**Expert Systems with Applications**

**1. Introduction**

Những năm gần đây robot nói chung và robot di động nói riêng là một lĩnh vực đầy tiềm năng. Lĩnh vực này mang đến những công cụ hỗ trợ đắc lực cho con người. Và vấn đề về khả năng điều hướng là một lĩnh vực nghiên cứu cơ bản, thiết yếu quan trọng đối với sự phát triển công nghệ robot được ứng mang đến những giải pháp hiện đại như tự động hóa robot công nghiệp (Yao-An Lu, Kai Tang, Cheng-Yong Wang , 2021), trong y tế phẫu thuật (Rafael Valencia-Garcia, Rodrigo Martinez-Be´jar, Alessandro Gasparetto, 2005), robot tự hành, ...

Hoạch định đường đi của robot là một trong những hoạt động cơ bản cẩn thiết để thực hiện sự điều hướng của robot ( Barraquand, Latombe, 1991; Choset, 2005). Giải pháp của nó cung cấp một con đường không va chạm khả thi để đi từ nơi này đến nơi khác. Trong những năm gần đây lĩnh vực này có những phát triển vượt bậc và trong đó phải kể đến các thuật toán lấy mẫu, các thuật toán dựa trên ý tưởng này đang được chú ý như PRMs. (Gildardo Sanchez,Jean-Claude Latombe, 2002; Yan wu,Amato, 1996; Pankaj K. Agarwal, Lydia E. Kavraki, Matthew T. Mason, 1998) Và thuật toán RRT cũng là một đại diện tiêu biểu của lấy mẫu (Steven M. LaValle, James J. Kuffner, Jr., 2001; LaValle, 1998). Đó là thuật toán ngẫu nhiên được trình bày để giải các bài toán lập kế hoạch đường dẫn. Nguyên lí này là việc xây dựng từng bước cây ngẫu nhiên (RRT) khám phá nhanh từ điểm bắt đầu tuy nhiên tuật toán này có thời gian và chi phí khá cao. RRT\* (Sertac Karaman, Emilio Frazzoli, 2011) xây dựng trên ý tưởng về RRT đã có khả năng tối ưu đường dẫn tới mức tệm cân nếu thời gian vô hạn, thuật toán này đánh một dấu mốc quan trọng trong sự phát triển của các biến thể RRT. Biến thể tiếp theo RRT\*-smart (Nasir, Islam, Malik, 2013)với mục đích đẩy nhanh tốc độ hội tụ hơn RRT\* dựa trên lấy mẫu thông minh băng cách thêm số lượng nút cụ thể, thuật toán này là một giải pháp gần tối ưu hoặc tối ưu. Tuy nhiên sau quá trình lấy mẫu thông minh số lượng các nút tăng thêm sẽ gây khó khăn cho qua trình xử lý của máy tính có bộ nhớ hạn chế. Để khắc phục điều này thì thuật toán RRT\*FN ra đời với việc loại bỏ các nút không hữu ích. Kết quả cho thấy thuật toán của này vượt trội hơn RRT và gần bằng RRT \* về độ tối ưu của đường dẫn trả về, trong khi cần số lượng nút được lưu trữ trong cây ít hơn nhiều (Olzhas Adiyatov, Huseyin Atakan Varol, 2013). F-RRT \* một giải pháp nhanh hơn RRT \*. F-RRT \* tối ưu hóa chi phí của các đường đi bằng cách tạo ra các nút cha ngẫu nhiên gần chướng ngại vật hơn. Quá trình tạo có thể được chia thành hai bước FindReachest và CreatNode, Mặc dù quá trình của việc tạo được thực hiện trong quá trình thêm từng nút ngẫu nhiên vào cây, nó không phải lúc nào cũng thành công (Bin Liao, F0angyi Wan a,Yi Hua ,Ruirui Ma , Shenrui Zhu, Xinlin Qing, 2020).

Một lý do cơ bản khiên tốc độ hội tụ chậm và tốn nhiều bộ nhớ là việc tạo ra thêm nút lấy mẫu. Dựa trên ý tưởng tối ưu đường dẫn của RRT\*-smart (Nasir, Islam, Malik, 2013) bằng cách kết nối những nút có thể nhìn thấy trực tiếp với nhau, bài báo này đề xuất một thuật toán mới RRT\* ...... Ngoài việc tối ưu đường dẫn, chúng tôi tối ưu tối đa đường dẫn mà không cần tạo ra thêm nút lấy mẫu mới. Bằng cách chia nhỏ đường dẫn ban đầu đường dẫn tìm được giống như RRT\* với số lần hữu hạn sẽ tạo ra các nút gần chướng ngại vật hơn đường dẫn mới tìm được sẽ dần cải thiện tới mức tiệm cận với tốc độ hội tụ nhanh.

**2. Background**

**2.1. Problem Formulation**

Trong phần này, bài báo sẽ trình bày một vài vấn đề về định nghĩa ký hiệu sẽ liên quan đến phần còn lại của bài báo. Đặt là một không gian cấu hình, trong đó . Đặt là vùng vật cản. là không gian tự do . Đặt và lần lượt là điểm bắt đầu và đích. Các vùng không gian được mô tả như trên hình 1.

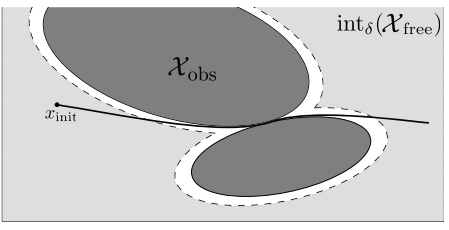


Figure 1. Không gian di chuyển của robot

**Định nghĩa 1 (Lập kế hoạch đường đi khả thi)**

Một đường dẫn được định nghĩa là một hàm liên tục : thuộc không gian cấu hình . Đường dẫn được gọi là một đường dẫn khả thi nếu nó đảm bảo: , và . Xét trên toàn bộ không gian mà robot di chuyển, the *total variation* của một đường dẫn là chiều dài của đường dẫn đó hay là khoảng cách giữa hai điểm trong (the Euclidean distance traversed by the path in ). Theo [RRT\_2011] the *total variation* được cho bởi:

với(1)

Đặt là tập hợp tất cả các đường dẫn và tập hợp tất cả con đường khả thi. We have:

(2)

Xét two paths và trong không gian cấu hình . Đặt denote their concatenation. Không mất tính tổng quát một đường dẫn có thể có nhiều đường dẫn khả thi trong đó. Đặt là cost function của đường dẫn khả thi: . , if only if , . We have:

, (3)

