ĐẠI HỌC QUỐC GIA
THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

CS14003 – Cơ Sở Trí Tuệ Nhân Tạo

BÁO CÁO ĐỒ ÁN 02
Hashiwokakero

Họ tên	MSSV
Nguyễn Thị Khánh Linh	23127082
Mạch Quốc Tấn	23127115
Nguyễn Thành Dâng	23127334
Trương Thành Đạt	23127344

Giảng viên hướng dẫn

Bùi Duy Đăng — Nguyễn Thanh Tình
Lê Nhựt Nam — Huỳnh Lâm Hải Đăng

Mục lục

1 Thông tin chung

1.1 Thông tin đồ án

- Tên học phần: Cơ sở trí tuệ nhân tạo
- Giảng viên hướng dẫn: Bùi Duy Đăng, Nguyễn Thanh Tình, Lê Nhựt Nam, Huỳnh Lâm Hải Đăng
- Đồ án thực hiện: Logic - Hashiwokakero
- Video Demo chạy đồ án:
- Source code: Github

1.2 Mô tả đồ án

Trong đồ án này, ta cần cài đặt một chương trình giải bài toán Hashiwokakero bằng cách sử dụng Conjunctive Normal Form (CNF). Đây là một trò chơi giải đố trên lưới chữ nhật, trong đó một số ô sẽ chứa các con số từ 1 đến 8 – gọi là "đảo". Nhiệm vụ của ta là nối tất cả các hòn đảo bằng cách vẽ các "cây cầu" tuân theo những quy tắc cụ thể:

- Cầu phải nối giữa hai hòn đảo khác nhau bằng đường ngang hoặc đường dọc, không cắt qua bất kỳ hòn đảo hay cầu nào khác.
- Giữa 2 hòn đảo, tối đa nối bởi 2 cây cầu.
- Số cầu nối tới một hòn đảo phải bằng đúng số ghi trên đảo đó.
- Các hòn đảo phải được nối thành một khối liên thông duy nhất.

Các bước thực hiện:

- Xác định biến logic: Mỗi ô trong ma trận sẽ được gán một biến logic.
- Biểu diễn ràng buộc dưới dạng CNF: Viết các ràng buộc cần thỏa mãn của trò chơi dưới dạng các mệnh đề CNF, sau đó bỏ đi các mệnh đề trùng lặp.
- Tự động tạo CNF: Tự động tạo ra các mệnh đề CNF từ input của đề bài.
- Giải bằng PySAT: Sử dụng thư viện pysat để tìm nghiệm và suy ra kết quả.
- Áp dụng A Search Algorithm:* Cài đặt thuật toán A* để giải bài toán dựa trên CNF.
- So sánh với các phương pháp khác: Viết thêm thuật toán brute-force và backtracking để so sánh tốc độ (thời gian chạy) và hiệu suất với A*.

1.3 Thông tin thành viên

MSSV	Họ và tên	Email
23127115	Mạch Quốc Tấn	mqtan23@clc.fitus.edu.vn
23127334	Nguyễn Thành Dâng	ntdang23@clc.fitus.edu.vn
23127344	Trương Thành Đạt	ttdat23@clc.fitus.edu.vn
23127082	Nguyễn Thị Khánh Linh	ntklinh23@clc.fitus.edu.vn

Bảng 1: Bảng thông tin thành viên

1.4 Bảng phân chia công việc

Công việc	**Phụ trách**	**Trạng thái**	%
Định nghĩa các mệnh đề và điều kiện mệnh đề	Quốc Tấn	Hoàn Thành	100%
Tạo CNF tự động	Quốc Tấn Thành Dâng	Hoàn Thành	100%
Sử dụng thư viện pySAT giải quyết CNF	Quốc Tấn Thành Dâng	Hoàn Thành	100%
Thực thi thuật toán và viết báo cáo về Astar	Khánh Linh	Hoàn Thành	100%
Thực thi thuật toán và viết báo cáo về Brute-force	Thành Đạt Thành Dâng	Hoàn Thành	100%
Thực thi thuật toán và viết báo cáo backtracking (DPLL)	Quốc Tấn	Hoàn Thành	100%
Viết báo cáo so sánh các thuật toán	Thành Dâng	Hoàn Thành	100%
Thực hiện video chạy thử chương trình	Khánh Linh	Hoàn Thành	100%
Vẽ flow-chart các thuật toán	Thành Đạt	Hoàn Thành	100%

Bảng 2: Bảng phân chia công việc

2 Đánh giá kết quả đồ án

Trong quá trình thực hiện đồ án cho đến khi đạt được kết quả mong muốn, nhóm chúng em đã học hỏi được rất nhiều điều thông qua đồ án cũng như cài đặt thành công và cải tiến các thuật toán trở nên tối ưu hơn. Qua đó, nhóm chúng em thấy rằng nhóm đã hoàn thành tốt các yêu cầu đề ra, tự đánh giá với số điểm là 10/10, cụ thể ở các tiêu chí sau:

Mô tả các nguyên tắc logic đúng để tạo CNF: Nhóm chúng em đã mô phỏng được các thành phần trong trò chơi Hashiwokakero thành các mệnh đề. Ngoài ra, chúng em còn định nghĩa lại được cách điều kiện của trò chơi thành các điều kiện mệnh đề, điều này sẽ được thể hiện rõ hơn trong phần báo cáo bên dưới.

Tạo CNF tự động: Sau khi định nghĩa được các điều kiện thành các mệnh đề logic, chúng em đã chuyển hóa được các mệnh đề điều kiện thành dạng hội chuẩn (Conjunctive Normal Form).Khi đưa vào một bản đồ Hashiwokakero, chương trình sẽ tự động tạo nên các điều kiện của CNF để hỗ trợ trong việc giải mã trò chơi.

