

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FALTY OF COMPUTER SCIENCE AND ENGINEERING



**CAPSTONE PROJECT
COMPUTER ENGINEERING**

Motion Planning around Obstacles

Semester 251

Instructor: Assoc. Prof. Le Hong Trang

STT	Full name	ID	Note
1	Nguyen Tran Dang Khoa	2211635	
2	Ho Dang Khoa	2211588	

Ho Chi Minh City, December 2025

Tóm tắt

Hoạch định chuyển động cho robot trong môi trường chứa vật cản vốn là một bài toán tối ưu hóa phi tuyến đầy thách thức do tính chất không lồi của không gian cấu hình. Trong khi các phương pháp dựa trên lấy mẫu truyền thống thường gặp hạn chế về tính tối ưu và khả năng xử lý các ràng buộc động lực học phức tạp, bài báo này đề xuất một phương pháp tiếp cận mới dựa trên Đồ thị các Tập Lồi (Graphs of Convex Sets - GCS). Bằng cách phân rã không gian tự do thành các vùng lồi hữu hạn và mô hình hóa quỹ đạo robot sử dụng đường cong Bézier, chúng tôi thiết lập bài toán dưới dạng tối ưu hóa hỗn hợp nguyên. Cuối cùng ta áp dụng kỹ thuật nới lỏng lồi (convex relaxation) chặt chẽ để chuyển đổi bài toán phức tạp này thành bài toán Lập trình nón bậc hai (Second-Order Cone Programming - SOCP) dễ giải. Kết quả thực nghiệm trên các hệ thống có số chiều cao chứng minh phương pháp này vượt trội hơn so với các thuật toán PRM truyền thống, giảm thiểu đáng kể thời gian tính toán trong khi vẫn đảm bảo tìm được quỹ đạo tối ưu toàn cục. Chúng tôi kiểm chứng GCS trên một loạt các môi trường với độ phức tạp tăng dần, từ các vật cản 2D đơn giản và mê cung phức tạp đến các thiết bị bay quadrotor động lực học và tay máy 7 bậc tự do (7-DoF) trong không gian nhiều chiều. Các thực nghiệm số chứng minh rằng GCS đạt hiệu năng vượt trội một cách nhất quán so với các bộ hoạch định dựa trên lấy mẫu được sử dụng rộng rãi (ví dụ: PRM). Cụ thể, trong các nhiệm vụ nhiều chiều, GCS giảm thời gian truy vấn trực tuyến lên đến một bậc so với PRM tiêu chuẩn, đồng thời tìm ra các quỹ đạo tối ưu toàn cục ngắn hơn đáng kể (giảm tối 40% độ dài) so với các bộ hoạch định dựa trên lấy mẫu đã được xử lý hậu kỳ [1].

Contents

1	Introduction	1
1.1	Motivation and Technical Challenges	1
1.2	Problem statement	3
1.3	Goals	4
1.4	Scope	4
1.5	Thesis structure	4
2	Theoretical Background	6
2.1	Convex Analysis and Optimization	6
2.1.1	Convex Sets	6
2.1.2	Convex Functions	7
2.1.3	Convex Optimization	8
2.1.4	Conic Optimization	9
2.1.5	Homogenization	10
2.1.6	Mixed-Integer Programs	11
2.2	Graph Theory and the Shortest Path Problem	12
2.2.1	Graph Theory Fundamentals	12
2.2.1.1	Basic Definitions and Concepts	12
2.2.1.2	Combinatorial Optimization on Graphs .	13
2.2.2	The Shortest Path Problem (SPP)	14
2.2.2.1	Definition and Classical Algorithms	14
2.2.2.2	Network Flow Formulation for the Shortest Path Problem	14

