

ASSESSMENT

COACHING

ARTIFICIAL
INTELLIGENCE

Các thành viên trong team

Họ và tên

MSSV

Lê Quang Tùng

1810784

Trương Công Thành

1810766

Vũ Minh Dương

1810885

Lê Long

1812881

Mục lục

we are here!



01 N puzzle - Depth First Search

02 Sudoku - Genetic Algorithm

03 Path Finding - A* Algorithm

N Puzzle

State

Sinh random một Puzzle có thể giải được (sinh ngẫu nhiên từ trạng thái đích bằng cách hoán đổi ngẫu nhiên giá trị 0 với giá trị kề nó sau một số lần).

N Puzzle

Initial State

Ví dụ:

$N = 3$

Puzzle sinh ra (ngẫu nhiên):

1	7	8
2	3	4
5	6	0

N Puzzle

Goal State

Ví dụ:

$N = 3$

Puzzle sinh ra (ngẫu nhiên):

[0 1 2]

[3 4 5]

[6 7 8]

N Puzzle

Legal Move

Những hướng (trên, dưới, trái, phải) mà giá trị 0 có thể hoán đổi được (không hoán đổi với giá trị ngoài bảng)

N puzzle - DFS (Recursion)

DFS(state):

list_action: Lưu trữ các hướng trong lúc đệ quy

Mục đích dùng để in kết quả

Đánh dấu state đã được duyệt

Nếu state là trạng thái kết thúc:

in kết quả theo list_action và dừng

Duyệt các hướng(dir) có thể trong LegalMoves(state):

Trạng thái mới khi đi theo hướng (dir)

next_state = result(state, dir)

Nếu next_state chưa được duyệt:

Thêm dir vào list_action

DFS(next_state)

Xóa dir ra khỏi list_action

N puzzle - DFS (No Recursion)

Vì thuật toán đệ quy sẽ sử dụng rất nhiều bộ nhớ của stack mà stack memory của python thì rất nhỏ, cho nên thuật toán sử dụng đệ quy không chạy được do tràn bộ nhớ, vì độ sâu của thuật toán DFS sẽ đi theo hàm giao thừa của N^N . Do đó, chúng ta sẽ dùng phương pháp khử đệ quy để khắc phục điều này.

Mục lục

01 N puzzle - Depth First Search

we are
here!



02 Sudoku - Genetic Algorithm

03 Path Finding - A* Algorithm

Initial State - Generate

Sinh ngẫu nhiên một bảng Sudoku bằng cách điền ngẫu nhiên một Sudoku thỏa mãn yêu cầu của một trò Sudoku 9×9 thông thường (các số trong bảng là các số từ 1-9 các lưới 3×3 , dòng, cột đều khác nhau). Sau đó ta xóa bớt các giá trị trong bảng để tạo ra một trạng thái ban đầu có trạng thái khởi đầu.



Initial State - Example

VD: vẽ bảng Sudoku 9*9 định dạng bằng mảng 2 chiều ô trống mang giá trị = 0

5	3			7				
6			1	9	5			
	9	8				6		
8			6					3
4		8	3					1
7			2					6
	6			2	8			
		4	1	9				5
			8		7	9		



Goal State

VD: về bảng Sudoku 9*9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



SUDOKU

Some definition

Chromosome (cá thể): Một cá thể ứng với một trạng thái được khởi tạo đầy đủ của một ma trận 9×9 .

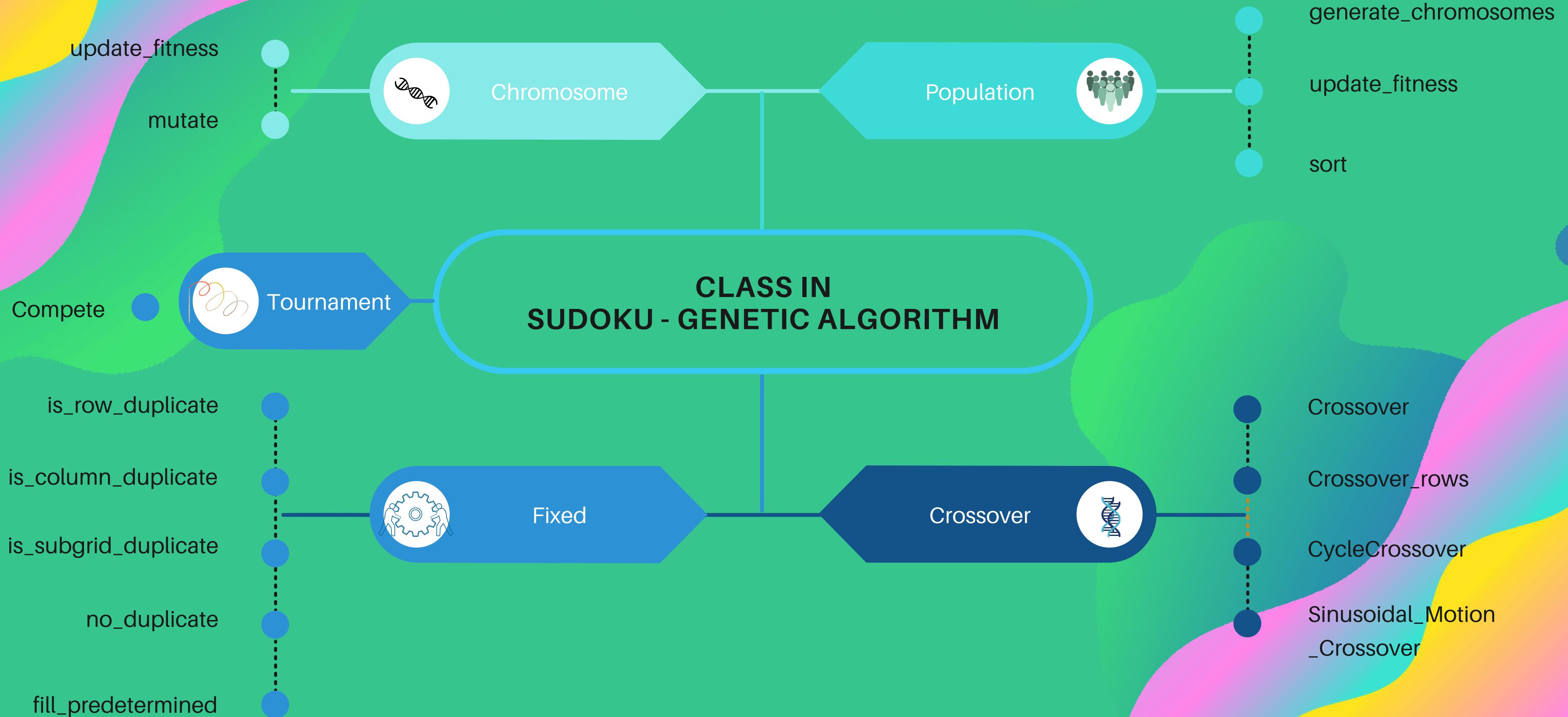
Gene: Là một gen của một cá thể, ở bài toán này ta xét một gen là một hàng (1×9) của ma trận.

