Bài thuyết trình Quá trình 2

Môn: Nhập môn trí tuệ nhân tạo

Nhóm: 8

Người hướng dẫn: **Ths Nguyễn Thành An**

Câu 1: Problem

Đề bài yêu cầu ta xử lý một bài toán tìm kiếm trong không gian 2D, trong đó mỗi trạng thái được đại diện bởi các giá trị x,y và giá trị evaluation của trạng thái ứng với giá trị z.

Ta có các phương thức chính của lớp problem để giải quyết bài toán như sau:

- **init(self, filename)**: Phương thức khởi tạo nhận vào một tên file ảnh, sử dụng OpenCV để đọc và xử lý ảnh thành không gian trạng thái 2D.
- **load_state_space(self, filename)**: Phương thức này sử dụng OpenCV để đọc ảnh từ file, sau đó thực hiện các bước tiền xử lý như resize và Gaussian blur để tạo không gian trạng thái.
- **show(self)**: Phương thức này trực quan hóa không gian trạng thái 3D bằng matplotlib và hiển thị đường cong 3D.
- **draw_path(self, path)**: Phương thức này nhận vào một đường đi (path) và vẽ nó trên mặt phẳng trạng thái.

Câu 1: Problem(cont)

- **get_random_state(self)**: Phương thức này trả về một trạng thái ngẫu nhiên trong không gian trạng thái.
- **evaluate_state(self, current_state)**: Phương thức này đánh giá một trạng thái bằng cách trả về giá trị z.
- **get_the_best_neighbor(self, current_state):** Phương thức này tìm và trả về láng giềng tốt nhất của một trạng thái, dựa trên giá trị z.
- **initial_state(self):** Phương thức này tạo ra một trạng thái ban đầu ngẫu nhiên và trả về cặp (state, state_value).
- **random_neighbor(self, current_state):** Phương thức này trả về một láng giềng ngẫu nhiên của một trạng thái cùng với giá trị của láng giềng đó.
- **get_k_randomly_state(self, k):** Phương thức này tạo ra một danh sách gồm k trạng thái ngẫu nhiên trong không gian trạng thái.

Câu 1: Problem(cont)

- **neighbors(self, current_state):** Phương thức này trả về tất cả các láng giềng hợp lệ của một trạng thái.

Câu 2: Random Restart Hill-Climbing

- **Mục tiêu:** Mục tiêu của thuật toán Random Restart Hill Climbing là tìm kiếm một giải pháp tốt nhất trong không gian trạng thái bằng cách sử dụng phương pháp leo đồi (hill climbing), nhưng với một số lần khởi đầu ngẫu nhiên (random restarts).
- **Quy Trình:**
  - **Khởi tạo**: Bắt đầu với một trạng thái ngẫu nhiên trong không gian trạng thái.
  - **Leo đồi (Hill Climbing)**: Từ trạng thái khởi tạo, thuật toán tìm kiếm các trạng thái láng giềng và chọn trạng thái có giá trị đánh giá (evaluation value) cao nhất. Quá trình này tiếp tục cho đến khi không còn trạng thái nào láng giềng có giá trị đánh giá cao hơn.
  - **Kiểm tra điều kiện dừng**: Kiểm tra xem trạng thái hiện tại có là giải pháp tốt nhất hay không. Nếu có, thuật toán kết thúc. Ngược lại, chuyển sang bước tiếp theo.
  - **Random Restart**: Lặp lại quá trình trên với một trạng thái khởi tạo ngẫu nhiên khác.
  - **Kết thúc**: Thuật toán kết thúc khi đã thực hiện đủ số lần khởi đầu ngẫu nhiên hoặc khi đã đạt được một giải pháp chấp nhận được.