2.3	Kinodynamic Trajectory Generation	16
2.3.1	Kinodynamic Formulation	16
2.3.2	Smoothness and Continuity Constraints	17
2.3.3	Background on Bézier Curves	18
2.4	Convex Decomposition	20
2.4.1	Định nghĩa	20
2.4.2	Phân loại Phương pháp Phân hoạch Lồi	20
2.4.2.1	Phân hoạch Lồi Chính xác (Exact Convex Decomposition – ECD)	21
2.4.2.2	Phân hoạch Lồi Xấp xỉ (Approximate Convex Decomposition – ACD)	21
3	Related works	23
3.1	Sampling-Based Methods	23
3.2	Discussion	23
4	Proposed Solution	24
4.1	Phân hoạch không gian an toàn	24
4.2	Các Phương pháp Cốt lõi cho Phân hoạch Không gian Tự do	25
4.2.1	Phân hoạch Lồi Xấp xỉ (ACD) của Đa giác	25
4.2.1.1	Các Khái niệm Cơ bản trong Phân rã Lồi	25
4.2.1.2	Ý tưởng và Chi tiết Thuật toán	26
4.2.1.3	Các Thước đo Độ lõm	28
4.2.1.4	Đặc tính của Thuật toán	29
4.2.2	Iterative Regional Inflation by Semi-Definite Programming (IRIS)	29
4.2.2.1	Ý tưởng và Chi tiết Thuật toán	29
4.2.2.2	Đặc tính của Thuật toán	35

4.2.3	Gieo mầm Tự động qua Lớp phủ Clique Khả kiến (VCC)	36
4.2.3.1	Ý tưởng và Chi tiết Thuật toán	37
4.2.3.2	Đặc tính và Tính tối ưu	40
4.3	Graphs of convex sets	40
4.3.1	Shortest Path Formulation in Graph of Convex Sets	40
4.3.2	Complexity Analysis	42
4.3.3	Detailed Formulation of Vertices and Edges . . .	42
4.3.4	Cost Function Specifications	44
4.3.5	Mixed-Integer Convex Programming Formulation via Perspective Relaxation	45
4.4	Proposed solution	47
4.5	Experiment setup	47
4.5.1	Implementation details	47
4.6	Results and analysis	47
5	Conclusion	48
5.1	Summary	48
5.2	Future Work	48
References		49
Appendix A	...	51
A.1	51
A.2	51

List of Figures

Figure 1.1 Boston Dynamics' Atlas and Amazon's Robin	2
Figure 2.1 Convex Sets Examples	6
Figure 2.2 Convex Hull Examples	7
Figure 4.1 (a) Nếu $x \in \partial P_i > 0$, hàm Resolve gộp biên ∂P_i vào đa giác P_0 . (b) Nếu $x \in \partial P_0$, hàm Resolve tách đa giác P thành hai đa giác P_1 và P_2 . (c) Độ lõm tại điểm x thay đổi sau khi đa giác được phân rã.	27
Figure 4.2 Quá trình đệ quy tiếp tục cho đến khi đạt đến giới hạn độ lõm đầu vào của người dùng τ	27
Figure 4.3 Giả sử r là điểm lõm có độ lõm lớn nhất. Sau khi giải quyết (resolve) r , độ lõm của s tăng lên. Nếu $\text{concave}(r) < \tau$, vùng s sẽ không bao giờ được xử lý, ngay cả khi $\text{concave}(s)$ lớn hơn τ	28
Figure 4.4 Độ lõm SL có thể xử lý được túi trong (a) một cách chính xác vì không có hướng chuẩn nào của các đỉnh trong túi ngược với hướng chuẩn của cầu. Tuy nhiên, túi trong (b) có thể dẫn đến độ lõm giảm không đơn điệu.	29
Figure 4.5 Siêu phẳng Phân cách	30
Figure 4.6 Ellipsoid Nội tiếp Cực đại	33

Figure 4.7 Lắp lại	34
Figure 4.8 Độ phức tạp thời gian	35
Figure 4.9 Lấy mẫu & Xây dựng Đồ thị Khả kiến (Visibility Graphs)	37
Figure 4.10 Tìm Lớp phủ Clique	38
Figure 4.11 Khởi tạo Ellipsoid	38
Figure 4.12 Lấp đầy thành Đa diện	39
Figure 4.13 Lắp lại	39

List of Tables

Chapter 1

Introduction

In chapter 1, the overview, objectives and goals of the research’s project are illustrated. The outline of the report is also presented.