Sử dụng thư viện PySAT để giải chính xác điều kiện CNF: Nhóm chúng em đã vận dụng tốt việc sử dụng thư viện có sẵn để tìm ra lời giải cho ma trận các hòn đảo. Qua đó giúp chúng em có được một mô hình giải quyết bài toán cơ bản để có thể áp dụng cho các thuật toán sau.

Thực hiện thuật toán A* không sử dụng thư viện hỗ trợ sẵn: Nhóm chúng em thực hiện thuật toán A* thành công với độ tối ưu cao khi kết hợp với các hàm cải tiến, bổ trợ trong quá trình tìm ra lời giải CNF.

Thực hiện thuật toán Brute-force và Backtracking, so sánh tốc độ với A*: Nhóm chúng em thực hiện tốt trong quá trình cài đặt thêm 2 thuật toán mới, ngoài ra, trong quá trình tìm hiểu, chúng em có cải tiến thêm để thuật toán có thể cho ra kết quả với thời gian tốt hơn. Thực hiện so sánh các thuật toán với dữ liệu là thời gian chạy của mỗi thuật toán trên 10 testcase khác nhau, trực quan hóa dữ liệu so sánh qua các biểu đồ giúp việc thấy được sự khác biệt của các thuật toán một cách rõ ràng hơn.

Thực hiện báo cáo và phân tích: Sau khi thành công trong quá trình thực thi các thuật toán, nhóm chúng em phân tích các đặc trưng mà thuật toán sở hữu, biết thế mạnh và khuyết điểm của thuật toán. Ngoài ra, thực hiện các thuật toán trên các bộ dữ liệu với các kích thước khác nhau từ nhỏ như 7x7 đến rất lớn như 20x20. Qua đó thấy được tính chính xác và tốc độ thực thi của mỗi thuật toán. Báo cáo thực hiện rõ ràng, liên mạch, minh họa thuật toán bởi các diagram góp phần tạo nên sử dễ hiểu cho người đọc.

3 Định nghĩa CNF

3.1 Tổng quan về bài toán

Hashiwokakero là một trò chơi giải đố với các quy tắc rõ rệt, đòi hỏi người chơi phải tuân theo và thỏa mãn các điều kiện đó để giành chiến thắng bằng cách giải mã được trò chơi. Trò chơi bắt đầu với các hòn đảo có đánh trọng số, tương ứng với số lượng cây cầu nối đến hòn đảo đó, người chơi sẽ cố gắng nối các hòn đảo bằng các cây cầu, với các điều kiện như không quá 2 câu nối giữa 2 đảo, các đảo phải được kết nối với nhau. Bởi tuân theo các điều kiện cụ thể, thế nên ta có thể tiếp cận giải quyết bài toán với các mệnh đề logic thông qua việc trừu tượng hóa các điều kiện thành mệnh đề.

3.2 Quy ước và định nghĩa mệnh đề

Mô hình hóa bài toán khi ta biết được các sự vật xuất hiện trong thế giới Hashiwokakero đơn thuần là các hòn đảo và các cây cầu, nên ta sẽ tập trung khai thác mệnh đề từ các sự vật đó.

Ban đầu, bản đồ của chúng ta chỉ các các hòn đảo, và sau khi thêm lần lượt các cây cầu, chúng ta sẽ giải quyết được trò chơi, vậy nên, ta có thể thấy việc tồn tại cây cầu ở một vị trí nào đó sẽ ảnh hưởng đến kết quả, nên ta có thể dùng sự xuất hiện của cây cầu ở vị trí cụ thể là một mệnh đề.

Trừu tượng hóa bản đồ là một ma trận 2 chiều, với các hòn đảo có một tọa độ xác định, với số cột và số dòng bắt đầu từ 0. Ngoài tọa độ ra, các hòn đảo còn có trọng số tương ứng với số cây cầu nối đến đảo. Gọi một đảo được lưu thông tin trong một tuple dạng (x, y, w) với x, y lần lượt là vị trí dòng và cột, w là trọng số của hòn đảo.

Dựa vào cách định nghĩa phía trên, ta sẽ có một cầu cầu nối giữa 2 đảo A = (xA, yA, wA) và B = (xB, yB, wB) được ký hiệu là:

$$(A, B) = ((x_A, y_A, w_A); (x_B, y_B, w_B))$$

Ngoài ra, để thuận tiện trong việc biểu thị số cầu nối giữa 2 đảo, sẽ thêm một hệ số phụ k vào ký hiệu. Vậy khi có cầu giữa 2 đảo A và B sẽ là ((xA, yA, wA),(xB, yB, wB)), có k cầu giữa 2 đảo A và B sẽ được biểu thị là:

$$((A, B), k) = (((x_A, y_A, w_A); (x_B, y_B, w_B)), k)$$

Như vậy, việc định nghĩa các mệnh đề đủ để sử dụng trong việc giải quyết bài toán đã hoàn tất, đến với các điều kiện mà các mệnh đề phải tuân theo.

3.3 Quy Tắc

Trong Hashiwokakero, có các quy tắc mà người chơi phải tuân thủ, và dựa vào các quy tắc đó, ta sẽ xây dựng các điều kiện để từ đó giải quyết bài toán.

Điều kiện đầu tiên là số cầu nối giữa 2 đảo được phép tối thiểu là 0 và tối đa là 2. Với điều kiện này, ta sẽ sử dụng hệ số k để biểu thị số cầu nối giữa 2 đảo. Ngoài ra, ta cũng có thể hình dung được rằng, nếu có cầu nối giữa 2 đảo, thì chỉ có thể là 1 cầu hoặc 2 cầu và việc có 1 cầu và 2 cầu không đồng thời xảy ra với nhau.