Câu 2: Random Restart Hill-Climbing(cont)

- **Mục tiêu cuối cùng** của thuật toán Random Restart Hill Climbing là tìm ra một giải pháp tốt nhất trong không gian trạng thái mặc dù có thể bị rơi vào các cục bộ tối ưu trên đường đi. Bằng cách thực hiện nhiều lần khởi đầu ngẫu nhiên, thuật toán có cơ hội lớn hơn để thoát ra khỏi các điểm cục bộ và tiến tới giải pháp toàn cục tốt nhất.
- **Ưu điểm:**
  - **Dễ triển khai:** Thuật toán RRHC không đòi hỏi quá nhiều tài nguyên tính toán và có cấu trúc đơn giản, dễ triển khai trong nhiều bài toán tìm kiếm.
  - **Dễ dàng tìm được giải pháp tốt nhất trong các lần chạy:** Nhờ vào việc thực hiện nhiều lần khởi đầu ngẫu nhiên, thuật toán có khả năng cao để tìm ra giải pháp tối ưu toàn cục, ít bị ảnh hưởng bởi các điểm cục bộ.
  - **Khả năng thoát khỏi cục bộ tối ưu:** Bằng cách khởi đầu lại từ các điểm khác nhau, thuật toán có khả năng cao để thoát ra khỏi các điểm cục bộ và tiến tới giải pháp toàn cục.
  - **Hiệu quả với không gian trạng thái lớn:** RRHC hoạt động tốt với các không gian trạng thái lớn mà không cần lưu trữ toàn bộ không gian trạng thái trong bộ nhớ.

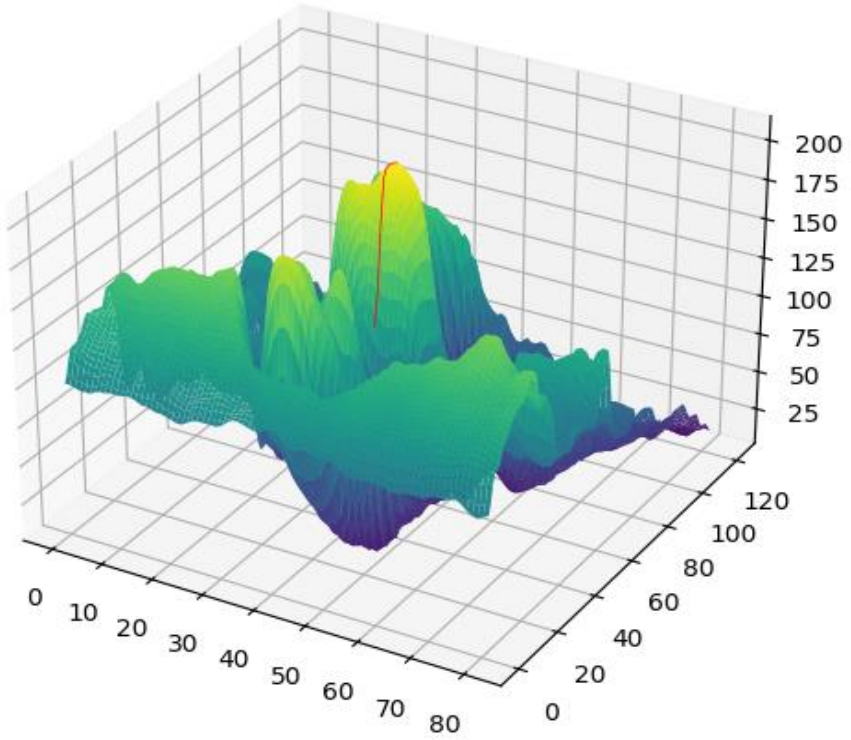
Câu 2: Random Restart Hill-Climbing(cont)

- **Nhược điểm:**
  - **Tốn thời gian:** RRHC có thể tốn nhiều thời gian khi cần thực hiện nhiều lần khởi đầu ngẫu nhiên, đặc biệt là trong các không gian trạng thái lớn.
  - **Không hiệu quả với bài toán có nhiều cục bộ:** Trong các bài toán có nhiều điểm cực đại cục bộ, RRHC có thể không hiệu quả vì việc thực hiện nhiều lần khởi đầu ngẫu nhiên cũng không đảm bảo tìm ra giải pháp toàn cục.
  - **Không đảm bảo tìm kiếm toàn cục tối ưu:** Mặc dù RRHC giúp giảm thiểu nguy cơ rơi vào cực tiểu cục bộ, nhưng không có đảm bảo tìm kiếm được giải pháp toàn cục tối ưu.