1.1 Motivation and Technical Challenges

Making optimal decisions in real-time is a cornerstone of modern engineering. The necessity of tightly coupling **discrete decisions** with **continuous control** is vividly illustrated in advanced robotics:

- **Humanoid Robots (e.g., Boston Dynamics’ Atlas):** Navigating complex terrain, such as a parkour course, requires the robot to simultaneously select footstep locations (discrete) and compute the trajectories for its center of mass and limbs (continuous). Currently, solutions often rely on “hand-coding” the discrete components, which severely limits the robot’s autonomy in novel environments [2].
- **Industrial Manipulators (e.g., Amazon’s Robin):** Sorting efficiency depends on jointly optimizing the sequence of bins (discrete) and the arm’s trajectory (continuous) [3]. A mere **1% increase** in operating speed through superior optimization translates to millions of additional packages processed annually.



Figure 1.1: **Left:** Boston Dynamics’ Atlas robot performing parkour, requiring discrete footstep planning and continuous trajectory optimization. **Right:** Amazon’s Robin robotic arm sorting packages, where discrete bin selection and continuous arm trajectory must be optimized jointly.

However, solving these coupled problems simultaneously remains an open challenge, typically necessitating manual engineering interventions or settling for suboptimal solutions.

Algorithmically, selecting a motion planning method currently forces researchers to compromise between conflicting features: dimensionality, dynamic constraints, and completeness.

- **Trajectory Optimization:** These methods excel at handling robot dynamics and high-dimensional spaces. However, when facing obstacle-cluttered environments—which render the problem inherently non-convex—they often get trapped in **local minima**, failing to identify a collision-free trajectory.[4]
- **Sampling-based Planners:** To overcome these limitations, the robotics community frequently resorts to sampling-based methods (e.g., RRT, PRM) due to their “probabilistic completeness”. However, this comes at a cost: the resulting trajectories are often significantly **suboptimal**, and imposing **continuous differential constraints** on discrete samples remains notoriously difficult.[5], [6]

Mixed-Integer Convex Programming (MICP) has emerged as a promis-

ing solution that bridges this gap. It combines the strengths of both worlds: the completeness of sampling-based algorithms and the dynamic handling capabilities of trajectory optimization, while guaranteeing **global optimality** within a unified framework. Nevertheless, the adoption of MICP is severely limited by its prohibitive computational costs, often requiring minutes to solve even small-scale problems.

The objective of this research is to break this computational barrier. We focus on a crucial class of motion planning problems involving differential constraints and propose an MICP-based planner capable of reliably solving high-dimensional problems in a matter of seconds through a **Single Convex Program**.

1.2 Problem statement

In the ever-evolving world of technology, the concept of an e-commerce website employing a recommender system to customize the user experience is not unfamiliar, particularly in the gadget and computer sector. Despite the prevalence of e-commerce platforms as standard marketing and sales channels for these firms, there is a need to improve the current methods of user interaction, specifically in facilitating users to effectively retrieve their desired products. While these websites offer a wide range of products, the existing user interface and navigation system present challenges for customers to quickly and efficiently find specific computers and gadgets, resulting in a subpar user experience.

The current user interaction design of e-commerce websites in the computer and gadgets sector lacks the necessary features and functionalities to enhance the browsing and search experience. Users often struggle to locate their desired products due to limited search filters, inadequate categorization, and a lack of intuitive navigation options. This leads to frustration,

increased browsing time, and a potential loss of interest in exploring the full range of offerings.

One of the specific challenges lies in the use of filters to narrow down search results, which often requires users to possess a certain level of domain knowledge in gadgets, electronics, or computers. This can cause confusion for users who are new to this field and may not be familiar with the specific terminology or technical specifications. Similarly, searching using full-text search poses difficulties, as users need to have a good understanding of how to formulate effective search queries to retrieve products that meet their needs.