**Định nghĩa 2: (hoạch định con đường tối ưu)** đường dẫn là đường dẫn tối ưu phải thỏa mãn cost function và

**Định nghĩa 3: (con đường tối ưu nhanh)** là tìm con đường tối ưu trong thời gian ít nhất có thể

**3. Related work**

**3.1. RRT\* algorithm**

Phần này chúng tôi giới thiệu thuật toán RRT\* [xx] là một biến thể của thuật toán RRT [xx] . RRT\* là một thuật toán lập kế hoạch đường đi dựa trên việc lấy mẫu, RRT\* có các đường dẫn tối ưu ở mức tiệm cận và đảm bảo thời gian nhanh hơn RRT. RRT \* là một cây , các đỉnh được lấy mẫu từ không gian trạng thái không có chướng ngại vật và các cạnh kết nối các đỉnh này với nhau. Đầu vào của RRT\* bao gồm điểm bắt đầu (, điểm đích ( và môi trường . Đầu tiên tập hợp các điểm ngẫu nhiên được tạo ra trong không gian cấu hình : . Do robot không thể di chuyển vào vùng chứa chướng ngại vật do đó tập hợp lấy mẫu cần thêm điều kiện Vì vậy, hàm là dãy con của {chỉ chứa các mẫu trong  :

(4)

Như vậy cây RRT\* phát triển bằng việc lấy mẫu từ hàm trả về các chuỗi điểm ngẫu nhiên độc lập trong . Mỗi một điểm ngẫu nhiên đó trong được ký hiệu: . Bước tiếp theo trong thuật toán là tìm điểm gần nhất với từ các đỉnh đã có trước đó . Hàm trả về đỉnh trong V gần nhất với  về mặt khoảng cách. Hàm tìm điểm gần nhất được cho bởi

(5)

Điểm v ∈ V  được trả ra từ hàm ký hiệu là . Đỉnh tiếp theo trong tập của cây RRT\* là một điểm mới được tạo ra từ hai điểm và bằng function Steer :

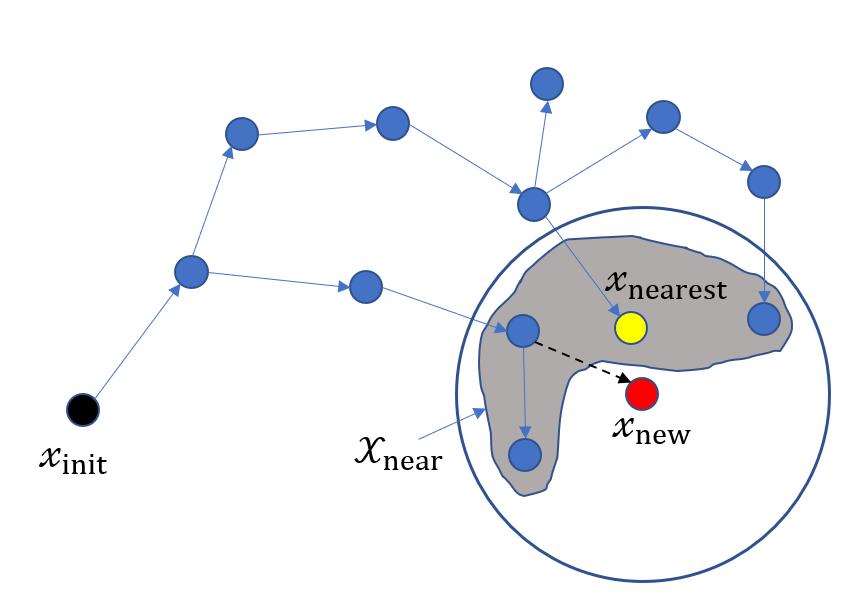
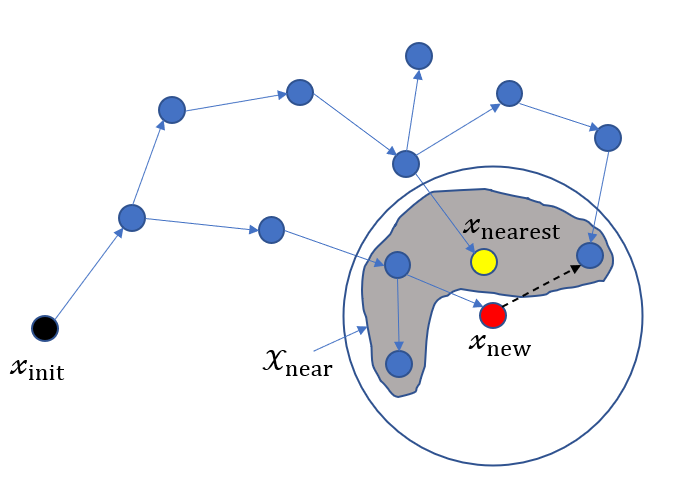
(6)

where  là a ball of radius  , centered at . Nếu nằm phía ngoài the ball thì nằm trên biên ball sao cho: , nếu nằm trong ball thì

Sau đó thuật toán tiếp tục kiểm tra sự kết nối giữa hai điểm và bằng hàm: . Function trả về giá trị 1 (TRUE) nếu đoạn thẳng không đi qua vật cản ( ). trả về giá trị 0 (FALSE) nếu đoạn thẳng đi qua vật cản ( ). Tiếp theo thuật toán tìm và trả về tập hợp tất cả các điểm thuộc hình cầu :

(4)

So với RRT, RRT\* cải tiến hơn khi có thêm hai quy trình ChooseParent (xem hình 1) và Rewire (xem hình 2).

Trong function ChooseParent, điểm nhận tất cả các điểm làm bố mẹ tạm thời, xét chi phí của tất cả các con đường đó từ điểm đến thông qua các điểm (hàm CollisionFree được sử dụng để đảm bảo rằng không có tồn tại vật cản giữa các điểm đang xét):

(5)

Trong đó hàm là chi phí từ đến , hàm là hàm đoạn thẳng nối hai điểm và , hàm kí hiệu khoảng cách của đoạn thẳng

Trong các con đường được giả định kết nối, con đường nào có chi phí thấp nhất thì nhận điểm thuộc con đường đó làm bố mẹ. Function ChooseParent được mô tả như thuật toán sau

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm 1**  **Input**:  **Output**: | | | |
| 1: |  | | |
| 2: |  | | |
| 3: | **for** each **do** | | |
| 4: |  | **if** CollisionFree Cost **then** | |
| 5: |  |  |  |
| 6: |  |  |  |
| 7: |  | **end if** | |
| 8: | **end for** | | |
| 9: | **return** | | |

