

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN KHOA
CÔNG NGHỆ THÔNG TIN



fit@hcmus

**Đồ án 2
HASHIWOKAKERO**

Nhóm sinh viên thực hiện:

Lê Gia Bảo **MSSV:23127325**

Vũ Anh **MSSV:23127321**

Hồ Gia Huy **MSSV:23127376**

Nguyễn Phan Thế Vinh

MSSV:23127520

Môn: Cơ Sở Trí Tuệ Nhân Tạo

Năm học: 2024-2025

TP.HCM, tháng 4 năm 2025

I. Bảng phân công	2
II. Self-evaluation.....	2
III. Nguyên lý Logic	4
IV. Thuật toán A*.....	5
V. Thuật toán Backtracking	7
VI. Thuật toán Brute-Force	8
VII.Bảng so sánh kết quả	8
VIII. References	9

I. Bảng phân công

Thành Viên	Công việc
Vũ Anh	Tham gia làm CNF, viết báo cáo, so sánh thời gian chạy, hàm ghi output ra file.
Lê Gia Bảo	Trình bày nguyên lí logic sinh ra các mệnh đề CNF, viết chương trình tự động sinh CNF
Hồ Gia Huy	Thuật toán A*, Backtracking, giải các mệnh đề CNF bằng pySAT, hàm đọc input từ file
Nguyễn Phan Thể Vinh	Thuật toán Brute Force, làm các file input, quay video

II. Self-evaluation

1. Mức độ hoàn thành:

- Các thuật toán hoạt động đúng với yêu cầu đề án đưa ra.
- Có đầy đủ các test case của các trường hợp có thể xảy ra.
- Link hoàn thành đồ án:

<https://github.com/Vincenthevinh/Hashiwokakero.git>

- Video hướng dẫn đồ án: <https://youtu.be/3-7EKNwq8KE>

Yêu cầu	Mức độ hoàn thành	Ghi chú
Tạo biến logic và biểu diễn bài toán dưới dạng CNF	<input checked="" type="checkbox"/> Hoàn thành	Đã xây dựng biến cho từng câu 1 hoặc 2 câu giữa các đảo, mã hóa đủ 5 điều kiện logic
Tự động sinh CNF	<input checked="" type="checkbox"/> Hoàn thành	Hệ thống sinh CNF từ lưới đầu vào, sử dụng ID biến và bảng hash ánh xạ logic
Giải CNF bằng thư viện PySAT	<input checked="" type="checkbox"/> Hoàn thành	Dùng solver Clucose3 của PySAT
Cài đặt thuật toán A* để giải CNF	<input checked="" type="checkbox"/> Hoàn thành	Sử dụng heuristic dựa trên tần suất biến và ràng buộc đơn vị
Cài đặt brute-force và backtracking	<input checked="" type="checkbox"/> Hoàn thành	Dùng để so sánh với A*
Tạo 10 test	<input checked="" type="checkbox"/> Hoàn thành	

case đầu vào		
--------------	--	--

2. Khó khăn:

- Tối ưu hiệu suất cho A* và Backtracking (Nhất là A* phải dùng tận 2 heuristic)
- Không thể tạo CNF liên thông.

III. Nguyên lý Logic

1. Mô tả các biến:

- a, b, c, d: là các đảo trong lưới.
- $X_{a, \beta, 1}$: biến logic đại diện cho **cầu thứ nhất** giữa đảo a và b. (True \Leftrightarrow có tồn tại)
- $X_{a, \beta, 2}$: biến logic đại diện cho **cầu thứ hai** giữa đảo a và b.
- $f(a)$: số cầu cần kết nối tại đảo a (theo số đã cho).
- $L(a, b)$: đảo a và b nằm **trên cùng một hàng**.
- $C(a, b)$: đảo a và b nằm **trên cùng một cột**.
- $Near(a, b)$: đảo a và b **liền kề trực tiếp** (khoảng cách 1 ô).
- $Cr(a, b, c, d)$: cầu từ a đến b **giao nhau** với cầu từ c đến d

2. Ràng buộc điều kiện

- Ràng buộc 1: Không được thêm cầu thứ 2 nếu không có cầu thứ 1
 \Rightarrow Nếu tồn tại cầu thứ 2 giữa a và b thì phải tồn tại cầu thứ nhất

$$\forall a, b \in X_{a,b,2} \Rightarrow X_{a,b,1} \equiv \neg X_{a,b,2} \vee X_{a,b,1}$$

