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

BÁO CÁO

PROJECT 2: COLORING PUZZLE

MÔN HỌC: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

GIÁO VIÊN HƯỚNG DẪN

TS. Nguyễn Hải Minh

TS. Nguyễn Ngọc Thảo

Trợ giảng. Nguyễn Thái Vũ

Thành phố Hồ Chí Minh – 2022

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

ꕥ♠♣♦♥📖♥♦♣♠ꕥ

BÁO CÁO

PROJECT 2: COLORING PUZZLE

MÔN HỌC: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

THÀNH VIÊN NHÓM

20127061 – Lưu Minh Phát

20127166 – Nguyễn Huy Hoàn

20127662 – Nguyễn Đình Văn

Thành phố Hồ Chí Minh – 2022

MỤC LỤC

MỤC LỤC 1
THÔNG TIN CÁC THÀNH VIÊN 2
BẢNG PHÂN CÔNG 3
GIỚI THIỆU VỀ ĐỒ ÁN 4
PHƯƠNG HƯỚNG TIẾP CẬN 5
THỰC NGHIỆM 8
KẾT LUẬN 9
TÀI LIỆU THAM KHẢO 11

THÔNG TIN CÁC THÀNH VIÊN

MSSV	Họ và tên	Chú thích
20127662	Nguyễn Đình Văn	20127662@student.hcums.edu.vn
20127166	Nguyễn Huy Hoàn	20127166@student.hcums.edu.vn
20127061	Lưu Minh Phát	20127061@student.hcums.edu.vn

BẢNG PHÂN CÔNG

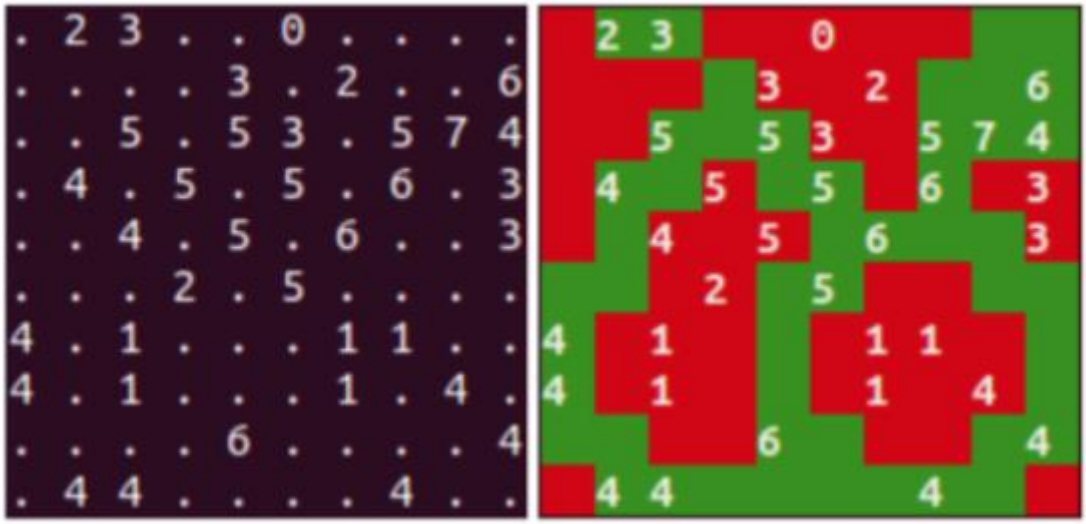
MSSV – Họ tên	Công việc	Mức độ hoàn thành
20127662 – Nguyễn Đình Văn	Phân chia công việc cho các thành viên trong nhóm. Code thuật toán PySat Code thuật toán A* Tạo CNF	90%
20127166 – Nguyễn Huy Hoàn	Code thuật toán Backtracking Code thuật toán BruteForce Chạy thực nghiệm Demo	90%
20127061 – Lưu Minh Phát	Thiết kế giao diện GUI, kiểm tra thuật toán cần chạy Tạo các test để thực thi và kiểm tra lỗi Viết báo cáo	90%

➔ Tuy vẫn có lỗi hay một số vấn đề phát sinh còn nhiều nhưng nhìn chung các công việc đều được hoàn thành khá tốt.

