HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

—————o0o—————

BÀI TẬP LỚN TOÁN RỜI RẠC 2

Đề tài: "THUẬT TOÁN BATCH-INFORMED RRT* (BIT*)"

Giảng Viên Hướng Dẫn: TS. Nguyễn Kiều Linh
Nhóm thực hiện: Nhóm 3
Lớp: Nhóm 14 Toán rời rạc 2
Hệ đào tạo: Đại học chính quy

Hà Nội, 05/2025

Bảng phân công công việc

STT	Họ và tên	Mã sinh viên	Công việc
1	Nguyễn Đức Toàn	B23DCCN831	Code báo cáo
2	Trần Thu Thiên	B23DCCN785	Làm slide, thuyết trình
3	Hoàng Xuân Vinh	B23DCCN929	Code báo cáo
4	Nguyễn Thị Uyên	B23DCCN912	Soạn nội dung
5	Trần Đăng Dương	B23DCCN227	Soạn nội dung
6	Nguyễn Văn Bằng	B23DCCN067	Code thuật toán, soạn nội dung
7	Đại Đức Hiếu	B23DCCN297	Code báo cáo
8	Nguyễn Quế Hoàng	B23DCCN339	Soạn nội dung

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:**)

Hà Nội, ngày tháng năm 20...

Giảng viên hướng dẫn

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Điểm: **(Bằng chữ:**)

Hà Nội, ngày tháng năm 20...

Giảng viên phản biện

Mục lục

Mở đầu

Bài toán lập kế hoạch đường đi (path planning) là một bài toán tìm kiếm một chuỗi các cấu hình hợp lệ để di chuyển một đối tượng từ một trạng thái bắt đầu (source/start configuration S hoặc x_{init}) đến một trạng thái đích (destination/goal configuration G hoặc x_{goal} hoặc goal region G(x_{goal})). Một cách cụ thể hơn, bài toán lập kế hoạch đường đi là tính toán một đường đi liên tục tối ưu kết nối trạng thái bắt đầu S và trạng thái đích G, đồng thời tránh va chạm với các chướng ngại vật đã biết trong không gian trạng thái tự do X_{free} Bài toán lập kế hoạch tối ưu được định nghĩa là việc tìm kiếm đường đi tối thiểu hóa một hàm chi phí (c() hoặc s()) trên đường đi đó, trong số tất cả các đường đi hợp lệ. Hàm chi phí thường là chiều dài của đường đi đã đi, nhưng cũng có thể là các tiêu chí khác như thời gian thực hiện hoặc năng lượng tiêu thụ.

Trong đề tài của phần này chúng em sẽ trình bày về thuật toán Batch-Informed RRT* (BIT*) nhằm giải quyết bài toán trên

Nội dung gồm các phần sau :

Mục 1: Giới thiệu: Giới thiệu qua về Bài toán lập kế hoạch chuyển động và tình hình của một số thuật toán điển hình khác. Giới thiệu BIT* và chỉ ra điểm đặc biệt của BIT*.

Mục 2: Nền tảng chuẩn bị : Trình bày các kiến thức nền tảng và kỹ năng cần thiết cho việc nghiên cứu và phát triển các thuật toán lập kế hoạch chuyển động.

Mục 3: Thuật toán BIT*: Giới thiệu ý tưởng, giải thích và trình bày thuật toán. Đồng thời nêu mối quan hệ với các thuật toán khác

Mục 4: Các mã giả code, kèm link code c++ và demo

Mục 5: So sánh và đánh giá : Dựa trên kết quả thực nghiệm, có thể rút

ra các so sánh giữa các thuật toán, đồng thời nêu đánh giá

Mục 6: Ứng dụng: Nêu ứng dụng của thuật toán BIT* trong các vấn đề được đặt ra.

Chương 1

Giới thiệu

I. Giới thiệu về bài toán lập kế hoạch chuyển động

Mục tiêu

Bài toán lập kế hoạch đường đi (path planning) là một vấn đề cốt lõi trong robot, trí tuệ nhân tạo và nhiều ứng dụng thực tiễn. Mục tiêu là tìm một đường đi liên tục từ trạng thái bắt đầu đến trạng thái đích, tránh va chạm với chướng ngại vật trong không gian trạng thái tự do và tối ưu hóa hàm chi phí, chẳng hạn như độ dài đường đi, thời gian hoặc năng lượng tiêu thụ.

Đầu vào:

Không gian cấu hình (configuration space): Biểu diễn tất cả trạng thái hợp lệ của robot.

Trạng thái bắt đầu và kết thúc.

Tập chướng ngại vật: Các vùng mà robot không được đi vào.

Đầu ra:

Một đường đi khả thi (có thể tối ưu hoặc không), đảm bảo tránh va chạm và có thể thực thi bởi robot.

II. Các phương pháp giải điển hình

A. Thuật toán lập kế hoạch dựa trên ô (Grid-based)

Ví dụ: **A*, Dijkstra**

Ý tưởng: Chia không gian thành các ô nhỏ và tìm đường như trên bản đồ.

Ưu điểm:: Dễ hiện thực, tìm được đường tối ưu.

Nhược điểm: : Không hiệu quả cho không gian lớn hoặc nhiều bậc tự do (high-dimensional C-space).

B. Thuật toán dựa trên mẫu (Sampling-based)

Đây là nhóm thuật toán hiệu quả cho không gian cấu hình lớn và phức tạp.

Thuật toán	Đặc điểm
PRM (Probabilistic Roadmap)	Tiền xử lý tạo đồ thị toàn cục → thích hợp cho môi trường tĩnh.
RRT (Rapidly-Exploring Random Tree)	Phát triển cây từ điểm bắt đầu, rất nhanh nhưng không tối ưu.
RRT*	Phiên bản cải tiến của RRT, đảm bảo hội tụ về lời giải tối ưu theo thời gian.
BIT*	Kết hợp RRT* và A*, dùng heuristic để hướng dẫn tìm kiếm → hội tụ nhanh hơn.
FMT* (Fast Marching Tree)	Duyệt theo front giống A*, nhanh và hiệu quả với nhiều ràng buộc động học.