In summary, by empowering users to find products that meet their needs without requiring extensive technical expertise or struggling to formulate what they need in proper words, the websites can attract a broader customer base, drive conversion rates, and establish a competitive advantage in the market.

1.3 Goals

1.4 Scope

1.5 Thesis structure

There are five chapters in this capstone project:

- Chapter 1 briefly describes the project about problem's motivation, as well as the project's objectives and scope
- Chapter 2 is dedicated to presenting the foundational knowledge about convex optimization, mixed-integer optimization, and graph theory

that are essential for understanding the proposed method.

- Chapter 3 arranges the discussion of related works on the same task to deepen the understanding of existing methods and their constraints.
- Chapter 4 describes the GCS framework and the formulation of our MICP. From that, we present the experimental results and analysis of our method in various environments with different complexity levels.
- Chapter 5 summarizes whole project and the plan for future development.

Chapter 2

Theoretical Background

In Chapter 2, it presents about preliminary knowledge in this project.

2.1 Convex Analysis and Optimization

2.1.1 Convex Sets

Before discussing optimization problems, we briefly define the fundamental geometric objects used in this work. A set $\mathcal{C} \subseteq \mathbb{R}^n$ is *convex* if the line segment between any two points in the set lies entirely within the set.

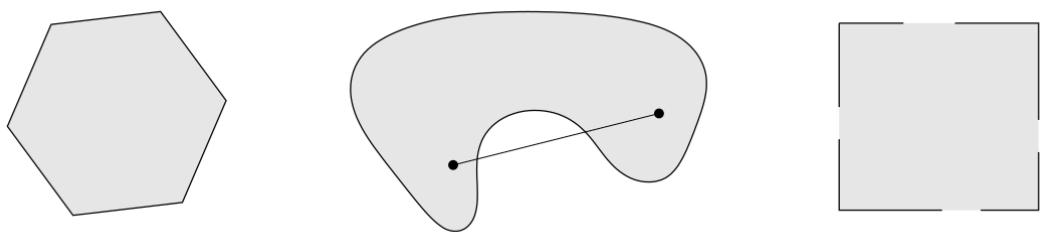


Figure 2.1: Examples of simple convex and nonconvex sets. Left: The hexagon, including its boundary, is convex. Middle: The kidney-shaped set is nonconvex, as the line segment connecting the two points lies outside the set. Right: The square contains some boundary points but not others, and is therefore not convex.[7]

Convex hulls are the smallest convex sets containing a given set of points. Formally, the convex hull of a set of points $\{x_1, x_2, \dots, x_m\} \subseteq \mathbb{R}^n$ is defined

as:

$$\text{conv}(\{x_1, x_2, \dots, x_m\}) = \left\{ \sum_{i=1}^m \lambda_i x_i \mid \lambda_i \geq 0, \sum_{i=1}^m \lambda_i = 1 \right\}.$$

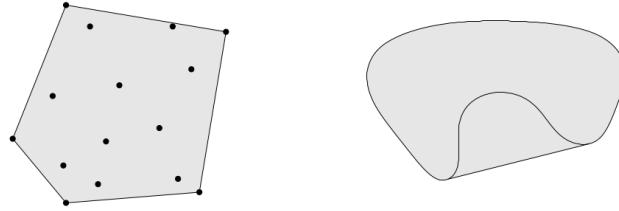


Figure 2.2: Convex hull of a set of points (shaded area).[7]

In the context of this thesis, we primarily focus on two types of convex sets:

- **Polyhedra:** Defined as the intersection of a finite number of half-spaces, $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$. Bounded polyhedra are called *polytopes*.
- **Ellipsoids:** Defined as the image of a unit ball under an affine transformation, typically represented as $\mathcal{E} = \{x \in \mathbb{R}^n \mid (x - c)^\top Q^{-1}(x - c) \leq 1\}$, where $Q \succ 0$.

These sets will serve as the regions (vertices) in our motion planning framework.