GIỚI THIỆU VỀ ĐỒ ÁN

Chủ đề: Coloring Puzzle

Nhóm được yêu cầu xây dựng một bộ giải để giải quyết Coloring Puzzle bằng cách chuyển đổi first order logic (FOL) thành CNF như mô tả sau: Cho một ma trận kích thước mxn, mà mỗi ô (cell) chứa giá trị không âm hoặc rỗng. Mỗi ô được coi là liền kề với chính nó và 8 ô bao xung quanh. Vấn đề của nhóm là cần tô màu tất cả các ô của ma trận với cả màu xanh hoặc đỏ, sao cho số bên trong mỗi ô tương ứng với số ô được tô màu (đỏ hoặc xanh) liền về với ô đó



Mức độ hoàn thành: 80%

PHƯƠNG HƯỚNG TIẾP CẬN

I. Tự động tạo CNFs

a). Ý tưởng

Đối với bài toán Color Puzzle ta tạo CNF bằng việc xét ràng buộc của mỗi ô chứa số với các ô xung quanh.

Với mỗi ô số có 9 ô phải xét và tô màu 8 ô xung quanh và bản thân ô chứa số, đặt theo thứ tự từ trên xuống, trái sang phải: 1, 2, 3, 4, ,5 ,6, 7, 8, 9.

Giả sử như xét ô chứa số là 3 thì ta sẽ chọn ra 3 ô trong số 9 ô sao cho thỏa mãn các điều kiện sau, để giá trị trong ô số bằng số ô màu xanh xung quanh ô số đó (bao gồm bản thân ô số đó).

- Nếu ta chọn 4 ô trong số 9 ô thì ta sẽ được ít nhất 1 ô không được chọn. Vậy ta sẽ có 9C4 mệnh đề CNFs:

  $(\neg 1|\neg 2|\neg 3|\neg 4)\&(\neg 1|\neg 2|\neg 3|\neg 5)\& \ldots \&(\neg 6|\neg 7|\neg 8|\neg 9)$

- Vì 3 ô được chọn nên sẽ có 6 ô không được chọn. Nếu ta chọn 7 ô trong số 9 ô thì sẽ có ít nhất 1 ô là được chọn. Vậy ta sẽ có 9C7 mệnh đề CNFs:

  $(1|2|3|4|5|6|7) \cap (2|3|4|5|6|7|8) \cap \ldots \cap (1|4|5|6|7|8|9)$

Sau khi kết hợp hai ràng buộc trên ta được CNF mong muốn thỏa yêu cầu đề bài.

Kết luận: Vậy muốn tạo CNF với mỗi ô số, ta xác định số ô xung quanh có thể đặt được (n) và giá trị của ô số (x). Tạo CNF với nC(x + 1) và nC(n – x + 1).

b) Sơ lược về tổ chức code

Nhóm chúng em tạo ra một class Color_Puzzle chứa các giải thuật bài toán Color Puzzle và GUI.

- *Các thư viện được sử dụng:*
  - Thư viện tkinter: thiết kế GUI
  - Thư viện pysat để giải CNF bằng thuật toán PySAT
  - Thư viện time để chỉnh delay
  - Thư viên itertool thư viện hỗ trợ việc lấy tổ hợp
- *Một số biến quan trọng*
  - *color*: list các điểm màu, mỗi khi chạy thuật toán bất kỳ thì sẽ gán lại cho danh sách (list) trong đó 0 – Màu đỏ, 1 – Màu xanh.

