A blue and black logo

Description automatically generatedA blue and white logo

Description automatically generated**KHOA CÔNG NGHỆ THÔNG TIN**



**4.0**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM

Điện thoại: (08) 38.354.266 – Fax: (08) 38.350.096



Cơ sở trí tuệ nhân tạo

**Báo cáo đồ án cá nhân**

Đề tài: Gem Hunter

🙦🙤

**Giảng viên hướng dẫn: Bùi Duy Đăng**

**Họ và tên:** Phan Công Châu

**MSSV:** 2212003

**Tp. Hồ Chí Minh, tháng 5 năm 2025**

MỤC LỤC

[**I. GIỚI THIỆU ĐỒ ÁN 3**](#_Toc198507963)

[**II. MÔ TẢ THUẬT TOÁN 3**](#_Toc198507965)

**1.** [**Các phương pháp 3**](#_Toc198507966)

**2.** [**Test case 3**](#_Toc198507970)

**3.** [**Mô tả thuật toán 4**](#_Toc198508010)

**4.** [**Tổng kết so sánh 12**](#_Toc198508010)

[**III. KẾT LUẬN 12**](#_Toc198507963)

[**IV. TỰ ĐÁNH GIÁ 5**](#_Toc198507965)

[**V. TÀI LIỆU THAM KHẢO 5**](#_Toc198507963)

1. **GIỚI THIỆU ĐỒ ÁN**

"**Gem Hunter**" yêu cầu người học xây dựng một trò chơi săn đá quý trên lưới, trong đó người chơi cần tìm đá quý và tránh bẫy dựa trên các con số gợi ý xung quanh. Bài toán được biểu diễn dưới dạng ràng buộc logic **CNF** và giải bằng thư viện **PySAT**. Ngoài ra, sinh viên còn cần hiện thực và so sánh hiệu suất của các thuật toán **brute-force**, **backtracking**. Đây là bài tập giúp rèn luyện tư duy logic, kỹ năng lập trình và áp dụng công cụ **AI** vào giải quyết bài toán suy luận.

* **Quy tác của Gem Hunter**
  + Các ô trống (\_) phải được gán giá trị: T (trap - bẫy) hoặc G (gem - ngọc).
  + Các ô chứa số tạo ra rang buộc: số trong ô này hiển thị cho tổng số lượng bẫy xung quanh ô đó.

1. **MÔ TẢ THUẬT TOÁN**
2. Các phương pháp

* **PySAT**: sử dụng thư viện **SAT** **solver** để tìm nghiệm hiệu quả.
* **Brute-force**: Thử tất cả các tổ hợp **T/G** cho các ô chưa biết.
* **Backtracking**: Gán giá trị từng bước và quay lui khi gặp mâu thuẫn

