

# Các Kỹ thuật Phân hoạch Lỗi cho Hoạch định Chuyển động Robot

Nhóm 2 thắng Khoa

Ngày 17 tháng 10 năm 2025

# Mục tiêu Buổi trình bày

## Các câu hỏi cần làm rõ

- Phân hoạch lỗi là gì? Tại sao nó quan trọng trong hoạch định chuyển động?
- Độ đo xấp xỉ của một vùng lỗi được đo lường ra sao?
- Trong bài báo, Thuật toán IRIS được sử dụng cụ thể ở đâu, trong trường hợp nào? Có những thuật toán nào khác được dùng cho từng trường hợp không?
- Các phương pháp được áp dụng trong môi trường 2D như thế nào?

## Mục tiêu chính

Buổi trình bày hôm nay sẽ tập trung giải đáp các câu hỏi trên, cung cấp cái nhìn tổng quan về các kỹ thuật phân hoạch lỗi và ứng dụng của chúng.

# Nội dung

- 1 Các Nguyên tắc Nền tảng
- 2 Phân hoạch Lỗi Xấp xỉ (ACD)
- 3 IRIS Algorithm
- 4 Visibility Clique Cover (VCC)
- 5 Ứng dụng trong GCS
- 6 Demo

# Định nghĩa Phân hoạch Lồi

## Định nghĩa

**Phân hoạch lồi (Convex Decomposition)** là quá trình chia một hình học phức tạp (không lồi) thành một tập hợp các **vùng con lồi** sao cho hợp của chúng khôi phục lại hình ban đầu và các vùng con không chồng lấn nhau (ngoại trừ biên chung).

## Công thức

Với một miền hình học  $P \subset \mathbb{R}^n$ , một *phân hoạch lồi* của  $P$  là tập hợp  $\{P_1, P_2, \dots, P_k\}$  sao cho:

$$P = \bigcup_{i=1}^k P_i, \quad P_i \cap P_j = \partial P_i \cap \partial P_j \text{ với mọi } i \neq j,$$

và mỗi  $P_i$  là một tập lồi.  $\partial P_i$  là biên của vùng  $P_i$ .

# Mục tiêu và Điều kiện Tồn tại

## Mục tiêu

- Tối thiểu hóa số lượng vùng lỗi
- Giảm thiểu chồng chéo
- Tránh vùng quá “mảnh” hoặc “dài”

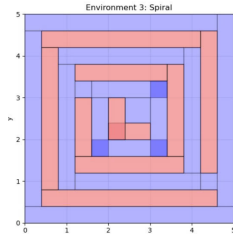
## Kết quả lý thuyết

Bất kỳ đa giác đơn nào đều có thể được phân hoạch thành tập hợp các vùng lỗi.

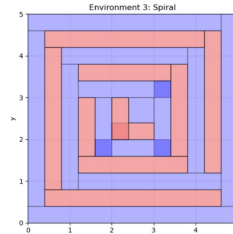
**Lý do:** Mọi đa giác đơn có thể tam giác hóa, mỗi tam giác là tập lỗi.

**Hạn chế:** Tạo ra số lượng lớn các thành phần, không thực tế cho ứng dụng.

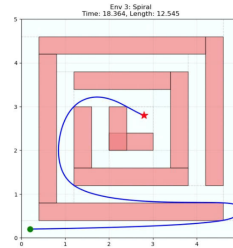
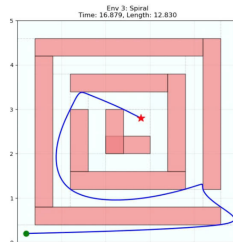
# Ví dụ về Phân hoạch Lỗi Trong 2D



(a) Case 1



(b) Case 2



## Exact Convex Decomposition (ECD)

- Thành phần hoàn toàn lồi
- Tái tạo chính xác
- NP-hard
- Số lượng vùng lớn
- Chủ yếu có trong lý thuyết

