UỶ BAN NHÂN DÂN

THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC SÀI GÒN**



**BÁO CÁO PHÂN TÍCH  
MÔ PHỎNG VÀ ĐÁNH GIÁ TÁC NHÂN THÔNG MINH,  
SO SÁNH HIỆU SUẤT CÁC THUẬT TOÁN TÌM KIẾM TRONG MÊ CUNG**

**Học phần: Trí tuệ nhân tạo nâng cao**

**Giảng viên hướng dẫn: TS. Đỗ Như Tài**

*-Sinh viên tham gia:*

|  |  |
| --- | --- |
| * *Cao Tấn Đạt* | *3122410069* |
| * *Lê Nguyễn Anh Đức* | *3122410087* |
| * *Nguyễn Trọng Tấn Sang* | *3121560076* |
|  |  |

**Thành phố Hồ Chí Minh, 10/2025**

# Bảng phân công nhiệm vụ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Tuần | Tên bài tập | Thành viên tham gia | % đóng góp (nếu trên 1 người) |
| 1 | 3,4 | Làm lab02\_01\_agents và Comparisons lab02\_02\_search | Đức | 33.33% |
| 2 | 3,4 | Làm assignment và Heuristics lab02\_02\_search | Đạt | 33.33% |
| 3 | 3,4 | Làm wor\_slide báo cáo kết quả | Sang | 33.33% |

***Mục lục***

[**1. TỔNG QUAN** 1](#_Toc210592560)

[**2. PHÂN TÍCH FILE 1: robot\_vacuum.ipynb (Robot Hút Bụi Tự Động)** 1](#_Toc210592561)

[**3. PHÂN TÍCH FILE 2: lunar\_lander.ipynb (Tàu Đổ Bộ Mặt Trăng)** 3](#_Toc210592562)

[**4. SO SÁNH VÀ KẾT LUẬN CHUNG** 4](#_Toc210592563)

[**1. Giới thiệu** 6](#_Toc210592564)

[**2. Phương pháp Nghiên cứu** 6](#_Toc210592565)

[**3. Kết quả Thực nghiệm** 6](#_Toc210592566)

[**4. Phân tích và Thảo luận** 7](#_Toc210592567)

[**5. Kết luận** 7](#_Toc210592568)

# *Mô phỏng và đánh giá tác nhân thông minh*

**1. TỔNG QUAN**

-Báo cáo này trình bày phân tích chi tiết về hai file mã nguồn Jupyter Notebook: robot\_vacuum.ipynb và lunar\_lander.ipynb. Cả hai file đều tập trung vào việc xây dựng và đánh giá các **tác nhân thông minh (Intelligent Agents)**, một khái niệm cốt lõi trong lĩnh vực Trí tuệ nhân tạo.

* robot\_vacuum.ipynb xây dựng một môi trường mô phỏng đơn giản (phòng có bụi bẩn) và so sánh hiệu suất của ba loại tác nhân phản xạ khác nhau.
* lunar\_lander.ipynb sử dụng một môi trường mô phỏng phức tạp hơn có sẵn từ thư viện Gymnasium để điều khiển một con tàu vũ trụ và đánh giá các tác nhân phản xạ dựa trên quy tắc.

-Mục tiêu của báo cáo là làm rõ mục đích, cấu trúc, kết quả và ý nghĩa của từng file mã nguồn, từ đó đưa ra một cái nhìn tổng thể về việc ứng dụng các loại tác nhân thông minh trong các bài toán khác nhau.

**2. PHÂN TÍCH FILE 1: robot\_vacuum.ipynb (Robot Hút Bụi Tự Động)**

**2.1. Mục tiêu**

-Notebook này nhằm mục đích:

* Xây dựng một môi trường mô phỏng cho bài toán robot hút bụi trong một căn phòng dạng lưới.
* Triển khai ba loại tác nhân với mức độ "thông minh" tăng dần: Tác nhân ngẫu nhiên, Tác nhân phản xạ đơn giản, và Tác nhân phản xạ dựa trên mô hình.
* Thực hiện một nghiên cứu mô phỏng để so sánh hiệu suất (số bước để dọn sạch phòng) của các tác nhân này trên các kích thước môi trường khác nhau.