1. Test case
   1. 5x5

\_, 3, \_, \_, 2

\_, 4, \_, \_, \_

\_, \_, 3, \_, 3

2, \_, \_, \_, \_

\_, 1, \_, 3, \_

* 1. 11x11

2, \_, \_, 1, \_, \_, 2, \_, \_, 1, \_

\_, 4, \_, \_, 3, \_, \_, 2, \_, \_, 1

\_, \_, 3, \_, \_, 2, \_, \_, 3, \_, \_

1, \_, \_, 3, \_, \_, 2, \_, \_, 2, 1

\_, 2, \_, \_, 4, \_, \_, 3, \_, \_, 2

\_, \_, 3, \_, \_, 4, \_, \_, 3, \_, \_

1, 2, \_, \_, 3, \_, \_, 4, \_, \_, 2

\_, \_, 2, \_, \_, 3, \_, \_, 2, \_, \_

1, \_, \_, 2, \_, \_, 2, \_, \_, 3, 1

\_, 3, \_, \_, 2, \_, \_, 3, \_, \_, 2

\_, \_, 1, \_, \_, 1, \_, \_, 2, \_, \_

* 1. 20x20

\_, \_, 2, \_, \_, \_, 3, \_, \_, \_, 2, \_, \_, 1, \_, \_, \_, 2, \_, \_

\_, 3, \_, \_, 2, \_, \_, \_, \_, 3, \_, \_, \_, \_, 2, \_, \_, \_, \_, 2

\_, \_, \_, 2, \_, \_, \_, 3, \_, \_, \_, 2, \_, \_, \_, \_, \_, 2, \_, \_

2, \_, \_, \_, \_, 3, \_, \_, \_, 2, \_, \_, 2, \_, \_, \_, 3, \_, \_, \_

\_, \_, 2, \_, \_, \_, \_, 2, \_, \_, \_, \_, 3, \_, \_, 2, \_, \_, 3, \_

\_, \_, \_, 3, \_, \_, \_, \_, 2, \_, \_, \_, \_, 2, \_, \_, \_, 3, \_, \_

\_, 2, \_, \_, \_, 2, \_, \_, \_, 3, \_, \_, \_, \_, \_, 3, \_, \_, \_, \_

3, \_, \_, \_, \_, \_, \_, 2, \_, \_, \_, 3, \_, \_, \_, \_, \_, 2, \_, \_

\_, \_, 3, \_, \_, 2, \_, \_, \_, \_, \_, \_, 2, \_, \_, 2, \_, \_, 3, \_

\_, \_, \_, 2, \_, \_, \_, 3, \_, \_, \_, \_, \_, 3, \_, \_, \_, 2, \_, \_

2, \_, \_, \_, \_, 3, \_, \_, 2, \_, \_, \_, 2, \_, \_, \_, \_, \_, \_, 2

\_, 3, \_, \_, \_, \_, 2, \_, \_, \_, 3, \_, \_, \_, \_, \_, 3, \_, \_, \_

\_, \_, \_, 3, \_, \_, \_, \_, \_, 2, \_, \_, \_, \_, 2, \_, \_, \_, 2, \_

\_, 2, \_, \_, \_, 2, \_, \_, \_, \_, \_, \_, 3, \_, \_, \_, \_, \_, \_, \_

3, \_, \_, \_, \_, \_, \_, 2, \_, 2, \_, \_, \_, 2, \_, \_, \_, 3, \_, \_

\_, \_, 3, \_, \_, \_, \_, 3, \_, \_, \_, 2, \_, \_, \_, 2, \_, \_, \_, \_

\_, \_, \_, 2, \_, \_, \_, \_, 2, \_, \_, \_, \_, 3, \_, \_, \_, \_, 3, \_

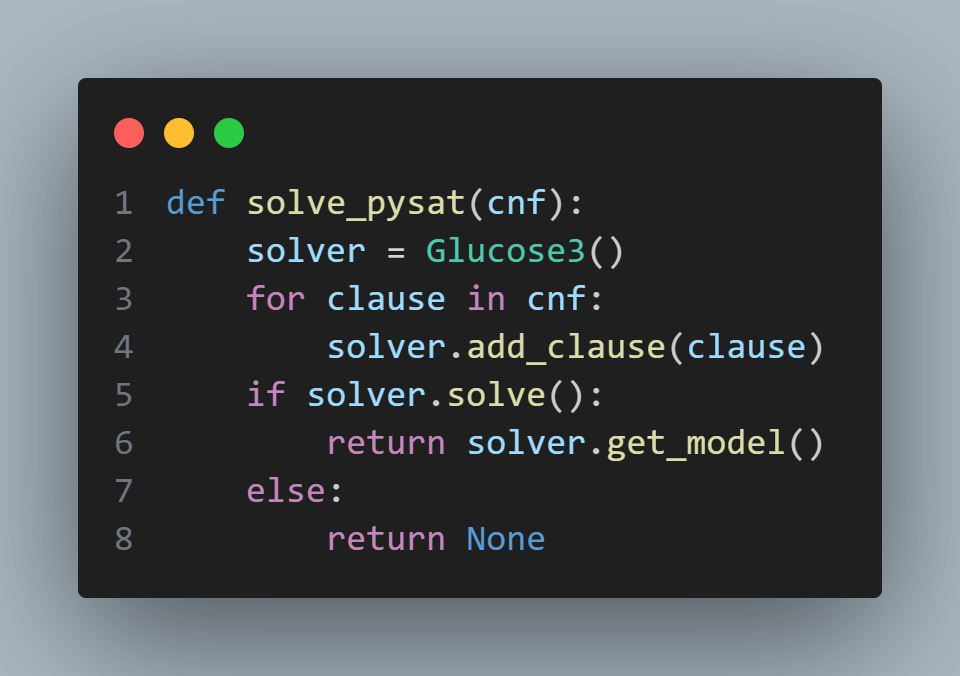
\_, 2, \_, \_, \_, 3, \_, \_, \_, \_, \_, \_, 2, \_, \_, 2, \_, \_, \_, \_