Trong thuật toán 1 là chi phí của con đường nhỏ nhất qua đến . Điểm là bố mẹ của điểm thỏa mãn:

CollisionFree Cost (6)

Trong function Rewire, cũng giống như ChooseParent trong the ball giả định rằng tất cả các điểm nhận điểm làm bố mẹ tạm thời. Nếu đường dẫn nào thỏa mãn: thì nếu không có đường dẫn nào thỏa mãn điều kiện trên thì giữ nguyên bố mẹ ban đầu

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm**  **Input**: G  **Output:** | | | |
| 1: | **for** each **do** | | |
| 2: |  | **if** CollisionFree Cost **then** | |
| 3: |  |  |  |
| 4: |  |  |  |
| 5: |  | **end if** | |
| 6: | **end for** | | |
| 7: | **return** | | |

Như vậy thuật toán RRT\* được mô tả như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm 1** RRT\*  **Input**:  **Output**: | | | |
| 1: | ; E | | |
| 2: | **for** i=1 to n **do** | | |
| 3: |  |  | |
| 4: |  |  | |
| 5: |  |  | |
| 6: |  | **if** CollisionFree **then** | |
| 7: |  |  |  |
| 8: |  |  |  |
| 9: |  |  | ; E |
| 10: |  |  |  |
| 11: |  | **end if** | |
| 12: | **end for** | | |
| 13: | **Return** | | |

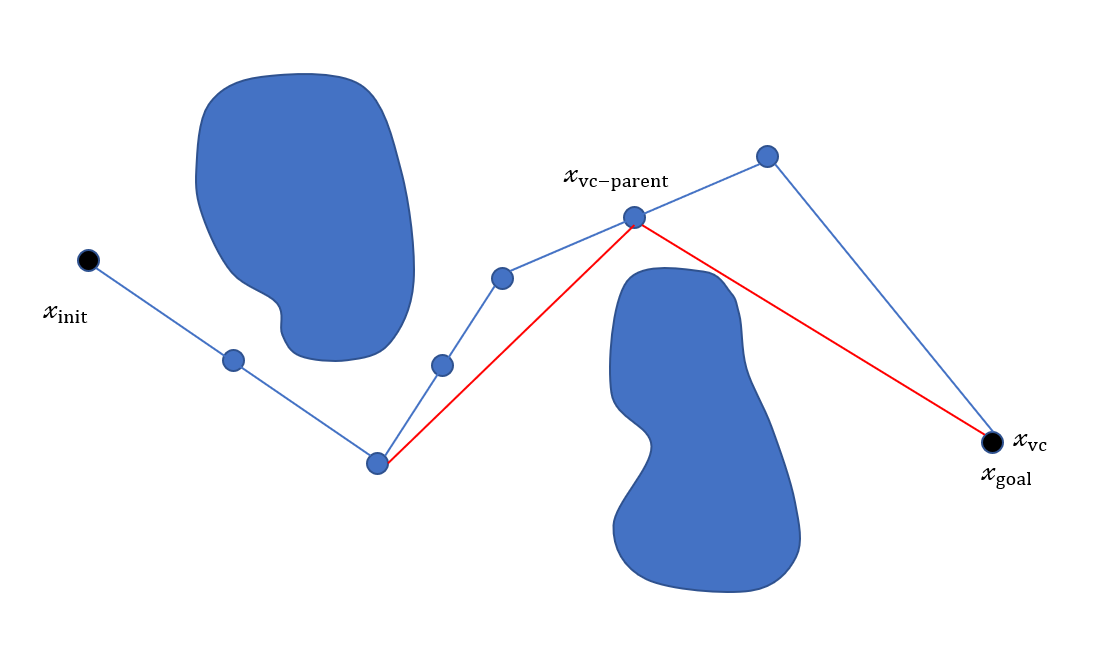
**3.2. Thuật toán RRT\*smart**

Phần này mô tả thuật toán cải tiến của RRT\*[...]. RRT\*Smart. RRT\*Smart cũng có các bước đầu tiên tương tự như RRT \*. Tuy nhiên, các giá trị chi phí của đường dẫn được trả về để tạo ra trên đường dẫn đó các điểm lấy mẫu tiềm năng thông qua hàm ( là các đỉnh của đường dẫn tối ưu từ hàm ). RRT\*-Smart được mô tả như trên thuật toán 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm 4**  **Input:**  **Output:** | | | | | | |
| 1: | ;E | | | | | |
| 2: | **for** i=1 to N **do** | | | | | |
| 3: |  | **if** i=n+b, n=2b, n+3b ... **then** | | | | |
| 4: |  |  |  | | | |
| 5: |  | **else** | | | | |
| 6: |  |  |  | | | |
| 7: |  |  |  | | | |
| 8: |  |  |  | | | |
| 9: |  |  | **if** CollisionFree(, ) **then** | | | |
| 10: |  |  |  |  | | |
| 11: |  |  |  |  | | |
| 12: |  |  |  | ; E | | |
| 13: |  |  |  |  | | |
| 14: |  |  |  | **if** InitialPathFound **then** | | |
| 15: |  |  |  |  |  | |
| 16: |  |  |  |  |  | |
| 17: |  |  |  |  | **if** | |
| 18: |  |  |  |  |  |  |
| 19: |  |  |  |  | **end if** | |
| 20: |  |  |  | **end if** | | |
| 21: |  |  | **end if** | | | |
| 22: |  | **end if** | | | | |
| 23: | **end for** | | | | | |
| 24: | **return** | | | | | |

Sau khi tìm được đường dẫn khả thi đầu tiên như RRT\*, đường dẫn có chi phí thấp hơn được tạo thành bởi việc kết nối các điểm lấy mẫu tiềm năng biasing points, có chi phí ngắn nhất mà đảm bảo không đi qua vật cản như mô tả trong thuật toán 5 và hình x

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm 5 | | | | |
| **Input:** | | | | |
| **Output:** | | | | |
| 1: | **while** | | | |
| 2: |  |  | | |
| 3: |  | **while** | | |
| 4: |  |  | **if** CollisionFree(, ) **then** | |
| 5: |  |  |  |  |
| 6: |  |  |  |  |
| 7: |  |  | **end if** | |
| 8: |  | **end while** | | |
| 9: | **end while** | | | |
| 10: | **return** | | | |