C.Thuật toán dựa trên quy hoạch toán học (Optimization-based)

Ví dụ: **CHOMP, TrajOpt, STOMP**

Ý tưởng: Tối ưu hóa quỹ đạo ban đầu để giảm va chạm và chi phí

Ưu điểm: Đường đi mượt và khả thi động học.

Nhược điểm: Dễ mắc kẹt ở cực trị cục bộ.

III. Giới thiệu thuật toán BIT*

BIT* là một thuật toán lập kế hoạch chuyển động hiện đại, thuộc nhóm **sampling-based** (dựa trên lấy mẫu), được đề xuất để **kết hợp ưu điểm của hai hướng tiếp cận mạnh nhất hiện nay:**

Từ **RRT***: có khả năng mở rộng cây trong không gian cấu hình và hội tụ về đường đi tối ưu theo thời gian.

Từ **A***: sử dụng thông tin heuristic để dẫn hướng quá trình tìm kiếm một cách hiệu quả.

Điểm đặc biệt của BIT*

Đặc điểm	**Mô tả**
Batch Sampling (Lấy mẫu theo lô)	Thay vì lấy từng điểm một như RRT*, BIT* lấy mẫu theo từng lô → tăng hiệu quả và kiểm soát tốt hơn việc mở rộng.
Sử dụng heuristic như A*	BIT* đánh giá chi phí tiềm năng $f = g + h$, trong đó: – g: chi phí thực đến một đỉnh trong cây – h: ước lượng chi phí còn lại đến đích ⇒ giúp cây mở rộng theo hướng "thông minh", tránh lan tỏa mù mờ như RRT.
Cấu trúc hàng đợi ưu tiên	Cạnh và đỉnh được mở rộng theo thứ tự ưu tiên thấp nhất về chi phí dự kiến (giống A*) ⇒ tìm được đường ngắn hơn nhanh hơn.
Chỉ mở rộng những phần có tiềm năng cải thiện lời giải	BIT* chỉ xem xét những điểm có tổng chi phí nhỏ hơn lời giải tốt nhất hiện tại c_{best}, giúp loại bỏ các nhánh không hiệu quả.
Hội tụ nhanh đến đường đi tối ưu	Nhờ những cơ chế trên, BIT* thường tìm được lời giải ban đầu nhanh, rồi dần cải thiện về tối ưu giống RRT* nhưng nhanh hơn đáng kể.

Chương 2

Nền tảng chuẩn bị

Mục này trình bày các kiến thức nền tảng và kỹ năng cần thiết cho việc nghiên cứu và phát triển các thuật toán lập kế hoạch chuyển động. Nội dung được chia thành năm phần chính: toán học nền tảng, kỹ năng lập trình, cấu trúc dữ liệu và thuật toán, cũng như các kiến thức bổ sung nâng cao có thể áp dụng trong thực tế. Mỗi phần cung cấp cơ sở vững chắc giúp người học hiểu rõ nguyên lý hoạt động, khả năng triển khai và tối ưu hóa thuật toán trong môi trường robot phức tạp.

1. Các định nghĩa và ký hiệu cơ bản

- Không gian trạng thái (X): Là tập hợp tất cả các cấu hình có thể có của robot hoặc hệ thống, thường là một tập con của $\mathbb{R}^d$, trong đó d là số chiều của không gian.
- Không gian chướng ngại vật (X_{obs}): Vùng không gian bị chiếm bởi các chướng ngại vật.
- Không gian tự do (X_{free}): Phần không gian mà robot có thể di chuyển mà không va chạm, được tính bằng $X_{\text{free}} = X \setminus X_{\text{obs}}$.
- Trạng thái bắt đầu (x_{start}): Cấu hình ban đầu của robot, $x_{\text{start}} \in X_{\text{free}}$.
- Trạng thái đích (x_{goal}): Cấu hình mà robot cần đạt tới, $x_{\text{goal}} \in X_{\text{free}}$.
- Đường đi (σ): Một chuỗi liên tục các trạng thái kết nối từ trạng thái bắt

đầu và trạng thái đích. Một đường đi khả thi là một đường đi không có va chạm và thỏa mãn các điều kiện bắt đầu và kết thúc.

- Hàm chi phí ($c(\sigma)$): Một hàm biểu thị chi phí của việc đi theo đường đi. Chi phí này thường là khoảng cách Euclid giữa hai điểm không có chướng ngại vật trên đường đi, hoặc vô cùng nếu có va chạm.

2. Bán kính kết nối RGG

Bán kính kết nối RGG, ký hiệu là r, xác định khoảng cách tối đa mà hai điểm mẫu được xem xét để kết nối với nhau. Việc lựa chọn bán kính này ảnh hưởng lớn đến tính chất của đồ thị được tạo ra (ví dụ: mật độ kết nối) và khả năng hội tụ của thuật toán đến lời giải tối ưu, cần thiết để đảm bảo tính tối ưu tiệm cận giảm khi số lượng mẫu tăng lên.

Công thức:

$$r = \gamma \left(\frac{\log q}{q} \right)^{1/d}$$

Trong đó:

- γ: Một hằng số điều chỉnh. Giá trị của γ phụ thuộc vào đặc tính của không gian trạng thái và hàm chi phí.
- q: Tổng số điểm đang được xem xét (bao gồm các đỉnh trong cây V và các mẫu chưa kết nối X_{unconn}).

$$q = |V| + |X_{\text{sample}}|$$

- d: Số chiều của không gian trạng thái.

Công thức này cho thấy bán kính r giảm khi số lượng điểm q tăng lên. Điều này đảm bảo rằng khi có nhiều mẫu hơn, thuật toán chỉ cần xem xét các kết nối trong một vùng lân cận nhỏ hơn, giúp giảm chi phí tính toán cho việc kiểm tra va chạm và tìm kiếm lân cận.

3. Hàm Heuristic

Hàm heuristic ($h(x)$) cung cấp ước lượng chi phí từ một trạng thái x hiện tại đến trạng thái đích x_{goal}. Trong BIT*, heuristic được sử dụng để định hướng quá trình tìm kiếm, ưu tiên mở rộng các đỉnh và xem xét các cạnh có tiềm năng nằm trên đường đi tối ưu.

