TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

---□□□---

CƠ SỞ TRÍ TUỆ NHÂN TẠO
NĂM HỌC 2024 – 2025
HỌC KỲ II

BÁO CÁO LAB CÁ NHÂN: GEM HUNTER GAME

Giảng viên hướng dẫn: Nguyễn Ngọc Đức
Nguyễn Trần Duy Minh
Nguyễn Thanh Tình

Sinh viên thực hiện: Nguyễn Thị Kiều Tiên

TPHCM, 16/05/2025

MỤC LỤC

I. GIỚI THIỆU

Dựa trên yêu cầu biểu diễn trò chơi Gem Hunter (thợ săn đá quý) bằng cách vận dụng Conjunctive Normal Form (CNF) xác định vị trí trap (bẫy) và gem (đá quý) trên lưới hai chiều, báo cáo này trình bày lập luận và các bước xây dựng CNF, và ứng dụng thư viện pysat để giải quyết bài toán. Đồng thời, bên cạnh việc sử dụng thư viện, chương trình giải CNF bằng các thuật toán Brute Force (vét cạn) và Backtracking (đệ quy quay lui) cũng sẽ được cài đặt để đặt lên bàn cân, từ đó rút ra các kết luận sơ bộ.

II. GIẢI PHÁP XÂY DỰNG CNF

A. Bài toán

Trong bài toán Gem Hunter, nhiệm vụ cần phải thực hiện chính là xác định vị trí của bẫy và đá quý trên lưới hai chiều, dựa trên các trọng số được đánh dấu trên lưới.

Quy ước để giải bài toán như sau:

- Lưới nhận vào sẽ gồm các con số từ 1 - 8 thể hiện số bẫy xuất hiện xung quanh vị trí đó, và các ký tự '_' đại diện cho các "ẩn số" người chơi cần phải khám phá.
- Mỗi ký tự '_' có thể là một bẫy hoặc một đá quý.
- Bẫy được thể hiện bởi ký tự 'T' viết tắt của Trap và đá quý được thể hiện bởi ký tự 'G' viết tắt của Gem
- Kết quả của bài toán sẽ là lưới với các vị trí '_' đều đã được xác định bởi T hoặc G sao cho phù hợp với các trọng số ban đầu.

Bằng cách sử dụng dữ liệu có được từ các trọng số trên lưới, một CNF có thể được sinh ra bằng cách xây dựng các ràng buộc luận lý và tìm ra bộ dữ liệu về T, G thỏa mãn (nếu tồn tại).

B. Ý tưởng:

Để xây dựng CNF, cần phải biết rằng CNF là một dạng chuẩn của biểu thức logic. Biểu thức logic CNF là một biểu thức gồm nhiều mệnh đề (clause) được kết hợp với nhau bởi phép AND, bản thân mỗi mệnh đề này lại là một phép OR của giá trị của các biến logic (literal).

Ý tưởng xây dựng CNF trên bài toán này dựa trên ràng buộc số lượng T và G xung quanh một vị trí có trọng số. Nói rõ hơn, khi xung quanh một vị trí có trọng số bằng n, có k ô trống, thì trong số k ô trống đó bắt buộc phải có đúng n bẫy và đúng m đá quý, sao cho tổng m + n luôn bằng k. Chỉ cần tồn tại bộ giá trị thỏa mãn ràng buộc này cho tất cả các vị trí thì tìm được kết quả cho bài toán.

Giả sử tại vị trí (x, y) trọng số n có k ô trống xung quanh (k = n), khi này m = 0 và tất cả các ô trống xung quanh vị trí (x, y) bắt buộc đều phải là bẫy. Còn trong trường hợp (k > n), vậy trong số các ô trống xung quanh (x, y) bắt buộc phải có đúng n bẫy, và m đá quý (vị trí chưa xác định). Khi đó, chọn ra n + 1 ô trống bất kỳ, phải có ít nhất một trong số đó là đá quý, tương tự, chọn ra m + 1 ô trống bất kỳ, phải có ít nhất một trong số đó là bẫy.

Ràng buộc ta cần có là "bất kỳ" tổ hợp nào trong số các ô trống cũng đều thỏa mãn điều kiện, tức là có C_k^{n+1} tổ hợp chứa ít nhất một đá quý, và C_k^{k-n+1} tổ hợp chứa ít nhất một bẫy (m = k - n). Từ đây ta có được các phép tuyển (OR) là mỗi tổ hợp được sinh ra (chỉ cần một giá trị đúng là cho ra giá trị đúng), và một CNF được tạo thành từ phép hội (AND) của tất cả phép tuyển trên (tất cả tổ hợp thỏa mãn mới cho ra giá trị đúng.