## Approximate Convex Decomposition (ACD)

- Gần lồi với dung sai  $\tau$
- Hiệu quả tính toán cao
- Giảm số lượng vùng
- Kiểm soát mức chi tiết
- Ứng dụng thực tế

## Chiến lược chia để trị đệ quy

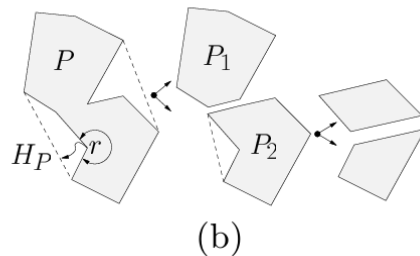
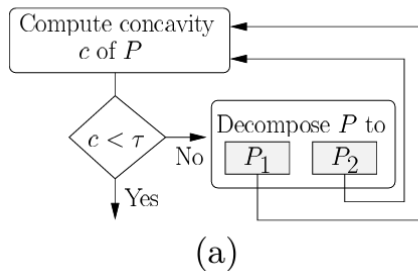
- ❶ **Đo độ lõm:** Xác định đỉnh lõm có độ lõm lớn nhất
- ❷ **Kiểm tra dung sai:** Nếu độ lõm  $< \tau$ , kết thúc
- ❸ **Giải quyết đỉnh lõm:** Thêm đường cắt, chia đa giác
- ❹ **Đệ quy:** Áp dụng cho các thành phần mới

## Độ phức tạp

$O(nr^2)$ , với  $n$  là số đỉnh,  $r$  là số đỉnh lõm

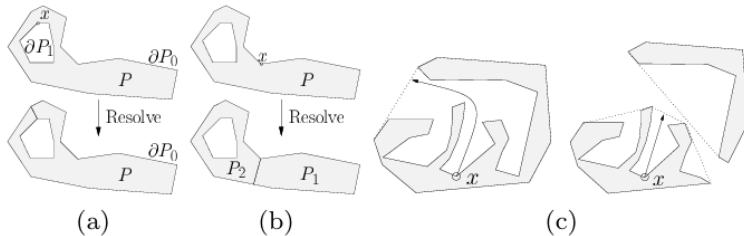


# Minh họa thuật toán



**Hình:** Quá trình đệ quy tiếp tục cho đến khi đạt đến giới hạn độ lõm đầu vào của người dùng  $\tau$ .

# Quá trình Phân rã Đệ quy



## Quy trình

- (a) Gộp biên vào đa giác
- (b) Tách đa giác thành hai
- (c) Độ lõm thay đổi sau phân rã

Quá trình tiếp tục đến khi  $\leq \tau$

## Định nghĩa các thành phần

- **Vỏ Lồi ( $H_P$ ):** Tập lồi nhỏ nhất chứa  $P$

$$P \text{ lồi} \Leftrightarrow P = H_P$$

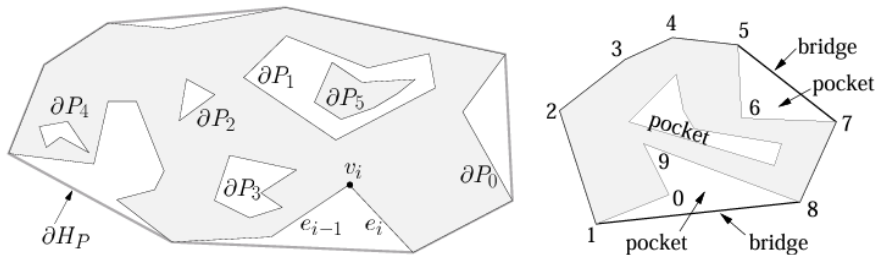
- **Khía (Notches):** Đỉnh không trên  $H_P$ , góc trong  $> 180^\circ$
- **Cầu (Bridges):** Cạnh của  $\partial H_P$  nối hai đỉnh không liền kề

$$\text{BRIDGES}(P) = \partial H_P \setminus \partial P$$

- **Túi (Pockets):** Chuỗi cạnh không thuộc  $\partial H_P$

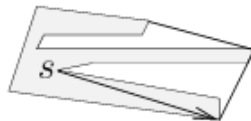
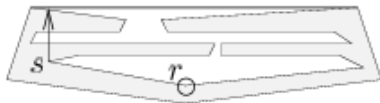
$$\text{POCKETS}(P) = \partial P \setminus \partial H_P$$