\_, \_, \_, 4, \_, \_, \_, \_, \_, 2, \_, \_, \_, \_, 3, \_, \_, \_, \_, \_

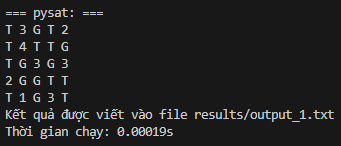
\_, 2, \_, \_, \_, 2, \_, 3, \_, \_, \_, 2, \_, \_, \_, \_, \_, \_, 3, \_

1. Mô tả thuật toán

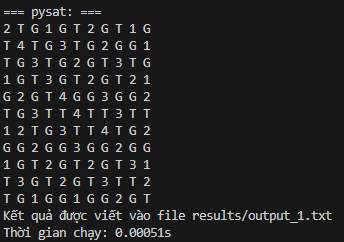
* Bước 1: Dùng **read\_grid\_from\_file(filename)** để đọc từng dòng trong file, chuyển thành ma trận 2 chiều gồm các kí hiệu:
  + “\_”: vị trí chưa biết.
  + Số: số lượng bẫy xung quanh.
* Bước 2: Dùng **get\_neighbors(grid, i, j)** để xác định các vị trí lân cận
  + Duyệt qua tất cả các vị trí xung quanh ô **(i, j)**
  + Kiểm tra các vị trí lân cận đó, nếu hợp lệ thì thêm vào danh sách **neighbors.**
* Bước 3: Dùng **var(i, j, cols)** để ánh xạ vị trí ô trong grid thành 1 biến nguyên dương duy nhất. Điều này giúp biểu diễn mỗi ô bằng 1 biến logic khi xây dựng các mệnh đề **CNF**.
* Bước 4: Dùng **create\_cnf(grid)** để chuyển ma trận lưới thành các mệnh đề logic (**CNF**), duyệt từng ô trong grid:
  + Nếu là “**T**”: thêm mệnh đề khẳng định đây là ô bẫy.
  + Nếu là “**G**”: thêm mệnh đề khẳng định đây là ô ngọc.
  + Nếu là **số**:
    - Lấy các ô lân cận.
    - Đếm số bẫy đã biết (**known\_traps**)
    - Lấy các ô chưa biết (**unknown**).
    - Tính số bẫy còn lại (**remaining**).
    - Nếu **remaining == 0**: tất cả các ô chưa biết xung quanh chắc chắn không phải bẫy.
    - Nếu **remaining == len(unknown):** Tất cả các ô chưa biết xung quanh chắc chắn là bẫy.
    - Nếu không, sẽ sinh các mệnh đề tổ hợp để đảm bảo đúng số lượng bẫy.
* Chuyển grid thành bài toán logic để giải bằng **SAT Solver**
* Bước 5: Giải bài toán
  + Dùng thư viện **PySAT**:



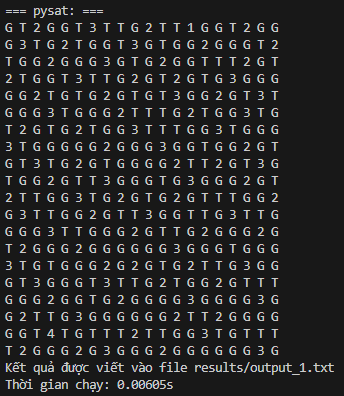
* + - Nhận đầu vào **cnf** là danh sách các vế (**clause**), mỗi vế là danh sách số nguyên
    - Tạo solver **Glucose3()**, thêm từng về vào Solver.
    - Gọi **Solver()** để tìm được lời giải
      * Nếu có, trả về model (danh sách các giá trị, ví dụ [1, -2] là x1 = true, x2 = False).
      * Nếu không có tả về **None.**
* **solve\_pysat(cnf)** nhận công thức logic, tìm cách gán **True/False** để công thức đúng và trả kết quả.
  + - Kết quả:
      * 5x5:



* + - * 11x11:



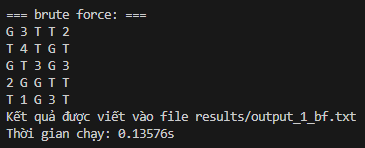
* + - * 20x20:



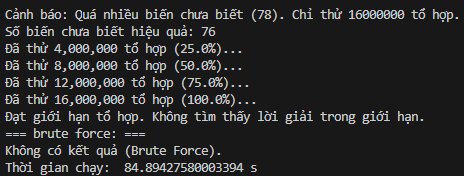
* + Dùng **Brute-Force:**



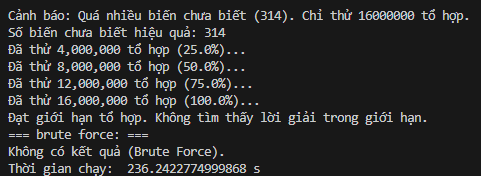
* + - Đầu vào:
      * **cnf**: Danh sách các mệnh đề , mỗi mệnh đề là một danh sách các **lit**
      * **unknown\_vars**: Danh sách các biến (số nguyên dương) chưa biết giá trị, cần gán để kiểm tra
    - Kiểm tra trường hợp đơn giản:
      * Nếu unknown\_vars rỗng, kiểm tra biểu thức **CNF** có được thỏa mãn khi không có biến nào thay đổi. Trả về **{}** nếu đúng, ngược lại trả về **None**.
    - Giới hạn số tổ hợp cần thử:
      * Để tránh duyệt quá lâu, hàm giới hạn tối đa số tổ hợp có thể thử (**max\_combinations = 16\_000\_000**). Nếu số biến chưa biết vượt quá **24**, sẽ in cảnh báo.
    - Tận dụng unit clause để cố định biến:
      * Duyệt qua các mệnh đề đơn, từ đó suy ra các biến buộc phải **True** hoặc **False**. Điều này giúp giảm số biến cần gán sau này.
    - Xác định tập biến hiệu quả:
      * Loại bỏ các biến đã được cố định từ **unknown\_vars,** chỉ giữ lại các biến cần gán (**effective\_vars**).
    - Duyệt tất cả các tổ hợp có thể:
      * Sử dụng **itertools.product** để sinh ra tất cả tổ hợp **True/False** cho **effective\_vars.**
      * Gộp tổ hợp này với các biến đã cố định để tạo ra assignment hoàn chỉnh.
    - Kiểm tra tính đúng đắn của assignment:
      * Với mỗi **assignment**, kiểm tra xem có mệnh đề nào không thỏa mãn không.
      * Một mệnh đề được thỏa mãn nếu tồn tại ít nhất một lit đúng.
    - Dừng khi tìm được lời giải đầu tiên:
      * Nếu tìm được assignment thỏa mãn toàn bộ **CNF**, trả về **assignment** đó.
      * Nếu đã thử hết số tổ hợp hoặc vượt quá giới hạn, trả về **None**.
    - Kết quả
      * 5x5:



* + - * 11x11:



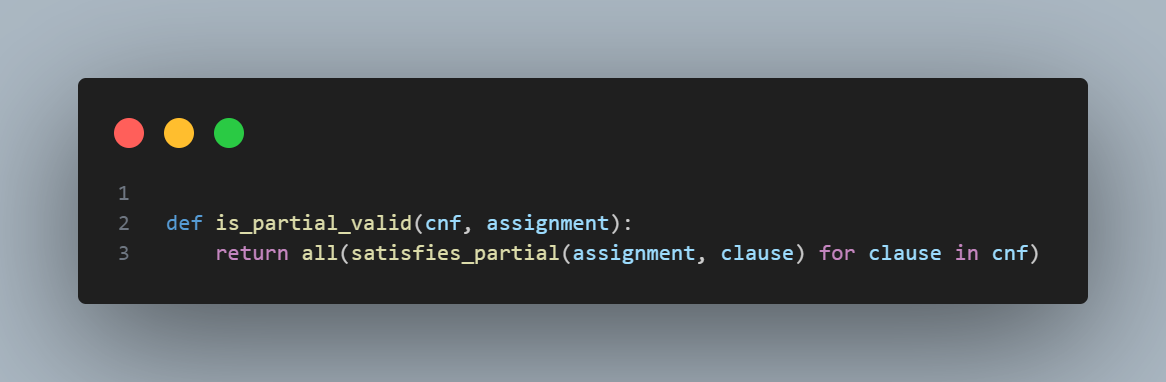
* + - * 20x20:



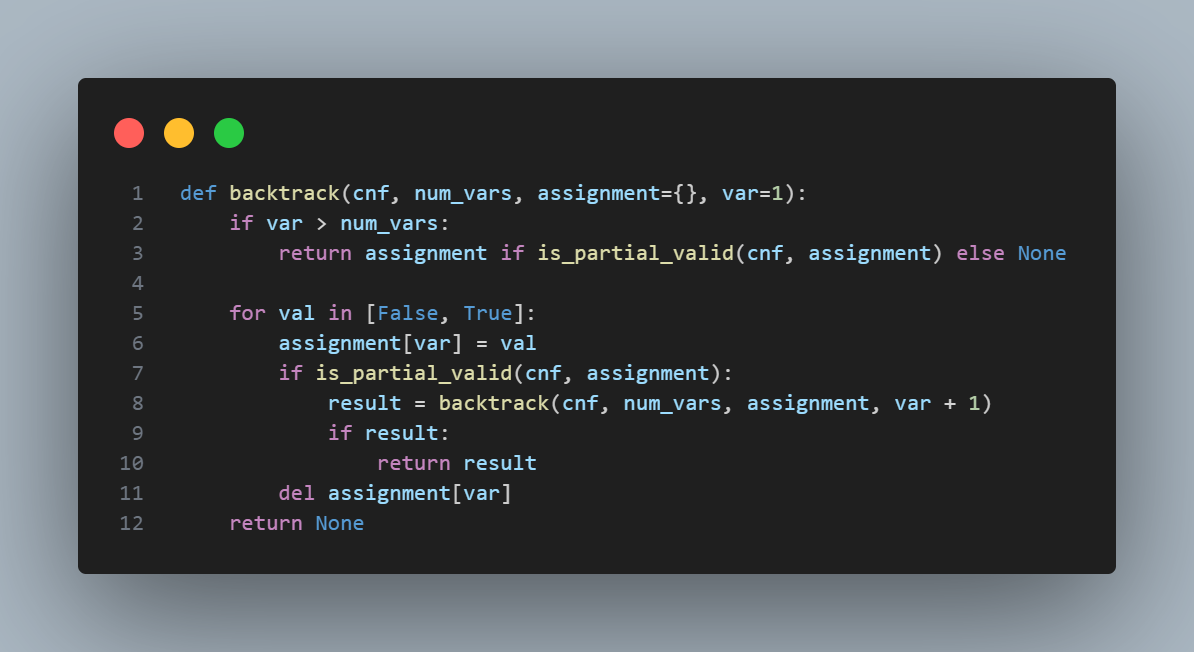
* + Dùng **Backtracking:**



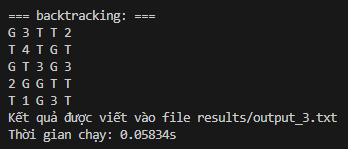
* + - **satisfies(assignment, clause)** kiểm tra một mệnh đề có được thõa mãn hoặc chưa chắc chắn bị sai dưới một gán hiện tại
    - Đầu vào:
      * **assignment:** ánh xạ biến giá trị **True/False**
      * **clause:** danh sách các lit đại diện một mệnh đề **CNF**
    - Duyệt từng **lit** trong **clause:**
      * Nếu chưa được gán, đánh dấu **undecided.**
      * Nếu lit được thõa, đánh dấu **satisfied = True.**
    - Trả về **True** nếu mệnh đề thõa hoặc còn biến vẫn còn cơ hội để thõa mãn, trả về **False** nếu mệnh đề chắc chắn sai với gán hiện tại



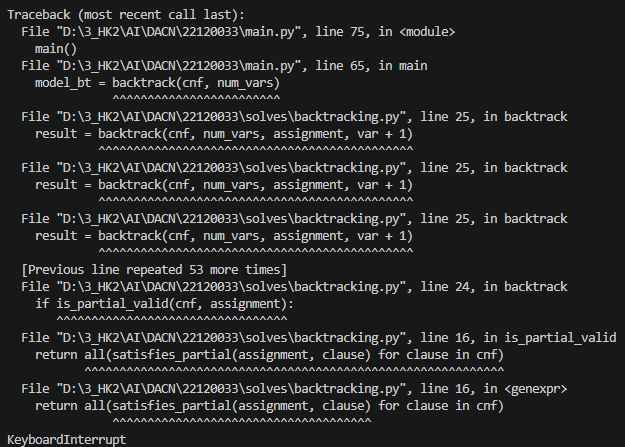
* + - Xác định xem toàn bộ **CNF** có vẫn còn hợp lệ dưới một gán hiện tại.
    - Đầu vào:
      * cnf: Danh sách các mệnh đề.
      * Gán hiện tại của các biến.
    - Gọi **satisfies\_partial** cho từng mệnh đề trong **CNF**.
    - Nếu tất cả mệnh đề đều được thỏa mãn hoặc chưa chắc chắn sai, trả về **True**.
    - Nếu ít nhất một mệnh đề chắc chắn sai, trả về **False**.