Câu 2: Random Restart Hill-Climbing (psuedocode)

```
function random-restart-hill-climbing(problem, num-trial) returns path
    path ← an empty list that stores a unique path from the starting point until the local maximum is found
    cur-best-state ← stores the best state during algorithm execution
    for each trial from 1 to num-trial do
        cur-path ← an empty list to store the path for the current trial
        current-state ← MAKE-NODE(problem.get-random-state)
        append current-state to cur-path
        loop do
            the-best-neighbor ← a highest-valued successor of current
            if the-best-neighbor is None then
                if problem.evaluate-state(current-state) > current-best-state then
                    cur-best-state ← problem.evaluate_state(current-state)
                    path ← cur-path
                break the loop
            current-state ← the-best-neighbor
            append current-state to cur-path
    return path
```

Câu 2: Random Restart Hill-Climbing



Hình 1: Biểu đồ trực quan Random Restart Hill-Climbing Search

Câu 3: Simulated Annealing Search

- **Mục tiêu**: thuật toán Simulated Annealing là tìm kiếm một giải pháp tối ưu (hoặc gần tối ưu) trong một không gian trạng thái lớn, trong khi vượt qua các cực tiểu cục bộ và tránh rơi vào các điểm cố định. Cụ thể, thuật toán này cố gắng mô phỏng quá trình luyện kim(annealing) trong lĩnh vực vật lý. Ý tưởng là bắt đầu từ một nhiệt độ cao và dần giảm nhiệt độ theo thời gian. Quá trình này giúp thuật toán có cơ hội lớn hơn để thoát ra khỏi các điểm cực tiểu cục bộ và tiến tới giải pháp tốt nhất
- **Quy trình:**
  - **Khởi tạo:** Bắt đầu từ một giải pháp ngẫu nhiên trong không gian trạng thái.
  - **Tìm kiếm láng giềng:** Từ một trạng thái hiện tại, thuật toán chọn một trạng thái láng giềng ngẫu nhiên.
  - **Đánh giá sự thay đổi**: So sánh giá trị đánh giá (evaluation value) của trạng thái mới và trạng thái hiện tại.

Câu 3: Simulated Annealing Search(cont)

- **Quyết định chấp nhận hoặc từ chối trạng thái mới**: Dựa trên một xác suất chấp nhận (probability of acceptance), thuật toán quyết định xem có chấp nhận trạng thái mới hay không. Xác suất chấp nhận giảm dần theo thời gian, theo một lịch trình giảm nhiệt độ.
- **Lặp lại quá trình từ bước 2 đến khi điều kiện dừng được đáp ứng**: Quá trình này tiếp tục cho đến khi đạt được một điều kiện dừng được xác định trước hoặc đến khi không còn cải thiện nào có thể đạt được.

• **Mục tiêu cuối cùng** của thuật toán Simulated Annealing là tìm ra một giải pháp tối ưu (hoặc gần tối ưu) trong không gian trạng thái, bằng cách khai thác các cơ hội thoát ra khỏi các điểm cực tiểu cục bộ và tận dụng sự linh hoạt của quá trình làm mát.

Câu 3: Simulated Annealing Search(cont)

- **Ưu điểm**:
  - **Khả năng thoát khỏi cực tiểu cục bộ:** Simulated Annealing có khả năng vượt qua các điểm cực tiểu cục bộ bằng cách chấp nhận các giải pháp kém hơn với một xác suất nhất định, giúp thuật toán khám phá không gian trạng thái một cách toàn diện hơn.
  - **Khả năng tìm giải pháp tối ưu toàn cục:** Nhờ vào cơ chế chấp nhận giải pháp kém hơn, Simulated Annealing có khả năng tìm ra giải pháp tối ưu toàn cục, đặc biệt là trong không gian trạng thái lớn và phức tạp.
  - **Dễ dàng thực hiện và điều chỉnh:** Thuật toán Simulated Annealing có cấu trúc đơn giản và dễ triển khai, đồng thời có thể điều chỉnh các tham số như nhiệt độ ban đầu và lịch trình giảm nhiệt độ để tối ưu hóa hiệu suất tìm kiếm.
  - **Hiệu quả với không gian trạng thái lớn:** Simulated Annealing hoạt động tốt với các bài toán có không gian trạng thái lớn mà không cần lưu trữ toàn bộ không gian trạng thái trong bộ nhớ.

