TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG – HCM

Khoa Công Nghệ Thông Tin

LAB: GEM HUNTER

Môn học: Cơ Sở Trí Tuệ Nhân Tạo

Sinh viên: 22120216 – Bùi Tấn Thành Nam

Học Kỳ - Niên học: HK2 – 2024 - 2025

Thành phố Hồ Chí Minh – 2025

Mục Lục

1. Giới thiệu.

Gem Hunter là một trò chơi tư duy logic, nơi người chơi khám phá một lưới ô vuông để tìm kiếm các viên đá quý ẩn (gems) trong khi tránh các bẫy (traps). Mỗi ô trong lưới có thể chứa một con số, biểu thị số lượng bẫy xung quanh nó (trong 8 ô kề bên). Các ô trống chưa xác định sẽ được suy luận là bẫy hoặc đá quý dựa trên thông tin có sẵn trong lưới.

Trong Lab 01, nhiệm vụ mô hình hóa bài toán Gem Hunter thành các biểu thức logic dưới dạng Chuẩn hội (CNF – Conjunctive Normal Form), sau đó áp dụng các thuật toán suy luận để tìm lời giải. Đây là một ứng dụng thực tiễn của các kỹ thuật suy diễn logic trong trí tuệ nhân tạo. Phân tích và giải bài toán, từ việc chuyển đổi lưới trò chơi thành CNF cho đến triển khai và so sánh ba phương pháp giải khác nhau: Brute-force, Backtracking và thư viện PySAT. Qua các thực nghiệm với lưới có kích thước tăng dần (5×5, 11×11, 20×20), báo cáo đánh giá hiệu năng, khả năng mở rộng và độ chính xác của từng phương pháp.

Thông qua việc thực hiện đồ án này, củng cố tư duy logic, hiểu rõ cách biểu diễn bài toán dưới dạng CNF, cũng như ứng dụng hiệu quả các công cụ và thuật toán trong lĩnh vực Trí tuệ Nhân tạo.

2. Mô tả bài toán.

2.1. Luật chơi

- Trò chơi diễn ra trên một lưới ô vuông gồm nhiều hàng và cột.
- Một số ô chứa số nguyên từ 1 đến 8, biểu thị số lượng bẫy (T) xung quanh ô đó.
- Các ô không chứa số sẽ được suy luận là bẫy (T) hoặc đá quý (G).
- Mục tiêu là xác định chính xác loại của từng ô chưa rõ, sao cho thỏa mãn toàn bộ các ràng buộc số đã cho.

2.2. Mục tiêu

- Mỗi ô chưa xác định cần được gán một trong hai giá trị: T (bẫy) hoặc G (đá quý).
- Kết quả cuối cùng là một lưới đầy đủ, đảm bảo tất cả các ô có số phản ánh đúng số bẫy lân cận.

3. Xử lý CNF

3.1. Mệnh đề và biến trong CNF

Trong logic mệnh đề, một literal là một biến Boolean hoặc phủ định của nó. Trong thư viện PySAT (thư viện giải SAT trong Python), literal được biểu diễn dưới dạng số nguyên:

- Biến x1 → 1
- Phủ định ¬x1 → -1

Một mệnh đề (clause) là phép OR của các literal, và trong PySAT, nó được biểu diễn bằng một danh sách các số nguyên. Ví dụ:

- $(\neg x1 \vee \neg x2 \vee x3) \rightarrow [-1, -2, 3]$

Một CNF (Conjunctive Normal Form) là phép AND của nhiều mệnh đề, được biểu diễn như một danh sách các danh sách:

- $(\neg x1 \vee x2) \wedge (x1 \vee x3 \vee \neg x4) \rightarrow [[-1, 2], [1, 3, -4]]$

Đây là định dạng chuẩn để các SAT solver sử dụng khi đánh giá tính khả thỏa.

3.2. Nguyên tắc logic chính xác để sinh CNF

Để biểu diễn bài toán Gem Hunter dưới dạng CNF, ta cần chuyển thông tin từ các ô số (chỉ số lượng bẫy xung quanh) thành các ràng buộc logic.