- *info*: list chứa các ô số để ta có thể thể truy suất thông tin khi thực hiện tính toán.
- clauses: list chứa các mệnh đề CNFs
- heur: chứa giá trị Heuristic của thuật toán A*

- *Một số hàm quan trọng:*
  - *makeCNF():* thực hiện tìm và gán các mệnh đề CNFs vào list clauses để dùng cho việc tính toán sau này.
  - *pySat():* thực hiện giải thuật pySat và trả kết quả về list color
  - *calcH(cell):* nhận vị trí của ô và tính toán và trả về giá trị heuristic của ô hiện tại.
  - *AStar():* Thực hiện giải thuật A* và trả kết quả về list color
  - *SAT():* sẽ kiểm tra trạng thái hiện tại của list color có phải là giải pháp cần tìm hay không nếu có trả về True nếu không trả về False
  - *Backtracking():* Thực hiện giải thuật Backtracking và trả kết quả về list color
  - *bruteForce():* Thực hiện giải thuật BruteForce và trả kết quả về list Color

II. Các thuật toán

❖ **PySat:**
- *Ý tưởng:* Sử dụng các hàm có sẵn của thư viện pySat và thêm vào các mệnh đề của CNFs (Các ô được đánh dấu bằng một giá trị ID trong CNF ID = X * số cột + Y + 1 – Với X, Y là vị trí của ô hiện tại).
- *Kết quả:* Luôn cho ra kết quả đúng và nhanh chóng gần như ngay lập tức

❖ **A*:**
- *Ý tưởng:* Duyệt từng ô chứ số để xác định các ô xung quanh của nó và Heuristic của từng ô đó (Heuristic được tính bằng cách xác định dựa vào các mệnh đề CNF đã thực hiện trước đó, với ô mà có càng nhiều ràng buộc thì độ ưu tiên càng cao – giá trị Heuristic càng thấp). Sau đó thêm những ô vào hàng chờ theo thứ tự của H. Và tiến hành lặp đi lặp lại cho đến khi ra kết quả.
- *Kết quả:* Cho ra kết quả đúng, tuy nhiên với input càng lớn hơn 5x5 thì kết quả ra được càng lâu.

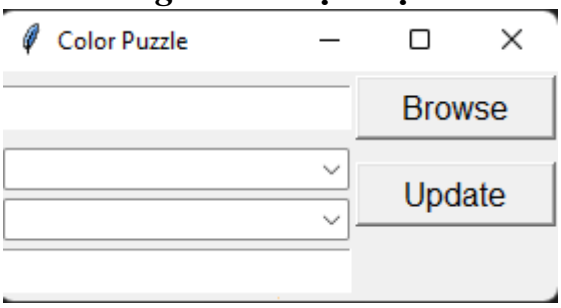
❖ **Backtracking**
- *Ý tưởng:* Duyệt từng ô số và tìm ra các ô tổ hợp xung quanh của ô số đó với mỗi tổ hợp ta chọn các ô đó và tiến hành đệ qui tiếp cho đến khi tìm ra giải pháp. Nếu không tìm ra thì sẽ bỏ chọn các ô tổ hợp đó và qua tổ hợp tiếp theo. Và cứ tiến hành đệ qui cho đến khi tìm ra giải pháp.
- *Kết quả:* Cho ra kết quả đúng, tuy nhiên cũng tương tự A* với input càng lớn thì kết quả càng lâu

❖ **Bruce Force**

- *Ý tưởng:* Duyệt tất cả các ô có trong ma trận tiến hành với mỗi ô ta tiến hành đệ qui 2 lần để tìm ra giải với với lần 1 là ô hiện tại không được chọn và lần 2 là ô hiện tại được chọn (Đây là một giải thuật vét cạn, nó sẽ duyệt tất cả các trường hợp có thể xảy ra). Và với mỗi ô ta tiến hành lặp đi lặp lại cho đến khi tìm thấy giải pháp.
- *Kết quả:* Cho ra kết quả đúng, tuy nhiên cũng tương tự A* và Backtracking với input càng lớn thì kết quả càng lâu.