**2.2. Mô tả bài toán (PEAS)**

-Bài toán được định nghĩa rõ ràng theo mô hình PEAS (Performance, Environment, Actuators, Sensors):

* **Performance (Hiệu suất):** Tổng số hành động (năng lượng) tiêu tốn để dọn sạch hoàn toàn căn phòng. Càng ít hành động càng tốt.
* **Environment (Môi trường):** Một căn phòng dạng lưới n x n (với n=5, 10, 100). Bụi được đặt ngẫu nhiên với xác suất 20%. Tác nhân khởi đầu ở một vị trí ngẫu nhiên.
* **Actuators (Bộ truyền động):** Robot có 5 hành động: suck (hút), north, east, south, west (di chuyển).
* **Sensors (Cảm biến):** Cảm biến va chạm ở bốn hướng và cảm biến bụi tại vị trí hiện tại.

**2.3. Cấu trúc mã nguồn**

**-Môi trường (init\_environment)**

Hàm này mô phỏng toàn bộ hoạt động của bài toán:

1. Khởi tạo một ma trận (mảng 2D NumPy) đại diện cho căn phòng, với giá trị 1 là bẩn và 0 là sạch.
2. Khởi tạo vị trí ngẫu nhiên cho tác nhân.
3. Trong một vòng lặp, nó cung cấp thông tin cảm biến (va chạm, bụi bẩn) cho hàm tác nhân.
4. Nhận hành động trả về từ tác nhân và cập nhật trạng thái môi trường (vị trí robot, trạng thái bụi bẩn).
5. Đếm số hành động và dừng lại khi phòng sạch hoàn toàn hoặc đạt giới hạn bước.
6. Trả về tổng số hành động đã thực hiện.

**-Các Tác nhân (Agents)**

1. **simple\_randomized\_agent:** Đây là tác nhân cơ bản nhất, hoạt động hoàn toàn ngẫu nhiên. Nó bỏ qua mọi thông tin từ cảm biến và chọn một trong năm hành động một cách ngẫu nhiên.
2. **simple\_reflex\_agent:** Tác nhân này thể hiện hành vi phản xạ đơn giản. Quy tắc của nó là:
   * Nếu ô hiện tại bẩn, hành động là suck.
   * Nếu không, nó sẽ chọn ngẫu nhiên một hướng di chuyển không bị chặn bởi tường. Đây là một cải tiến lớn so với tác nhân ngẫu nhiên vì nó sử dụng thông tin cảm biến để hành động hợp lý hơn.
3. **model\_based\_reflex\_agent:** Đây là tác nhân thông minh nhất. Nó có một "mô hình" nội bộ về thế giới, thể hiện qua việc nó có thể truy cập vào trạng thái của toàn bộ căn phòng (room, agent\_position). Quy tắc của nó là:
   * Nếu ô hiện tại bẩn, hành động là suck.
   * Nếu không, nó tìm tất cả các ô còn bẩn trong phòng, xác định ô gần nhất (dựa trên khoảng cách Manhattan) và di chuyển về hướng đó. Tác nhân này không chỉ phản xạ mà còn có mục tiêu rõ ràng, giúp nó hiệu quả hơn nhiều.

