ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

CƠ SỞ TRÍ TUỆ NHÂN TẠO

LAB01: N-queens problem

Sinh viên thực hiện : 21127083_Hoàng Đức Kiên

Lớp : 21CLC03

Giáo viên hướng dẫn : Nguyễn Ngọc Thảo

Lê Ngọc Thành

Nguyễn Trần Duy Minh

Nguyễn Hải Đăng

Mục lục

1. Tổng quan

Bài toán N-quân hậu với trạng thái ban đầu là các quân hậu nằm trên các cột khác nhau cùng với đó mỗi trạng thái con của bàn cờ được tạo ra từ việc di chuyển một quân hậu. Để tìm ra lời giải cho bàn toán này nghĩa là ta cần tìm ra bàn cờ hợp lệ là không có bất kì căp quân hậu nào tấn công nhau trên cùng hàng , cột, đường chéo chính-phụ. Và để giải quyết bài toán này ta sẽ sử dụng các thuật toán tìm kiếm :

- Uniform-cost search.
- Graph-search A* với hàm MIN-CONFLICT heuristic: “Số cặp quân hậu tấn công nhau”.
- Genetic algorithm với hàm objective function được định nghĩa từ hàm heuristic.

2. Cài đặt chương trình

Thư viện sử dụng

_Import random : sử dụng các hàm random.

_Import heapq : sử dụng cấu trúc dữ liệu heap.

_Import time : sử dụng các hàm tính thời gian để thuật toán chạy

_Import statistics : sử dụng hàm tính giá trị trung bình.

_Import tracemalloc : sử dụng hàm tính bộ nhớ đã sử dụng trong quá trình chạy thuật toán.

_From typing import List, Tuple, Callabe : sử dụng List , Tuple , Callabe

_From itertools import combinations : sử dụng để tạo các tổ hop745 của một danh sách.

1.1 Uniform-cost search

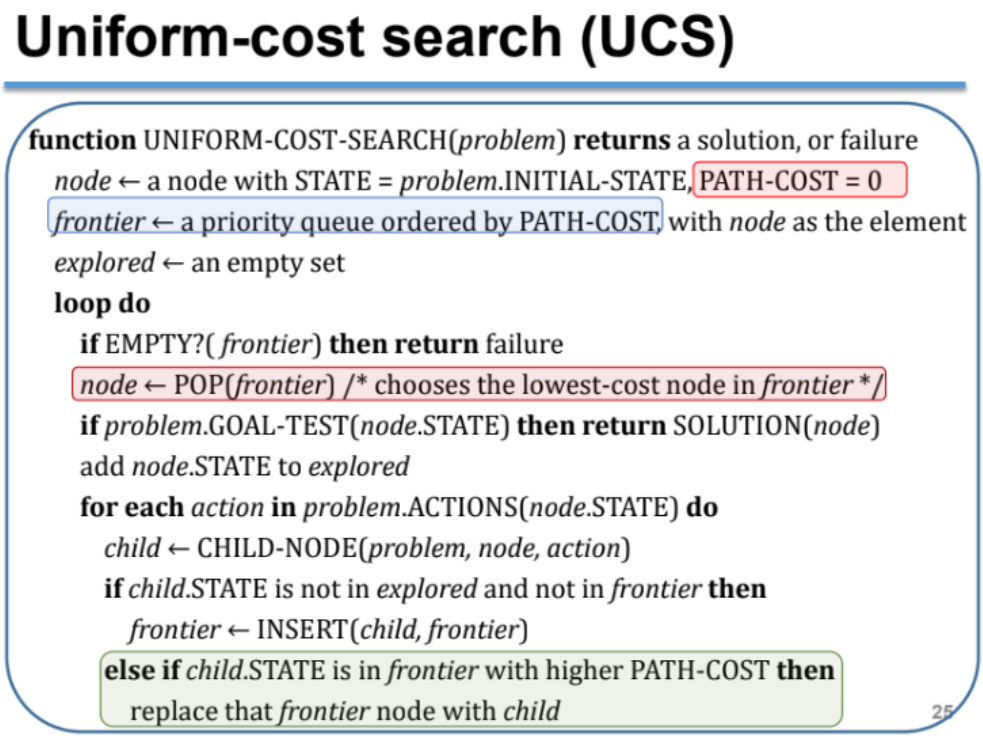
Các hàm được sử dụng :

_def is_valid_state() : kiểm tra tính hợp lệ của bàn cờ.

_def generate_successor_states() : tạo ra danh sách các trạng thái con khi di chuyển 1 quân hậu.