III. GUI

a) Chức năng có thể thực hiện



Lấy dữ liệu từ một đường link, thông qua trình duyệt (nút Browse)

Hiển thị ma trận kích thước của tệp vào, hiện cái ô có số và ô rỗng thành công

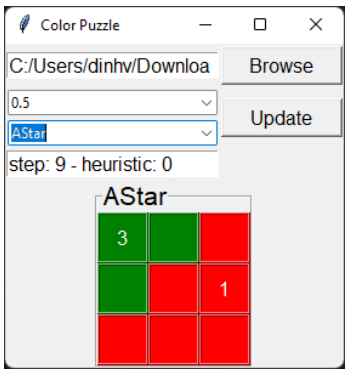
Thanh chọn thuật toán ('AStar', 'Backtracking', 'Brute force', 'PySAT')

Thanh chọn delay time ('0.5', '0.75', '1.0', '1.25', '1.5') và mặc định là 0.5 nếu không chọn

Có thể thay đổi màu khi đang chạy thuật toán bất kỳ tuy nhiên, một số trường hợp có thể không hiện thay đổi màu liên tục. Tuy nhiên kết quả cuối cùng được thể hiện màu chính xác.

Thanh thể hiện step và heuristic hoạt động tuy nhiên một số trường hợp step chạy nhưng có thể bảng màu không thay đổi.

Kết quả giao diện sau khi chạy:



b) Chưa thể hiện thực hiện được

Nút dừng làm cho chương trình dừng ở một trường hợp nào đó để xem rõ bảng màu.

THỰC NGHIỆM

Sau quá trình chạy thực nghiệm ta có bảng sau:

Test case	PySat	A*	Backtracking	BruteForce
3x3	0.151s	3.429s	0.508s	35.202s
5x5	0.142s	Too long (>20')	9.487s	Too long (>30')
9x9	0.282s	Too long (>30')	Too long (>30')	Too long (>30')
11x11	0.481s	Too long (>30')	Too long (>30')	Too long (>30')
15x15	0.666s	Too long (>30')	Too long (>30')	Too long (>30')
20x20	2.192s	Too long (>30')	Too long (>30')	Too long (>30')

Nhận xét:

Đối với pySat do sử dụng các hàm có sẵn của thư viện PySat. Kết quả cho được chính xác và trong khoảng thời gian rất nhanh, lâu hơn 1 chút với bộ test 15x15, 20x20 nhưng không đáng kể.

Đối với A* và Backtracking thì Backtracking cho ra kết quả nhanh hơn A*.

Đối với A* và BruteForce thì A* cho kết quả nhanh hơn.

Đối với 3 thuật toán A*, Backtracking và BruteForce thì Backtracking nhanh nhất và chậm nhất là BruteForce (Speech – Backtracking < A* < BruteForce). Từ 5x5 trở lên thì thời gian chạy rất lâu.

Vì A*, Back Tracking vẫn xảy ra một số bug nên với test lớn chạy khá lâu và có thể chạy không ra. Brute force chạy bình thường nhưng do duyệt hết nên thời gian rất lâu.

KẾT LUẬN

Thông qua đồ án lần này nhóm chúng em rút ra được bài học và kết luật như sau:

1. Học về cách chuyển đổi các mệnh đề về dạng CNFs bằng ngôn ngữ lập trình (phụ thuộc vào độ phức tạp của các ràng buộc).
2. Biết về thư viện, công cụ hỗ trợ mạnh mẽ để giải bài toán (PySat) cho kết quả chính xác và vô cùng nhanh chóng.
3. Phân biệt tốc độ xử lý, sự khác nhau giữa các thuật toán thông qua các thực nghiệm để so sánh.
4. Nhận ra được điểm mạnh và hạn chế của từng thuật toán.

LINK DEMO

https://drive.google.com/drive/folders/1ZpUGRdF9ALZMvn-YRhPC38HT8GXhkq5t?usp=sharing

TÀI LIỆU THAM KHẢO

https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/solving.html

https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/cnf2.html