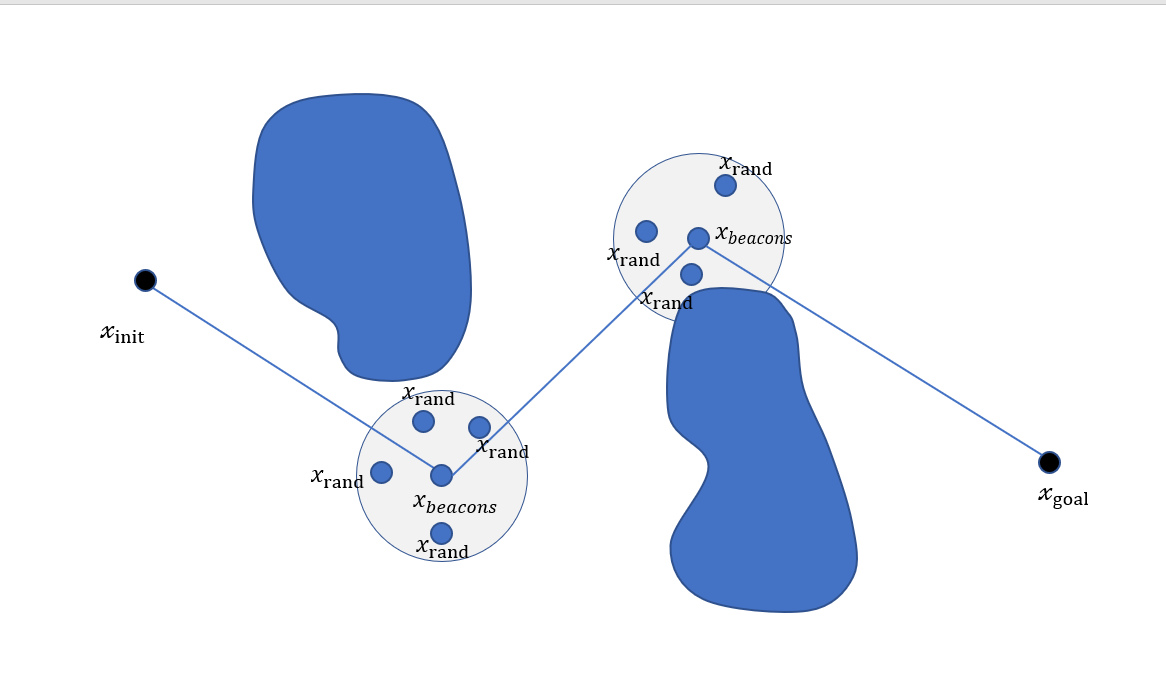


Hình x

Khoảng thời gian lấy mẫu thông minh xen kẽ với các khoảng thời gian lấy mẫu thưởng được điều chỉnh bởi một hằng số b. Những điểm mới được tạo ra nếu chi phí của con đường mới thấp hơn chi phí của con đường cũ. Việc lấy mẫu thông minh của thuật toán được thực hiện bằng cách lấy ngẫu nhiên các điểm xung quanh các điểm tiềm năng. Hàm là dãy con của {chỉ chứa các mẫu trong the ball (tâm bán kính ):

(4)

Việc lấy mẫu từ hàm trả về các chuỗi điểm ngẫu nhiên độc lập trong như trên hình y.

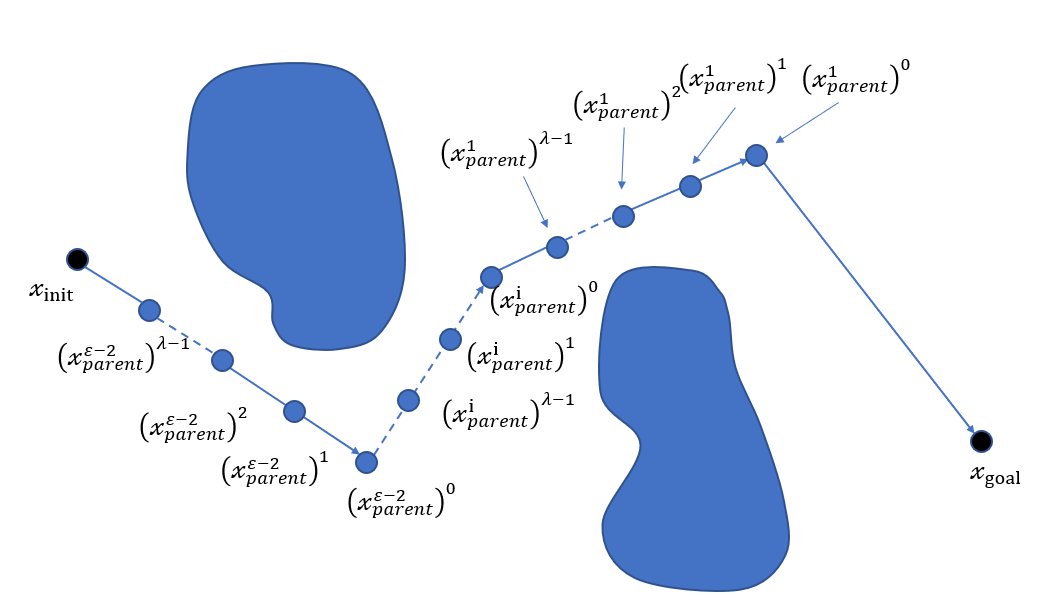
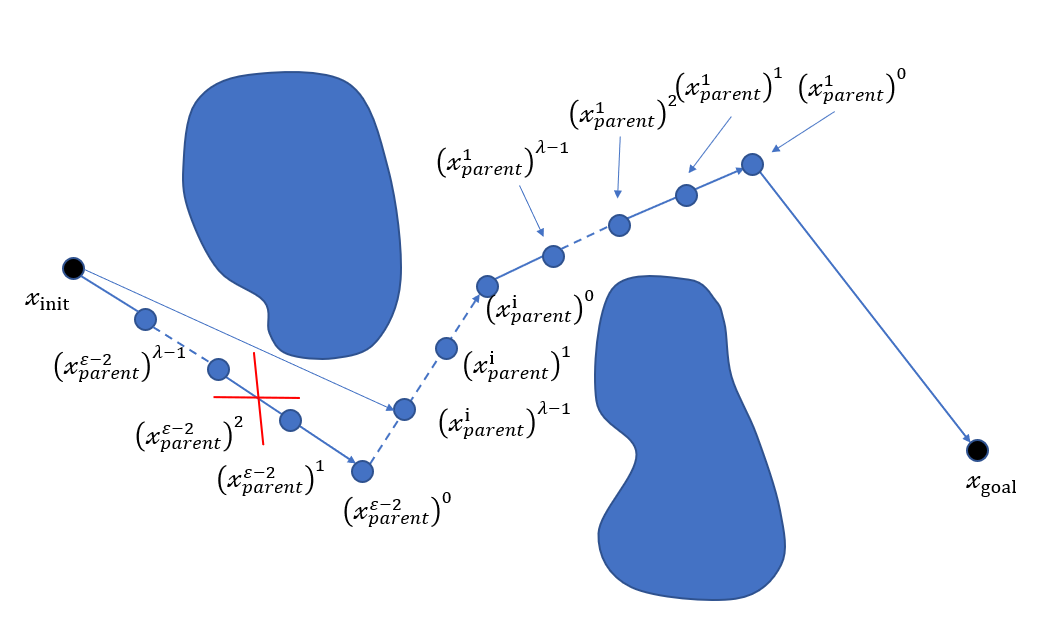