# Minh họa Các Khái niệm



Hình: Vỏ lồi, khía, cầu và túi trong đa giác không lồi

# Các Thước đo Độ lõm

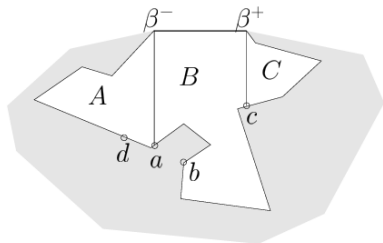


## SL-Concavity

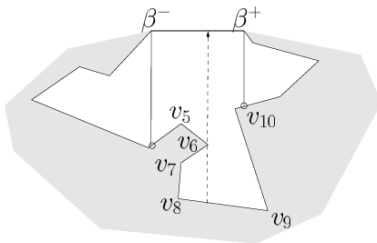
Khoảng cách Euclid đến cầu nổi (bridge).

- + Nhanh
- Không đảm bảo tính đơn điệu

# Các Thước đo Độ lõm



(a)



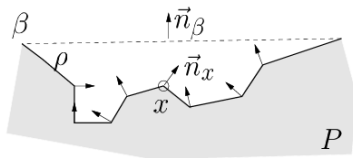
(b)

## SP-Concavity

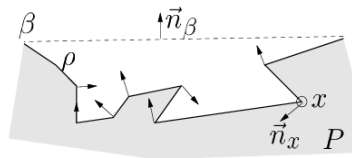
Đường đi ngắn nhất trong túi (pocket) đến cầu nối tương ứng.

- Chậm hơn
- + Đảm bảo tính đơn điệu
- + Xử lý tất cả đặc điểm lõm quan trọng

# Các Thước đo Độ lõm



(a)



(b)

## H-Concavity

Lai ghép SL và SP - Mặc định dùng SL, chuyển sang SP khi phát hiện không đơn điệu (kiểm tra:  $n_\beta \cdot n_i < 0$ ).

## Iterative Regional Inflation by Semidefinite Programming

**Câu hỏi cốt lõi:** Với điểm mầm (seed), vùng lồi lớn nhất là gì?

### Quy trình lặp 4 bước

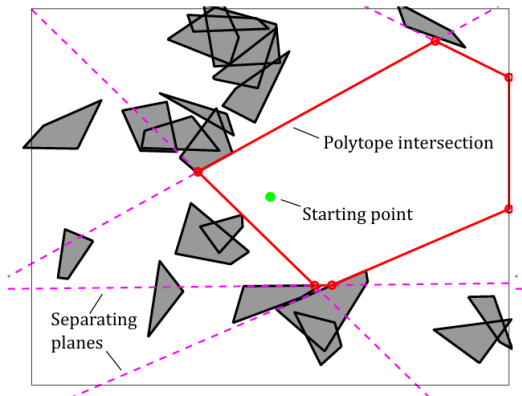
- 1 Khởi tạo hình cầu rất nhỏ tại seed
- 2 Tìm siêu phẳng phân cách (QP)
- 3 Tìm ellipsoid nội tiếp cực đại (SDP)
- 4 Lặp lại đến hội tụ