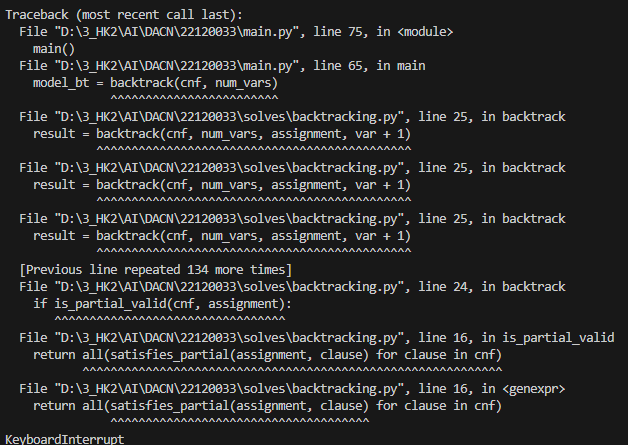
* + - **backtrack(cnf, num\_vars, assignment={}, var=1)** tìm cách gán giá trị cho **num\_vars** biến thỏa mãn công thức **CNF**
    - Đầu vào:
      * **cnf**: Danh sách các mệnh đề **CNF**, mỗi mệnh đề là danh sách các **lit**
      * **num\_vars:** Số lượng biến trong biểu thức
      * assignment: Ánh xạ biến, giá trị **(True/False).**
      * **var:** Biến đang xét (từ 1 đến **num\_vars**)
    - Nếu đã gán hết biến, kiểm tra biểu thức có hợp lệ không, trả về **assignment** hoặc **None**.
    - Thử gán **False** và **True** cho biến **var**.
    - Nếu gán tạm thời không làm mệnh đề nào sai **(is\_partial\_valid)**, đệ quy xét biến tiếp theo.
    - Nếu không có gán nào hợp lệ, quay lui (xóa gán) và thử nhánh khác.
    - Kết quả
      * 5x5:



* + - * 11x11:



* + - * 20x20:



1. Tổng kết so sánh

|  |  |  |  |
| --- | --- | --- | --- |
|  | PySAT | Brute-Force | Backtracking |
| 5x5 | Thời gian: 0.00019s  Nhanh nhất, do dùng giải thuật hiện đại, xử lý dễ dàng. | Thời gian: 0.13576s  Khá chậm, phù hợp bài toán nhỏ. | Thời gian: 0.05834s  Chạy tốt nhờ cắt nhánh sớm khi phát hiện mâu thuẫn. |
| 11x11 | Thời gian: 0.00051s  Chạy mượt, tận dụng heuristic và học mệnh đề. | Bị treo máy khá lâu và không tìm được kết quả. Quá nhiều tổ hợp (2¹²¹), không thể chạy trong thời gian hợp lý. | Không thể chạy được trong thời gian hợp lý (quá nhiều nhánh). |
| 20x20 | Thời gian:0.00605s  Duy nhất có thể giải được. Chạy trong vài giây đến vài phút tùy độ phức tạp. | Bị treo máy khá lâu và không tìm được kết quả. Bất khả thi do 2⁴⁰⁰ tổ hợp. Không thể thử hết. | Không thể hoàn thành. Quay lui vẫn quá tốn thời gian. |

1. **KẾT LUẬN**

* **PySAT**: hiệu quả nhất, nhờ sử dụng các thuật toán hiện đại, giải nhanh và hiệu quả ngay cả với các bài toán phức tạp
* **Brute-Force**: đơn giản, dễ cài đặt nhưng chỉ phù hợp với các bài toán quy mô nhỏ. Với **O(2ⁿ)**, thời gian xử lý tăng theo cấp số mũ và trở nên không khả thi.
* **Backtracking**: cải thiện so với **Brute-Force** nhờ khả năng cắt nhánh sớm, nhưng hiệu quả vẫn bị hạn chế

=> Nên dùng **PySAT**, Backtracking phù hợp với kích thước nhỏ vừa, hạn chế dùng **Brute-Force**

1. **TỰ ĐÁNH GIÁ**
2. **TÀI LIỆU THAM KHẢO**