- Ràng buộc 2: Đảo không được nối với chính nó

$$\forall a: \neg X_{a,a,1}$$

- Ràng buộc 3: Cầu chỉ được phép nằm ngang hoặc dọc

$$\forall a, b: X_{a,b,1} \Rightarrow (L(a, b) \vee C(a, b)) \wedge \neg(L(a, b) \oplus C(a, b))$$

Chuyển về CNF:

$$\neg X_{a,b,1} \vee L(a, b) \vee C(a, b)$$

$$\neg X_{a,b,1} \vee \neg L(a, b) \vee \neg C(a, b)$$

- Ràng buộc 4: Không được nối cầu giữa 2 đảo liền kề

$$\forall \mathbf{a}, \mathbf{b}: \text{Near}(\mathbf{a}, \mathbf{b}) \Rightarrow \neg X_{\mathbf{a}, \mathbf{b}, 1} \equiv \neg \text{Near}(\mathbf{a}, \mathbf{b}) \vee \neg X_{\mathbf{a}, \mathbf{b}, 1}$$

- Ràng buộc 5: Không được để các cầu giao nhau

+ Ràng buộc này được xác định bằng cách kiểm tra hình học trong lối (vector hướng và vị trí giao nhau)

$$\forall \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}: \text{Cr}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \Rightarrow \neg X_{\mathbf{a}, \mathbf{b}, 1} \vee \neg X_{\mathbf{c}, \mathbf{d}, 1}$$

- Ràng buộc 6: Tổng số cầu đúng bằng số trên đảo

+ Ràng buộc này sử dụng thư viện pysat.card.CardEnc.equals() để mã hóa CNF tương đương

$$\forall \mathbf{a}: \sum_{\mathbf{b} \text{ là hàng xóm của } \mathbf{a}} (X_{\mathbf{a}, \mathbf{b}, 1} + X_{\mathbf{a}, \mathbf{b}, 2}) = f(\mathbf{a})$$

IV. Thuật toán A*

1. State

- Mỗi trạng thái trong A* gồm:

+ *assignment*: ánh xạ từ biến => giá trị True/False

+ *unsat_clause*: tập chỉ số các mệnh đề chưa được thỏa mãn

2. Hàm f(n)

- g(n): số bước gán biến đã thực hiện

- h(n): số mệnh đề chưa thỏa mãn trong trạng thái hiện tại – heuristic

⇒ Cách thức hoạt động: tìm 1 *assigment* sao cho *unsat_clauses* = 0
(tức là toàn bộ CNF được thỏa mãn)

3. Cải tiến và tối ưu

- Áp dụng unit propagation để rút gọn trạng thái nhanh chóng trước khi mở rộng

- Thêm heuristic chọn biến:

+ Với mỗi biến chưa gán:

+ Tính tần suất xuất hiện của biến trong các mệnh đề chưa thỏa mãn

- + Nếu biến xuất hiện nhiều thì ưu tiên gán sớm
- + Ưu tiên các biến thuộc các ràng buộc

```

var_scores = defaultdict(float)
bridge_vars = set()

for clause_idx in unsat:
    clause = self.cnf[clause_idx]
    if any(abs(lit) in self.freq and self.freq[abs(lit)] > 5 for lit in clause):
        for lit in clause:
            var = abs(lit)
            if var not in current_assignment:
                var_scores[var] += 2.0
                bridge_vars.add(var)

```

Biến xuất hiện trong **các mệnh đề chưa thỏa mãn**, đặc biệt là các biến liên quan đến số cầu ($\text{freq} > 5$), sẽ được cộng điểm ưu tiên cao hơn → chọn trước.

Nếu không tìm được biến thuộc dạng đặc biệt, fallback:

```

if not var_scores:
    for clause_idx in unsat:
        clause = self.cnf[clause_idx]
        for lit in clause:
            var = abs(lit)
            if var not in current_assignment:
                var_scores[var] += 1.0 / len(clause)

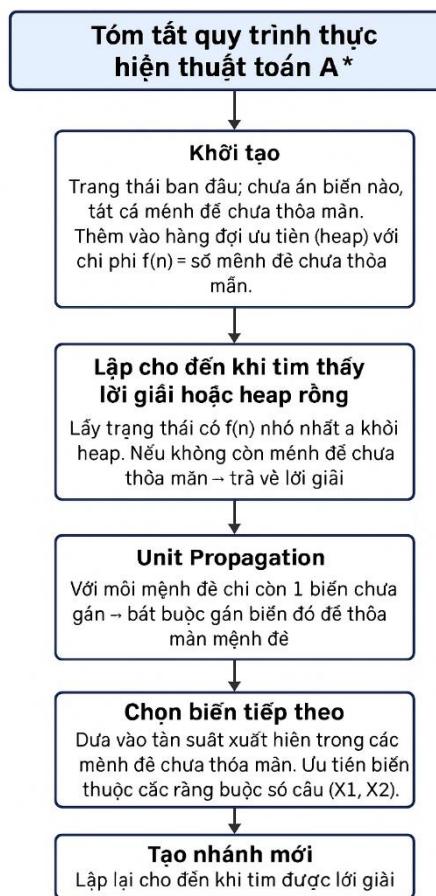
```