### Output

Một vùng lồi lớn nhất từ seed point



## IRIS Bước 2: Tìm Siêu phẳng



### Biến đổi không gian

Từ ellipsoid sang quả cầu đơn vị:

$$\tilde{x} = C^{-1}(x - d)$$

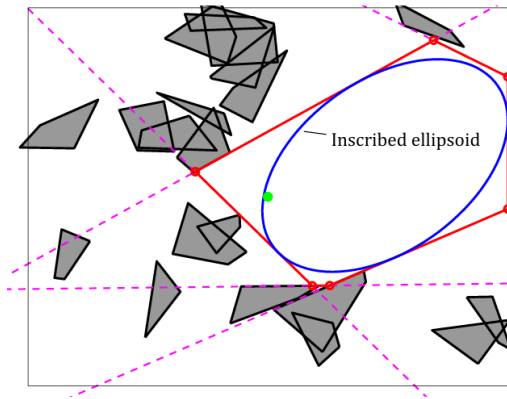
### QP Problem

$$\begin{aligned} \min_{\tilde{x}, w} \quad & \|\tilde{x}\|^2 \\ \text{s.t.} \quad & [\tilde{v}_{j,i}]w = \tilde{x} \\ & w_i \geq 0 \end{aligned}$$

Pháp tuyến:

$$a_j = 2C^{-1}C^{-\top}(x^* - d)$$

# IRIS Bước 3: Ellipsoid Nội tiếp



## Ellipsoid Definition

$$\mathcal{E} = \{C\tilde{x} + d : \|\tilde{x}\|_2 \leq 1\}$$

## SDP Problem

$$\begin{aligned} \max_{C,d} \quad & \log \det C \\ \text{s.t.} \quad & \|a_i^\top C\|_2 + a_i^\top d \leq b_i \\ & C \succ 0 \end{aligned}$$

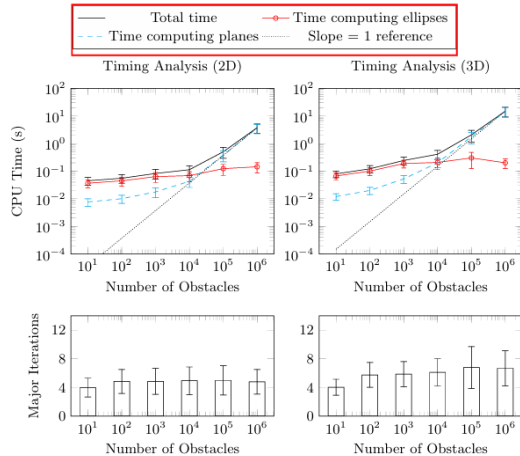
Tối đa hóa thể tích ellipsoid trong đa diện

## Thuộc tính chính

- **Loại:** Phương pháp xấp xỉ
- **Hội tụ:** 4-8 vòng lặp
- **Độ phức tạp:** Tuyến tính theo số obstacles

## Xử lý obstacles không lồi

- **Work space:** Bao lồi hoặc phân hoạch trước
- **Configuration space:** IRIS-NP, C-IRIS



Hình: Thời gian thực thi IRIS tăng tuyến tính với số lượng obstacles.

# VCC - Giải quyết Vấn đề Seeding

## Hạn chế của IRIS

Cần điểm mầm tốt - việc lấy mầm ngẫu nhiên không hiệu quả

## Giải pháp VCC

Tự động hóa quy trình seeding thông qua cấu trúc toàn cục của không gian tự do

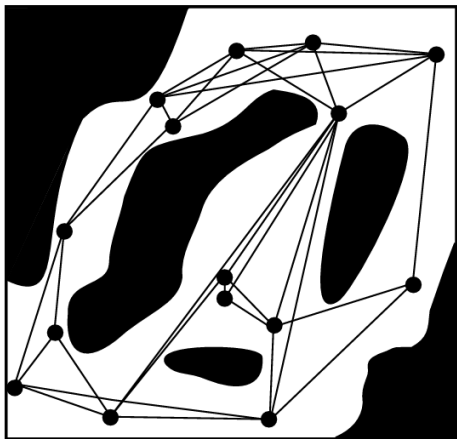
## Quy trình 4 bước

- 1 Lấy mẫu  $n$  điểm trong  $C_{free}$
- 2 Xây dựng đồ thị khả kiến (visibility graph)
- 3 Tìm lớp phủ clique (MAXCLIQUE + ILP)
- 4 Khởi tạo ellipsoid và lấp đầy (IRIS-like)