Nếu xem bẫy là các giá trị nguyên bản và đá quý là phủ định của nó thì CNF được tạo thành có dạng như sau:

$$\bigwedge_{j=1}^{C(k,n+1)} \left(\bigvee_{i=1}^{n+1} \neg x_{ji} \right) \wedge \bigwedge_{j=1}^{C(k,k-n+1)} \left(\bigvee_{i=1}^{k-n+1} x_{ji} \right)$$

C. Cài đặt

Hàm tạo CNF có thể được cài đặt như sau:

- Duyệt qua các vị trí trong lưới
- Nếu gặp vị trí có trọng số, tính số ô trống xung quanh vị trí
- Nếu số ô trống đúng bằng trọng số, lần lượt thêm vào CNF các đơn mệnh đề khẳng định ô trống là bẫy.
- Ngược lại, sử dụng `combinations` tạo ra các tổ hợp bao gồm tổ hợp chập n + 1 của k các phủ định và tổ hợp chập k - n + 1 của k các khẳng định vào thêm CNF

Theo quy tắc của thư viện pysat, số dương được coi là khẳng định, số âm được coi là phủ định. Để đơn giản hóa, các vị trí trên mảng hai chiều (lưới) sẽ được mã hóa thành dạng mảng một chiều có thứ tự.

Tức là thay vì:

(0,0) (0,1) (0,2)

(1,0) (1,1) (1,2)

(2,0) (2,1) (2,2)

Thì vị trí trên lưới được mã hóa thành:

1 2 3

4 5 6

7 8 9

=> 1 2 3 4 5 6 7 8 9

Với số dương đại diện cho bẫy và số âm đại diện cho đá quý. Sau khi thu được nghiệm từ việc giải CNF, ô trống tương ứng với số dương sẽ là bẫy, còn ô trống tương ứng với số âm sẽ là đá quý.

III. GIẢI CNF

A. Sử dụng thư viện pysat

1. Glucose3 solver:

Thuật toán của pysat dùng để giải bài toán tìm nghiệm cho CNF (cụ thể là Glucose3) hoạt động dựa trên nền tảng thuật toán Conflict Driven Clause Learning (CDCL) gồm các bước chính:

- **Lựa chọn biến quyết định:** Khi chưa có giá trị nào được xác định trong CNF, thuật toán sẽ tiến hành chọn ra biến triển vọng, có ảnh hưởng quyết định.
- **Lan truyền đơn vị:** Khi xác định giá trị một biến logic, các mệnh đề trong CNF được lan truyền ảnh hưởng từ biến được xác định ấy.

Ví dụ: Ta có CNF gồm (A ∨ B), (¬A ∨ C), khi chọn A khẳng định, vậy để mệnh đề (¬A ∨ C) đúng, C bắt buộc phải có giá trị khẳng định.

- **Phân tích xung đột:** Trong quá trình chọn biến và lan truyền, có thể tìm thấy các xung đột báo hiệu rằng hướng đi này không chính xác, để kịp thời từ bỏ hướng tìm kiếm này.

  Ví dụ: Ta có CNF gồm (A ∨ B), (¬A ∨ C), (¬B ∨ ¬C ∨ D), (¬D ∨ ¬A), khi chọn A khẳng định, ta lan truyền thu được C khẳng định, sau đó tiếp tục gán B khẳng định. Khi này, xung đột xuất hiện. Để mệnh đề (¬B ∨ ¬C ∨ D) cho giá trị đúng, D phải khẳng định, trong khi vế (¬D ∨ ¬A) cần phủ định D để nhận giá trị đúng ⇒ Hướng này không chính xác.

- **Học mệnh đề:** Khi tìm phát hiện xung đột trong hướng đi, trước khi lựa chọn hướng đi khác, một mệnh đề mới được thêm vào để bổ sung tri thức cho quá trình giải, tránh lặp lại sai phạm cũ.

  Ví dụ: Ta chọn A khẳng định, lan truyền được giá trị của C, tiếp tục chọn B khẳng định, thế nhưng điều này gây xung đột khi chọn giá trị của D ⇒ Một trong các lựa chọn trước đó không đúng, ta cần bổ sung một phép tuyển gồm phủ định của các giá trị đã chọn, chẳng hạn trong trường hợp này sẽ là (¬A ∨ ¬B)

- **Quay lui:** Sau khi học mệnh đề mới vì phát hiện xung đột, thuật toán lùi một bước và thay đổi lựa chọn so với trước đó.

  Ví dụ: Lựa chọn A khẳng định rồi B khẳng định gây ra mâu thuẫn, học được mệnh đề mới (¬A ∨ ¬B), tức là khi này B phải là phủ định.

Trong quá trình tìm lời giải, Glucose3 còn sử dụng chỉ số Literal Block Distance (LDB) để làm thang đánh giá "chất lượng" mệnh đề. LDB thể hiện số mức quyết định khác nhau của các literal trong mệnh đề.

Các biến chịu ảnh hưởng lan truyền từ biến được chọn nhận số mức quyết định ngang với biến được chọn.

Ví dụ, khi chọn A đầu tiên, A nhận số mức quyết định 1, kéo theo C mang cùng mức. Kế tiếp, B được chọn, do đó mang số mức quyết định 2. Mệnh đề được tạo ra bởi A và B sẽ có LDB = 2.

LDB càng thấp càng có giá trị cao, bởi nó giúp việc xác định giá trị của các biến trong mệnh đề tiết kiệm thời gian và công sức hơn. Bởi vì, nếu LDB = 2, khi xác định được giá trị được một biến, biến còn lại lập tức chịu ràng buộc mạnh mẽ, chứ không phải chia đều ràng buộc giữa nhiều biến.

- **Khởi động lại:** Chính vì có giá trị LDB, chương trình có thể tự đánh giá chất lượng hướng đi. Nếu LDB trung quá cao, có thể nhận thấy hướng đi này kém hiệu quả, nên thực hiện việc tạm dừng với tái khởi động với hướng đi mới với kỳ vọng mới.

Bên cạnh những điều trên, công tác tiền xử lý của Glucose3 thực sự hiệu quả, khiến không gian tìm kiếm giải pháp trên CNF được làm sạch đáng kể, giảm gánh nặng cho quá trình xử lý các mệnh đề.

2. Triển khai giải CNF với thư viện pysat

Việc ứng dụng thư viện pysat để giải CNF vô cùng đơn giản, chỉ việc khởi tạo solver và nạp các mệnh đề cần thiết vào, phần còn lại chỉ là đợi nhận kết quả.

Nếu tồn tại mô hình thỏa mãn những ràng buộc, ta có thể gọi `get_model()` từ solver để nhận về mô hình, sau đó gán các giá trị theo đúng kết quả vào lưới trò chơi. Tất cả bẫy và đá quý đã được tìm thấy.

B. Thuật toán Brute Force

1. Khái niệm vét cạn:

Yếu quyết của thuật toán này chỉ một: Nếu đáp án có tồn tại, chỉ cần thử qua tất cả trường hợp thì ta chắc chắn tìm ra nó.

Ứng dụng Brute Force giải CNF chỉ xoay quanh việc lần lượt thử từng mô hình có thể có cho các biến logic, khi kiểm tra thấy có một mô hình thỏa mãn tất cả ràng buộc, lập tức trả mô hình ấy về. Trong trường hợp bế tắc, khi đã kiểm tra tất cả mô hình mà không tìm được đáp án, Brute Force bấy giờ mới có thể trả lời rằng không có lời giải cho bài toán này.

2. Triển khai giải CNF với Brute Force

Để tìm ra giá trị T hoặc G cho các ô trống, ta có thể sử dụng một dãy bit để biểu diễn. Với bit thứ i bật lên 1, tức là ô trống thứ i là bẫy, và tất nhiên bit thứ i được biểu diễn bởi 0 là một đá quý.

Để duyệt qua toàn bộ các tổ hợp giá trị của n vị trí trống trên lưới, ta cần khởi tạo vòng lặp duyệt 2^n để kiểm thử lần lượt các mô hình và trả về mô hình thỏa mãn nếu tìm được.

C. Thuật toán Backtracking

1. Thuật toán DPLL:

Thay vì Backtracking thông thường, tại đây, thuật toán DPLL sẽ được sử dụng.

Theo đó, trước khi tiến hành thử giá trị và đệ quy quay lui, thuật toán DPLL có các bước xử lý giúp đơn giản hóa các mệnh đề của CNF:

- **Unit Propagation:** Nói một cách đơn giản, để các mệnh đề chỉ có duy nhất một literal (hay còn gọi là đơn mệnh đề) đúng, biến logic này bắt buộc phải mang giá trị đó ⇒ Để CNF thỏa mãn, trong mô hình tất yếu phải chứa literal này.
- **Pure Literal Elimination:** Khi tồn tại một biến mà nó có cùng dạng với nhau ở mọi mệnh đề (tức là cùng khẳng định hoặc cùng phủ định), hiển nhiên biến ấy cũng mang giá trị đó trong mô hình.

Việc xác định được giá trị các biến thông qua hai con đường trên, các mệnh đề được đơn giản hóa rất nhiều. Khi một biến được xác định, trong số các mệnh đề còn lại, mệnh đề nào mang cùng dạng với nó đều có thể lược bỏ, không cần xét (vì đã đúng), và trái dạng của nó trong các mệnh đề còn lại cũng mặc nhiên biến mất (chắc chắn sai).

2. Triển khai giải CNF với Backtracking

Cài đặt của thuật toán này trong chương trình có thể diễn giải như sau:

- Nếu không còn mệnh đề nào chưa được thỏa mãn trong CNF, mô hình phù hợp đã được tìm thấy, có thể trả về mô hình.
- Khi có bất cứ mệnh đề nào rỗng trong CNF, tức là một mệnh đề không thể thỏa mãn được, có thể xác nhận rằng không có mô hình nào thỏa mãn cho bài toán.
- Thực hiện tìm kiếm các mệnh đề đơn và xác định giá trị của nó trong mô hình rồi đơn giản hóa CNF.

  Trong bước này, nếu tìm thấy một mệnh đề đơn gây mâu thuẫn, ví dụ tìm thấy một mệnh đề đơn A khẳng định, tuy nhiên trong mô hình đã tồn tại A phủ định, điều này gây mâu thuẫn, có thể lập tức báo không có lời giải.

- Thực hiện tìm kiếm các biến chỉ tồn tại cùng một dạng, xác định giá trị của nó trong mô hình, đơn giản hóa CNF.
- Trong các biến chưa xác định giá trị, ta mở nhánh khẳng định, gán giá trị khẳng định cho biến và đẩy vào mô hình, gọi đệ quy để đi sâu vào nhánh này, trả về mô hình thỏa mãn nếu tìm được.
- Nếu nhánh trên không tìm được đáp án, ta gỡ giá trị khẳng định của biến ra khỏi mô hình, mở nhánh phủ định, gán giá trị phủ định cho biến và đẩy vào mô hình, gọi đệ quy để đi sâu vào nhánh này. Đến đây không còn giá trị khác để thử, trả về mô hình thỏa mãn nếu tìm được hoặc trả về None để thông báo không có lời giải.
- Nếu qua tất cả bước trên vẫn không tìm ra được mô hình, bài toán này không có lời giải.

IV. THỰC NGHIỆM VÀ PHÂN TÍCH

A. Thực nghiệm

Đơn vị thời gian tính bằng giây (s)

Kích thước ma trận	5x5	11x11	20x20
Thư viện pysat (Glucose3)	0.0001063347	0.0002176762	0.0008494854
Brute Force	0.0003049374	TIMEOUT	TIMEOUT
Backtracking (DPLL)	0.0000982285	0.0033876896	0.0855123997

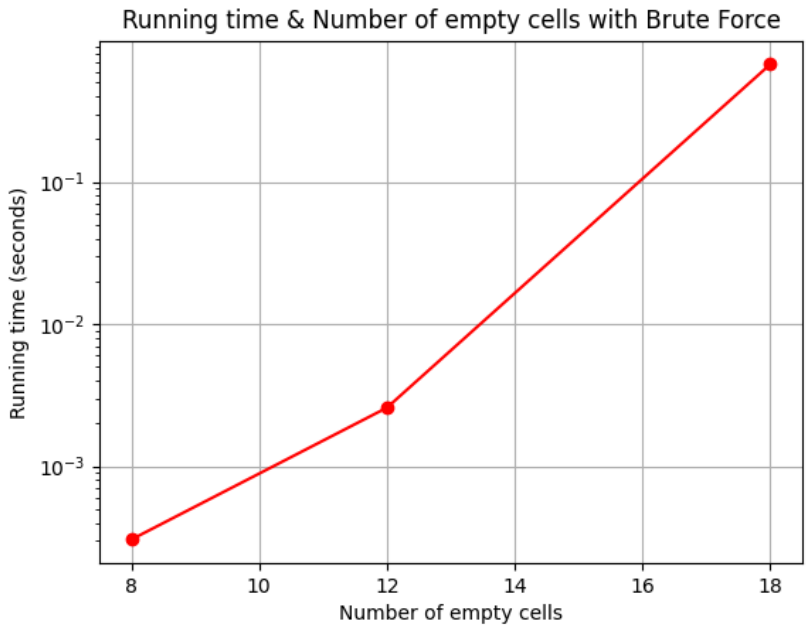
B. Phân tích

1. Nhận xét từ thực nghiệm

- Dễ dàng nhận thấy rằng thời gian thực thi của pysat tăng không đáng kể khi kích thước ma trận tăng. Dù ban đầu thời gian thực thi của pysat có vẻ còn chậm hơn Backtracking một chút, tuy nhiên đến ma trận 20x20 đã cho thấy sự khác biệt rõ rệt.
- Tuy cùng có độ phức tạp về thời gian cơ bản là 2^n (n là số ô trống) nhưng nhờ xử lý tiên tiến mà Glucose3 của thư viện pysat và Backtracking theo hướng DPLL đem lại hiệu suất cao hơn xa so với Brute Force.
- Thuật toán Brute Force chỉ duyệt và kiểm tra một cách đơn thuần nên đã "vỡ trận" khi nhận vào một ma trận 11x11.

⇒ Thuật toán pysat có khả năng mở rộng cao nhất, bài toán càng lớn càng tỏ rõ sự ưu việt. Trong khi đó, Backtracking có khả năng đáp ứng khá ổn cho các bài toán dạng trung, còn Brute Force là "tối hậu sách", cách trong không có cách, chỉ có thể giải những bài toán nhỏ, còn gặp những bài toán chỉ nhỉnh hơn một chút đã phải chịu thua.

2. Lý giải kết quả timeout của Brute Force



- Khi chạy thử ma trận 6x6 (12 ô trống) và ma trận 7x7 (18 ô trống), thêm ma trận 5x5 (8 ô trống) trước đó, ta thu được biểu đồ như trên, cho thấy rằng lượng thời gian thực thi của Brute Force tăng nhanh khi lượng ô trống càng nhiều về sau. Khi chạy ma trận kích thước 8x8 đã vượt hơn 120 giây (trong video demo).
- Theo cài đặt, mỗi lần lặp, Brute Force cần lặp thêm n lần (n là số trống) và kiểm thử một mô hình với số lần lặp là m (m là số mệnh đề có trong CNF), độ phức tạp về thời gian của thuật toán này là $O(2^n \times n \times m)$.
- Xem như m không đáng kể, ta có thể ước tính thời gian thực thi của Brute Force khi giải ma trận 11x11 (giả sử số ô trống chiếm ⅓ ma trận) như sau:

$c \times 2^8 \times 8 = 0.0003049374$ (s) → $c = \approx 1.49 \times 10^{-7}$ (s)

⇒ Với n = 11x11:3 thì thời gian là $1.49 \times 10^{-7} \times 2\text{^}40 \times 40 \approx 6.55 \times 10^6$ (s) ≈ 75 ngày

Chính vì vậy không thể chờ kết quả từ Brute Force

3. Độ phức tạp về không gian:

- PySAT: O(n + c) với n là số ô trống và c là số lượng mệnh đề. Các mệnh đề đươc mở rộng thêm trong quá trình thực thi.
- Brute Force: O(n) với n là số ô trống. Chỉ lưu trữ cấu hình hiện tại đang được kiểm tra.
- Backtracking: O(n) với n là số ô trống. Độ sâu stack khi đệ quy tỷ lệ thuận với n.

V. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

Yêu cầu	Mức độ hoàn thành
Lập luật giải pháp và mô tả ý tưởng tạo CNF	100%
Tạo CNF một cách tự động	100%
Sử dụng thư viện PySAT để giải CNF	100%
Cài đặt thuật toán Brute Force và so sánh	100%
Cài đặt thuật toán Backtracking và so sánh	100%
Kiểm thử ít nhất 3 trường hợp (5x5, 11x11, 20x20), đưa ra phân tích và đánh giá kết quả thử nghiệm. So sánh các thuật toán.	100%

VI. DANH MỤC THAM KHẢO

[1] Tutorial pysat

https://pysat.readthedocs.io/en/latest/tutorial.html

[2] SAT-based MaxSAT algorithms

https://www.sciencedirect.com/science/article/pii/S000437021300012X