Gọi A, B lần lượt là mệnh đề về việc có 1 cầu, có 2 cầu nối giữa 2 đảo X và Y. Chúng ta sẽ có được các mệnh đề như sau:

$$(\neg A \vee \neg B)$$

Biểu thị rằng một trong hai sẽ không được tồn tại do cả hai không được đồng thời xảy ra Lưu ý rằng, nếu 2 đảo có 1 đảo có trọng số là 1, thì chúng ta không cần xem xét điều kiện này. Như vậy với điều kiện trên, ta kiểm soát được việc chỉ có tối đa 2 cầu tồn tại giữa 2 đảo.

Điều kiện tiếp theo là tổng số cầu nối đến đảo phải đúng bằng trọng số của đảo. Với việc thao tác trên các mệnh đề, sẽ rất khó khăn trong việc thực hiện phép cộng để biết rằng tổng số cầu có bằng số đảo không, thay vào đó, hướng tiếp cận sẽ là việc tìm ra tất cả các trường hợp để mà tổng số cầu nối đến được thỏa mãn.

Đầu tiên, chúng ta sẽ tìm tập hợp tất cả các cạnh từ các đảo khác có thể nối đến đảo đang xét, và với mỗi cạnh, ta sẽ lại có một hệ số k để biểu thị số cầu, k có thể là 1 hoặc 2. Ta sẽ tạo nên tập hợp cạnh đó, và cũng đồng thời xem xét rằng có trường hợp có 2 cầu không bởi lẽ nếu trọng số của một đảo là 1 thì chỉ có thể có trường hợp có 1 cầu nối giữa 2 đảo.

Khi đã có tập hợp các cây cầu khả thi của đảo đang xét, ta sẽ thực hiện tạo nên các tập hợp con, với mỗi tập hợp con là một trường hợp khả thi khi tổng số cầu của tập hợp đảo đó bằng trọng số của đảo.

Giả sử tập hợp có mệnh đề (các cầu) là C1, C2, . . . , Cn. Thì các mệnh đề này phải cùng xảy ra để thỏa mãn, nói cách khác là C1 và C2 và . . . và Cn là đúng. Và giả sử ta có n tập hợp để thỏa mãn điều kiện là P1, P2, . . . , Pn, thì chỉ cần 1 trong các tập hợp này xảy ra, nghĩa là P1 hoặc P2 hoặc . . . hoặc Pn. Và tất nhiên, các tập hợp này cùng không được đồng thời xảy ra, thế nhưng việc tạo nên mệnh đề cho điều kiện nếu 1 tập hợp này xảy ra, thì các mệnh đề khác không được xảy ra khá phức tạp, nên ta xem xét tạo nên tập hợp các cầu sao cho tổng số các cầu này lớn hơn trọng số của đảo. Và ta sẽ đi phủ định từng tập hợp này. Giả sử các mệnh đề (các cầu) trong tập hợp là Y1, Y2, . . . , Yn thì ta sẽ có điều kiện là không Y1 hoặc không Y2 hoặc . . . hoặc không Yn. Và tương tự như thế, ta sẽ xây dựng điều kiện cho tất cả các đỉnh trong bản đồ.

Điều kiện thứ 3 chính là các cầu không được cắt nhau.Với điều kiện này, giả sử A, B là mệnh đề cho việc có cầu giữa 2 đảo bất kì, và 2 cầu này cắt nhau. Vậy có nghĩa là A và B không được đồng thời xảy ra, nên ta sẽ có điều kiện là:

$$(\neg A \vee \neg B)$$

Ngoài ra, có một vấn đề xảy ra là A, B chỉ biểu thị việc có cầu, và nếu biểu thị thêm số cầu, thì 1 điều kiện không cắt nhau cho một cặp cầu sẽ sinh ra 4 điều kiện (vì có thể có 1 cầu và 2 cầu). Vậy nên ta sẽ sinh ra thêm một điều kiện nữa, giả sử A, A1, A2 là mệnh đề có cầu, 1 cầu, 2 cầu giữa 2 đảo, ta sẽ có điều kiện A tương đương với A1 hoặc A2, biểu thị cho việc có cầu giữa 2 đảo khi có 1 cầu hoặc 2 cầu giữa 2 đảo. Ta chuyển đổi điều kiện mệnh đề tương đương, ta sẽ có được (không A hoặc A1 hoặc A2) và (không A1 hoặc A) và (không A2 hoặc A).

$$(\neg A \vee \neg A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A) \wedge (\neg A_2 \vee A)$$

Cứ như vậy, ta tìm tất cả các cặp cầu cắt nhau và áp đặt điều kiện lên, từ đó kiểm soát được việc các cầu không cắt nhau.

Ngoài ra, còn một điều kiện là các đảo phải được kết nối với nhau, hay trong ngôn ngữ đồ thị gọi là liên thông, thì với điều kiện này, việc chuyển thành mệnh đề logic khó khăn, nên sẽ xét kết hợp điều kiện thi thực thi chung với thuật toán.

Như vậy, với 3 điều kiện trên, ta sẽ kết hợp bằng cách hội các điều kiện tạo nên điều kiện mệnh đề tổng thể. Sau đó chuyển sang bước tiếp theo là chuyển đổi các mệnh đề thành dạng hội chuẩn (Conjunctive Normal Form).

3.4 Chuyển đổi các điều kiện thành CNF

Để chuyển điều kiện tổng thành dạng CNF, tức có nghĩa là dạng các mệnh đề hợp hội với nhau, ta đã hoàn thành thành được bước đầu tiên khi các mệnh đề đang ở dạng hội vì ta kết hợp các điều kiện bằng cách hội lại.

Và bước tiếp theo, các mệnh đề hội lại với nhau thì các mệnh đề phải ở dạng là các literal hợp với nhau. Qua đó, tạo cho ta việc chuyển đổi các mệnh đề từ bất kì dạng nào dạng hợp với nhau.