Công thức: Hàm heuristic thường là khoảng cách Euclid giữa điểm hiện tại và điểm đích.

$$h(n) = \sqrt{(x_{\text{goal}} - x_n)^2 + (y_{\text{goal}} - y_n)^2}$$

(với $h(n)$ bằng $h(x)$, x_n bằng x, $y_n = y$)

- Trong bối cảnh tổng quát của BIT*, chi phí ước lượng của một nút hoặc cạnh được kết hợp với chi phí thực tế đã biết từ điểm bắt đầu. Tổng chi phí ước lượng của một đỉnh x là:

  $$f(x) = g(x) + h(x)$$

  trong đó $g(x)$ là chi phí thực tế từ x_{start} đến x.

4. Lấy mẫu có thông tin

Kỹ thuật lấy mẫu có thông tin tập trung việc lấy mẫu vào một vùng không gian hẹp hơn, cụ thể là bên trong một hình ellipsoid, sau khi một đường đi khả thi đầu tiên được tìm thấy. Vùng ellipsoid này được định nghĩa bởi điểm bắt đầu, điểm đích và chi phí của đường đi tốt nhất hiện tại (c_{best}). Mục đích là chỉ lấy mẫu ở những vị trí có khả năng cải thiện giải pháp hiện tại, loại bỏ các vùng không gian không cần thiết và tăng hiệu quả hội tụ đến nghiệm tối ưu.

Công thức: Các điểm mẫu được lấy sao cho tổng khoảng cách từ điểm mẫu đến điểm bắt đầu và điểm đích nhỏ hơn hoặc bằng c_{best}:

$$\|x - x_{\text{start}}\| + \|x - x_{\text{goal}}\| \leq c_{\text{best}}$$

Trong đó:

- c_{best}: Chi phí của đường đi tốt nhất hiện tại.
- $\|\cdot\|$: Ký hiệu cho chuẩn Euclid.

5. Cắt tỉa (Pruning)

Mô tả và ý nghĩa: Cắt tỉa là quá trình loại bỏ các đỉnh và cạnh trong cây tìm kiếm cũng như các điểm mẫu không còn khả năng cải thiện chi phí của đường đi tốt nhất hiện tại (c_{best}). Điều này giúp giảm kích thước của cây và tập mẫu, tập trung tài nguyên tính toán vào các khu vực có tiềm năng tìm thấy đường đi tốt hơn.

Công thức: Một đỉnh x hoặc một mẫu x sẽ bị cắt tỉa nếu thỏa mãn điều kiện:

$$g(x) + h(x) \geq c_{\text{best}}$$

Trong đó:

- $g(x)$: Chi phí thực tế từ x_{start} đến điểm x.
- $h(x)$: Chi phí ước lượng từ điểm x đến x_{goal} (heuristic).
- c_{best}: Chi phí của đường đi tốt nhất hiện tại.

6. Kết nối lại (Rewiring)

- Kết nối lại là một bước trong thuật toán BIT* (kế thừa từ RRT*) nhằm tối ưu hóa cấu trúc của cây tìm kiếm. Khi một đỉnh mới x_{new} được thêm vào cây hoặc chi phí đến một đỉnh thay đổi, thuật toán kiểm tra các đỉnh lân cận (x_{neighbor}) của x_{new} để xem liệu việc thay đổi cha của x_{neighbor} thành x_{new} có tạo ra một đường đi từ gốc đến x_{neighbor} có chi phí thấp hơn hay không. Nếu có và đường đi mới không có va chạm, cây sẽ được cập nhật bằng cách thay đổi cạnh nối đến x_{neighbor}.
- **Công thức:** Chi phí đến một đỉnh x_{new} thông qua một đỉnh lân cận x_{near} được tính và so sánh với chi phí hiện tại đến x_{new}:

Công thức Rewiring:

$$\text{cost}(x_{\text{new}}) = \min \{\text{cost}(x_{\text{near}}) + c(x_{\text{near}}, x_{\text{new}})\}$$

Giải thích:

- x_{new}: điểm mới muốn thêm vào cây.
- x_{near}: các điểm "gần" trong cây hiện tại.
- $c(x_{\text{near}}, x_{\text{new}})$: chi phí từ x_{near} đến x_{new}.
- $\text{cost}(x_{\text{new}})$: tổng chi phí tốt nhất từ gốc đến x_{new}.

Chương 3

Thuật toán BIT*

3.1 Ý tưởng chính của BIT*

BIT* (Batch Informed Trees*) là một thuật toán lập kế hoạch chuyển động tối ưu, kết hợp giữa:

- **RRT***: khả năng tìm đường trong không gian lớn và hội tụ về lời giải tối ưu theo thời gian.
- **A***: sử dụng heuristic (ước lượng chi phí) để hướng dẫn việc mở rộng cây theo hướng ưu tiên.

Ý tưởng chính:

- Lấy mẫu ngẫu nhiên theo lô (*batch sampling*) thay vì từng điểm một.
- Chỉ tập trung vào vùng có khả năng cải thiện lời giải hiện tại.
- Duyệt các đỉnh và cạnh theo chi phí dự kiến $f = g + h$ như trong A*.

3.2 Các bước của thuật toán BIT*

Khởi tạo:

- Tạo cây tìm đường chứa đỉnh xuất phát.
- Đặt chi phí tốt nhất hiện tại $c_{best} = \infty$.
- Khởi tạo hàng đợi ưu tiên.

Lặp lại cho đến khi hết thời gian hoặc hội tụ:

1. Nếu hàng đợi rỗng:
   - Tạo lô mẫu mới trong không gian cấu hình.
   - Lọc các mẫu có $f(x) = g(x) + h(x) < c_{best}$.
   - Thêm các mẫu vào tập đỉnh chưa mở rộng.
   - Cập nhật danh sách cạnh khả thi.
2. Trong khi hàng đợi chưa rỗng:
   - Lấy cạnh có chi phí nhỏ nhất từ hàng đợi.
   - Nếu $g(v) + h(x) > c_{best}$: bỏ qua cạnh.
   - Nếu x chưa có trong cây:
     - Thêm đỉnh x vào cây qua cạnh tốt nhất (v, x).
     - Nếu x là đích $\rightarrow$ cập nhật c_{best}.

Kết thúc:

Trả về đường đi ngắn nhất tìm được.

3.3 Giải thích thuật toán

- **Batch sampling** giúp duyệt không gian hiệu quả hơn so với RRT*.
- BIT* giống A* hoạt động trên không gian liên tục, kết hợp với cấu trúc cây của RRT.
- Thay vì mở rộng tất cả điểm như PRM hay RRT*, BIT* chỉ mở rộng những điểm có tiềm năng cải thiện lời giải hiện tại.
- Sau khi có lời giải đầu tiên, BIT* tiếp tục cải thiện, hội tụ dần về tối ưu.

3.4 Mối quan hệ với các thuật toán khác

Thuật toán	Mối quan hệ với BIT*
RRT*	BIT* kế thừa khung cây mở rộng từ RRT* và đảm bảo hội tụ tối ưu. Tuy nhiên, BIT* mở rộng theo batch và có hướng dẫn heuristic.
A*	BIT* mượn chiến lược duyệt đỉnh theo chi phí $f = g+h$, ưu tiên mở rộng đỉnh hứa hẹn hơn, giúp nhanh hội tụ hơn.
PRM*	PRM* xây dựng đồ thị toàn cục trước rồi mới tìm đường. BIT* duyệt theo từng batch, hiệu quả hơn trong môi trường phức tạp.
FMT*	FMT* cũng duyệt theo chi phí như A*, nhưng BIT* sử dụng hàng đợi ưu tiên và mở rộng đỉnh linh hoạt hơn.

Chương 4

Các mã giả code, kèm link code Python và demo

Link code:

https://github.com/nvbangg/TRR2_BITstar/blob/main/Nh%C3%B3m%2014.3/BITstar.py

```python
import pygame
import random
import heapq
import math
import time
from collections import defaultdict

# Màu sắc
WHITE, RED, BLUE, GREEN, BLACK = (
    (255, 255, 255),
    (255, 0, 0),
    (0, 0, 255),
    (0, 255, 0),
    (0, 0, 0),
)

```

```
17.  # Hằng số
18.  INF = float("inf")
19.  FPS = 60
20.  WIDTH, HEIGHT = SIZE_MAP = (1200, 600)
21.  SO_VAT = 20
22.  SIZE_VAT = (100, 100)
23.  START = (200, HEIGHT - 200)
24.  GOAL = (WIDTH - 200, 200)
25.  MAX_SO_LAN_LAP = 10
26.  SIZE_LO = 25
27.  R_NUT = 3
28.  DO_RONG_CANH = 2
29.
30.
31.  class Node:
32.      def __init__(self, state):
33.          self.state = state
34.          self.g = self._khoang_cach(state, START)
35.          self.h = self._khoang_cach(state, GOAL)
36.          self.f = self.g + self.h
37.          self.g_T = INF
38.          self.parent = self
39.
40.      def __lt__(self, other):
41.          self_cost = self.cost()
42.          other_cost = other.cost()
43.          if self_cost == other_cost:
44.              return self.g_T < other.g_T
45.          return self_cost < other_cost
46.
47.      def cost(self):
48.          return min(self.g_T + self.h, INF)
49.
50.      @staticmethod
51.      def _khoang_cach(pos1, pos2):
52.          return math.hypot(pos1[0] - pos2[0], pos1[1] - pos2[1])
53.
```

```
55. class BITStar:
56.     def __init__(self):
57.         pygame.init()
58.         self.screen = pygame.display.set_mode(SIZE_MAP)
59.         pygame.display.set_caption("BIT* Path Planning")
60.         self.font = pygame.font.SysFont("Arial", 24)
61.         self.clock = pygame.time.Clock()
62.
63.         self.V = []
64.         self.E = []
65.         self.X_unconn = []
66.         self.vat_vat = []
67.         self.edge_dict = defaultdict(list)
68.
69.         self.nut_start = Node(START)
70.         self.nut_start.g_T = 0
71.         self.nut_end = Node(GOAL)
72.         self.V.append(self.nut_start)
73.
74.         self.r = INF
75.         self.Q_E = []
76.         self.Q_V = []
77.         self.V_old = []
78.         self.path_new = []
79.         self.path_old = []
80.         self.c_i = INF
81.         self.running = True
82.         self.so_lan_lap = 0
83.         self.finished = False
84.
85.     def trong_vat(self, state):
86.         return not any(obs.collidepoint(state) for obs in self.vat_vat)
87.
88.     def trong_elip(self, state):
89.         if self.c_i == INF:
90.             return True
91.         dist_start = math.hypot(state[0] - START[0], state[1] - START[1])
92.         dist_goal = math.hypot(state[0] - GOAL[0], state[1] - GOAL[1])
93.         return dist_start + dist_goal <= self.c_i
94.
```

```
95.     def co_va_cham(self, v, w):
96.         x1, y1 = v.state
97.         x2, y2 = w.state
98.         dist = Node._khoang_cach((x1, y1), (x2, y2))
99.         steps = min(int(dist / 5) + 5, 50)
100.
101.        for i in range(steps):
102.            u = i / (steps - 1)
103.            pos = (u * x2 + (1 - u) * x1, u * y2 + (1 - u) * y1)
104.            if not self.trong_vat(pos):
105.                return True
106.        return False
107.
108.    def khoang_cach(self, v, w):
109.        return Node._khoang_cach(v.state, w.state)
110.
111.    def chi_phi_canh(self, v, w):
112.        return INF if self.co_va_cham(v, w) else self.khoang_cach(v, w)
113.
114.    def cap_nhat_r(self):
115.        n = max(1, len(self.V) + len(self.X_unconn))
116.        return (
117.            2
118.            * 1.1
119.            * ((1 + 1 / 2) * (WIDTH * HEIGHT / math.pi) ** 0.5)
120.            * ((math.log(n) / n) ** 0.5)
121.        )
122.
123.    def gia_tri_tot_nhat(self, queue):
124.        if not queue:
125.            return INF
126.        if isinstance(queue[0], tuple):
127.            return queue[0][0]
128.        else:
129.            return queue[0].cost()
```

```
131.     def tao_vat(self):
132.         for _ in range(SO_VAT):
133.             while True:
134.                 pos = (
135.                     random.randint(0, WIDTH - SIZE_VAT[0]),
136.                     random.randint(0, HEIGHT - SIZE_VAT[1]),
137.                 )
138.                 obs = pygame.Rect(pos, SIZE_VAT)
139.                 if not (obs.collidepoint(START) or obs.collidepoint(GOAL)):
140.                     self.vat_vat.append(obs)
141.                     break
142. 
143.     def cat_tia(self):
144.         # Cắt đỉnh và mẫu nằm ngoài elip
145.         self.X_unconn = [x for x in self.X_unconn if x.f < self.c_i]
146.         self.V = [v for v in self.V if v.f <= self.c_i]
147. 
148.         # Lưu cạnh cũ và khởi tạo lại danh sách cạnh
149.         old_E = self.E
150.         self.E = []
151.         self.edge_dict.clear()
152. 
153.         # Xử lý các cạnh cũ
154.         for v, w in old_E:
155.             if v.f <= self.c_i and w.f <= self.c_i:
156.                 self.E.append((v, w))
157.                 self.edge_dict[v].append(w)
158.             else:
159.                 w.g_T = INF
160.                 w.parent = w
161. 
162.         to_move = [v for v in self.V if v.g_T == INF]
163.         self.X_unconn.extend(to_move)
164.         self.V = [v for v in self.V if v.g_T < INF]
165.         self.update_display(wait=True)
166. 
167.     def them_canh(self, v, w):
168.         if v.g_T < INF:
169.             w.g_T = v.g_T + self.khoang_cach(v, w)
170.             w.parent = v
171.             self.E.append((v, w))
172.             self.edge_dict[v].append(w)
```

```
174.     def hang_doi_canh(self, v, x):
175.         if v.g_T < INF:
176.             cost_val = v.g_T + self.khoang_cach(v, x) + x.h
177.             heapq.heappush(self.Q_E, (cost_val, (v, x)))
178.
179.     def hang_doi_dinh(self, v):
180.         heapq.heappush(self.Q_V, (v.cost(), v))
181.
182.     def mo_rong_dinh(self):
183.         _, v = heapq.heappop(self.Q_V)
184.         if v.g_T == INF:
185.             return
186.
187.         kc_limit = self.r
188.         g_T_v = v.g_T
189.         g_T_dest = self.nut_end.g_T
190.
191.         # Xử lý đỉnh chưa kết nối
192.         for x in self.X_unconn:
193.             kc = self.khoang_cach(v, x)
194.             if (
195.                 kc <= kc_limit
196.                 and g_T_v + kc + x.h < g_T_dest
197.                 and not self.co_va_cham(v, x)
198.             ):
199.                 self.hang_doi_canh(v, x)
200.
201.         # Xử lý đỉnh đã có trong V
202.         if v not in self.V_old:
203.             for w in self.V:
204.                 if v != w and w not in self.edge_dict.get(v, []):
205.                     kc = self.khoang_cach(v, w)
206.                     if (
207.                         kc <= kc_limit
208.                         and g_T_v + kc + w.h < g_T_dest
209.                         and g_T_v + kc < w.g_T
210.                         and not self.co_va_cham(v, w)
211.                     ):
212.                         self.hang_doi_canh(v, w)
213.
```

```
214.     def duong_den_dich(self):
215.         if self.nut_end.g_T == INF:
216.             return []
217.         path, v = [], self.nut_end
218.         while v != self.nut_start:
219.             path.append(v)
220.             v = v.parent
221.         path.append(self.nut_start)
222.         return path
223.
224.     def xu_ly_lo(self):
225.         # Khởi tạo
226.         if self.so_lan_lap == 0:
227.             self.V, self.X_unconn = [self.nut_start], [self.nut_end]
228.         else:
229.             self.cat_tia()
230.
231.         self.Q_E, self.Q_V = [], []
232.         j = 0
233.
234.         while True:
235.             # Kiểm tra điều kiện dừng
236.             if not self.Q_E and not self.Q_V:
237.                 j += 1
238.                 if (
239.                     j > 5
240.                     or self.so_lan_lap >= MAX_SO_LAN_LAP
241.                     or self.nut_end.g_T < self.c_i
242.                 ):
243.                     if self.nut_end.g_T < self.c_i:
244.                         self.so_lan_lap += 1
245.                     else:
246.                         self.running = False
247.                     break
248.
249.                 # Lấy mẫu mới
250.                 new_samples = self.lay_mau()
251.                 if not new_samples:
252.                     print("Không tạo được mẫu mới.")
253.                     self.running = False
254.                     break
255.
```

```
                self.X_unconn.extend(new_samples)
                self.cat_tia()
                self.V_old = self.V.copy()
                for v in self.V:
                    self.hang_doi_dinh(v)
                self.r = self.cap_nhat_r()
                continue

            # Mở rộng đỉnh tốt nhất
            best_edge = self.gia_tri_tot_nhat(self.Q_E)
            while self.Q_V and self.gia_tri_tot_nhat(self.Q_V) <= best_edge:
                self.mo_rong_dinh()
                best_edge = self.gia_tri_tot_nhat(self.Q_E)

            if not self.Q_E:
                continue

            # Xử lý cạnh tốt nhất
            _, (v_m, x_m) = heapq.heappop(self.Q_E)
            g_T_vm = v_m.g_T
            kc = self.khoang_cach(v_m, x_m)

            if g_T_vm + kc + x_m.h >= self.nut_end.g_T or g_T_vm + kc >= x_m.g_T:
                continue

            if not self.co_va_cham(v_m, x_m) and g_T_vm + kc + x_m.h < self.nut_end.g_T:
                # Cập nhật đồ thị
                if x_m in self.V:
                    self.E = [(v, w) for v, w in self.E if w != x_m]
                    for v in list(self.edge_dict.keys()):
                        if x_m in self.edge_dict[v]:
                            self.edge_dict[v].remove(x_m)
                    x_m.parent, x_m.g_T = x_m, INF
                else:
                    self.X_unconn.remove(x_m)
                    self.V.append(x_m)
                    self.hang_doi_dinh(x_m)
                self.them_canh(v_m, x_m)

                x_m_g_T = x_m.g_T
                self.Q_E = [
                    (c, (v, x))
                    for c, (v, x) in self.Q_E
                    if x != x_m or v.g_T + self.khoang_cach(v, x) < x_m_g_T
                ]
                heapq.heapify(self.Q_E)
```

```
303.                 if not self.Q_E or not self.Q_V:
304.                     self.update_display(wait=True)
305.             else:
306.                 self.Q_E, self.Q_V = [], []
307.
308.         # Cập nhật đường đi
309.         if self.c_i < INF:
310.             self.path_old = self.path_new.copy()
311.         new_path = self.duong_den_dich()
312.         if new_path:
313.             self.path_new, self.c_i = new_path, self.nut_end.g_T
314.         self.update_display(wait=True)
315.
316.     def update_display(self, wait=False):
317.         self.screen.fill(WHITE)
318.
319.         # Vẽ các vật và điểm đầu cuối
320.         pygame.draw.circle(self.screen, RED, START, R_NUT + 5)
321.         pygame.draw.circle(self.screen, RED, GOAL, R_NUT + 5)
322.         for obs in self.vat_vat:
323.             pygame.draw.rect(self.screen, BLACK, obs)
324.
325.         # Vẽ elip
326.         if self.c_i != INF:
327.             sx, sy, gx, gy = START[0], START[1], GOAL[0], GOAL[1]
328.             cx, cy = (sx + gx) / 2, (sy + gy) / 2
329.             dx, dy = gx - sx, gy - sy
330.             dist = math.hypot(dx, dy)
331.             if dist > 0:
332.                 angle = math.atan2(dy, dx)
333.                 a = self.c_i / 2
334.                 b = math.sqrt(max(0, a**2 - (dist / 2) ** 2))
335.
336.                 surf = pygame.Surface((2 * a, 2 * b), pygame.SRCALPHA)
337.                 pygame.draw.ellipse(surf, BLACK, surf.get_rect(center=(a, b)), width=2)
338.                 rot_surf = pygame.transform.rotate(surf, -math.degrees(angle))
339.                 self.screen.blit(rot_surf, rot_surf.get_rect(center=(cx, cy)))
340.
341.         # Vẽ đồ thị
342.         for v in self.V:
343.             if v.g_T < INF:
344.                 pygame.draw.circle(self.screen, BLUE, v.state, R_NUT)
```

```
345.
346.         for v, w in self.E:
347.             if v.g_T < INF and w.g_T < INF:
348.                 pygame.draw.line(self.screen, BLUE, v.state, w.state, DO_RONG_CANH)
349.
350.         for x in self.X_unconn:
351.             pygame.draw.circle(self.screen, GREEN, x.state, R_NUT)
352.
353.         # Vẽ đường đi tốt nhất
354.         if self.path_new:
355.             prev = self.path_new[0]
356.             for v in self.path_new[1:]:
357.                 pygame.draw.line(
358.                     self.screen, RED, prev.state, v.state, DO_RONG_CANH + 2
359.                 )
360.                 pygame.draw.circle(self.screen, RED, v.state, R_NUT + 2)
361.                 prev = v
362.
363.         # Hiển thị chi phí
364.         cost_text = (
365.             f"Best cost: {self.c_i:.2f}" if self.c_i != INF else "Best cost: INF"
366.         )
367.         self.screen.blit(self.font.render(cost_text, True, BLACK), (10, 10))
368.
369.         pygame.display.update()
370.
371.         # Xử lý sự kiện
372.         if wait and not self.finished:
373.             for event in pygame.event.get():
374.                 if event.type == pygame.QUIT:
375.                     pygame.quit()
376.                     exit()
377.             self.clock.tick(FPS)
378.             time.sleep(0.1)
379.
```

```
380.     def run(self):
381.         # Khởi tạo và chạy thuật toán
382.         self.tao_vat()
383.         self.update_display()
384.
385.         while True:
386.             # Xử lý sự kiện
387.             for event in pygame.event.get():
388.                 if event.type == pygame.QUIT:
389.                     pygame.quit()
390.                     return
391.
392.             self.update_display()
393.
394.             # Cập nhật thuật toán
395.             if self.running and not self.finished:
396.                 prev_cost = self.c_i
397.                 self.xu_ly_lo()
398.
399.                 # Hiển thị kết quả
400.                 if self.path_new and self.c_i != prev_cost:
401.                     print(f"Iteration {self.so_lan_lap}: Best cost = {self.c_i:.2f}")
402.
403.                 if not self.running:
404.                     self.finished = True
405.                     print(
406.                         f"Final path cost: {self.c_i:.2f}"
407.                         if self.path_new
408.                         else "No path found to goal"
409.                     )
410.
411.             self.clock.tick(FPS)
412.             time.sleep(0.1)
413.
414.     def lay_mau(self):
415.         samples = []
416.         attempts = 0
417.         max_attempts = 1000
418.         while len(samples) < SIZE_LO and attempts < max_attempts:
419.             attempts += 1
420.             state = (random.randint(0, WIDTH), random.randint(0, HEIGHT))
421.             if self.trong_vat(state) and self.trong_elip(state):
422.                 samples.append(Node(state))
423.         return samples
```

```
426.  if __name__ == "__main__":
427.      BITStar().run()
```