Hình y

**4. Thuật toán ABC RRT\***

Trong phần này sẽ trình bày chi tiết về thuật toán ...RRT\*. Ý tưởng của thuật toán này được hình thành trên ý tưởng là cải tiến về đường dẫn sử dụng các điểm tổ tiên của RRT\*-smart. Từ đường dẫn ban đầu được hình thành trên cơ sở của RRT\*, đặt là các đỉnh V của đường dẫn bất kỳ , ,, là tổng số đỉnh của đường dẫn. Chúng tôi phân đoạn đường dẫn theo hàm , là các đoạn thẳng bằng nhau giữa hai điểm nối liên tiếp , ( , ). Kết quả được các điểm ( ). Giá trị tăng dần theo sự phát triển của các thế hệ điều đó có nghĩa rằng các điểm gần với xgoal theo đường dẫn thì được coi là các điểm tổ tiên cao nhất, cụ thể được mô tả trong thuật toán 6 và hình 1a.

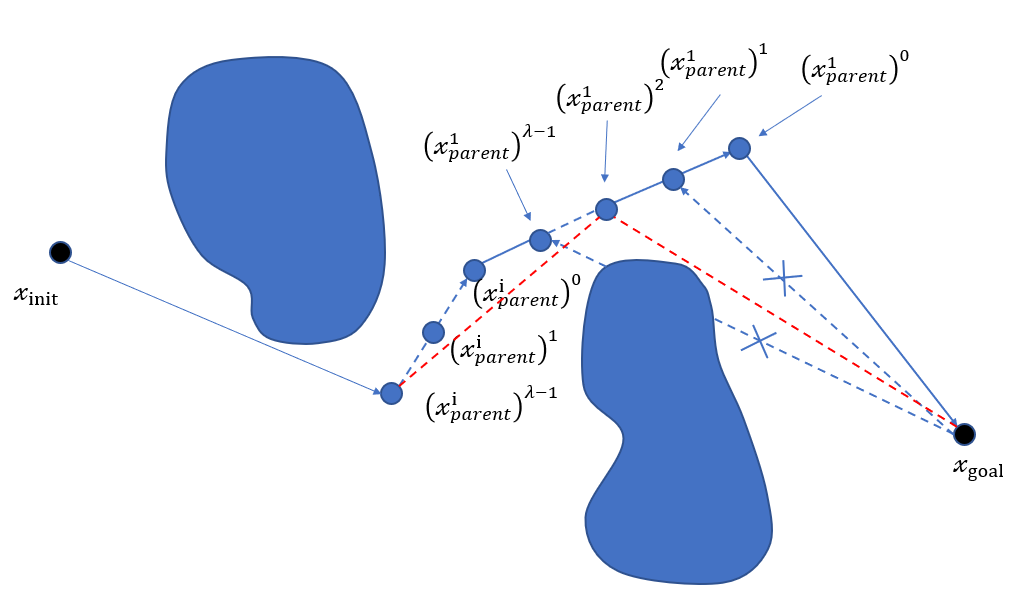
|  |  |  |
| --- | --- | --- |
| **Algorithm 6** | | |
| **Input:** | | |
| **Output:** | | |
| 1: | **for** to **do** | |
| 2: |  |  |
| 3: | **end for** | |
| 4: | **return** | |

Hình 1.a 1.b

Trên cơ sở các điểm ancester point được chia trên đường dẫn, điểm được kết nối đến lần lượt các điểm tổ tiên theo sự tăng dần các thế hệ (hình 1b). Nếu đường nối xảy ra va chạm với vật cản thì dừng lại ở điểm con cháu gần nhất , điểm không kết nối qua các điểm từ điểm đến điểm mà kết nối trực tiếp với (Thuật toán 7).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm 7** | | | | | |
| **Input:** | | | | | |
| **Output:** | | | | | |
|  | | | |
| **while** | | | |
|  | **for** to **do** | | |
|  |  | **if** **then** | | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  | **break**; | |
|  |  | **end if** | | |
|  | **end for** | | |
|  |  | | |
| **end while** | | | |
| **return** | | | |



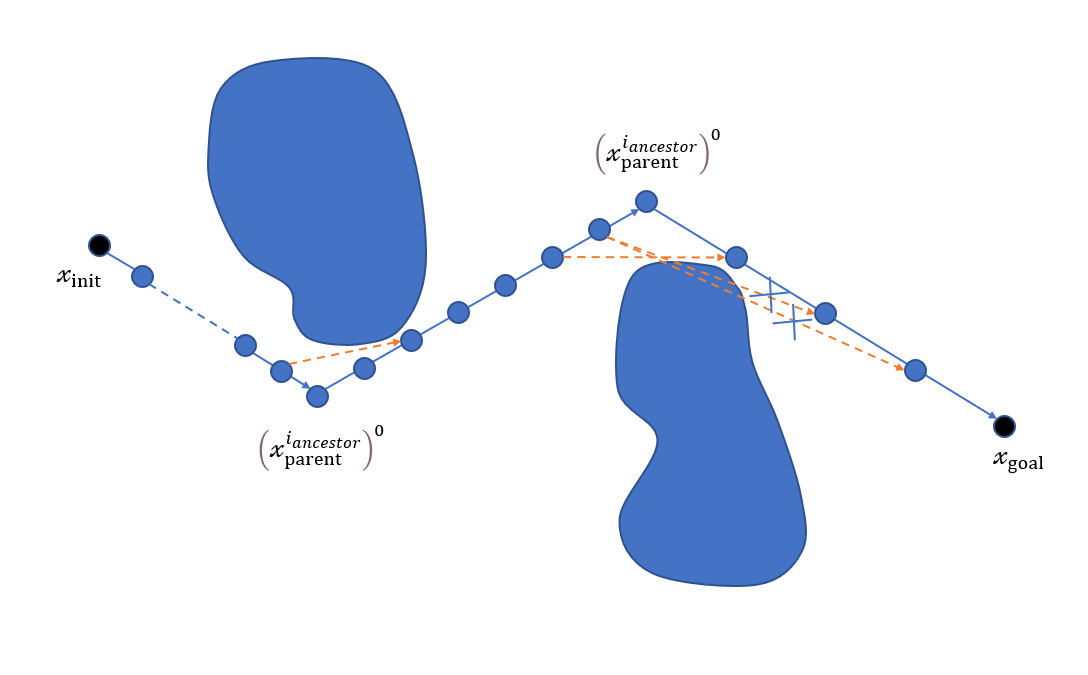
Hình 2

Tiếp theo, trên cơ sở các điểm ancester point được chia trên đường dẫn trả ra từ Algorithm 7, điểm được kết nối đến lần lượt các điểm tổ tiên theo sự giảm dần các thế hệ (hình 2). Nếu đường nối xảy ra va chạm với vật cản thì dừng lại ở điểm tổ tiên gần nhất , là kết quả đường dẫn mới theo thuật toán 8.