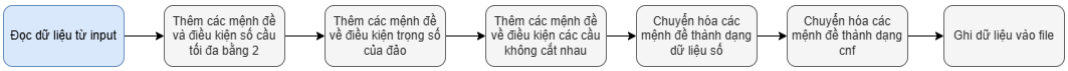
Thông thường ta sẽ gặp các trường hợp sau:

- Nếu mệnh đề được xét là ở dạng CNF, ta không cần thao tác gì, có thể bỏ đóng ngoặc của mệnh đề để khiến các phần mệnh đề bên trong trở thành mệnh đề của mệnh đề tổng.
- Nếu mệnh đề ở dạng là mệnh đề hợp với nhau, ta cũng không cần thao tác gì thêm.

- Nếu mệnh đề ở dạng là các mệnh đề hội với nhau, ta cũng bỏ đóng ngoặc mệnh đề để khiến các phần tử mệnh đề trở thành mệnh đề của mệnh đề tổng.
- Nếu mệnh đề xuất hiện ở dạng DNF (Disjunctive Normal Form), ta thực hiện thao tác phân phối các mệnh đề vào bên trong, cho đến khi có dạng CNF thì dừng.

Sau khi hoàn tất việc chuyển đổi sang CNF, thực hiện thao tác xét những điều kiện lặp để loại bỏ.

3.5 Diagram



Hình 1: Thực thi thuật toán

Chuyển hóa CNF để phục vụ các thuật toán. Khi đã có được mệnh tổng dưới dạng CNF, thực hiện thao tác tách thành các mệnh đề con và để vào trong một tập tin, như vậy, mỗi dòng của tập tin sẽ là một mệnh đề hợp.

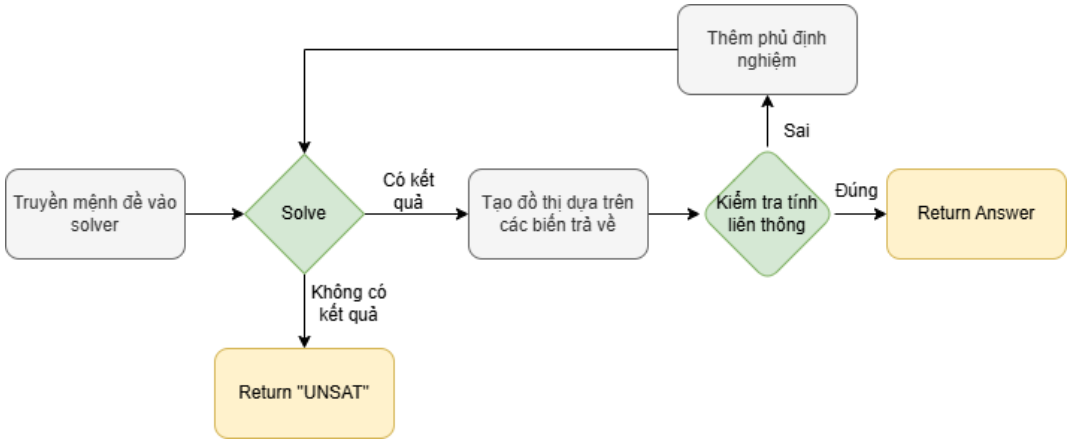
4 Áp dụng thư viện PySAT để giải CNF

Sau khi hoàn tất việc tạo ra các điều kiện CNF, ta thực nghiệm các thư viện có sẵn hỗ trợ việc giải các bài toán SAT (Boolean Satisfiability Problem).

Nhóm chúng em sử dụng thư viện PySAT phiên bản 1.8.dev16 và sử dụng SAT Solvers Glucose 3 để hỗ trợ trong việc tìm ra các nghiệm thỏa mãn CNF.

Bắt đầu bằng việc tải các điều kiện CNF đã được tạo ra khi nhập dữ liệu ma trận các đảo với trọng số vào, tiếp theo đưa các điều kiện đó vào SAT Solver và lấy ra các bộ nghiệm thỏa mãn.

Do đã đề cập ở trên việc kiểm tra tính liên thông sẽ được thực hiện khi có một bộ giá trị thỏa, kiểm tra bộ giá trị đó có thỏa tính liên thông không bằng cách sử dụng Disjoint Set Union - DSU để kiểm tra điều đó.



Hình 2: Thực thi thuật toán

Việc áp dụng thư viện PySAT để giải quyết CNF đặt tiền đề cho việc giúp chúng em thực thi các thuật toán sau một các dễ dàng hơn vì đã có được mô hình tổng quát để giải quyết bài toán.

5 Phân tích thuật toán

5.1 Thuật toán Brute Force

5.1.1 Định nghĩa

Brute force (tìm kiếm vét cạn) là một phương pháp đơn giản nhưng mạnh mẽ để tìm lời giải cho nhiều bài toán khác nhau.

Ý tưởng chính của phương pháp brute force là thử tất cả các tổ hợp giá trị có thể của các biến trong bài toán để kiểm tra xem tổ hợp nào thỏa mãn điều kiện cho trước. Đây là phương pháp dễ triển khai nhưng lại không hiệu quả đối với bài toán có số lượng biến lớn do số lượng tổ hợp tăng theo cấp số nhân.