Demo video:

https://www.youtube.com/watch?v=sq4HM6zaMdQ

Mã giả:

Algorithm 1 Thuật toán mở rộng đồ thị tìm đường đi tối ưu

Khởi tạo tập đỉnh $V \leftarrow \{X_{\text{start}}\}$; $E \leftarrow \emptyset$; $T \leftarrow (V, E)$
$X_{\text{unconn}} \leftarrow X_{\text{goal}}$ ▷ Tạo mẫu rỗng
$QV \leftarrow V$; $QE \leftarrow \emptyset$
$V_{\text{sol'n}} \leftarrow V \cap \{X_{\text{goal}}\}$; $V_{\text{unexpnd}} \leftarrow V$
$X_{\text{new}} \leftarrow X_{\text{unconn}}$
$V_{\text{open}} \leftarrow V \cup \{X_{\text{goal}}\}$
$V_{\text{expanded}} \leftarrow V$
$c_i \leftarrow \min_{v_{\text{goal}} \in V_{\text{sol'n}}} \{g_T(v_{\text{goal}})\}$
while không dừng **do**
    if $QE = \emptyset$ **và** $QV = \emptyset$ **then**
        $X_{\text{reuse}} \leftarrow \text{Prune}(T, X_{\text{unconn}}, c_i)$
        $X_{\text{sampling}} \leftarrow \text{Sample}(m, X_{\text{start}}, X_{\text{goal}}, c_i)$
        $X_{\text{new}} \leftarrow X_{\text{reuse}} \cup X_{\text{sampling}}$
        $X_{\text{unconn}} \leftarrow X_{\text{unconn}} \cup X_{\text{new}}$
        $QV \leftarrow V$
    end if
    while BestQueueValue(QV) $\leq$ BestQueueValue(QE) **do**
        ExpandNextVertex(QV, QE, c_i)
    end while
    $(v_{\min}, x_{\min}) \leftarrow \text{popBestInQueue}(QE)$
    if không va chạm $(v_{\min}, x_{\min})$ **và** $g_T(v_{\min}) + c(v_{\min}, x_{\min}) + \hat{h}(x_{\min}) < c_i$ **then**
        $c_{\text{edge}} \leftarrow c(v_{\min}, x_{\min})$
        if $g_T(v_{\min}) + c_{\text{edge}} + \hat{h}(x_{\min}) < c_i$ **then**
            if $g_T(v_{\min}) + c_{\text{edge}} < g_T(x_{\min})$ **then**
                if $x_{\min} \in V$ **then**
                    Cập nhật cạnh cha mới, xóa cạnh cũ
                    $E \leftarrow \text{parent}(x_{\min})$
                else
                    $X_{\text{unconn}} \leftarrow X_{\text{unconn}} \setminus \{x_{\min}\}$
                    $V \leftarrow V \cup \{x_{\min}\}$
                    $QV \leftarrow QV \cup \{x_{\min}\}$
                    $V_{\text{unexpnd}} \leftarrow V_{\text{unexpnd}} \cup \{x_{\min}\}$
                end if
                if $x_{\min} \in X_{\text{goal}}$ **then**
                    $V_{\text{sol'n}} \leftarrow V_{\text{sol'n}} \cup \{x_{\min}\}$
                end if
                $E \leftarrow E \cup \{(v_{\min}, x_{\min})\}$
                $c_i \leftarrow \min_{v_{\text{goal}} \in V_{\text{sol'n}}} \{g_T(v_{\text{goal}})\}$
            else
                $QE \leftarrow \emptyset$; $QV \leftarrow \emptyset$
            end if
        end if
    end if
end while
return T

Chương 5

So sánh và đánh giá

So sánh

Dựa trên kết quả thực nghiệm, có thể rút ra các so sánh chính sau:

BIT* vs RRT và các biến thể (Informed RRT*, RRT#, SORRT*)

- **Tốc độ hội tụ và chất lượng giải pháp:** BIT* thường hội tụ nhanh hơn và đạt được giải pháp có chi phí thấp hơn trong cùng một khoảng thời gian, đặc biệt rõ rệt ở số chiều cao. Điều này là do BIT* sử dụng heuristic để sắp xếp thứ tự tìm kiếm một cách hiệu quả (giống A*) và xử lý mẫu theo lô, thay vì tìm kiếm ngẫu nhiên như RRT*. Informed RRT* và SORRT* cố gắng cải thiện RRT* bằng cách tập trung lấy mẫu hoặc sắp xếp mẫu, nhưng BIT* với việc xem xét nhiều kết nối trên đồ thị ẩn RGG tỏ ra hiệu quả hơn trong các bài toán phức tạp.

- **Thời gian giải pháp ban đầu:** BIT* có thể mất nhiều thời gian hơn một chút để tìm giải pháp ban đầu so với RRT* trong một số trường hợp đơn giản (do chi phí quản lý lô và hàng đợi), nhưng thường nhanh hơn trong các bài toán phức tạp hoặc số chiều cao nhờ tìm kiếm có hướng.

- **Khả năng mở rộng:** BIT* thể hiện khả năng mở rộng tốt hơn RRT* khi số chiều tăng lên.

BIT* vs FMT*

- **Tính Anytime:** BIT* là thuật toán anytime, liên tục cải thiện giải pháp theo thời gian, trong khi FMT* thì không (chỉ trả về giải pháp khi hoàn thành tìm kiếm trên số mẫu cố định).
- **Hiệu quả tìm kiếm:** Cả hai đều sử dụng tìm kiếm có thứ tự trên tập mẫu. FMT* tìm kiếm theo chi phí đến (cost-to-come), còn BIT* tìm kiếm theo chi phí giải pháp ước tính (giống A*). BIT* có thể hiệu quả hơn nhờ heuristic và khả năng tái sử dụng thông tin giữa các lô.
- **Chất lượng giải pháp:** Với cùng số lượng mẫu hoặc thời gian tính toán tương đương, BIT* thường cho kết quả cuối cùng tốt hơn hoặc tương đương FMT* nhờ khả năng cải thiện liên tục và tập trung vào vùng informed set.

BIT* vs RRT-Connect

- **Tốc độ vs. Tối ưu:** RRT-Connect thường là thuật toán nhanh nhất tìm ra một giải pháp khả thi ban đầu do tìm kiếm hai chiều. Tuy nhiên, giải pháp này thường xa tối ưu và không được cải thiện. BIT* tuy có thể chậm hơn một chút ban đầu nhưng đảm bảo hội tụ về giải pháp tối ưu và liên tục cải thiện chất lượng. Trong nhiều trường hợp phức tạp (ví dụ: HERB 14-DOF), BIT* thậm chí còn nhanh hơn RRT-Connect để tìm giải pháp ban đầu.

Ảnh hưởng của kích thước lô trong BIT*

- Kích thước lô nhỏ (ví dụ: $m = 1$) làm BIT* hoạt động giống Informed RRT*.
- Kích thước lô lớn làm BIT* hoạt động giống FMT* hơn (nếu chỉ chạy 1 lô).
- Kích thước lô trung bình (ví dụ: $m = 100$) dường như mang lại sự cân bằng tốt giữa tốc độ tìm giải pháp ban đầu và tốc độ hội tụ (Hình 16 trong [**?**]). Việc giảm kích thước lô có thể giảm thời gian tìm giải pháp ban đầu đến

một ngưỡng nhất định nhưng cũng có thể làm chậm tốc độ hội tụ, đặc biệt ở số chiều cao.

Đánh giá

BIT* nổi lên như một thuật toán lập kế hoạch đường đi mạnh mẽ, kết hợp thành công các ưu điểm của phương pháp dựa trên tìm kiếm đồ thị (có thứ tự, hiệu quả) và phương pháp dựa trên lấy mẫu (anytime, tối ưu tiệm cận, khả năng mở rộng).

Ưu điểm của BIT*

- **Tối ưu tiệm cận và hoàn chỉnh xác suất:** Đảm bảo về mặt lý thuyết sẽ hội tụ đến giải pháp tối ưu nếu tồn tại.
- **Hiệu suất Anytime:** Nhanh chóng tìm ra giải pháp khả thi và liên tục cải thiện chất lượng theo thời gian, phù hợp cho các ứng dụng có giới hạn thời gian.
- **Tìm kiếm có thông tin và hiệu quả:** Việc sử dụng heuristic và tập trung lấy mẫu vào vùng informed set giúp BIT* tìm kiếm hiệu quả hơn nhiều so với tìm kiếm ngẫu nhiên của RRT*, đặc biệt trong không gian trạng thái phức tạp và số chiều cao. Nó tránh lãng phí tài nguyên vào các vùng không gian không hứa hẹn.
- **Xử lý theo lô và tái sử dụng thông tin:** Cho phép tìm kiếm có thứ tự trên không gian liên tục và tái sử dụng thông tin từ các lô trước, tăng tốc độ hội tụ.
- **Khả năng mở rộng tốt:** Các thí nghiệm cho thấy BIT* hoạt động tốt hơn các thuật toán khác khi số chiều của không gian trạng thái tăng lên.
- **Ít nhạy cảm với tham số:** So với RRT* (phụ thuộc nhiều vào độ dài cạnh tối đa), BIT* (với kích thước lô $m = 100$) tỏ ra ổn định trên nhiều loại bài toán [**?**].

Nhược điểm của BIT*

- **Độ phức tạp:** BIT* phức tạp hơn trong việc triển khai so với RRT* do quản lý hai hàng đợi, xử lý lô và các cơ chế tỉa cây, đánh dấu trạng thái.
- **Ảnh hưởng của Heuristic:** Hiệu suất của BIT* phụ thuộc vào chất lượng của hàm heuristic. Một heuristic tốt sẽ giúp hội tụ nhanh hơn, nhưng một heuristic kém có thể không mang lại nhiều lợi ích so với tìm kiếm không có thông tin. Tuy nhiên, ngay cả với heuristic zero (tương đương Dijkstra), BIT* vẫn hoạt động.
- **Chi phí quản lý lô:** Việc xử lý mẫu theo lô và cập nhật hàng đợi có thể tạo ra chi phí tính toán nhất định, có thể làm chậm thời gian tìm giải pháp ban đầu trong các bài toán rất đơn giản so với RRT*.
- **Kết nối cạnh:** BIT* xem xét nhiều kết nối tiềm năng cho mỗi mẫu trong RGG ẩn. Điều này giúp tìm ra đường đi tốt hơn nhưng cũng có thể tốn chi phí nếu nhiều kết nối bị chặn bởi chướng ngại vật. Các biến thể như SORRT* hoặc cơ chế loại bỏ mẫu (sample removal) có thể giảm bớt vấn đề này nhưng có thể ảnh hưởng đến việc khai thác các mẫu hữu ích.

Chương 6

Ứng dụng

Ứng dụng của BIT*

- **Tìm đường đi tối ưu trong đồ thị có trọng số:** BIT* có thể áp dụng để tìm đường ngắn nhất có ràng buộc, tương tự như Dijkstra hoặc A*, nhưng hiệu quả hơn khi không gian trạng thái lớn nhờ kết hợp heuristic và sampling.
- **Tối ưu bài toán rời rạc có không gian trạng thái lớn:** BIT* sử dụng heuristic và sampling có định hướng để tìm nghiệm gần tối ưu cho các bài toán như lập lịch, phân công tài nguyên, routing trong mạng.
- **Mô hình hóa bài toán điều khiển rời rạc:** BIT* giúp tìm đường đi cho robot hoặc hệ thống trong không gian lưới, tránh chướng ngại vật và tối ưu chi phí di chuyển.
- **Giảng dạy kỹ thuật heuristic và cấu trúc dữ liệu:** BIT* là ví dụ minh họa rõ ràng cho việc kết hợp hàng đợi ưu tiên, heuristic, và cập nhật động trong các bài toán tổ hợp.

Kết luận

BIT* là một bước tiến quan trọng trong lĩnh vực lập kế hoạch đường đi dựa trên lấy mẫu. Bằng cách tích hợp tìm kiếm có hướng heuristic vào một quy trình

xử lý mẫu theo lô và tính chất *anytime*, BIT* cung cấp sự cân bằng tuyệt vời giữa tốc độ hội tụ, chất lượng giải pháp và khả năng mở rộng.

Thuật toán này đặc biệt phù hợp cho các bài toán lập kế hoạch phức tạp, có số chiều cao trong robotics, nơi việc tìm kiếm đường đi tối ưu hoặc gần tối ưu một cách nhanh chóng là rất quan trọng. Các kết quả thực nghiệm trên cả môi trường mô phỏng và hệ thống robot thực tế đã chứng minh tính ưu việt của BIT* so với nhiều thuật toán tiên tiến khác.

Tài liệu tham khảo

Bài báo gốc về BIT*

- **Tên bài:** *"Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs"*
- **Tác giả:** Jonathan D. Gammell, Siddhartha S. Srinivasa, Timothy D. Barfoot
- **Năm:** 2015
- **Link:** https://arxiv.org/abs/1405.5848

Đây là tài liệu **quan trọng nhất**, trình bày đầy đủ ý tưởng, phân tích và so sánh BIT* với các thuật toán khác.

Sách tham khảo

[label=•]

- ***Planning Algorithms*** – **Steven M. LaValle**
  - Một cuốn sách **kinh điển**, bao quát nhiều chủ đề về lập kế hoạch chuyển động: motion planning, discrete planning, planning under uncertainty,...
  - **Link tải miễn phí** (tác giả cung cấp): http://lavalle.pl/planning/

Khóa học và bài giảng online

[label=•]

- ***CS287: Advanced Robotics – UC Berkeley***
  - Giảng viên: Pieter Abbeel
  - Có bài giảng video về sampling-based planning và trajectory optimization.
  - YouTube playlist: https://www.youtube.com/playlist?list=PLkFD6_40KJIw