Ràng buộc Exactly-K

Giả sử một ô có số "k" nghĩa là có đúng k bẫy xung quanh. Gọi danh sách các biến tương ứng với các ô xung quanh là neighbors_vars. Để biểu diễn "chính xác k biến là True", ta phân rã thành:

xactly K $\Leftrightarrow$ (At least K) $\wedge$ (At most K)

- **Ràng buộc Tối đa K (At most K)**

Để đảm bảo không có nhiều hơn K biến trong neighbors_vars là True (tức là bẫy), ta thực hiện:

Lọc ra tất cả tập con k+1 phần tử.

Với mỗi tập con, tạo một mệnh đề phủ định tất cả các biến trong tập con → ngăn không cho cùng lúc k+1 biến là True.

Công thức:

Với mọi tập con (k+1) phần tử trong neighbors_vars:

clause = [¬l1, ¬l2, ..., ¬l(k+1)]

- **Ràng buộc Tối thiểu K (At least K)**

Để đảm bảo ít nhất K biến là True (tức là bảy), ta:

Lọc ra tất cả các tập con (n-K+1) phần tử, với n là kích thước neighbors_vars.

Mỗi mệnh đề là OR các biến trong tập con, nghĩa là ít nhất một trong số đó phải là bảy.

Công thức:

Với mọi tập con (n-K+1) trong neighbors_vars:

clause = [l1, l2, ..., l(n-K+1)]

3.3. Sinh mệnh đề

Dựa trên biểu diễn Exactly-K, với mỗi ô trong lưới có số:

- Lấy ra danh sách các ô trống xung quanh (tương ứng với các biến).
- Áp dụng At-Most-K và At-Least-K để tạo ra các mệnh đề.

Lặp lại quá trình trên cho từng ô có số trong lưới và thêm vào tập CNF chung

3.4. Sinh CNF tự động

Quy trình sinh CNF tự động:

1. Gán biến cho mỗi ô trống trong lưới.
2. Duyệt qua lưới:
3. Với mỗi ô chứa số:
   + Lấy ra danh sách các ô trống xung quanh.
   + Áp dụng bước 4.3 để sinh mệnh đề.
4. Loại bỏ mệnh đề trùng lặp để tối ưu.
5. Trả về CNF dạng danh sách các mệnh đề (list of lists), sẵn sàng dùng với PySAT hoặc SAT solver khác.

4. Xử lý CNF

Sau khi sinh được CNF từ lưới trò chơi, ta áp dụng ba phương pháp khác nhau để giải và so sánh hiểu quả:

4.1. Brute-force

Duyệt tất cả các cách gán có thể cho n biến logic. là một chiến lược tìm kiếm đơn giản, toàn diện. Tuy nhiên, do độ phức tạp thời gian cao, các kỹ thuật không hiệu quả đối với các vấn đề quy mô lớn.

Thực hiện:

- Với mỗi mô hình gán, kiểm tra xem có thỏa tất cả các mệnh đề trong CNF không.
- Nếu có mô hình hợp lệ, trả về kết quả.

Nhược điểm:

- Tốn thời gian tính toán rất lớn khi số lượng biến tăng.
- Không khả thi với lưới lớn (ví dụ 11x11 trở lên).

4.2. Backtracking

Thuật toán Backtracking (quay lui) là một phương pháp tìm kiếm và liệt kê các cấu hình có thể của một bài toán bằng cách thử tất cả các khả năng và quay lại bước trước đó nếu gặp lựa chọn không phù hợp.

Duyệt theo chiều sâu, gán từng biến và kiểm tra sớm để loại bỏ các nhánh sai.

Thực hiện:

- Áp dụng thuật toán DPLL (Davis–Putnam–Logemann–Loveland).

- Tự động đơn giản hóa CNF tại mỗi bước gán để rút gọn bài toán.
- Khi một mệnh đề trở nên rỗng, quay lui.

Ưu điểm:

- Nhanh hơn brute-force do loại bỏ được nhiều nhánh sai.
- Có thể giải được bài toán với quy mô trung bình (như 11x11 hoặc 20x20 nếu tối ưu tốt).

Nhược điểm:

- Vẫn có thể bị quá tải với bài toán lớn.
- Cần cài đặt chi tiết và đúng thuật toán DPLL.