Câu 3: Simulated Annealing Search(cont)

- **Khả năng điều chỉnh tốc độ tìm kiếm**: Bằng cách điều chỉnh nhiệt độ ban đầu và lịch trình giảm nhiệt độ, người dùng có thể kiểm soát tốc độ của quá trình tìm kiếm và mức độ khám phá không gian trạng thái.
- **Áp dụng linh hoạt:** Simulated Annealing có thể áp dụng cho nhiều loại bài toán tối ưu hóa, từ tối ưu hóa hàm mất mát đến lập lịch và tối ưu hóa cấu trúc.

- **Nhược điểm:**
  - **Độ phức tạp của việc điều chỉnh tham số:** Simulated Annealing yêu cầu người dùng điều chỉnh một số tham số quan trọng như nhiệt độ ban đầu, lịch trình giảm nhiệt độ và hàm xác suất chấp nhận. Việc điều chỉnh các tham số này có thể khá phức tạp và đòi hỏi sự hiểu biết sâu sắc về bài toán cụ thể.

Câu 3: Simulated Annealing Search(cont)

- **Độ phức tạp của việc điều chỉnh tham số**: Simulated Annealing yêu cầu người dùng điều chỉnh một số tham số quan trọng như nhiệt độ ban đầu, lịch trình giảm nhiệt độ và hàm xác suất chấp nhận. Việc điều chỉnh các tham số này có thể khá phức tạp và đòi hỏi sự hiểu biết sâu sắc về bài toán cụ thể.
- **Tốn thời gian:** Quá trình tìm kiếm của Simulated Annealing có thể tốn nhiều thời gian, đặc biệt là với các bài toán có không gian trạng thái lớn hoặc có nhiều cực tiểu cục bộ.
- **Khả năng rơi vào cực tiểu cục bộ:** Mặc dù có cơ chế chấp nhận giải pháp kém hơn với một xác suất nhất định, Simulated Annealing vẫn có thể rơi vào các điểm cực tiểu cục bộ, đặc biệt là khi khaông tìm được các tham số tối ưu.
- **Chọn ngẫu nhiên các trạng thái láng giềng:** Quá trình chọn ngẫu nhiên các trạng thái láng giềng có thể dẫn đến việc không tìm ra được các trạng thái tốt trong quá trình tìm kiếm.

Câu 3: Simulated Annealing Search(cont)

- **Khả năng không tìm ra giải pháp tối ưu:** Mặc dù Simulated Annealing có khả năng tìm ra giải pháp tối ưu toàn cục, nhưng không có đảm bảo rằng giải pháp tối ưu sẽ được tìm thấy trong mọi trường hợp. Điều này phụ thuộc nhiều vào cách điều chỉnh các tham số và cấu trúc của bài toán.

Câu 3: Simulated Annealing Search(pseudocode)

function SIMULATED-ANNEALING(problem, schedule) **returns** path

Inputs: *problem,* a problem

schedule, a mapping from time to “temperature”

path ← an empty list that stores a unique path from the starting point until the local maximum is found

current, current-value ← MAKE-NODE(problem.INITIAL-STATE)

append current to path

initialize t where t is iteration step (t ← 0.000001)

loop do

T ← *schedule*(t)

if T < 0.000001 **then** break loop

next, next_value ← problem.random_neighbor(current)

ΔE ← next.VALUE – current.VALUE

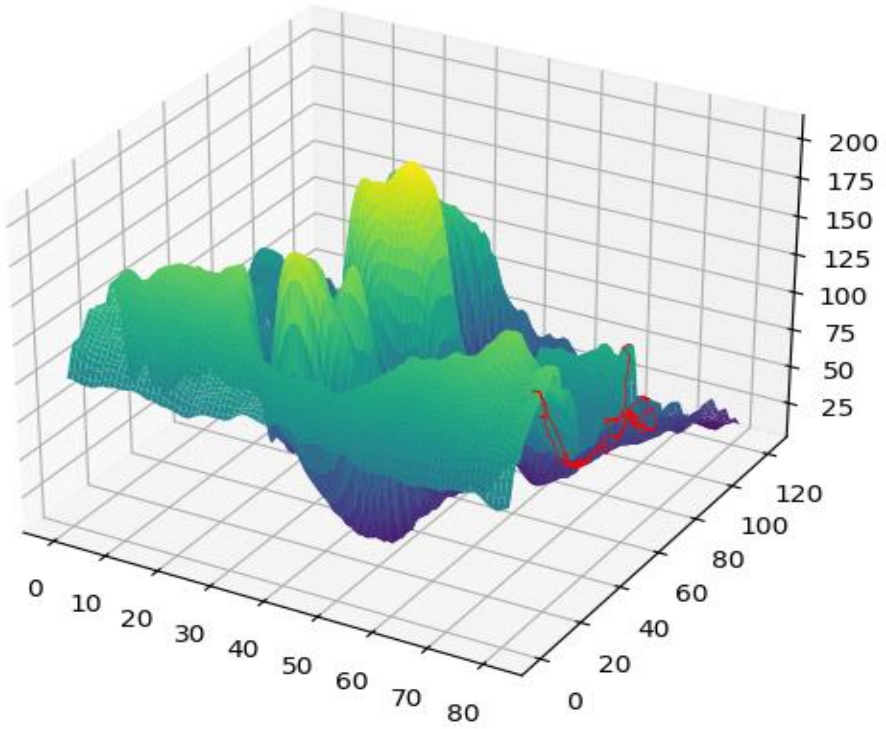
if ΔE > 0 **then** current ← next

else current ← next only with probability $e^{\Delta E/T}$

increase t ← t*1.1

return path

Câu 3: Simulated Annealing Search



Hình 2: Biểu đồ trực quan Simulated Annealing Search

Câu 4: Local Beam Search

- **Mục tiêu**: thuật toán local beam search là tìm kiếm một lời giải tốt nhất trong một không gian tìm kiếm lớn bằng cách tạo ra và duy trì một tập hợp các trạng thái (hoặc nút) tiềm năng. Thuật toán này thực hiện việc mở rộng từ các trạng thái hiện tại, đánh giá các trạng thái mới được tạo ra, và lựa chọn một tập hợp con tốt nhất của chúng để tiếp tục tìm kiếm.
  - *Các mục tiêu cụ thể của thuật toán local beam search:*
  - **Tìm kiếm trong không gian lớn**: Mục tiêu chính của thuật toán local beam search là tìm kiếm trong không gian tìm kiếm lớn. Điều này áp dụng cho các bài toán mà không gian tìm kiếm có kích thước lớn và không thể duyệt hoàn toàn.
  - **Tìm kiếm gần lời giải tối ưu**: Mặc dù không thể đảm bảo tìm được lời giải tối ưu, mục tiêu của thuật toán local beam search là tìm kiếm và tiến gần đến lời giải tối ưu nhất có thể trong một thời gian hợp lý.

Câu 4: Local Beam Search(cont)

- **Tìm kiếm nhanh chóng:** Thuật toán local beam search thường được thiết kế để tìm kiếm nhanh chóng. Thay vì tìm kiếm hoàn toàn trong toàn bộ không gian tìm kiếm, nó chỉ tập trung vào một tập hợp con các trạng thái tiềm năng, làm giảm đáng kể thời gian tìm kiếm so với các phương pháp tìm kiếm toàn diện.
  - **Tìm kiếm địa phương**: Thuật toán này thường tập trung vào việc tìm kiếm "gần" các trạng thái tốt nhất trong không gian tìm kiếm, thay vì tìm kiếm toàn bộ không gian. Do đó, nó thường thực hiện các bước mở rộng từ các trạng thái hiện tại và chọn ra các trạng thái tiếp theo từ các trạng thái mở rộng này.