## Ý tưởng cốt lõi

Điểm “nhìn thấy” nhau  $\Rightarrow$  có khả năng cùng vùng lỗi

## Visibility Graph –



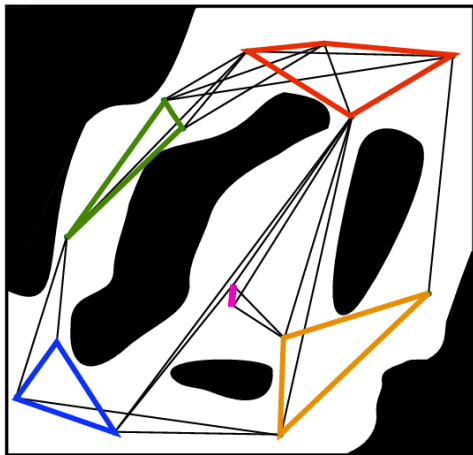
### Visibility Graph Construction

- Lấy mẫu  $n$  điểm ngẫu nhiên trong  $C_{free}$
- Tạo cạnh giữa hai điểm nếu đoạn thẳng nối chúng không va chạm
- Xây dựng đồ thị khả kiến (visibility graph)

### Định nghĩa

**Visibility graph:** Đồ thị có cạnh nối hai đỉnh khi đoạn thẳng nối chúng nằm hoàn toàn trong không gian tự do

## ► Clique Cover



### Clique Cover Problem

- Tìm tập hợp các clique lớn (đồ thị con đầy đủ)
- Sử dụng MAXCLIQUE algorithm
- Giải bài toán Integer Linear Programming (ILP)

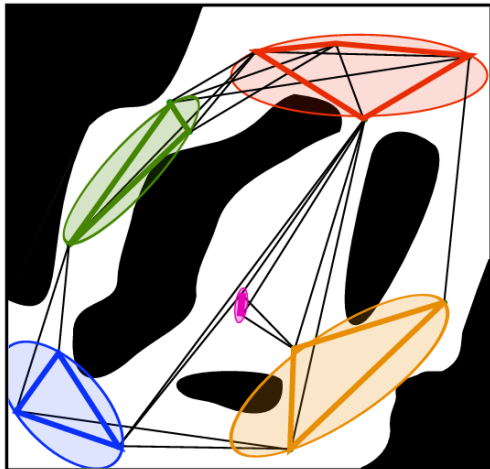
### Ý tưởng cốt lõi

Các điểm "nhìn thấy" nhau có khả năng nằm trong cùng một vùng lỗi

*Con đường thẳng = an toàn*



## Ellipsoids



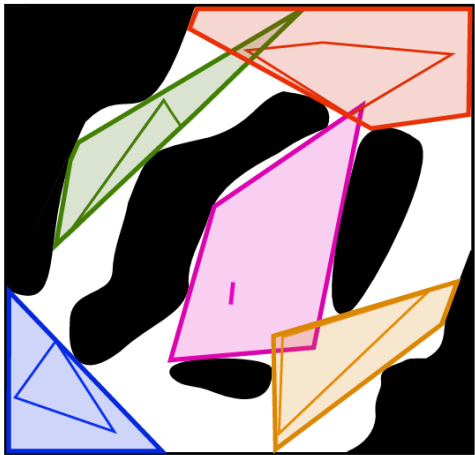
### Ellipsoid Initialization

- Tính ellipsoid có thể tích nhỏ nhất bao quanh tất cả điểm trong clique
- Cung cấp tâm (seed point) cho bước tiếp theo
- Xác định hình dạng ban đầu (các trục chính)

### Lợi ích

Ellipsoid đã được "thông báo" về hình học cục bộ  
⇒ Hiệu quả hơn IRIS tiêu chuẩn