5.1.2 Thực thi

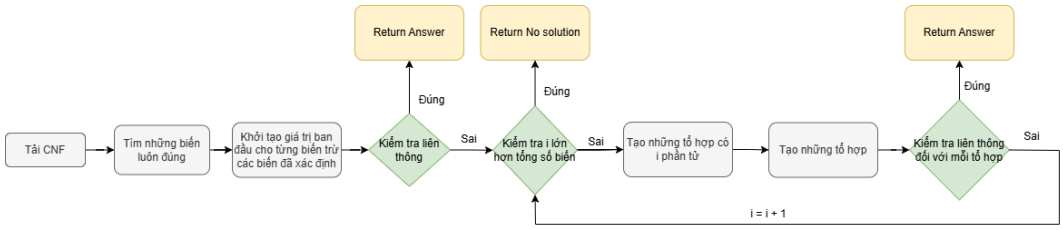
Brute force giải quyết bài toán bằng cách thử tất cả các cách gán giá trị có thể cho các biến, kiểm tra từng trường hợp để xác định xem có thỏa mãn công thức CNF hay không. Khởi tạo cho tất cả các biến là True. với mỗi giá trị trong tổ hợp ta đổi các biến trong tổ hợp sang False và kiểm tra.

Ví dụ, giả sử bài toán có 3 biến , chúng ta thử lần lượt các tổ hợp sau: (1, 1, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 0, 0), (0, 1, 0), (0, 0, 1), (0, 0, 0). Mỗi tổ hợp cần được đánh giá dựa trên từng mệnh đề của CNF để xác định xem nó có thỏa mãn công thức hay không. Độ phức tạp: Do mỗi biến có hai trạng thái (True/False), với n biến ta có tổng cộng tổ hợp cần kiểm tra. Do đó, độ phức tạp của thuật toán này là $O(2^n$

Cải tiến Brute Force:

- Xác định giá trị cố định: Với các điều kiện đơn giản (ví dụ: trong bài toán logic, một biến phải luôn là True), ta gán ngay giá trị cho biến đó. Clause có 2 biến và 1 biến đã được xác định. Nếu biến đã xác định đó là False thì biến còn lại phải là True.
- Loại bỏ dần biến: Nếu một biến đã được xác định (True hoặc False), ta loại nó khỏi quá trình duyệt, từ đó giảm số tổ hợp cần kiểm tra.

5.1.3 Diagram



Hình 3: Thực thi thuật toán

5.1.4 Ưu điểm

- Phương pháp dễ triển khai, không yêu cầu kỹ thuật phức tạp, chỉ cần thử tất cả các khả năng có thể. Đảm bảo tìm được lời giải nếu tồn tại, vì ta kiểm tra toàn bộ không gian tìm kiếm.
- Ngoài ra, brute force không đòi hỏi hiểu biết chuyên sâu về bài toán, do không cần sử dụng các phương pháp tối ưu hóa hay phân tích trước.

5.1.5 Nhược điểm

- Hiệu suất kém do phải duyệt qua tất cả tổ hợp với n càng lớn thời gian chạy càng tăng lên theo cấp số nhân, khiến thuật toán nhanh chóng trở nên không khả thi.
- Phương pháp này tiêu tốn nhiều tài nguyên, yêu cầu bộ nhớ lớn và sức mạnh tính toán đáng kể.

5.2 Thuật toán Backtracking

5.2.1 Định nghĩa

Trong quá trình tìm hiểu về thuật toán backtracking ứng dụng vào để giải quyết bài toán SAT với dữ liệu CNF đã có, có nhiều loại thuật toán thuộc dạng đệ quy quay lui, từ cơ bản đơn giản như việc thử sai với từng literal, cho đến các thuật toán nâng cao hơn khi kết hợp thêm các yếu tố khác, chúng em quyết định thực thi thuật toán DPLL (Davis-Putnam-Logemann-Loveland) bởi tính mới mẻ và thường được vận dụng cho việc giải quyết bài toán SAT.

Thuật toán Davis-Putnam-Logemann-Loveland, viết tắt là DPLL, là một thuật toán dạng backtracking, thường được vận dụng nhiều trong việc giải quyết các bài toán SAT bởi tính hiệu quả ổn định và đơn giản trong việc thực thi, với ý tưởng kết hợp việc thử và sai với việc rút gọn CNF trước khi đưa vào đệ quy.

5.2.2 Thực thi

Trước khi thực hiện thuật toán, ta sẽ cài đặt 2 hàm bổ trợ là hàm lan truyền đơn vị (**Unit Propagation**) và loại bỏ biến thuần (**Pure Literal Elimination**).

Unit Propagation ý tưởng chính là ta xét các mệnh đề chỉ có một biến duy nhất, và khi mệnh đề đó phải đúng đồng nghĩa rằng biến đó phải đúng (đúng so với biến, nếu biến đó ở dạng phủ định thì biến đó phải sai). Tạm gọi biến có giá trị cố định là a, duyệt qua tất cả mệnh đề, nếu mệnh đề có biến a thì không cần xem xét mệnh đề đó nữa vì mệnh đề đó đã đúng, ngược lại, nếu chứa phủ định của biến a, thì ta loại bỏ phủ định của biến a để xét các biến còn lại. Điều này giúp ta có thể rút gọn tổng số mệnh đề xuống và có được các biến có giá trị cố định, giúp lần tiếp theo đệ quy sẽ với số lượng mệnh đề ít hơn.

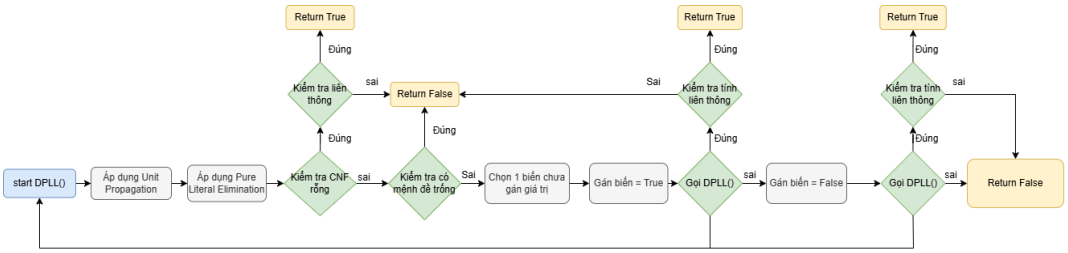
Pure Literal Elimination ý tưởng là ta xét kiểm tra xem có biến nào chỉ xuất hiện một dạng duy nhất, hay nói cách khác nếu biến a tồn tại thì không tồn tại phủ định của biến a, ta xem biến a là biến thuần và gán giá trị tương ứng sao cho biến a đúng. Điều này giúp ta có thể giảm được số lượng mệnh đề cần xét bởi khi gán giá trị cho biến thuần sẽ không gây xung đột với các mệnh đề khác do không tồn tại dạng còn của biến.

Bắt đầu thuật toán, ta sẽ thực hiện thao tác rút gọn với 2 hàm trên, sau đó thực hiện một số kiểm tra, với việc kiểm tra xem rằng số mệnh còn lại có bằng 0 không, nếu có, điều này đồng nghĩa với việc các giá trị literal đã thỏa mãn bài toán. Tiếp theo, kiểm tra xem rằng có mệnh đề nào trống không, nếu điều này xảy ra, tức có nghĩa đã có sự xung đột trong việc gán giá trị cho các biến.

Tiếp đến ta tiến hành chọn một biến ngẫu nhiên và gán giá trị cho biến đó lần lượt là True và False, tạo nên bộ CNF và bộ giá trị mới, đồng thời gọi đệ quy, khi có kết quả trả về, nếu nếu trả về True tức có nghĩa là gán đúng và thỏa mãn bài toán với bộ giá trị các biến, ta tiến hành kiểm tra xem liệu rằng với bộ giá trị đó có thỏa mãn điều kiện liên thông của các đảo không, nếu có, ta hoàn thành bài toán về trả về kết quả, nếu sau 2 lần gán mà vẫn không có kết quả True, ta trả về False.

Giải thích thêm về việc tồn tại mệnh đề trống, khi ta thử gán một giá trị cho một biến a bất kì, đồng thời tạo nên bộ CNF mới sẽ loại bỏ các mệnh đề có chứa biến được gán a do mệnh đề đã đúng, và loại bỏ phủ định của biến a trong các mệnh đề, nếu mệnh đề chỉ còn có phủ định của biến a, điều này sẽ tạo nên một mệnh đề trống, lúc này, chính mệnh đề trống là dấu hiệu cho việc gán biến đã sai nói cách khác là không còn cách gán biến nào để thỏa mãn mệnh đề trống đó.

5.2.3 Diagram



Hình 4: Thực thi thuật toán

5.2.4 Ưu điểm

Với sự hỗ trợ từ 2 hàm rút gọn, thuật toán thể hiện mạnh mẽ hơn các thuật toán backtracking thông thường khác với độ phức tạp theo hàm lũy thừa. Thực hiện đệ quy quay lui khiến cho thuật toán dễ dàng trong việc cài đặt bởi cấu trúc ngắn gọn, xúc tích.

5.2.5 Nhược điểm

Do vẫn mang thiên hướng xử lý trên các mệnh đề, thuật toán vẫn nằm trong nhóm các thuật toán thử sai, không tối ưu bằng các thuật toán có tính tự học như CDCL

Thuật toán thực hiện thử sai nên đôi khi sẽ gây nên những tình huống gọi đệ quy nhiều lần dẫn đến tràn vùng nhớ, thiếu đi việc lựa chọn dựa trên sự tối ưu.

5.3 Thuật toán A*

5.3.1 Định nghĩa

Thuật toán A* (Astar) là một thuật toán tìm kiếm heuristic thường được dùng để tìm đường hoặc giải quyết các bài toán tối ưu trong việc tìm kiếm trong không gian lớn. Khi áp dụng vào bài toán SAT (Boolean Satisfiability Problem), mục tiêu của thuật toán là tìm ra một cách gán giá trị True hoặc False cho các biến (literal) sao cho mọi mệnh đề (clauses) trong công thức đều được thỏa mãn. Trong bài toán này, thuật toán A* giúp chúng ta gán các biến theo cách tối ưu nhất để tìm ra kết quả nhanh chóng.

5.3.2 Thực thi

Đọc file và chuẩn bị dữ liệu

- Đầu tiên, chương trình cần phải đọc file điều kiện (chứa các mệnh đề) để biết được phải xử lý những gì. `parse_clauses()` đọc từng dòng trong file điều kiện và chuyển mỗi dòng thành một mệnh đề, tức là một list các literal.
- `load_variable_mapping()` dùng để tạo ra một bảng ánh xạ giữa số nguyên và các biến cụ thể. Ví dụ như số 1 tương ứng với A, số 2 tương ứng với B.
- Đồng thời, chương trình cũng gọi `read_matrix()` và `find_islands()` để lấy thông tin về các nhóm kết nối (islands). Những hàm này được dùng sau khi tìm ra lời giải để kiểm tra xem các nhóm có kết nối với nhau đúng cách không.

Khởi tạo và tiền xử lý

- Sau khi đọc dữ liệu xong, chương trình tạo ra một tập hợp chứa tất cả các biến được tạo ra (all_variables) từ các mệnh đề.
- Ngoài ra, nó cũng tạo ra:
  - `priority_queue` (Hàng đợi ưu tiên): Danh sách chờ, được sắp xếp theo thứ tự sao cho những trạng thái có điểm f nhỏ nhất được xử lý trước.
  - `closed_set` (Tập đóng): Nơi lưu trữ những trạng thái đã duyệt qua rồi để tránh duyệt lại.
  - `unsatisfied_cache`: Ghi nhớ số lượng mệnh đề chưa thỏa mãn cho một trạng thái nhất định. Giúp tối ưu tốc độ vì không phải tính toán lại nhiều lần.
  - Khởi đầu thì phép gán trong `initial_assignment` là rỗng , vì ta chưa quyết định biến nào True, biến nào False.