Câu 4: Local Beam Search(cont)

- **Ưu điểm:**
  - **Khả năng tránh khỏi cạm bẫy tối ưu cục bộ**: Trong một số bài toán, đặc biệt là các bài toán tìm kiếm tối ưu trong không gian lớn, có thể tồn tại các cạm bẫy tối ưu cục bộ. Local Beam Search, bằng cách chọn nhiều trạng thái khởi đầu ngẫu nhiên, có khả năng tránh được các cạm bẫy này bằng cách tiến xa hơn trong không gian tìm kiếm.
  - **Tốc độ nhanh và tiết kiệm bộ nhớ:** Vì chỉ lưu trữ một số lượng nhỏ các trạng thái con tốt nhất,Local beam search có thể tìm kiếm và giải quyết các bài toán lớn một cách nhanh chóng và tiết kiệm.
  - **Khả năng tùy chỉnh**: Local Beam Search có thể dễ dàng được điều chỉnh thông qua việc chọn số lượng k trạng thái khởi đầu và việc lựa chọn hàm đánh giá. Điều này làm cho thuật toán linh hoạt và có thể được tinh chỉnh để phù hợp với nhiều loại bài toán khác nhau.

Câu 4: Local Beam Search(cont)

- **Nhược điểm**
  - **Rủi ro rơi vào cạm bẫy tối ưu cục bộ**: Local Beam Search có thể dễ dàng rơi vào cạm bẫy tối ưu cục bộ, đặc biệt khi số lượng trạng thái ban đầu được chọn là nhỏ. Nếu các trạng thái khởi đầu không bao quát được không gian tìm kiếm một cách đủ đại diện, thuật toán có thể bị kẹt ở một khu vực cục bộ của không gian tìm kiếm và không thể tiếp tục tiến hóa.
  - **Không đảm bảo lời giải tối ưu**: Local Beam Search không đảm bảo tìm kiếm lời giải tối ưu. Do thuật toán chỉ tập trung vào một số lượng hữu hạn các trạng thái, có thể bỏ lỡ lời giải tốt hơn nằm ngoài tập hợp này.
  - **Yêu cầu lựa chọn số lượng trạng thái ban đầu chính xác**: Hiệu suất của Local Beam Search phụ thuộc rất nhiều vào việc lựa chọn số lượng trạng thái ban đầu (k). Nếu chọn k quá nhỏ, thuật toán có thể không thể khám phá đủ sâu trong không gian tìm kiếm. Ngược lại, nếu chọn k quá lớn, sẽ tốn nhiều bộ nhớ và thời gian tính toán.

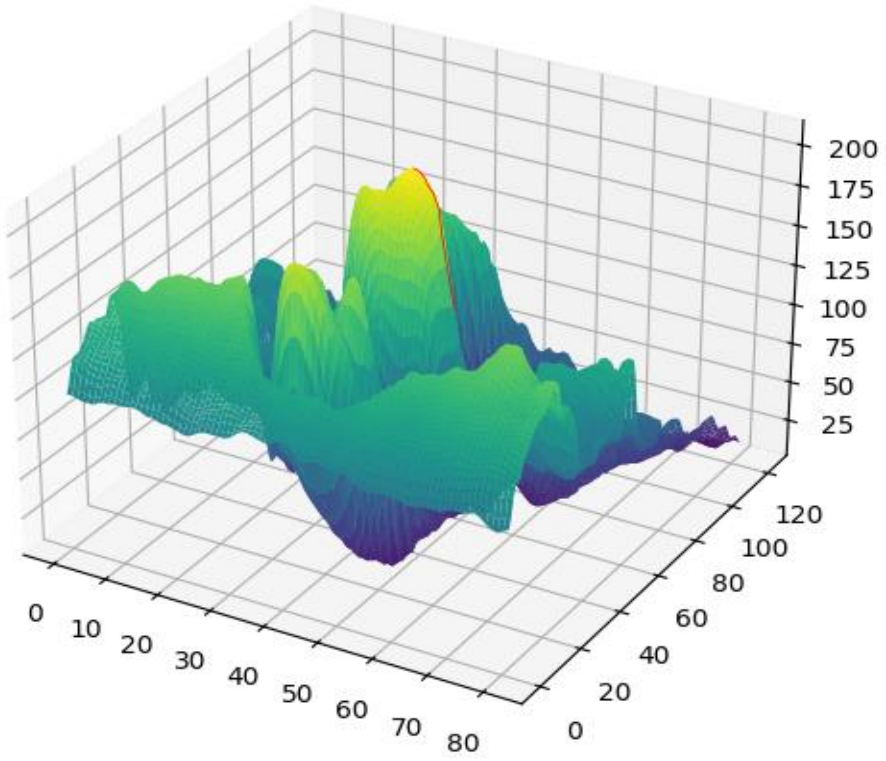
Câu 4: Local Beam Search(cont)

- **Cần lựa chọn hàm đánh giá phù hợp**: Hiệu suất của Local Beam Search phụ thuộc vào việc chọn một hàm đánh giá phù hợp để đánh giá chất lượng của các trạng thái. Nếu hàm đánh giá không phản ánh được mức độ tốt của các trạng thái, thuật toán có thể không tìm được lời giải tốt.
- **Khả năng rơi vào local minimum**: Trong một số trường hợp, Local Beam Search có thể rơi vào *local minimum* hoặc *plateau* - các vùng không có sự cải thiện đáng kể trong đánh giá. Trong những trường hợp này, thuật toán có thể mất thời gian lớn hoặc không thể tìm ra lời giải tốt.

Câu 4: Local Beam Search(cont)

```
function local_beam_search(problem, k) returns path
    path ← an empty list to store the final path
    current_state ← MAKE-NODE(problem.get-randomly-state)
      loop do
                next_states ← an empty list to store all successors of the k current states
                // Generate all successors of the k current states
                next-state.extend(all successor of the k current states)
                // Sort the current states and the next states
                sorted_current ← sort current_states by the evaluation function value in descending order
                sorted_next_state ← sort next_states by the evaluation function value in descending order
                // Get the maximum current state and the maximum next state
                current_max ← get the first item in sorted-current list
                next_max ← get the first item in sorted-next list
                // Check if reaching the goal state or not
                if problem.evaluate_state(next_max.state) ≤ problem.evaluate_state(current_max.state) then
                    save the current max node
                    break the loop
                if the goal not found, select k best successor from the sorted_next_state and repeat
      Reconstruct the path from the maximum node to the root
return path
```

Câu 4: Local Beam Search



Hình 3: Biểu đồ trực quan Local Beam Search

Phân công công việc

Họ và tên	MSSV	Email	Phần phụ trách	Mức độ hoàn thành
Mã Trường Quang	52100925	Matruongquang1@gmail.com	Câu 1,2: code problem, hill-climbing serch	100%
Huỳnh Tấn Đạt	52200152	huynhtandat184@gmail.com	Làm slide và nội dung câu 3,4	100%
Lê Như Đạt	52200160	lenhudat181104@gmail.com	Câu 3,4: code simulated-annealing, local beam search	100%
Ngô Đức Anh Tuấn	51800951	anhtuan.11020@gmail.com	Làm slide và nội dung câu 1,2	100%

Tài liệu tham khảo

- Ths Nguyễn Thành An. 2024. Giáo trình Nhập môn Trí tuệ nhân tạo. Khoa Công nghệ thông tin. Trường ĐH Tôn Đức Thắng
- Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
- Lê Hoài Bắc, Tô Hoài Việt. 2014. Giáo trình Cơ sở Trí tuệ nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.
- Nguyễn Ngọc Thảo, Nguyễn Hải Minh. 2020. Bài giảng Cơ sở Trí tuệ Nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.

Thank you