### ► Inflate Polytopes -



#### Inflation Process

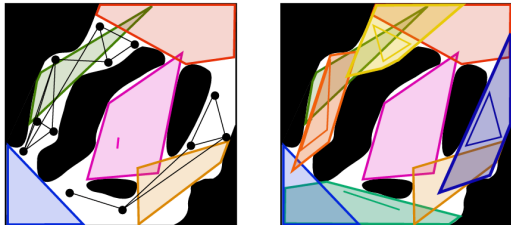
- Sử dụng thuật toán tương tự IRIS
- Một vòng lặp duy nhất (thay vì nhiều vòng)
- Tạo đa diện lồi cuối cùng từ ellipsoid

#### Hiệu quả cao hơn IRIS

Ellipsoid ban đầu đã được “thông báo” về hình học cục bộ

⇒ Loại bỏ nhiều vòng lặp tốn kém

## Repeated Iterations



## Iteration Process

- Lấy mẫu mới từ không gian trống còn lại
- Lặp lại các bước 1-4 cho vùng chưa phủ
- Tiếp tục đến khi đạt độ phủ  $\alpha$  đủ

## Điều kiện dừng

Quá trình kết thúc khi:

- Đạt độ phủ mong muốn  $\alpha$
- Không còn không gian đáng kể để phủ
- Số vùng lỗi đã đủ cho ứng dụng

# VCC - Đặc tính và Ưu điểm

## Đặc tính

- **Loại:** Phương pháp xấp xỉ
- **Mục tiêu:** Tối thiểu hóa số vùng với độ phủ  $\alpha$
- **Phương pháp:** Lai ghép hình học và ILP

## Ưu điểm

- Tự động hóa hoàn toàn
- Không phụ thuộc hình dạng obstacles
- Sử dụng kiểm tra va chạm chung

# Vai trò trong Graph of Convex Sets

## Khuôn khổ GCS

Yêu cầu phân hoạch lỗi trước khi hoạch định chuyển động

## Vai trò phân hoạch

- Các vùng lỗi = đỉnh trong đồ thị  $G$
- Chất lượng phân hoạch  $\Rightarrow$  hiệu suất GCS
- Phân hoạch kém  $\Rightarrow$  đồ thị phức tạp

## Chuỗi phụ thuộc

Thuật toán phân hoạch tiên tiến  $\Rightarrow$  GCS khả thi  $\Rightarrow$  Giải quyết bài toán DOF cao

# Phương pháp theo Môi trường & Độ phức tạp

## 2D đơn giản - Exact Decomposition

- **Đa giác:** 12 vùng V-polytope  $\rightarrow$  H-polytope
- **Mê cung:** Grid decomposition  $50 \times 50$

## 3D Quadrotor - Box Decomposition

- Axis-aligned box decomposition
- Thu nhỏ theo kích thước robot ( $0.2m$  radius)

## High-DOF (7-DOF, 14-DOF) - IRIS Approximate

- IrisInConfigurationSpace từ Drake
- Seed poses thủ công qua Inverse Kinematics
- Kết hợp qua tích Descartes (dual arms)

# Ví dụ 2D - Phân hoạch Chính xác

```
48 # vertices of the safe regions
49 vertices = [
50     np.array([
51         [ 0.4, 0.],
52         [ 0.4, 5.],
53         [ 0. , 5.],
54         [ 0. , 0.]
55     ]),
56     np.array([
57         [0.4, 2.4],
58         [1. , 2.4],
59         [1. , 2.6],
60         [0.4, 2.6]
61     ]),
62     np.array([
63         [1.4, 2.2],
64         [1.4, 4.6],
65         [1. , 4.6],
66         [1. , 2.2]
67     ]),
```