→ Biến trong mệnh đề ngắn sẽ được ưu tiên hơn.

4. Ví dụ minh họa

$$(x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$$

Thuật toán sẽ gán $x_1 = \text{True}$ (do đơn mệnh đề) → propagation $x_2 = \text{True} \rightarrow x_3 = \text{True}$



V. Thuật toán Backtracking

Backtracking là phương pháp duyệt các cấu hình có thể của bài toán Hashiwokakero sao cho thỏa mãn mọi mệnh đề bằng cách thử vào giá trị True/False và quay lui khi cấu hình hiện tại vi phạm ràng buộc.

Thuật toán khi khởi tạo cũng nhận vào tham số CNF. Sau đó cũng trích xuất các biến như thuật toán Brute Force, ngoài ra còn có biến assignment để lưu trữ gán hiện tại.

Thuật toán hoạt động như sau :

- Khởi tạo biến assignment rỗng
- Sử dụng đệ quy để tìm lời giải:
 - + Nếu tất cả Clause thỏa mãn trả về kết quả
 - + Chọn biến chưa gán
 - + Thử gán True cho biến, nếu không xung đột các ràng buộc tiếp tục đệ quy

+ Nếu giá trị True vi phạm, thử gán False

+ Nếu cả 2 giá trị True False không thoả, xoá giá trị đã gán và quay lui

VI. Thuật toán Brute-Force

- Brute Force là phương pháp giải quyết vấn đề bằng cách thử tất cả các tổ hợp có thể của các biến cho đến khi tìm được lời giải thỏa mãn tất cả các ràng buộc. Thuật toán sinh ra tất cả kết hợp có thể của các biến boolean. Mỗi biến nhận giá trị 1 hoặc 0.

Thuật toán khởi tạo với tham số là CNF. Đối với từng kết hợp được sinh ra, thuật toán kiểm tra xem có thỏa mãn tất cả các clause trong CNF không.

Sử dụng itertools.product để tạo tất cả các tổ hợp từ n biến trong tập variables, vì một biến có trạng thái biểu diễn là True/False nên n biến sẽ có 2^n tổ hợp => hoạt động tốt nhất khi CNF sinh ra không quá nhiều biến. Mỗi tổ hợp được chuyển thành dictionary Model để ánh xạ tới biến giá trị. Mỗi Model sẽ được kiểm tra xem có thỏa mãn không bằng hàm _check_satisfaction, hàm này hoạt động như sau :

- Mỗi Clause chỉ cần có ít nhất 1 biến là True thì cả Clause sẽ đúng.
- Literal dương nếu biến tương ứng là True, ngược lại âm nếu biến tương ứng là False
- Nếu tất cả các Clause là True trả về kết quả

VII. Bảng so sánh kết quả

Grid: 2, 0, 2, 0

0, 0, 0, 0

1, 0, 0, 0

0, 0, 1, 0

	pySAT	A*	Backtracking	Brute-Force
Thời gian	0.0000ms	0.9954ms	1.0600ms	5494.8411ms

Dung lượng	0.0469MB	0.0078MB	0.0	0.0
------------	----------	----------	-----	-----

⇒ Tốc độ pySAT nhanh nhất nhưng cũng tốn RAM nhiều nhất.

VIII. References

- Tham khảo ChatGPT
- [PySAT](#)
- [HASHIWOKAKERO](#)
- [Find Heuristic for CNF](#)
- [Hashiwokakero - T. Morsink - 2009](#)
- [Solving and generating puzzles - Gerhard van der Knijff - 2021](#)
- [Benchmark Instances and Branch-and-Cut Algorithm for the Hashiwokakero Puzzle](#)
- [Algorithms for the Satisfiability \(SAT\) Problem: A Survey](#)