**2.4. Phân tích kết quả thực nghiệm**

-Nghiên cứu mô phỏng (Task 4) so sánh hiệu suất trung bình của ba tác nhân sau 100 lần chạy trên các kích thước phòng khác nhau.

| Kích thước | Tác nhân Ngẫu nhiên (bước) | Tác nhân Phản xạ Đơn giản (bước) | Tác nhân Phản xạ Dựa trên Mô hình (bước) |
| --- | --- | --- | --- |
| 5x5 | 372.36 | 106.81 | **15.99** |
| 10x10 | 3192.69 | 916.22 | **70.00** |
| 100x100 | > 100,000 (Thất bại) | > 100,000 (Thất bại) | **7043.19** |

Xuất sang Trang tính

**-Phân tích:**

* **Hiệu suất vượt trội:** Tác nhân Phản xạ Dựa trên Mô hình cho thấy hiệu suất vượt trội ở mọi kích thước. Nó luôn di chuyển có mục đích đến vị trí bẩn gần nhất, tối ưu hóa đường đi và tránh các hành động thừa.
* **Sự kém hiệu quả của hành vi ngẫu nhiên:** Tác nhân Phản xạ Đơn giản tốt hơn nhiều so với tác nhân Ngẫu nhiên, nhưng cả hai đều dựa vào việc "lang thang" để tìm bụi. Khi kích thước phòng tăng lên, chiến lược này trở nên cực kỳ kém hiệu quả.
* **Khả năng mở rộng:** Với phòng 100x100, hai tác nhân đầu tiên không thể hoàn thành nhiệm vụ trong giới hạn 100,000 bước, cho thấy chúng không có khả năng mở rộng. Ngược lại, tác nhân dựa trên mô hình vẫn hoàn thành tốt nhiệm vụ, chứng tỏ tính hiệu quả của việc lập kế hoạch dựa trên trạng thái toàn cục.

**2.5. Phân tích độ bền (Robustness)**

-Task 5 trong file đưa ra phân tích định tính về hạn chế của các tác nhân khi môi trường thay đổi:

* **Phòng không rõ kích thước / hình dạng bất thường:** Tác nhân dựa trên mô hình sẽ thất bại vì nó phụ thuộc vào kiến thức toàn cục về môi trường. Các tác nhân phản xạ đơn giản và ngẫu nhiên vẫn hoạt động được (vì chúng chỉ phản ứng với cảm biến cục bộ) nhưng có thể bị kẹt hoặc bỏ sót khu vực.
* **Có vật cản:** Các tác nhân sẽ coi vật cản như tường và tránh chúng, nhưng hiệu suất sẽ giảm vì phải tìm đường vòng.
* **Cảm biến sai lệch:**
  + Nếu cảm biến bụi sai, robot có thể bỏ sót bụi hoặc hút lại ô đã sạch, làm giảm hiệu suất.
  + Nếu cảm biến va chạm sai, robot có thể lãng phí năng lượng khi cố di chuyển vào tường.

**3. PHÂN TÍCH FILE 2: lunar\_lander.ipynb (Tàu Đổ Bộ Mặt Trăng)**

**3.1. Mục tiêu**

-Notebook này nhằm mục đích:

* Sử dụng môi trường LunarLander-v3 từ thư viện Gymnasium, một công cụ tiêu chuẩn trong học tăng cường.
* Xây dựng và so sánh các tác nhân phản xạ đơn giản (dựa trên quy tắc) để giải quyết bài toán điều khiển tàu đổ bộ.
* Đánh giá hiệu suất của các tác nhân thông qua hệ thống điểm thưởng của môi trường.

**3.2. Mô tả môi trường và bài toán**

* **Môi trường:** Một bài toán tối ưu quỹ đạo kinh điển. Tàu vũ trụ phải hạ cánh an toàn xuống bãi đáp tại tọa độ (0,0). Không gian trạng thái là liên tục.
* **Actuators (Hành động):** Có 4 hành động rời rạc: không làm gì, bắn động cơ trái, bắn động cơ chính, bắn động cơ phải.
* **Sensors (Quan sát):** Trạng thái của môi trường được biểu diễn bằng một vector 8 chiều, bao gồm: tọa độ (x, y), vận tốc (vx, vy), góc nghiêng, vận tốc góc, và trạng thái tiếp đất của hai chân vịt.
* **Performance (Điểm thưởng):** +100 điểm cho hạ cánh an toàn, -100 điểm nếu bị rơi. Có các điểm thưởng/phạt trung gian cho việc bắn động cơ và tiếp đất.