## Two-Dimensional Example

### 12 vùng lỗi V-polytope

Chuyển đổi sang H-polytope:

$$\mathcal{R}_i = \{x \in \mathbb{R}^2 : A_i x \leq b_i\}$$

Sử dụng ConvexHull algorithm

**Kết quả:** Phù hợp hoàn hảo với GCS framework

# Ví dụ Maze - Grid Decomposition

```
regions = []
edges = []
for x in range(maze_size):
    for y in range(maze_size):
        regions.append(HPolyhedron.MakeBox([x, y], [x+1., y+1.]))
        C = y + x * maze.ny
        if not maze.map[x][y].walls['N']:
            edges.append((C, C + 1))
        if not maze.map[x][y].walls['S']:
            edges.append((C, C - 1))
        if not maze.map[x][y].walls['E']:
            edges.append((C, C + maze.ny))
        if not maze.map[x][y].walls['W']:
            edges.append((C, C - maze.ny))
```

## Maze Planning

**Lưới đều**  $50 \times 50$

Mỗi ô = vùng lồi  $Q_i$

**Graph connectivity:**

- Dựa trên cấu trúc tường
- Hai ô kề không tường  $\Rightarrow$  có cạnh
- Đơn giản nhưng hiệu quả



# Ví dụ 3D Quadrotor - Chi tiết Box Decomposition

## Thuật toán chia nhỏ không gian

- **Phòng (Indoor):** Hộp duy nhất cho mỗi ô

$$\text{size} = 2.5 - (\text{wall\_offset} + \text{quad\_radius})$$

- **Ngoài trời (Outdoor):**

- *Không cây:* Hộp lớn bao phủ toàn bộ
- *Có cây:* 4 hộp xung quanh (trên, dưới, trái, phải)

- **Khe hở tường:**

- *Cửa:* Hộp hẹp cao
- *Cửa sổ:* 1-2 hộp nhỏ
- *Không tường:* Hộp lớn nối phòng

# High-DOF: KUKA 7-DOF & Dual Arms 14-DOF

## Giải pháp IRIS-based

- Sử dụng `IrisInConfigurationSpace` (Drake)
- Xử lý chướng ngại vật không lỗi trong C-space
- Sinh đa diện lỗi  $Q_i$  quanh seed poses
- Song song hóa cho nhiều seed khác nhau

## Dual Arms: Kết hợp không gian

Tích Descartes hoặc hợp lỗi của các vùng từ từng cánh tay riêng lẻ

## Vấn đề Seeding thủ công

- IRIS cần seed poses được chọn thủ công
- Sử dụng Inverse Kinematics để tạo seed
- Không tự động, phụ thuộc kinh nghiệm người dùng

## Ý nghĩa thực tiễn

**Sự thành công của IRIS trong C-space nhiều chiều làm cho GCS khả thi cho robot thể giới thực**

# Kết luận

## Các phương pháp chính

- **ACD** cân bằng giữa chính xác và hiệu quả
- **IRIS** cho phép xử lý không gian DOF cao
- **VCC** tự động hóa seeding cho IRIS

## Ứng dụng theo môi trường

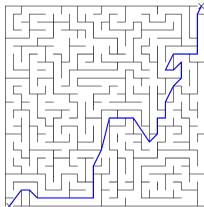
- 2D: Exact decomposition
- 3D: Box/Grid decomposition
- High-DOF: IRIS-based approximation

## Kết nối với GCS

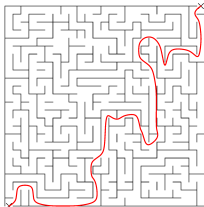
**GCS framework** phụ thuộc hoàn toàn vào chất lượng phân hoạch

# Demo 1

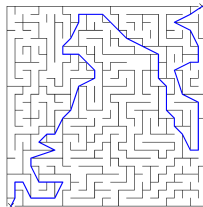
Seed 4 - Min Distance  
Cost: 54.71, Time: 1.08s



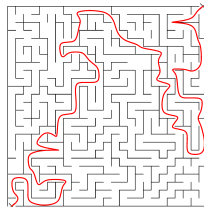
Seed 4 - Min Time  
Duration: 65.99s, Time: 82.50s



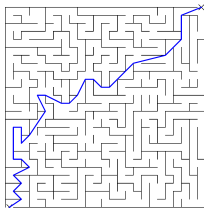
Seed 123 - Min Distance  
Cost: 97.51, Time: 1.45s



