



**fit@hcmus**

**KHOA CÔNG NGHỆ THÔNG TIN**  
**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 lab 2**

### **Gem hunter**

**Giảng viên hướng dẫn:**

Thầy Bùi Duy Đăng  
Thầy Nguyễn Thanh Tình

**Họ và tên :** Nguyễn Thanh Phong

**MSSV:** 22120265

**Học kỳ:** 2

**Năm học :** 2024-2025

*Thành phố Hồ Chí Minh, tháng 05, năm 2025*

## Mục lục

<b>I.</b>	<b>Bài toán</b>	3
1.	Mô tả bài toán	3
2.	Mục tiêu	3
<b>II.</b>	<b>Xử lý sinh CNF</b>	3
1.	Nguyên lý xây dựng ràng buộc	3
2.	Các bước xây dựng ràng buộc CNF	3
3.	Sinh CNF tự động	3
a.	Gán biến logic cho mỗi ô	3
b.	Sinh các mệnh đề CNF	4
4.	Áp dụng thư viện PySAT để giải CNF	4
<b>III.</b>	<b>Thực nghiệm</b>	5
1.	Lưới 5x5	5
2.	Lưới 11x11	6
3.	Lưới 20x20:	7
4.	Nhận xét	9
5.	Link video demo	9
6.	Tự đánh giá	10
<b>IV.</b>	<b>Tham khảo</b>	10

## I. Bài toán

### 1. Mô tả bài toán

Gem hunter là bài toán lập trình trong lĩnh vực trí tuệ nhân tạo, yêu cầu phát triển trò chơi tìm kiếm đá quý trên lưới ô vuông sử dụng logic dựa trên Conjunctive Normal Form – CNF. Trong trò chơi, người chơi sẽ khám phá một lưới ô vuông để tìm kiếm các viên đá quý ẩn (G) trong khi tránh các bẫy (T). Một số ô trên lưới ô vuông chứa một số nguyên từ 1 – 8, biểu thị số lượng bẫy xung quanh ô đó. Nhiệm vụ chính là xây dựng các ràng buộc CNF, từ đó áp dụng thư viện PySAT hoặc thuật toán brute-force, backtracking để xác định vị trí đá quý và bẫy.

### 2. Mục tiêu

- Xây dựng các ràng buộc CNF dựa trên thông tin số bẫy xung quanh một ô.
- Tự động sinh CNF từ lưới đầu vào.
- Sử dụng thư viện PySAT để tìm lời giải.
- Cài đặt thuật toán brute-force, backtracking, so sánh hiệu suất (thời gian thực thi) của chúng so với sử dụng thư viện PySAT.
- Phân tích và đánh giá hiệu quả của các phương pháp qua các test case với lưới ô vuông các kích thước khác nhau (5x5, 11x11, 20x20).

## II. Xử lý sinh CNF

### 1. Nguyên lý xây dựng ràng buộc

Lưới ô vuông được biểu diễn dưới dạng các ràng buộc logic trong dạng chuẩn tắc kết hợp (CNF). Mỗi ô trên lưới được gán một biến logic: True nếu ô đó chứa bẫy, False nếu ô đó chứa đá quý. Các ràng buộc CNF được xây dựng dựa trên các thông tin sau:

- Ô chứa số: chứa số tự nhiên  $k$  (từ 1 – 8) biểu thị có đúng  $k$  ô chứa bẫy trong các ô xung quanh (Exactly  $k$  True):

$$\text{Exactly } K \text{ true} \Leftrightarrow (\text{No more than } K \wedge \text{At least } K)$$

- Ô chứa bẫy (T): Biến logic tương ứng với ô này nhận giá trị True.
- Ô chứa đá quý (G): Biến logic tương ứng với ô này nhận giá trị False.

### 2. Các bước xây dựng ràng buộc CNF

Quá trình xây dựng CNF bao gồm các bước sau:

- Gán một biến logic duy nhất cho mỗi ô  $(i, j)$  trên lưới: sử dụng hàm `get_var_id(i, j, width)` từ `utils.py` để trả về một số nguyên duy nhất.
- Xác định các ô lân cận bằng hàm `get_neighbors(i, j, height, width)` từ `utils.py` để trả về danh sách các tọa độ của 8 ô xung quanh.
- Tạo các mệnh đề CNF dựa trên giá trị của ô (số, T, G, \_) và thông tin từ các ô lân cận.

### 3. Sinh CNF tự động

#### a. Gán biến logic cho mỗi ô

- Mỗi ô trên lưới được gán một biến logic  $var\_id_{i,j}$ , trong đó  $(i, j)$  là tọa độ của ô trên lưới ô vuông.
- Với lưới có kích thước  $m \times n$  thì có tổng cộng  $m.n$  biến logic.

- Ví dụ: Lưới có kích thước 3x3 ta có  $3 \cdot 3 = 9$  ô  $\Rightarrow$  có 9 biến logic  
 $2, \_, 1$   
 $\_, \_, 1$   
 $1, \_, 2$

#### b. Sinh các mệnh đề CNF

Duyệt qua toàn bộ ô vuông (grid), kiểm tra giá trị của nó:

- Nếu  $\text{grid}(i, j) = 'T'$ , ta thêm mệnh đề  $[\text{var\_id}]$  để đảm bảo biến logic đó nhận giá trị True.
- Nếu  $\text{grid}(i, j) = 'G'$ , ta thêm mệnh đề  $[-\text{var\_id}]$  để đảm bảo biến logic đó nhận giá trị False.
- Nếu  $\text{grid}(i, j) = k$ :
  - Lấy danh sách các ô lân cận bằng  $\text{get\_neighbors}(i, j, n\_rows, n\_columns)$ .
  - Lấy danh sách các ô chứa bẫy ( $n\_traps$ )
  - Tạo danh sách các biến logic cho các ô trống ( $\text{unknown\_neighbors}$ ).
  - Số bẫy còn lại cần xác định là  $\text{remaining} = k - \text{len}(n\_traps)$ .
  - Kiểm tra tính khả thi: nếu số bẫy còn lại ( $\text{remaining}$ ) nhỏ hơn 0 hoặc lớn hơn số ô chưa xác định lân cận ( $\text{unknown\_neighbors}$ ) thì bài toán không có lời giải, nếu số bẫy còn lại bằng 0 ta cần thêm mệnh đề  $[-x \text{ for } x \text{ in } \text{unknown\_neighbors}]$  để đảm bảo không có ô trống nào là bẫy, nếu số bẫy còn lại bằng đúng với số ô chưa xác định lân cận thì ta cần thêm mệnh đề  $[x \text{ for } x \text{ in } \text{unknown\_neighbors}]$  để đảm bảo tất cả các ô trống còn lại là bẫy.
  - Tạo các ràng buộc “at least k true”.
  - Tạo các ràng buộc “at most k true”.

#### 4. Áp dụng thư viện PySAT để giải CNF

- Sau khi tạo CNF, ta áp dụng PySAT để tìm nghiệm.
- Nghiệm là tập hợp giá trị (True or False) cho các biến  $\text{var\_id}_{i,j}$ , tương ứng với việc gán T hoặc G cho các ô trống.
- Hàm  $\text{solve\_cnf}()$  sử dụng PySAT:

```

1  from pysat.solvers import Glucose3
2  from utils import *
3
4  def solve_cnf(cnf):
5      solver = Glucose3()
6
7      for clause in cnf:
8          solver.add_clause(clause)
9
10     if solver.solve():
11         model = solver.get_model()
12         return model
13     else:
14         return None

```

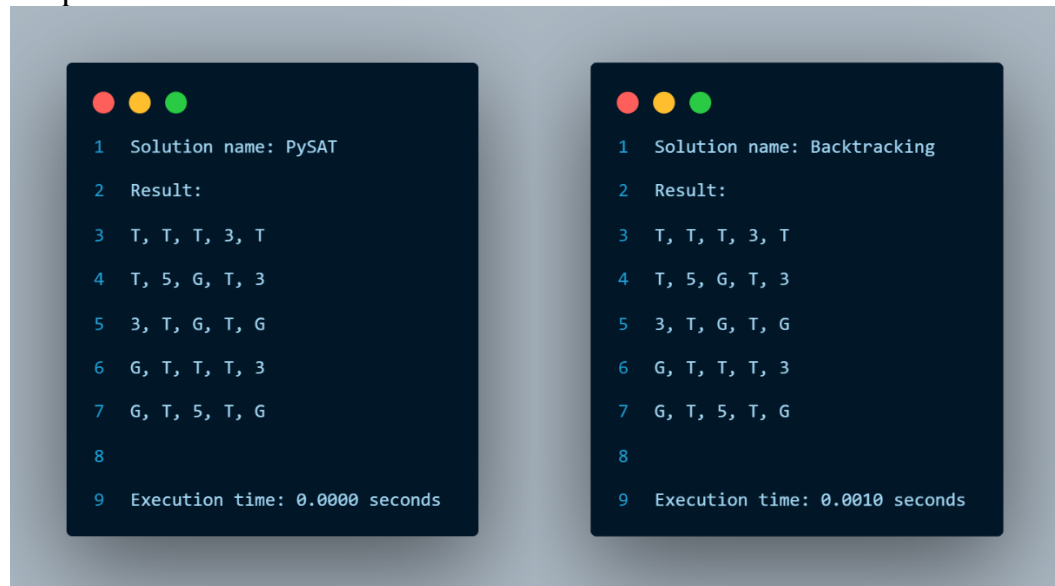
### III. Thực nghiệm

#### 1. Lưới 5x5

- Input:



- Output:



```
1 Solution name: Bruteforce
2 Result:
3 T, T, T, 3, T
4 T, 5, G, T, 3
5 3, T, G, T, G
6 G, T, T, T, 3
7 T, T, 5, T, G
8
9 Execution time: 1.9253 seconds
```

## 2. Lưới 11x11

- Input:

```
1 2, _, 2, 2, _, _, _, 3, 2
2 3, _, 3, 2, _, 5, _, 5, 4, _
3 _, 3, 2, 3, 4, _, 4, 3, _, 3, 2
4 2, 3, _, 2, _, _, 2, 2, 3, 2
5 1, _, 4, 4, 3, 3, 3, 2, 2, _, _
6 3, 5, _, 3, 1, 2, _, 2, 4, _
7 _, _, 4, 4, _, 2, 2, 2, 3, _
8 2, 3, 4, _, 4, 2, 2, 2, _, 4, 3
9 _, 1, 2, 4, _, 2, 1, _, 4, _, _
10 _, 1, _, 4, _, 3, 1, 3, _, 4, 2
11 _, 1, 1, 3, _, 2, _, 2, _, 2, _
```

- Output:



```
1 Solution name: PySAT
2 Result:
3 2, T, T, 2, 2, T, T, T, T, 3, 2
4 3, T, 3, 2, T, 5, T, 5, 4, T, T
5 T, 3, 2, 3, 4, T, 4, 3, T, 3, 2
6 2, 3, T, 2, T, T, T, 2, 2, 3, 2
7 1, T, 4, 4, 3, 3, 3, 2, 2, T, T
8 3, 5, T, T, 3, 1, 2, T, 2, 4, T
9 T, T, T, T, 4, T, 2, 2, 2, 3, T
10 2, 3, 4, T, 4, 2, 2, 2, T, 4, 3
11 G, 1, 2, 4, T, 2, 1, T, 4, T, T
12 G, 1, T, 4, T, 3, 1, 3, T, 4, 2
13 G, 1, 1, 3, T, 2, G, 2, T, 2, G
14
15 Execution time: 0.0011 seconds
```



```
1 Solution name: Backtracking
2 Result:
3 2, T, T, 2, 2, T, T, T, T, 3, 2
4 3, T, 3, 2, T, 5, T, 5, 4, T, T
5 T, 3, 2, 3, 4, T, 4, 3, T, 3, 2
6 2, 3, T, 2, T, T, T, 2, 2, 3, 2
7 1, T, 4, 4, 3, 3, 3, 2, 2, T, T
8 3, 5, T, T, 3, 1, 2, T, 2, 4, T
9 T, T, T, T, 4, T, 2, 2, 2, 3, T
10 2, 3, 4, T, 4, 2, 2, 2, T, 4, 3
11 G, 1, 2, 4, T, 2, 1, T, 4, T, T
12 G, 1, T, 4, T, 3, 1, 3, T, 4, 2
13 G, 1, 1, 3, T, 2, G, 2, T, 2, G
14
15 Execution time: 0.0057 seconds
16
```



```
1 Solution name: Bruteforce
2 No solution found!
3 Execution time: 30.5239 seconds
```

### 3. Lưới 20x20:

- Input:



```
1  1, 2, _, _, _, _, 2, 1, 3, _, 3, 1, 2, 1, 2, 2, _, 3, 2
2  _, 3, 2, _, 2, _, 3, 3, _, 4, _, 5, _, 3, _, 3, _, 4, _ _
3  _, 3, 2, 2, _, _, 2, _, 4, 4, _, _, 4, _, 4, 5, _, 4, 3, _
4  _, 2, 1, _, 3, 3, 5, _, _, 3, _, 5, _, 5, _, _, 3, _, 3, 2
5  1, _, _, _, _, _, _, 4, 2, _, 4, _, _, 5, 4, _, 3, _, _
6  2, 2, 2, _, _, 5, _, _, 5, 3, 3, _, 4, 5, _, _, 4, _, 5, _
7  _, _, _, 4, 4, _, _, _, _, 4, 4, _, 3, _, 5, _, _, _ 2
8  _, 5, _, 3, _, 4, _, 5, 5, 4, _, _, 2, _, 1, 4, _, _, _ 2
9  _, 6, 3, 4, _, _, 3, _, _, 4, 3, 2, 1, 1, 2, 5, _, _, 5, 3
10 _, _, _, 4, 3, 2, 4, _, _, _, 1, 1, 1, _, _, _ 6, _, _
11 _, _, _, 2, _, 4, _, _, 4, 4, _, 4, 4, 5, 6, _, _, _ _
12 5, _, _, 3, 2, _, 3, _, _, _, _, _, _, _, 4, 4, 3, 2
13 _, _, _, 2, 2, 3, 3, 3, _, _, 4, _, _, 6, _, 4, _, 4, _ 2
14 _, _, 3, 1, _, _, 2, 2, _, 2, 3, _, 4, _, 3, _, _, _ 3
15 5, _, 4, _, 4, _, _, 3, _, _, 1, 1, 2, _, 2, 3, 4, _, _
16 _, _, 2, _, 2, 3, 3, _, _, 2, 2, 2, 2, _, 3, _, _ 5, _
17 _, 4, 3, 2, _, 2, _, 3, 3, 4, _, _, 2, 1, 2, _, 5, _, _
18 1, _, 2, 1, _, 3, 2, 2, 2, _, 4, _, 1, 1, 2, _, 3, 2
19 3, 3, 4, _, 4, _ 3, 2, _ 3, 3, 4, 3, _ 2, 1, 1, _ 3, _
20 _, _ 3, _ 3, _ _ 1, 2, _ 2, _ 2, _ 1, _ 1, _ _
```

- Output:



```

1 Solution name: PySAT
2 Result:
3 1, 2, T, G, G, T, T, 2, 1, 3, T, 3, 1, 2, 1, 2, 2, T, 3, 2
4 T, 3, 2, T, 2, G, 3, 3, T, 4, T, 5, T, 3, T, 3, T, 4, T, T
5 T, 3, 2, 2, G, G, 2, T, 4, 4, T, T, 4, T, 4, 5, T, 4, 3, G
6 T, 2, 1, T, 3, 3, 5, T, T, 3, G, 5, T, 5, T, T, 3, T, 3, 2
7 1, G, G, G, T, T, T, T, 4, 2, T, 4, T, T, 5, 4, G, 3, T, T
8 2, 2, 2, T, T, 5, G, T, 5, 3, 3, T, 4, 5, T, T, 4, T, 5, G
9 T, T, G, 4, 4, T, G, T, T, T, 4, 4, T, 3, T, 5, T, T, T, 2
10 T, 5, T, 3, T, 4, T, 5, 5, 4, T, T, 2, G, 1, 4, T, T, T, 2
11 T, 6, 3, 4, T, G, 3, T, T, 4, 3, 2, 1, 1, 2, 5, T, T, 5, 3
12 T, T, T, 4, 3, 2, 4, T, T, T, G, 1, 1, 1, T, T, T, 6, T, T
13 T, T, T, T, 2, T, 4, T, G, 4, 4, T, 4, 4, 5, 6, T, T, G, T
14 5, T, T, 3, 2, G, 3, T, T, T, G, T, T, T, T, T, 4, 4, 3, 2
15 T, T, G, 2, 2, 3, 3, 3, G, T, 4, T, T, 6, T, 4, T, 4, T, 2
16 T, T, 3, 1, T, T, T, 2, 2, G, 2, 3, T, 4, G, 3, G, T, T, 3
17 5, T, 4, G, 4, T, G, T, 3, G, G, 1, 1, 2, T, 2, 3, 4, G, T
18 T, T, T, 2, T, 2, 3, 3, T, T, 2, 2, 2, 2, G, 3, T, T, 5, T
19 G, 4, 3, 2, G, 2, G, T, 3, 3, 4, T, T, 2, 1, 2, T, 5, T, T
20 1, T, 2, 1, G, T, 3, 2, 2, 2, T, T, 4, T, 1, 1, 2, T, 3, 2
21 3, 3, 4, T, 4, T, 3, 2, T, 3, 3, 4, 3, G, 2, 1, 1, G, 3, G
22 T, T, 3, T, 3, G, T, G, 1, 2, T, 2, T, 2, T, 1, G, 1, T, T
23
24 Execution time: 0.0025 seconds

```

```

1 Solution name: Backtracking
2 Result:
3 1, 2, T, G, G, T, T, 2, 1, 3, T, 3, 1, 2, 1, 2, 2, T, 3, 2
4 T, 3, 2, T, 2, G, 3, 3, T, 4, T, 5, T, 3, T, 3, T, 4, T, T
5 T, 3, 2, 2, G, G, 2, T, 4, 4, T, T, 4, T, 4, 5, T, 4, 3, G
6 T, 2, 1, T, 3, 3, 5, T, T, 3, G, 5, T, 5, T, T, 3, T, 3, 2
7 1, G, G, G, T, T, T, T, 4, 2, T, 4, T, T, 5, 4, G, 3, T, T
8 2, 2, 2, T, T, 5, G, T, 5, 3, 3, T, 4, 5, T, T, 4, T, 5, G
9 T, T, G, 4, 4, T, G, T, T, T, 4, 4, T, 3, T, 5, T, T, T, 2
10 T, 5, T, 3, T, 4, T, 5, 5, 4, T, T, 2, G, 1, 4, T, T, T, 2
11 T, 6, 3, 4, T, G, 3, T, T, 4, 3, 2, 1, 1, 2, 5, T, T, 5, 3
12 T, T, T, 4, 3, 2, 4, T, T, T, G, 1, 1, 1, T, T, T, 6, T, T
13 T, T, T, T, 2, T, 4, T, G, 4, 4, T, 4, 4, 5, 6, T, T, G, T
14 5, T, T, 3, 2, G, 3, T, T, T, G, T, T, T, T, T, 4, 4, 3, 2
15 T, T, G, 2, 2, 3, 3, 3, G, T, 4, T, T, 6, T, 4, T, 4, T, 2
16 T, T, 3, 1, T, T, T, 2, 2, G, 2, 3, T, 4, G, 3, G, T, T, 3
17 5, T, 4, G, 4, T, G, T, 3, G, G, 1, 1, 2, T, 2, 3, 4, G, T
18 T, T, T, 2, T, 2, 3, 3, T, T, 2, 2, 2, 2, G, 3, T, T, 5, T
19 G, 4, 3, 2, G, 2, G, T, 3, 3, 4, T, T, 2, 1, 2, T, 5, T, T
20 1, T, 2, 1, G, T, 3, 2, 2, 2, T, T, 4, T, 1, 1, 2, T, 3, 2
21 3, 3, 4, T, 4, T, 3, 2, T, 3, 3, 4, 3, G, 2, 1, 1, G, 3, G
22 T, T, 3, T, 3, G, T, G, 1, 2, T, 2, T, 2, T, 1, G, 1, T, T
23
24 Execution time: 0.2569 seconds

```

```

1 Solution name: Bruteforce
2 No solution found!
3 Execution time: 30.0331 seconds

```

#### 4. Nhận xét

- Brute-force: Hiệu quả thấp, không khả thi cho lưới vừa và lớn do tính chất của thuật toán vét cạn, thời gian thực thi rất lâu.
- Backtracking: Hiệu quả trung bình, phù hợp với lưới có kích thước vừa phải.
- PySAT: Hiệu quả cao nhất, thời gian thực thi nhanh (vài giây đến vài phút) với lưới có kích thước lớn.

#### 5. Link video demo

## 6. Tự đánh giá

Yêu cầu	Điểm	Hoàn thành
1. Solution description: Describe the correct logical principles for generating CNFs,	20%	90%
2. Generate CNFs automatically	10%	100%
3. Use PySAT library to solve CNFs correctly.	10%	100%
4. Program brute-force algorithm to compare with using library(speed).	10%	100%
5. Program backtracking algorithm to compare with using library (speed).	10%	100%
6. Documents and other resources that you need to write and analysis in your report: Thoroughness in analysis and experimentation. Give at least 3 test cases with different sizes (5x5, 11x11, 20x20) to check your solution Comparing result and performance	40%	90%

## IV. Tham khảo

- [1] <https://github.com/pysathq/pysat>
- [2] <https://gist.github.com/davefernig/e670bda722d558817f2ba0e90ebce66f>
- [3] <https://github.com/bgrohman/Brute-Force-Sudoku-Solver>
- [4] Slide in this course
- [5] Claude AI