**3.3. Cấu trúc mã nguồn**

**-Môi trường (Gymnasium)**

Mã nguồn không tự xây dựng môi trường mà sử dụng các hàm có sẵn từ thư viện Gymnasium:

* gym.make("LunarLander-v3"): Tạo một thực thể của môi trường.
* env.reset(): Đặt lại môi trường về trạng thái ban đầu, trả về quan sát đầu tiên.
* env.step(action): Thực thi một hành động, trả về quan sát mới, điểm thưởng, và các cờ trạng thái (kết thúc, cắt ngắn).

**-Các Tác nhân (Agents)**

+Các tác nhân là các hàm nhận vào observation (vector 8 chiều) và trả về một hành động.

1. **random\_agent\_function:** Tác nhân cơ bản, chọn một hành động ngẫu nhiên và bỏ qua mọi quan sát.
2. **rocket\_agent\_function:** Một tác nhân phản xạ rất đơn giản. Nó chỉ tuân theo một quy tắc duy nhất: nếu vận tốc dọc (vy) nhỏ hơn -0.3 (rơi quá nhanh), hãy bắn động cơ chính.
3. **better\_rocket\_agent\_function:** Tác nhân cải tiến với một tập hợp quy tắc phức tạp hơn:
   * Giảm tốc độ rơi nếu vy < -0.2.
   * Điều chỉnh góc nghiêng và vận tốc góc bằng động cơ trái/phải.
   * Điều chỉnh vị trí ngang (x) để hướng về bãi đáp.
   * Ổn định tàu khi đã tiếp đất. Đây là một tác nhân phản xạ điển hình, ánh xạ trực tiếp từ điều kiện (trạng thái) sang hành động.

**3.4. Phân tích và đánh giá**

* Mã nguồn định nghĩa hàm run\_episodes để chạy tác nhân trong nhiều lượt (mặc định 1000) và ghi lại điểm thưởng cuối cùng của mỗi lượt.
* Hiệu suất được đo bằng **điểm thưởng trung bình** và **tỷ lệ thành công** (số lần đạt 100 điểm).
* **Kết quả (dự kiến):**
  + Tác nhân random\_agent sẽ có điểm thưởng trung bình rất thấp và tỷ lệ thành công gần như bằng 0.
  + Tác nhân rocket\_agent\_function sẽ tốt hơn một chút nhưng vẫn có khả năng thất bại cao vì nó không kiểm soát được phương ngang và góc nghiêng.
  + Tác nhân better\_rocket\_agent\_function được kỳ vọng sẽ đạt điểm thưởng trung bình cao hơn đáng kể và có tỷ lệ thành công nhất định, vì các quy tắc của nó giải quyết các khía cạnh quan trọng của bài toán (giảm tốc, giữ thăng bằng, di chuyển ngang).
* **Phân tích:** Notebook này minh họa rằng ngay cả với một bài toán điều khiển trong không gian liên tục, một tập hợp các quy tắc phản xạ được thiết kế tốt cũng có thể đạt được hiệu suất khá tốt, dù không phải là tối ưu như các phương pháp học tăng cường.

**4. SO SÁNH VÀ KẾT LUẬN CHUNG**

**4.1. Điểm tương đồng**

* Cả hai notebook đều khám phá khái niệm về tác nhân thông minh, cụ thể là các tác nhân phản xạ.
* Đều sử dụng một tác nhân ngẫu nhiên làm cơ sở để so sánh.
* Đều có một quy trình đánh giá lặp đi lặp lại để tính toán hiệu suất trung bình.
* Đều cho thấy rằng việc bổ sung các quy tắc hoặc mô hình hợp lý sẽ cải thiện đáng kể hiệu suất so với hành vi ngẫu nhiên hoặc phản xạ quá đơn giản.

**4.2. Điểm khác biệt**

* **Môi trường:** robot\_vacuum tự xây dựng một môi trường rời rạc, đơn giản. lunar\_lander sử dụng một môi trường liên tục, dựa trên vật lý, phức tạp hơn từ một thư viện tiêu chuẩn.
* **Loại tác nhân:** robot\_vacuum giới thiệu một tác nhân "dựa trên mô hình" có kiến thức toàn cục. lunar\_lander chỉ tập trung vào các tác nhân phản xạ thuần túy, chỉ phản ứng với trạng thái quan sát được tại một thời điểm.
* **Độ phức tạp:** Bài toán Lunar Lander về bản chất phức tạp hơn do không gian trạng thái liên tục và động lực học vật lý, đòi hỏi các quy tắc điều khiển tinh vi hơn.

**4.3. Kết luận**

-Hai notebook này cung cấp những ví dụ minh họa xuất sắc về các loại tác nhân thông minh cơ bản.

1. **Sức mạnh của Quy tắc:** Cho thấy rằng các tác nhân phản xạ đơn giản, dựa trên các quy tắc "Nếu-Thì", có thể giải quyết các vấn đề từ đơn giản đến tương đối phức tạp. Hiệu suất của chúng phụ thuộc trực tiếp vào chất lượng và sự đầy đủ của bộ quy tắc.
2. **Giá trị của Mô hình:** Notebook về robot hút bụi chứng minh một cách thuyết phục rằng các tác nhân có mô hình nội bộ về thế giới (biết bản đồ) hoạt động hiệu quả và có khả năng mở rộng tốt hơn nhiều so với các tác nhân chỉ dựa vào phản xạ cục bộ.
3. **Nền tảng cho các khái niệm nâng cao:** Các bài toán này là ví dụ kinh điển và là bước đệm để hiểu các khái niệm phức tạp hơn như học tăng cường (Reinforcement Learning), nơi một tác nhân có thể tự động học các quy tắc tối ưu thay vì được lập trình sẵn.

***So sánh hiệu suất của các thuật toán tìm kiếm khác nhau để giải quyết mê cung***

**1. Giới thiệu**

-Nghiên cứu này khám phá bài toán tìm đường đi trong mê cung, một vấn đề kinh điển trong lĩnh vực trí tuệ nhân tạo. Mục tiêu chính là triển khai và đánh giá hiệu quả của các thuật toán tìm kiếm để tìm ra đường đi tối ưu từ điểm xuất phát (S) đến điểm đích (G) trong một môi trường có thể quan sát đầy đủ, rời rạc và xác định.

**2. Phương pháp Nghiên cứu**

-Nghiên cứu định nghĩa bài toán tìm kiếm với các thành phần sau:

* **Trạng thái ban đầu**: Vị trí của tác nhân tại điểm 'S' trong mê cung.
* **Hành động**: Tác nhân có thể di chuyển theo bốn hướng: Lên, Xuống, Trái, Phải, miễn là không đi vào tường.
* **Mô hình chuyển tiếp**: Một hành động sẽ đưa tác nhân đến một ô mới nếu hợp lệ.
* **Trạng thái đích**: Vị trí của ô 'G'.
* **Chi phí đường đi**: Mỗi bước di chuyển có chi phí là 1.

-Các thuật toán tìm kiếm được triển khai và so sánh bao gồm:

* **Tìm kiếm theo chiều rộng (BFS)**: Một thuật toán tìm kiếm không thông tin, đảm bảo tìm ra đường đi ngắn nhất.
* **Tìm kiếm theo chiều sâu (DFS)**: Một thuật toán tìm kiếm không thông tin khác, ưu tiên đi sâu vào một nhánh trước khi quay lại.
* **Tìm kiếm tham lam tốt nhất đầu tiên (GBFS)**: Một thuật toán tìm kiếm có thông tin, sử dụng hàm heuristic (khoảng cách Manhattan) để ưu tiên các nút gần đích hơn.
* *Tìm kiếm A (A-Star)*\*: Một thuật toán tìm kiếm có thông tin kết hợp chi phí đường đi đã biết (g(n)) và chi phí ước tính đến đích (h(n)) để tìm đường đi tối ưu một cách hiệu quả.

**3. Kết quả Thực nghiệm**

-Các thuật toán đã được chạy trên ba mê cung với kích thước khác nhau: nhỏ, vừa và lớn. Dưới đây là bảng tóm tắt hiệu suất:

**Mê cung nhỏ**

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuật toán** | **Chi phí đường đi** | **Số nút đã mở rộng** | **Bộ nhớ tối đa (số nút)** |
| **BFS** | **19** | **91** | **95** |
| **DFS** | **29** | **53** | **4** |
| **GBFS** | **29** | **39** | **49** |
| **A\*** | **19** | **45** | **61** |

**Mê cung vừa**

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuật toán** | **Chi phí đường đi** | **Số nút đã mở rộng** | **Bộ nhớ tối đa (số nút)** |
| **BFS** | **68** | **269** | **275** |
| **DFS** | **164** | **258** | **12** |
| **GBFS** | **74** | **78** | **86** |
| **A\*** | **68** | **222** | **235** |

**Mê cung lớn**

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuật toán** | **Chi phí đường đi** | **Số nút đã mở rộng** | **Bộ nhớ tối đa (số nút)** |
| **BFS** | **210** | **620** | **627** |
| **DFS** | **210** | **469** | **30** |
| **GBFS** | **210** | **466** | **508** |
| **A\*** | **210** | **549** | **565** |

**4. Phân tích và Thảo luận**

* **Sự đánh đổi giữa Tối ưu và Hiệu quả**:
  + **BFS** và **A\*** luôn tìm thấy đường đi tối ưu (chi phí thấp nhất).
  + **GBFS** là thuật toán hiệu quả nhất về số nút được mở rộng, nhưng không phải lúc nào cũng tìm ra đường đi tối ưu (như trong mê cung nhỏ và vừa).
  + **DFS** sử dụng ít bộ nhớ nhất nhưng thường tìm thấy các đường đi không tối ưu và có thể hoạt động rất kém trong các mê cung phức tạp.
* **Tác động của Heuristic**:
  + **A\***, với việc sử dụng hàm heuristic khoảng cách Manhattan, đã giảm đáng kể số lượng nút cần mở rộng so với BFS mà vẫn đảm bảo tính tối ưu. Điều này cho thấy tầm quan trọng của việc sử dụng thông tin (heuristic) trong tìm kiếm.
* **Độ phức tạp của bài toán**:
  + Hành vi của các thuật toán thay đổi rõ rệt tùy theo cấu trúc của mê cung. Trong mê cung lớn, nơi chỉ có một đường đi tối ưu duy nhất, tất cả các thuật toán đều tìm thấy cùng một giải pháp, mặc dù hiệu quả tính toán khác nhau.

**5. Kết luận**

-Nghiên cứu chứng minh rằng **thuật toán A\*** mang lại sự cân bằng tốt nhất giữa việc tìm ra giải pháp tối ưu và hiệu quả tính toán. Trong khi BFS đảm bảo tối ưu nhưng chậm hơn, GBFS nhanh nhưng có thể bỏ lỡ đường đi tốt nhất, và DFS tiết kiệm bộ nhớ nhưng không đáng tin cậy về chất lượng giải pháp.

-Sự lựa chọn thuật toán tìm kiếm phù hợp phụ thuộc vào các yêu cầu cụ thể của bài toán, chẳng hạn như sự cần thiết của tính tối ưu, tài nguyên tính toán và bộ nhớ có sẵn.