_def uniform_cost_search() : tìm kiếm bàn cờ hợp lệ

Mã giả :



_Frontier : sử dụng cấu trúc dữ liệu heap để lưu các bàn cờ (giá trị path-cost,bàn cờ). Ta thêm giá trị path-cost mặc dù ta không cần đến để khi được thêm vào heap , heap sẽ tự động sắp sếp theo thứ tự tăng dần theo phần tử đầu tiên của nó , nên thứ tự của bàn cờ trong frontier sẽ theo thứ tự lần lượt thêm bàn cờ vào.

_Explored : sử dụng phương thức set() để lưu các bàn cờ đã được pop ra khỏi frontier vì set() chỉ chứa các phần tử khác nhau nên khi gặp các bàn cờ đã tồn tại trong explored thì nó sẽ không thêm bàn cờ vô.

_Path – cost sẽ là như nhau , bằng 1 , dẫn đến việc UCS sẽ không quan tâm đến path-cost , sẽ chạy giống với BFS nhưng thuật toán sẽ dừng cho đến khi goal được pop ra khỏi frontier.

1.2 Graph search A*

Các hàm được sử dụng :

_def min_conflict_heuristic() : tính giá trị heuristic là các cặp quân hậu tấn công nhau.

_def generate_successor_states() : tạo ra danh sách sách các trạng thái con khi di chuyển 1 quân hậu.
_def graph_search_a_star() : tìm kiếm bàn cờ hợp lệ.

Mả giã : sẽ tương tự như thuật toán UCS nhưng chỉ khác cách thức tính giá trị path – cost.

_Frontier : sử dụng cấu trúc dữ liệu heap để lưu các bàn cờ với cấu trúc là 1 tuple (giá trị heuristic,bàn cờ). Khi thêm vào heap thì heap sẽ tự động sắp xếp theo thứ tự tăng dần của phần tử đầu tiên của nó – giá trị heuristic.

_Explored : sử dụng phương thức set() để lưu các bàn cờ đã được pop ra khỏi frontier vì set() chỉ chứa các phần tử khác nhau nên khi gặp các bàn cờ đã tồn tại trong explored thì nó sẽ không thêm bàn cờ vô.

_Path – cost lúc này sẽ là giá trị heuristic cũa trạng thái con của mỗi bàn cờ.

1.3 Genetic algorithm

Các hàm sử dụng:

_def initializ_population() : tạo danh sách quần thể.

_def compute_fitness() : tính điểm thích nghi của quần thể dựa trên nội dung hàm heuristic là số cặp quân hậu tấn công nhau , trả về danh sách điểm thích nghi.

_def select_parents() : hàm sẽ chọn ra hai bàn cờ cha mẹ từ quần thể hiện tại để tạo ra bàn cờ con mới trong quá trình lai ghép, thuật toán ‘roulette wheel”

để tính xác suất được chọn bằng tỷ lệ điểm thích nghi của nó với tổng điểm thích nghi của quần thể.

_def crossover() : giao phối cha mẹ được chọn ngẫu nhiên để tạo ra một con. Chọn một điểm cắt ngẫu nhiên trong dãy gen của bố mẹ, đưa dãy gen phía sau điểm cắt của mẹ ghép vào dãy gen trước điểm cắt của bố để tạo ra đời con.

_def mutate() : đột biến điểm, nếu giá trị ngẫu nhiên được sinh ra nhỏ hơn mutation_rate thì giá trị của phần tử đó sẽ bị đột biến thành một giá trị ngẫu nhiên mới nằm trong khoảng từ 0 đến n-1.

_def genetic_algorithm() :

Khởi tạo quần thể ban đầu.

Vòng lặp for duyệt từng cá thể trong quần thể từ 0 đến 1000.

- Tính điểm thích nghi của mỗi cá thể trong quần thể.
- Kiểm tra xem bàn cờ tốt nhất trong thế hệ hiện tại có số lượng xung đột có bằng 0 hay không. Nếu có thì trả về bàn cờ hợp lệ, không thì tiếp tục vòng lặp.
- Chọn ba mẹ để tiến hành lai ghép.
- Tạo cá thể con.
- Đánh giá điểm thích nghi
- Chọn ra các cá thể sinh tồn của thế hệ mới bao gồm cá thể con và các cá thể trong quần thể hiện tại bằng cách sắp xếp tăng dần điểm thích nghi.

Kết thúc vòng lặp và trả về trạng thái tốt nhất

Các giá trị được gán mặc định :

- Pop_size = 100: số lượng quần thể
- Generation = 1000: số lượng cá thể sẽ lấy ra từ quần thể
- Mutation_rate = 0.1 : xác suất đột biến.

3. Số liệu thống kê

	Running time (ms)			Memory (MB)		
Algorithms	N = 8	N = 100	N = 500	N = 8	N = 100	N = 500
UCS	9054.4171	Intractable	Intractable	0.3268	Intractable	Intractable
A*	14.9891	1969581.4401	Intractable	0.05014	0.167786	Intractable
GA	1107.6623	Intractable	Intractable	0.2171	Intractable	Intractable

Từ bảng số liệu thu thập được từ việc chạy các thuật toán , ta thấy rằng :

- Uniform – cost search

_ Với N = 8 thì thuật toán UCS tốn nhiều thời gian nhất cũng như là bộ nhớ so với 2 thuật toán còn lại vì nó phụ thuộc vào trạng thái ban đầu của bàn cờ khi bàn cờ được tạo ngẫu nhiên. Cùng với đó là các trạng thái con được tạo ra theo cấp số mũ và được thêm lần lượt vào nên tốn rất nhiều bộ nhớ.

_ Với N = 100 , 500 , thuật toán UCS tốn nhiều bộ nhớ đến mức tràn RAM nên ta không thể thu được số liệu khi chạy.

- Graph – search A^*

_ Với N = 8 thì thuật toán A^* tốn ít thời gian nhất cũng như là bộ nhớ so với hai thuật toán còn lại vì với mỗi trạng thái con của bàn cờ thì luôn được tính giá trị heuristic ứng với từng trạng thái con đó , và các trạng thái con được sắp xếp theo thứ tự tăng dần của giá trị heuristic , nên là ta sẽ luôn xét các trạng thái con có giá trị heuristic nhỏ nhất trước là các trạng thái con có số cặp quân hậu tấn công nhau ít nhất nên sẽ tiết kiệm rất nhiều thời gian và bộ nhớ để giải được bài toán.
_ Và với N = 100 , sau một khoảng thời gian tương đối dài thì cũng đã có thể giải được bài toán. Nhưng với N = 500 thì lại tốn nhiều bộ nhớ đến mức tràn RAM nên ta không thể thu được số liệu khi chạy.

- Genetic Algoritm

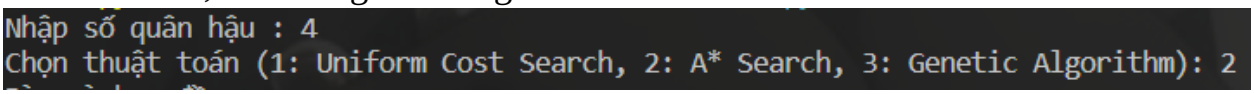
_ Với N = 8 , thì thuật toán genetic Algoritm có thời gian chạy và bộ nhớ được sử dụng ở mức trung bình so với 2 thuật toán còn lại. Bằng việc chọn lọc những cá thể có giá trị thích nghi cao để tiến hành lai ghép , đột biến điểm để từ đó tạo ra thế con tốt hơn nên thời gian chạy sẽ tương đối nhanh hơn so với UCS nhưng vẫn không nhanh bằng thuật toán A^*. Đồng thời việc chọn các giá trị về kích thước quần thể (pop_size) , số lượng cá thể trong quần thể (generations) , tỉ lệ độ biến (mutation_rate) cũng ảnh hưởng đến việc giải quyết bài toán.
_ Với N = 100 , N =500 thì thời gian chạy quá lâu so với mức cho phép nên ta không thể thu được số liệu này.

4. Những công việc đã hoàn thành và chưa hoàn thành

Công việc đã hoàn thành	Công việc chưa hoàn thành
_ Cài đặt đủ cả 3 thuật toán và chú thích từng thuật toán. _ Các thuật toán đều chạy được và mỗi thuật toán đều được chú thích rõ ràng trong file. _Cài đặt giao diện đơn giản dễ sử dụng	_ Vẫn chưa thể tối ưu thuật toán để có thể chạy với N = 100 , 500

5. Cách thức chương trình chạy

1. Nhập số lượng quân hậu (N)
2. Chọn thuật toán để chạy. Nhập 1 nếu là uniform-cost search , 2 nếu là A* search , 3 nếu là genetic algorithm.



Nguồn tham khảo

https://arxiv.org/ftp/arxiv/papers/1802/1802.02006.pdf

https://brilliant.org/wiki/a-star-search/