4.3. PySAT Solver

Sử dụng thư viện chuyên dụng đã tối ưu hóa để giải CNF.

Thực hiện:

- Khởi tạo solver với tập CNF đã sinh.
- Gọi hàm .solve() và lấy mô hình gán nếu có.

Ưu điểm:

- Rất nhanh, hiệu quả cao, phù hợp với bài toán lớn.
- Được tối ưu bằng các kỹ thuật hiện đại như clause learning, watched literals, heuristic.

Nhược điểm:

- Phụ thuộc vào thư viện bên ngoài.

5. Thực nghiệm

5.1. Input

Tất cả các tệp đầu vào đều nằm trong thư mục **input**. Tên của mỗi tệp theo định dạng "input x.txt" với x là số thứ tự của đầu vào.

Để đánh giá hiệu năng của ba phương pháp, tiến hành input với ba kích thước lưới:

- **5x5**: Lưới nhỏ, có thể giải bằng cả ba phương pháp.
- **11x11**: Lưới trung bình, brute-force bắt đầu thất bại.
- **20x20**: Lưới lớn, chỉ còn PySAT có thể xử lý hiệu quả.

5.2. Output

Tất cả các tập tin đầu ra đều nằm trong thư mục đầu ra. Tên của mỗi tập tin theo định dạng "output x.txt" với x là số thứ tự của đầu ra. Mỗi tập tin đầu ra chứa lưới đã được giải quyết bằng 3 phương pháp: thư viện Pysat, Brute force và backtracking.

5.3. Bảng so sánh

TestCase > algorithm_comparison.txt

```
Algorithm Comparison Summary
================================================

Test Case           SAT Time       DPLL Time      Brute Force Time     SAT Valid  DPLL Valid Brute Force Valid
-------------------------------------------------------------------------------------------------------
Test Case 1            0.0001s        0.0000s        0.0004s            Yes        Yes        Yes
Test Case 2            0.0001s        0.0007s        0.0133s            Yes        Yes        Yes
Test Case 3            0.0001s        0.0004s        Timeout            Yes        Yes        No
Test Case 4            0.0003s        0.0033s        Timeout            Yes        Yes        No

```

6. Phân Tích

Dựa trên kết quả thực hiện cùng một testcase với các giải pháp khác nhau, ta được được một bảng so sánh hiệu suất của các thuật toán.

Test case	Brute Force (s)	Backtracking (s)	PaySAT (s)
1	0.0004	0.0001	0.0001
2	0.0133	0.0007	0.0001
3	Timeout	0.0004	0.0001
4	Timeout	0.0033	0.0003

6.1. Hiệu năng theo từng phương pháp

Brute-force:

- Chỉ thực hiện tốt ở test case nhỏ (test 1, 2).
- Hoàn toàn không giải được các test case lớn hơn (3 và 4) do quá thời gian (Timeout).
- Thể hiện rõ nhược điểm về hiệu năng với độ phức tạp thời gian là O(2^n).

Backtracking:

- Giải quyết tốt tất cả test case từ nhỏ đến lớn.
- Dù thời gian tăng dần theo độ phức tạp của bài toán, nhưng vẫn ở mức dưới 10 mili giây.
- Cho thấy thuật toán DPLL là giải pháp hiệu quả hơn brute-force nhờ khả năng cắt nhánh sớm.

PySAT:

- Là phương pháp nhanh và ổn định nhất.
- Thời gian giải đều dưới 1 mili giây, ngay cả với test case lớn nhất.
- Đây là minh chứng cho hiệu suất tối ưu của SAT solver chuyên dụng sử dụng kỹ thuật hiện đại như conflict-driven learning, heuristic, v.v.

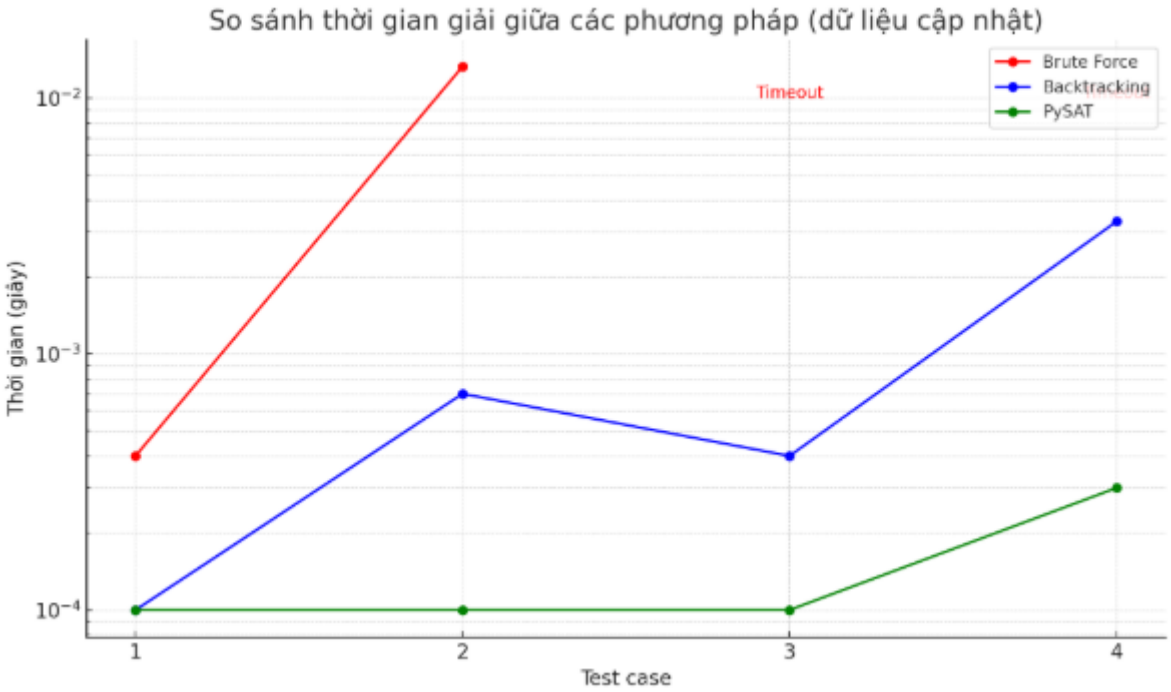
6.2. So sánh tổng quan

Brute-force chỉ phù hợp cho lưới nhỏ, mang tính tham khảo hoặc kiểm thử đơn giản.
Backtracking đạt hiệu quả tốt hơn nhiều và có thể áp dụng cho bài toán có quy mô vừa.
PySAT là lựa chọn hàng đầu cho cả quy mô vừa và lớn, vừa đảm bảo độ chính xác cao vừa có thời gian chạy nhanh nhất.

6.3. Biểu đồ minh họa.

Biểu đồ thời gian (trục tung là giây, trục hoành là test case) cho thấy:

- Đường thời gian của brute-force tăng đột ngột đến mức không giải được.
- Backtracking có độ dốc thời gian nhẹ, vẫn khả thi.
- PySAT hầu như nằm ngang → ổn định và hiệu quả vượt trội.



7. Đánh giá.

STT	Tiêu chí	Tỷ lệ	Mức độ hoàn thành
1	Mô tả giải pháp: Trình bày rõ ràng nguyên tắc logic chính xác để sinh dạng CNF.	20%	100%
2	Tự động sinh CNF từ dữ liệu đầu vào.	10%	100%
3	Sử dụng thư viện PySAT để giải CNF một cách chính xác.	10%	100%
4	Cài đặt thuật toán brute-force để so sánh với thư viện	10%	100%
5	Cài đặt thuật toán backtracking để so sánh với thư viện	10%	100%
6	Tài liệu và các phần phân tích cần trình bày trong báo cáo: - Phân tích kỹ lưỡng và thực nghiệm đầy đủ - Thực hiện ít nhất 3 bộ test với kích thước khác nhau (5x5, 11x11, 20x20) để kiểm tra lời giải - So sánh kết quả và hiệu năng giữa các phương pháp	40%	100%

8. Source code and Video demo

- Github repository
- Video

9. Tài Liệu Tham Khảo

- Antonio Morgado Alexey Ignatiev Joao Marques-Silva. Boolean formula ma nipulation (pysat.formula). Accessed: March 2025.