Population: Một quần thể gồm n cá thể (n cho trước).



SUDOKU

SUDOKU - Genetic Algorithm



SUDOKU - Genetic Algorithm

- 1 Lấy trạng thái ban đầu từ tệp tin dữ liệu.
- 2 Khởi tạo các thông số như: kích thước quần thể, kích thước elite, số lượng thế hệ,...
- 3 Sinh ra n cá thể cho thế hệ đầu tiên.
- 4 Tính độ “tốt” tốt nhất của quần thể hiện tại.
- 5 Lai tạo và đột biến các cá thể trong quần thể (Tạo ra cá thể mới và giữ lại các cá thể cũ tốt)
- 6 Nếu chưa tìm ra lời giải hoặc đạt tới ngưỡng số thế hệ cho trước thì dừng.
(Bài toán không giải được). Ngược lại, thì là đáp án của bài toán.

Mục lục

01 N puzzle - Depth First Search

02 Sudoku - Genetic Algorithm

03 Path Finding - A* Algorithm

we are
here!





Path Finding

State

Là tọa độ của một điểm bất kì thỏa:

- Điểm đó nằm trong bảng
- Tại điểm đó không phải là vật cản



Path Finding

Initial State

Cho điểm bắt đầu có tọa độ là (x_0, y_0) , như vậy trạng thái bắt đầu là (x_0, y_0)

Goal State

Cho điểm kết thúc có tọa độ là (x_n, y_n) , như vậy trạng thái kết thúc là (x_n, y_n)





Legal Move

Cho trạng thái hiện tại là (x,y) , các ô có thể đi là

- Lên trên $(x-1,y)$
- Xuống dưới $(x+1,y)$
- Sang trái $(x,y-1)$
- Sang phải $(x,y+1)$

nếu các ô trên nằm trong bảng và không phải vật cản

Finding Path Giải thuật A*

HÀM LƯỢNG GIÁ

HEURISTIC:

$$\text{Heuristic } (x,y) = \text{Abs}(x_n - x) + \text{Abs}(y_n - y)$$

COST:

Node color: White, step cost = 1

Path cost $(x_0, y_0) \rightarrow (x, y) =$

$\text{Path_cost}(x_0, y_0) \rightarrow (x', y') + \text{step cost } (x', y') \rightarrow (x, y)$

với (x', y') là tọa độ ô kế cận (x, y)

F_COST:

- fringe: Lưu node đã xử lý ngoài biên và F_cost của node
- frontier: Lưu node ngoài biên chưa xử lý
- explored: Lưu node đã xử lý

Finding Path - Giải thuật A*

1. Đưa start vào fringe, $f_cost = \text{heuristic}(\text{start})$
2. Đưa start vào frontier
3. Nếu frontier chưa rỗng:
 - a. $\text{current} = \text{frontier.pop()}$
 - b. Đưa current vào explored
 - c. Xóa current khỏi frontier
 - d. Nếu $\text{current} == \text{end}$, truy vết ngược về start, kết thúc giải thuật
 - e. Lấy neighbor của current dựa trên legal move, với mỗi neighbor không nằm trong explored dựng path_cost:
 - i. Nếu neighbor không nằm trong frontier: dựng f_cost , đưa vào fringe và frontier
 - ii. Nếu neighbor đã nằm trong frontier: so sánh path_cost mới với path_cost cũ của neighbor, update fringe và frontier



THANK YOU!

Thanks for watching :)