Hàm trả về điểm tổ tiên của nó nếu thì

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm 8 | | | | | |
| **Input:** | | | | | |
| **Output:** | | | | | |
| 1: |  | | | | |
| 2: |  | | | | |
| 3: |  | | | | |
| 4: | **while** | | | | |
| 5: |  |  | **for** j to **do** | | |
| 6: |  |  | | **if** **then** | |
| 7: |  |  | |  |  | |
| 8: |  |  | |  |  | |
| 9: |  |  | |  |  | |
| 10: |  |  | |  |  | |
| 11: |  |  | |  |  | |
| 12: |  |  | |  |  | |
| 13: |  |  | |  | **break** | |
| 14: |  |  | | **end if** | |
| 15: |  |  | **end for** | | |
| 16: |  |  |  | | |
| 17: | **end while** | | | | |
| 18 : | **return** | | | | |

Trên cơ sở đường dẫn mới được hình thành bởi thuật toán 8, ta chia nhỏ các cạnh , mỗi cạnh được chia thành các đoạn thẳng bằng nhau bắt cho tới điểm . Các điểm ancestor point được chia: , . Trong đó thể hiện đoạn thẳng cũng được xét đến. Tiếp theo, trên cơ sở các điểm ancestor point, hàm IntelligentPathOptimization bắt đầu từ   
, xgoal được kết nối đến lần lượt các điểm tổ tiên theo sự giảm dần các thế hệ (hình xx). Nếu đường nối xảy ra va chạm với vật cản thì dừng lại ở điểm tổ tiên gần nhất  theo thuật toán 9 và mô tả như ở hình xx.



Hình xx.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm 9** Intelligent | | | | | | | |
| **Input:** | | | | | | | |
| **Output:** | | | | | | | |
|  |  | | | | | | |
|  |  | | | | | | |
| 1: | **while** | | | | | | |
|  |  |  | **for** j**do** | | | | |
|  |  |  |  |  | | | | |
|  |  |  |  |  | | | | |
| 8: |  |  |  | **if** **then** | | | | |
|  |  |  |  |  |  | | | | |
|  |  |  |  |  | **if**  **then** | | | | |
|  |  |  |  |  |  |  | | | |
|  |  |  |  |  |  | **if** **then** | | | |
|  |  |  |  |  |  |  |  | | |
|  |  |  |  |  |  |  |  | | |
|  |  |  |  |  |  |  |  | | |
|  |  |  |  |  |  |  |  | | |
|  |  |  |  |  |  |  |  | | |
|  |  |  |  |  |  | **break** | | | |
|  |  |  |  |  |  | **end if** | | | |
|  |  |  |  |  | **else** | | | | |
|  |  |  |  |  |  |  | | | |
|  |  |  |  |  |  |  | | | |
|  |  |  |  |  |  |  | | | |
|  |  |  |  |  |  |  | | | |
|  |  |  |  |  |  |  | | | |
|  |  |  |  |  | **break** | | | | |
|  |  |  |  |  | **end if** | | | | |
|  |  |  |  | **end if** | | | | |
|  |  |  | **end for** | | | | |
|  | **end while** | | | | | | |
|  |  | | | | | | |
| 9: | **return** | | | | | | |

Hàm trả về chi phí của đường dẫn để đi từ đến

Như vậy thuật toán ABC RRT\* được mô tả như trên thuật toán 10.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm 10**  **Input:**  **Output:** | | | | | |
| 1: | ; E | | | | |
| 2: | **for** i=1 to n **do** | | | | |
| 3: |  |  | | | |
| 4: |  |  | | | |
| 5: |  |  | | | |
| 6: |  | **if** CollisionFree **then** | | | |
| 7: |  |  |  | | |
| 8: |  |  |  | | |
| 9: |  |  | ; E | | |
| 10: |  |  |  | | |
|  |  |  | **if** **then** | | |
|  |  |  |  |  | |
|  |  |  |  |  | |
|  |  |  |  |  | |
|  |  |  |  |  | |
|  |  |  |  | **while** | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | **end while** | |
|  |  |  | **end if** | | |
| 11: |  | **end if** | | | |
| 12: | **end for** | | | | |
| 13: | **return** | | | | |

**5 Mô phỏng kết quả**

Trong phần này, .....-RRT ∗ được so sánh với thuật toán hiện có RRT\* trong 4 map 2 chiều

Với các môi trường đơn giản Simple, mê cung Maze, lộn xộn Cluttered

Với các tham số trong mô phỏng là radius , ,

Để so sánh công bằng, các thông số mô phỏng giống nhau đối với các thuật toán. Trong thủ tục ChooseParent và Rewire thì = 150

mô phỏng là Matlab. Bốn map thử nghiệm có cùng kích thước.



1)Simple



2)Maze 1



3)Maze 2

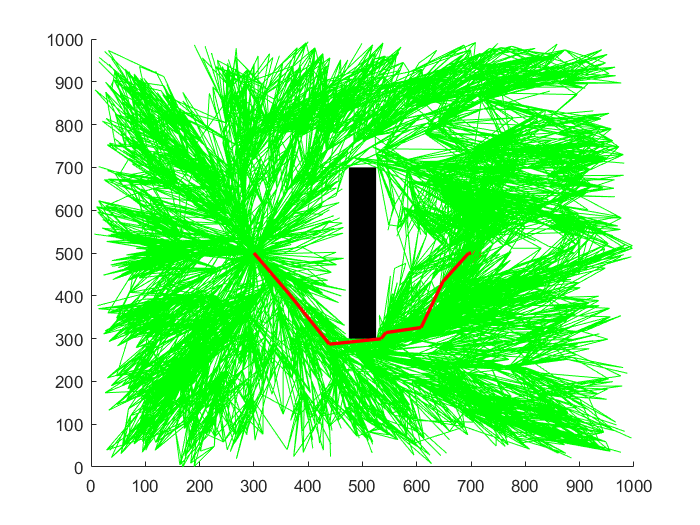


4)Cluttered

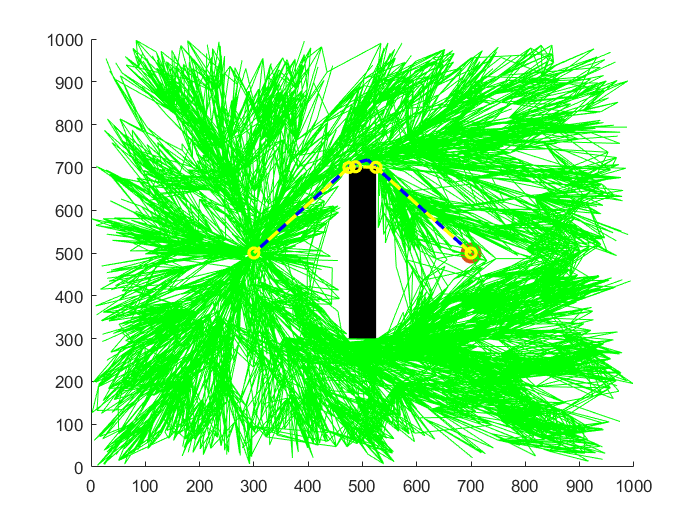
**1) map simple**

Hình 1.1 cho thấy đường dẫn trong môi trường simple của hai thuật toán với cùng N=2000, đường dẫn của RRT\* ( C=650.848988235233) ở hình 1.1a và đường dẫn của..RRT\* (C\_g=594.966695249414, C=581.772257351105) tối ưu hơn gần tới mức tiệm cận ở hình 1.1b

Như có thể thấy trong hình 1.2 chi phí của thuật toán theo số lần lặp N ..RRT\* gần như tối ưu ở cả con đường cho thấy thuật toán ...RRT\* hội tụ nhanh hơn rất nhiều ở môi trường simple



a)RRT\*C=650.848988235233



b)...RRT\*C\_g=594.966695249414, C=581.772257351105

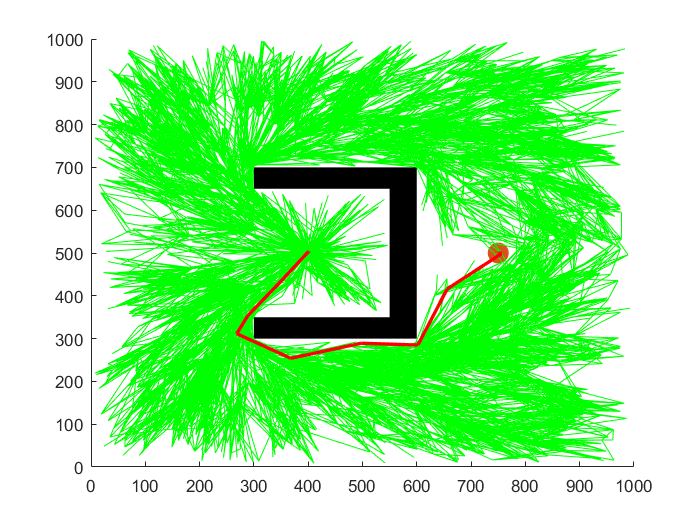
Hình 1.1: kết quả thực nghiệm của hai thuật toán trên môi trường simple

Hình 1.2: biểu đồ Cost theo số lần lặp N trong môi trường simple

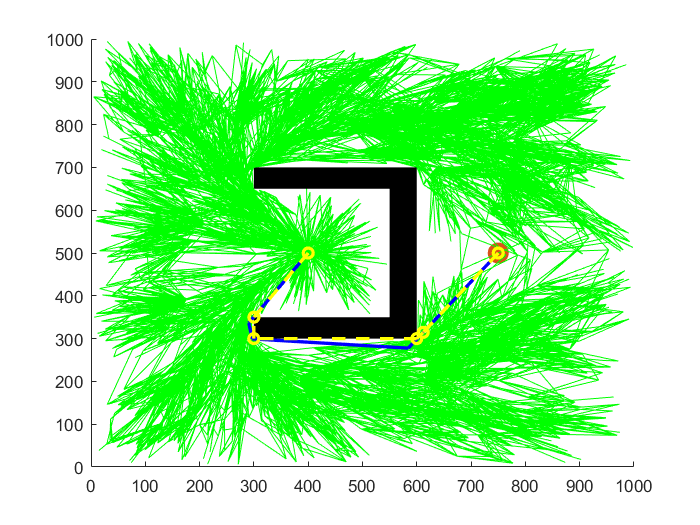
**2) map maze 1**

ở Hình 2.1 cho thấy đường dẫn trong môi trường maze đơn giản của hai thuật toán với cùng N=3000, cả 2 đường dẫn của..RRT\* (..rrt\*C\_g= 799.903795919975,C= 780.570004095919) ở hình 1.1b ta thấy có chi phí tôi ưu hơn nhiều so với đường dẫn của RRT\*( C= 904.553525650686)

trong hình 1.2 chi phí của cả hai con đường của thuật toán ..RRT và chi phí của đường dẫn theo thuật toán RRT\* thấy được chi phí của con đường tăng giảm không phụ thuộc vào số lần lặp N tuy nhiên cả hai con đường của thuật toán ...RRT\* đều tốt hơn nhiều so với đường dẫn RRT\* đường dẫn này giảm nhưng khá chậm khi số lần lặp tăng



1. RRT\* C= 904.553525650686



b)..rrt\*C\_g= 799.903795919975,C= 780.570004095919

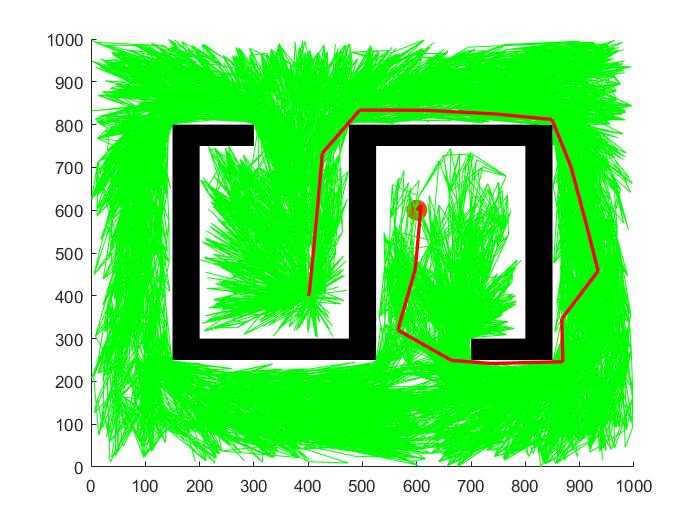
Hình 2.1: kết quả thực nghiệm của hai thuật toán trên môi trường maze1

Hình 2.2: biểu đồ Cost theo số lần lặp N trong môi trường maze1

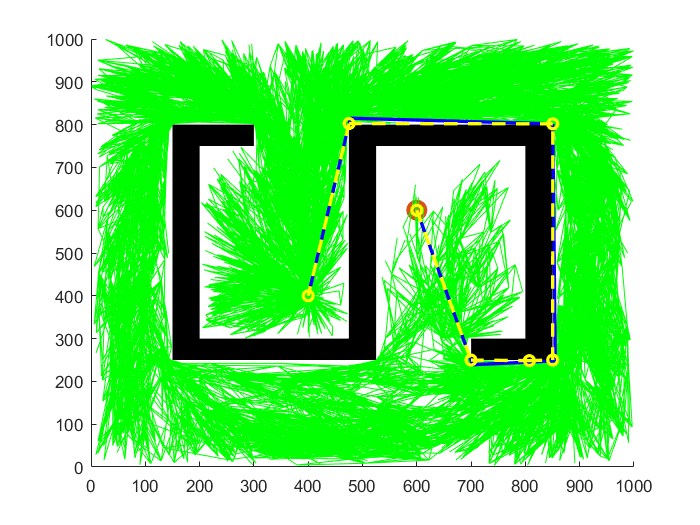
**3) map maze2**

ở hình 3.1 ta thấy được đường dẫn của cả hai thuật toán trong môi trường maze phức tạp với N=4000. Hình 3.1a đường dẫn của rrt\* (C= 2221.19787175945) có chi phí khá lớn, hình 3.1.b là hai đường dẫn của ...rrt\*(C\_g= 1876.83463602032,C= 1851.07602408406) chi phí ở gần mức tiệm cận

hình 3.2 là chi phí theo số lần lặp N của môi trường maze 2 ta thấy được chi phí của đường dẫn rrt\* giảm theo số lần lặp N tăng, khi càng tăng chi phí càng giảm ít, còn chi phí cuối cùng của đường dẫn ...rrt\* có chi phí tối ưu gần mức tiệm cận với N rất nhỏ



a)rrt\* C= 2221.19787175945



b)..rrt\*C\_g= 1876.83463602032,C= 1851.07602408406

Hình 3.1: kết quả thực nghiệm của hai thuật toán trên môi trường maze2

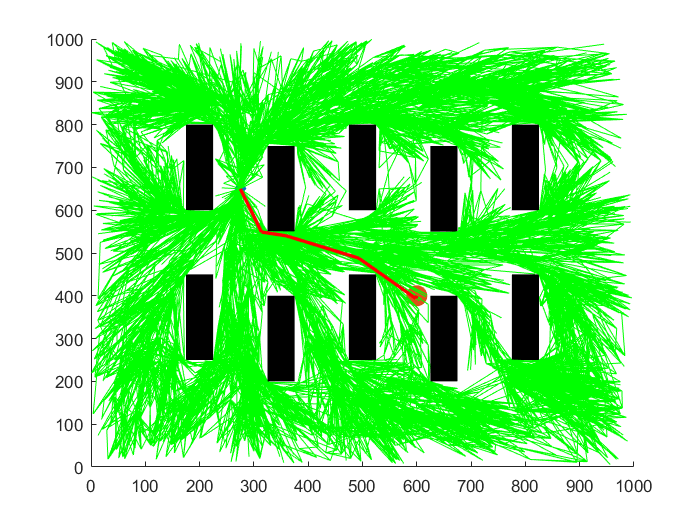
Hình 3.2: biểu đồ Cost theo số lần lặp N trong môi trường maze2

**4) map Cluttered**

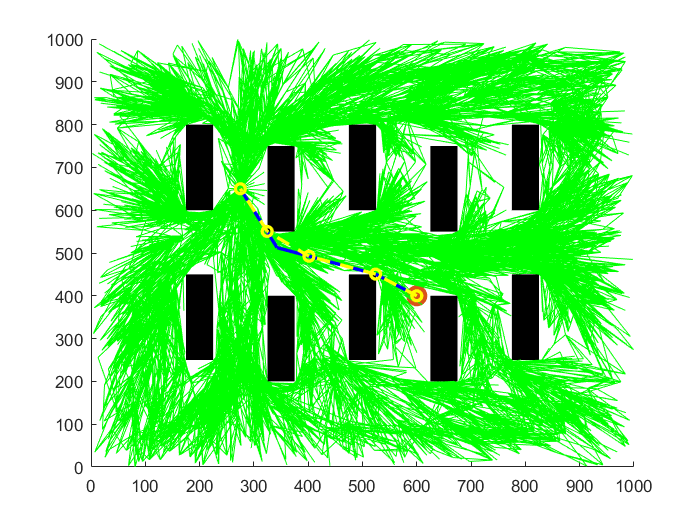
Hình 4.1a biểu diễn đường dẫn của thuật toán RRT\* (C= 496.224628358544) trong môi trường Cluttered hình 4.1.b biểu diễn hai đường dẫn của thuật toán.. rrt\* (C\_g= 436.207141069025,C= 428.396846432546) cả hai thuật toán đều chạy cùng số lần lặp là 3000

Hình 4.2 biểu diễn biểu đồ cost theo số lần lặp trong môi trường Cluttered của cả hai thuật toán

Tại ở môi trường này ta cũng có thể thấy đường dẫn của thuật toán RRT\* tốt hơn về mặt chi 3phí khi cùng với số lần lặp, đường dẫn của thuật toán ...RRT\* cũng có chi phí tối ưu gần tới mức tiệm cận với số lần lặp N nhỏ, N càng tăng thời gian càng tăng do thời gian phụ thuộc chủ yếu vào số lần lặp N do đó thuật toán của chúng tôi tìm được con đường có chi phí tối ưu gần tới mức tiệm cận nhanh hơn rất nhiều so với thuật toán RRT\*



1. rrt\* C= 496.224628358544



b)..rrt\*C\_g= 436.207141069025,C= 428.396846432546

Hình 4.1: kết quả thực nghiệm của hai thuật toán trên môi trường Cluttered

Hình 4.2: biểu đồ Cost theo số lần lặp N trong môi trường Cluttered

**5 kết luận**