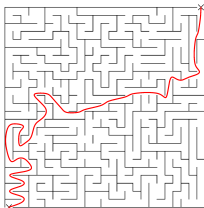
Seed 123 - Min Time  
Duration: 120.39s, Time: 69.10s



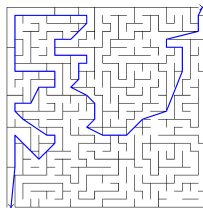
Seed 42 - Min Distance  
Cost: 53.14, Time: 1.60s



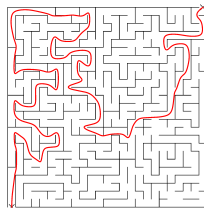
Seed 42 - Min Time  
Duration: 67.33s, Time: 92.52s



Seed 456 - Min Distance  
Cost: 107.51, Time: 1.03s



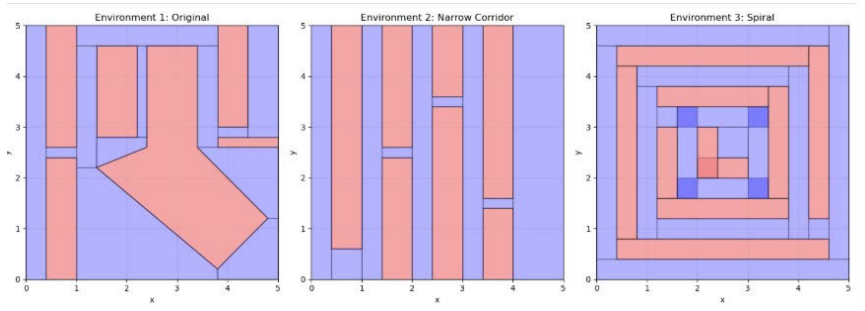
Seed 456 - Min Time  
Duration: 131.95s, Time: 88.40s



Hình:

Hình:

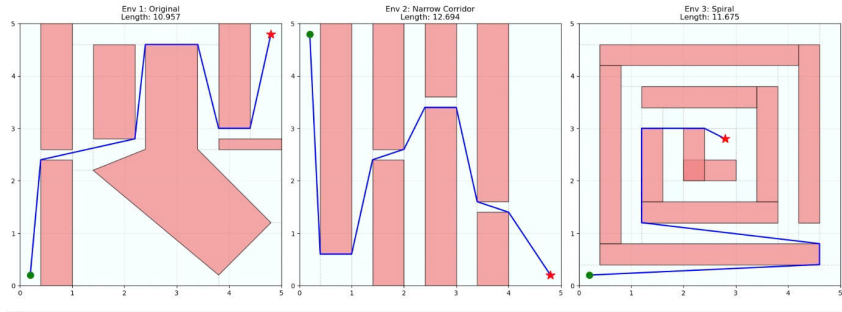
# Demo 2



Hình:

# Demo 2

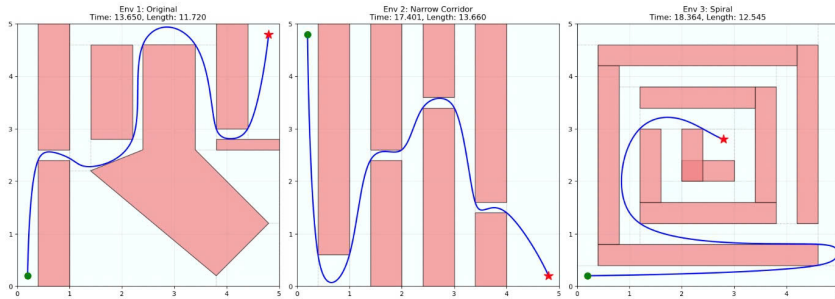
## Linear GCS (Minimum Distance) Results



Hình:

# Demo 2

## Bezier GCS (Order=6, Continuity=2) Results



Hình: