bằng solver như PySAT.

## 5. Sử dụng thư viện PySAT (Solve Using PySAT)

Trong phần này, chúng ta áp dụng **thư viện PySAT** để giải bài toán Hashiwokakero bằng cách biểu diễn trò chơi dưới dạng **biểu thức logic CNF (Chuyển về mệnh đề chuẩn tắc)** và sau đó sử dụng bộ giải SAT để tìm nghiệm thỏa mãn các ràng buộc.

### 5.1. Tạo biến logic

Mỗi cầu tiềm năng được biểu diễn bằng một hoặc hai biến logic:

- Một biến đại diện cho **cầu đơn**

- Một biến đại diện cho **cầu đôi**

Thêm vào đó, để đảm bảo tính liên thông giữa các đảo, chúng ta tạo thêm các **biến dòng chảy (flow variables)** giữa các đảo để mã hóa ràng buộc kết nối.

## 5.2. Biểu diễn ràng buộc dưới dạng CNF

Các ràng buộc logic được mã hóa bao gồm:

- **Giá trị đảo:** Tổng số cầu kết nối đến đảo bằng giá trị của đảo
- **Không giao nhau:** Hai cầu cắt nhau không thể tồn tại đồng thời
- **Ràng buộc kết nối:** Dựa trên mô hình dòng chảy, mọi đảo đều được kết nối với đảo gốc

Chúng ta sử dụng các công cụ như **CardEnc** và **PBEnc** từ PySAT để mã hóa các ràng buộc đếm vào CNF một cách hiệu quả.

## 5.3. Giải bằng PySAT

Sau khi sinh ra các mệnh đề CNF, chúng ta sử dụng **Glucose3** từ PySAT để tìm nghiệm:

```
from pysat.solvers import Glucose3

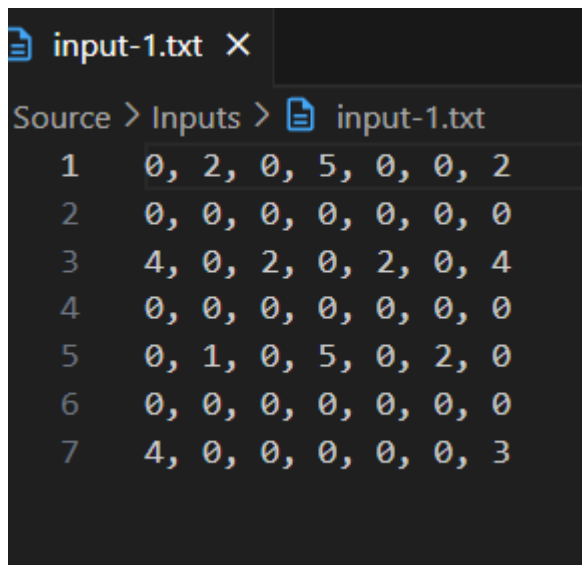
def (variable) solver: Glucose3
    solver = Glucose3()
    solver.append_formula(cnf.clauses)

    if solver.solve():
        model = solver.get_model()
        solver.delete()
        return model
    else:
        solver.delete()
        return None
```

Nếu tìm được nghiệm (model), chúng ta tiến hành **giải mã giá trị của các biến** để xác định xem mỗi câu có tồn tại hay không (câu đơn, đôi hay không có câu).

#### 5.4. Kết quả và minh họa

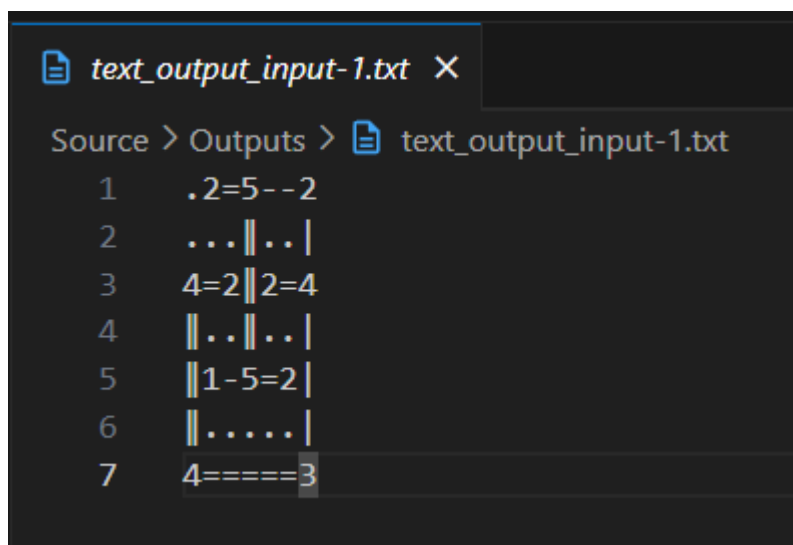
Chương trình đã tìm được lời giải hợp lệ cho một số lưới Hashiwokakero mẫu. Ví dụ, với lưới đầu vào:



The screenshot shows a text editor window titled 'input-1.txt'. The content is a 7x8 grid of numbers separated by commas. The grid is as follows:

	0	2	0	5	0	0	2
1	0	0	0	0	0	0	0
2	4	0	2	0	2	0	4
3	0	0	0	0	0	0	0
4	0	1	0	5	0	2	0
5	0	0	0	0	0	0	0
6	4	0	0	0	0	0	3
7							

Đầu ra dạng text sau khi giải và giải mã model:



The screenshot shows a text editor window titled 'text\_output\_input-1.txt'. The content is a 7x8 grid of text representing the solved puzzle. The grid is as follows:

	.	2	=	5	-	-	2
1	.	.	.		.	.	
2	4	=	2		2	=	4
3		.	.		.	.	
4		1	-	5	=	2	
5		.	.	.	.	.	
6	4	=	=	=	=	=	3
7							

## 5.5. Nhận xét

- Phương pháp giải bằng PySAT rất nhanh và chính xác trên các lưới nhỏ đến vừa
- Tách biệt phần sinh CNF và giải giúp dễ dàng mở rộng, thay thế hoặc kiểm tra các thuật toán khác như A\*, brute-force, v.v.

## 6. Thuật toán A\* (Apply A\* search algorithm)

Tại đây, chúng ta sử dụng thuật toán A\* để giải dạng biểu diễn CNF của Hashiwokakero.

### a. Biểu diễn trạng thái

- Coi mỗi trạng thái có thể xảy ra là một node trên bản đồ, kết nối với các node cách 1 nước đi
- `encode_state()` chuyển trạng thái thành mệnh đề đơn để đưa vào SAT

### b. Heuristic

- Sử dụng hàm chuẩn của A\*:  $f(n) = g(n) + h(n)$
- $g(n)$  là số bước (hoặc số cầu) đã đặt.
- $h(n)$  là số biến chưa gán, tức là `var_count - len(assign)`
- Heuristic admissible nhưng không tối ưu

### c. Tiến trình tìm kiếm

1. Khởi tạo
  - Trạng thái ban đầu là một dict rỗng
  - Tính  $f(n)$  và thêm vào hàng đợi frontier
2. Vòng lặp chính
  - Lấy trạng thái tốt nhất từ hàng đợi.
  - Kết hợp CNF gốc và các mệnh đề từ trạng thái hiện tại -> giải bằng PySAT (Glucose3).
  - Nếu tìm được nghiệm -> trả về mô hình (model).
  - Nếu chưa, chọn một biến chưa được gán, thử cả hai giá trị (True/False), tính lại  $f(n)$  và thêm trạng thái mới vào hàng đợi
3. Kết thúc



- Nếu hàng đợi rỗng mà không tìm được lời giải -> không có nghiệm thỏa mãn.

#### **d. Nhận xét**

Ưu điểm:

- Có định hướng, ít tốn thời gian hơn brute-force.
- Cho phép mở rộng với heuristic mạnh hơn (ví dụ: kiểm tra độ vi phạm ràng buộc)

Nhược điểm:

- Hiện tại sử dụng heuristic đơn giản nên tốc độ chưa tối ưu.
- Mỗi bước gọi lại SAT solver, gây tốn thời gian nếu không cache trạng thái.

## **7. Thuật toán Brute-force & Backtracking**

Để so sánh với A\*, nhóm đã triển khai hai thuật toán cơ bản là Brute-force và Backtracking. Cả hai phương pháp đều không sử dụng thuật toán tìm kiếm tối ưu mà thay vào đó là các phương pháp đơn giản và trực tiếp, giúp kiểm tra tính hợp lệ của tất cả các khả năng kết nối giữa các đảo.

### **a. Brute-force**

Cách thức hoạt động:

- Duyệt tất cả các khả năng: Thuật toán brute-force thử tất cả các kết hợp có thể của các cầu đơn và cầu đôi giữa các đảo.
- Các biến cầu được biểu diễn dưới dạng 1 (cầu có), -1 (cầu không có), sau đó chuyển thành các mệnh đề giả thuyết và đưa vào bộ giải SAT (Glucose3).

Chiến lược: Duyệt qua tất cả các tổ hợp có thể của các cầu đơn/đôi (sử dụng `itertools.product`), kiểm tra tính hợp lệ của mỗi tổ hợp, và trả về mô hình hợp lệ nếu tìm được.

Nhận xét:

- Ưu điểm: Đảm bảo tìm được tất cả các nghiệm có thể, chính xác tuyệt đối.
- Nhược điểm: Tốn rất nhiều thời gian với lưới lớn vì số khả năng tăng theo cấp số nhân.

### **b. Thuật toán Backtracking**

Cách thức hoạt động:

- Tìm kiếm theo chiều sâu: Thuật toán sử dụng phương pháp đệ quy để xây dựng giải pháp dần dần. Nếu tại một điểm nào đó, giải pháp không hợp lệ, thuật toán quay lại (backtrack) và thử một khả năng khác.

- Cấu trúc dữ liệu: Sử dụng một mảng assignment để lưu trữ các giá trị gán của các biến. Tại mỗi bước, thuật toán thử gán giá trị True (1) hoặc False (-1) cho các biến cần.
- Thuật toán đệ quy: Mỗi bước cố gắng gán giá trị cho một biến và gọi đệ quy để tiếp tục tìm kiếm. Nếu tất cả các biến được gán hợp lệ, thuật toán trả về nghiệm.

Nhận xét:

- Ưu điểm: Thử nghiệm từng bước, loại trừ nhanh chóng các khả năng không hợp lệ (tính năng "cắt tỉa").
- Nhược điểm: Mặc dù nhanh hơn brute-force, nhưng vẫn có thể gặp phải tình trạng bùng nổ khi không có chiến lược tối ưu.

## 8. Đánh giá & tổng kết

Trong dự án này, nhóm đã triển khai các thuật toán khác nhau để giải quyết bài toán Hashiwokakero, bao gồm các thuật toán PySAT, A\*, Brute-force, và Backtracking. Mỗi thuật toán có những ưu điểm và hạn chế riêng, và được áp dụng tùy vào kích thước của bài toán.

### a. So sánh hiệu năng

Phương pháp	Ưu điểm	Nhược điểm	Thời gian (4x4)	Bộ nhớ (4x4)	Thời gian (5x5)	Bộ nhớ (5x5)
<b>PySAT</b>	Nhanh, chính xác, mô-đun rõ ràng	Cần CNF chính xác, khó mở rộng	3ms	5KB	6ms	11KB
<b>A*</b>	Linh hoạt	Phụ thuộc vào heuristic	~0ms	4KB	2ms	7KB
<b>Brute-force</b>	Chính xác	Chậm, không khả thi khi lưới lớn	69ms	8KB	191ms	17KB
<b>Backtrac</b>	Loại trừ	Tốc độ	65ms	3KB	132ms	6KB

king	các khả năng sai	không ổn định				
------	------------------	---------------	--	--	--	--

## 9. Tham khảo

- <https://www.kakuro-online.com/hasi/>
- ChatGPT