Điểm cải tiến

- Áp dụng kỹ thuật DPLL:
  - `unit_propagation()`: Khi một mệnh đề chỉ còn một literal chưa được gán, mình sẽ ép buộc nó để mệnh đề đó thỏa mãn. Ví dụ: (1) thì gán 1 = True, (-2) thì gán 2 = False.
  - `pure_literal_elimination()`: Nếu một biến chỉ xuất hiện dưới một dạng duy nhất (True hoặc False) trong tất cả mệnh đề, mình gán nó để thỏa mãn tất cả các mệnh đề đó. Ví dụ: 3 xuất hiện dưới dạng 3 (không phải -3), thì gán 3 = True.
- Phân tích xung đột:
  - `conflict_analysis`:Khi tìm thấy một trạng thái không hợp lệ, thực hiện phân tích xung đột bằng cách thêm mệnh đề học được (learned clause) vào công thức.

Khởi tạo hàng đợi ưu tiên (priority queue) và hàm *Heuristic*:

- Dùng hàng đợi ưu tiên (priority queue) để quản lý các trạng thái theo giá trị hàm $f = g + h$. Trạng thái có giá trị f nhỏ hơn sẽ được ưu tiên.
- Trong công thức $f = g + h$, ta có:
  - g: Chi phí đã thực hiện (cost so far). Đây là số lượng phép gán đã được thực hiện từ trạng thái ban đầu đến trạng thái hiện tại.
  - h: Hàm heuristic. Hàm heuristic được thiết kế để đánh giá mức độ chưa thỏa mãn của công thức hiện tại. Có dạng như sau:

$$\boxed{h = \texttt{remaining\_vars} + 2 * \texttt{unsatisfied}}$$

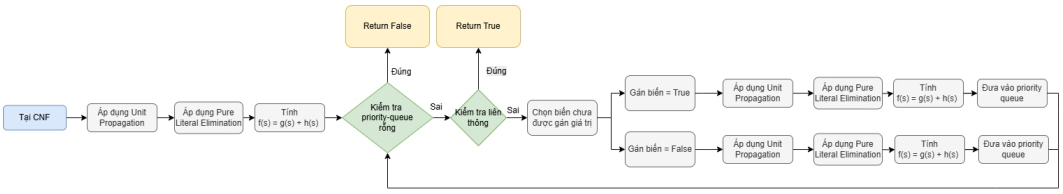
- Trong đó:
  - `unsatisfied` = `uncount_unsatisfied_clauses()`: Số mệnh đề chưa được thỏa mãn. Ta nhân 2 để tăng mức độ ưu tiên cho việc thỏa mãn mệnh đề thay vì chỉ đơn giản là gán hết các biến.
  - `remaining_vars`: Số lượng biến chưa được gán giá trị.

Vòng lặp tìm kiếm A*. *Thực hiện vòng lặp theo quá trình sau:*

- Đầu tiên lấy ra state (trạng thái) tốt nhất từ hàng đợi. State ở đây bao gồm `assignment` (Dict gồm các key - value là các literal và giá trị True/False), `f_score`, `h_score` và `g_score`.
- Kiểm tra trạng thái hiện tại: Nếu tất cả mệnh đề đều thỏa mãn và tất cả biến đều được gán giá trị, thì kết thúc.

- Kiểm tra trạng thái đã duyệt: Nếu trạng thái hiện tại đã có trong `closed_set`, bỏ qua không duyệt lại.
- Chọn biến để gán tiếp theo: hàm `next_literal()` chọn biến xuất hiện nhiều nhất trong các mệnh đề chưa thỏa mãn. Duyệt qua hai nhánh (True và False): Gán thử giá trị rồi áp dụng các hàm cải tiến (`unit_propagation()` và `pure_literal_elimination()`).
- Nếu phép gán không hợp lệ (`verify_assignment_validity()` trả về False), thì gọi `conflict_analysis()` để học thêm mệnh đề mới, tránh lỗi tương tự xảy ra.
- Tính toán `f_score` rồi thêm trạng thái mới vào `priority_queue` nếu chưa từng duyệt qua (không có trong `closed_set`).
- Đã hoàn thành một vòng lặp, quay lại xét tiếp đến khi ta tìm ra kết quả cuối cùng.

5.3.3 Diagram



Hình 5: Thực thi thuật toán

5.3.4 Ưu điểm

Việc kết hợp các kỹ thuật heuristic và DPLL giúp thuật toán tìm kiếm hiệu quả hơn trong không gian tìm kiếm lớn. Thuật toán được cái đặt tương đối ổn định, có thể giải được các bài toán lên đến 500 literals. Dễ dàng mở rộng thêm các kỹ thuật khác từ DPLL hoặc CDCL để tăng hiệu suất.

5.3.5 Nhược điểm

Do sử dụng bộ nhớ đệm và hàng đợi ưu tiên, chi phí bộ nhớ có thể tăng cao khi xử lý các bài toán lớn. Hàm heuristic chưa thật sự hiệu quả để tìm ra giải pháp được nhanh nhất. Kỹ thuật phân tích xung đột trong cài đặt hiện tại còn đơn giản, chưa đạt mức tối ưu như trong CDCL.

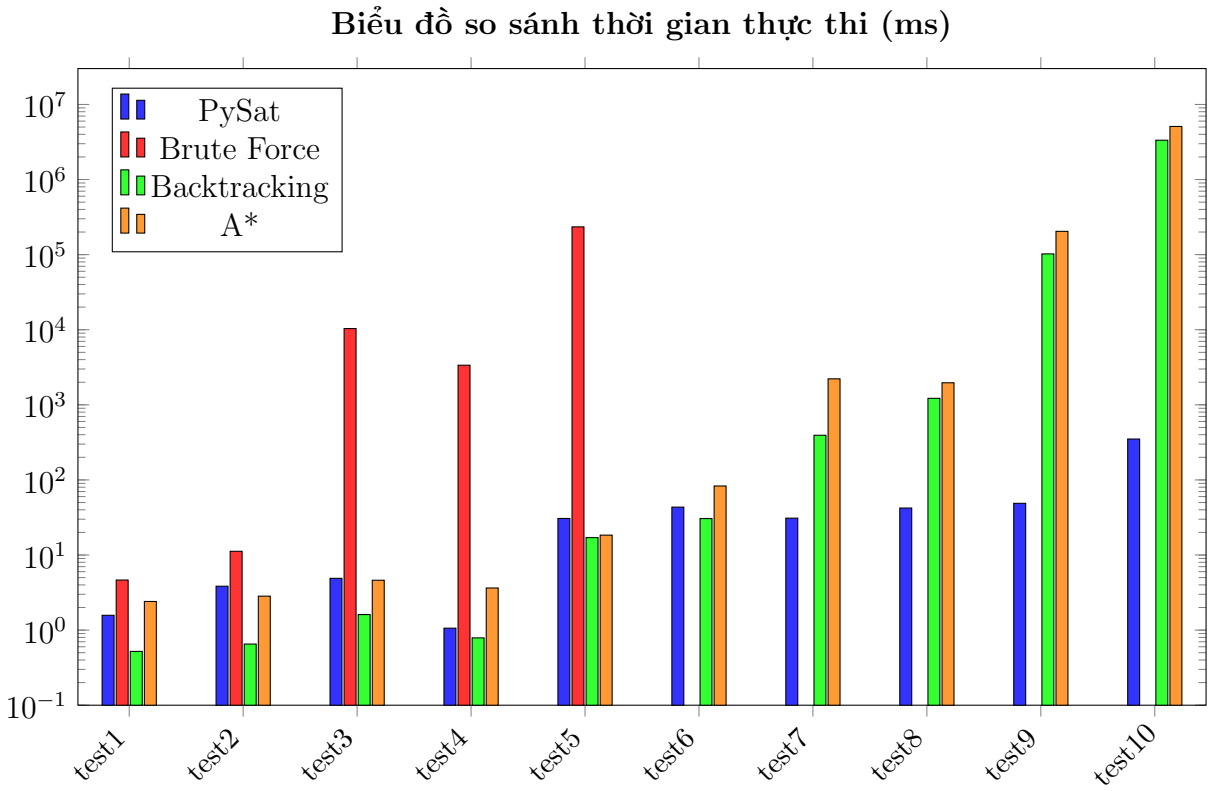
6 Kết quả thực nghiệm

6.1 Kết quả thuật toán

Test	PySat	Brute Force	Backtracking	A*
Test 1	1.5754	4.6439	0.5209	2.4068
Test 2	3.8413	11.2126	0.6521	2.832
Test 3	4.8942	10368.872	1.6104	4.6149
Test 4	1.0617	3368.6945	0.7881	3.6387
Test 5	30.678511	234332.195	17.0446	18.3995
Test 6	43.437481		30.563	83.0433
Test 7	30.953884		393.3704	2219.0608
Test 8	42.235851		1220.7102	1966.9417
Test 9	48.827887		102249.34	204292.285
Test 10	350.409985		3340393.191	5093422.244

Bảng 3: Bảng kết quả thời gian thực nghiệm (ms)

6.2 Biểu đồ



6.3 Phân tích

Với dữ liệu input đầu vào là các ma trận có kích thước khác nhau (7×7, 9×9, 11×11, 13×13, 17×17, 20×20) qua đó dựa trên kết quả thực nghiệm với các input trên, ta nhận thấy sự khác biệt đáng kể về thời gian thực thi giữa các thuật toán:

Với PySat có thời gian xử lý các test thấp nhất chỉ trong khoảng 22 ms đến 350 ms, thời gian ổn định cho từng test.

Với brute force hầu hết các test đều sẽ sử dụng rất nhiều thời gian để thực thi thậm chí đối với những test lớn phức tạp thuật toán brute force không thể đưa ra được kết quả. Qua đó thấy được độ phực tạp của brute force rất cao vì thuật toán đòi hỏi duyệt qua tất cả các trường hợp để tìm kết kết.

Backtracking thời gian thực thi đối với các test 1 đến test 8 có thời gian thực thi khá lý tưởng. Tuy nhiên, đối với các bài toán phức tạp hơn, số lần gọi đệ quy tăng đáng kể, dẫn đến thời gian thực thi kéo dài (có thể lên đến 3340 giây với test 10).

A* thuật toán áp dụng heuristic để xử lý cnf có thời gian xử lý tương tự backtracking với các test nhỏ, nhưng với các test lớn hơn thì lại có thời gian cao hơn nhiều so với backtracking điều này cho thấy thuật toán A* thực sự không phải là giải pháp tối ưu cho bài toán cnf.

Nhận xét: Đối với bài toán cnf các thuật toán như backtracking và A* sẽ có thể áp dụng cho những trường hợp test nhỏ và đơn giản. Đối với brute force thuật toán này về mặt lý thuyết là vẫn khả thi để giải quyết bài toán cnf nhưng đòi hỏi thời gian quá lớn nến trên thực tế sẽ không nên áp dụng. Vậy đối Với các test case lớn phức tạp thì Pysat là lựa chọn tối ưu và hữu ích nhất trong bài toán cnf vì thời gian giải quyết và đưa ra kết quả khá nhanh và hiệu quả.

7 Tham khảo

- **PySAT**
- **Hashi puzzles** | by @bridges-solver on Github
- **DPLL algorithm** | on Wikipedia