2.1.2 Convex Functions

A function $f : \mathcal{D} \rightarrow \mathbb{R}$ is defined as *convex* if its domain $\mathcal{D} \subseteq \mathbb{R}^n$ is a convex set and if, for all $x, y \in \mathcal{D}$ and $\theta \in [0, 1]$, the inequality

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

is satisfied. Geometrically, this condition implies that the line segment connecting any two points on the graph of the function lies above or on the graph itself. A function is termed strictly convex if this inequality holds strictly for all $x \neq y$ and $\theta \in (0, 1)$. Conversely, a function f is concave if $-f$ is convex.

Convex functions possess analytical properties that are fundamental to efficient optimization. The most critical of these is that any local minimum of a convex function is guaranteed to be a global minimum.

2.1.3 Convex Optimization

A **Convex Program (CP)** is an optimization problem of the form

$$\text{minimize } f(x) \quad (2.1a)$$

$$\text{subject to } x \in \mathcal{C}, \quad (2.1b)$$

where the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and the constraint set $\mathcal{C} \subset \mathbb{R}^n$ are convex. Sometimes we will refer to the set \mathcal{C} as the *feasible set* of the CP, and to its elements as *feasible solutions*. The CP is said to be *feasible* if a feasible solution exists, i.e., if $\mathcal{C} \neq \emptyset$. A feasible solution x^{opt} is *optimal* if $f(x^{\text{opt}}) \leq f(x)$ for all $x \in \mathcal{C}$. If an optimal solution exists, we call $f^{\text{opt}} := f(x^{\text{opt}})$ the *optimal value* of the CP.

A fundamental property of CPs is that any locally optimal solution (i.e., any point x^{opt} such that $f(x^{\text{opt}}) \leq f(x)$ for all x in a neighborhood of x^{opt}) is also an optimal solution according to the (global) definition just given [7]. A commonly satisfied assumption for the existence of an optimal solution is that the feasible set \mathcal{C} is nonempty and compact.

CPs are a fundamental class of optimization problems, with applications in essentially every engineering discipline. The great majority of the CPs that we encounter in practice can be solved efficiently and reliably using interior-point methods (or other specialized algorithms such as the simplex method). However, strictly speaking, it is not always true that a CP can be solved efficiently: for example, the set of nonnegative polynomials is a convex cone (in the space of the polynomial coefficients), but checking if a polynomial is nonnegative is NP-hard. In this thesis we will be a little

imprecise, and use the term “convex optimization problem” almost as a synonym of “optimization problem that is efficiently solvable”.

2.1.4 Conic Optimization

A **Conic Program (KP)** is a CP with linear objective function and constraint set in conic form:

$$\text{minimize} \quad c^\top x \tag{2.2a}$$

$$\text{subject to} \quad Ax + b \in \mathcal{K}, \tag{2.2b}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $\mathcal{K} \subseteq \mathbb{R}^m$ is a closed convex cone.

Special classes of KPs are:

- **Linear Program (LP)** when \mathcal{K} is the nonnegative orthant.
- **Second-Order Cone Program (SOCP)** when \mathcal{K} is the Cartesian product of second-order cones.
- **Semidefinite Program (SDP)** when \mathcal{K} is (isomorphic to) the semidefinite cone.

These classes of problems have increasing modelling power: every LP is an SOCP, and every SOCP is an SDP. SDPs are efficiently solvable, and cover the vast majority of the practical uses of convex optimization.

Another important class of CPs are **Quadratic Programs (QP)**, these are CPs in the form (2.1) with quadratic objective function f and polyhedral constraint set \mathcal{C} . Every QP can be formulated as an SOCP. However, in many cases, active-set algorithms specialized to this class of problems can be more effective than using an SOCP solver.

2.1.5 Homogenization

Homogenization is a technique used in convex analysis to transform a non-convex set into a convex one by introducing an additional dimension. This process is particularly useful in optimization problems where convexity is desired for efficient solution methods.

Given a set $\mathcal{S} \subseteq \mathbb{R}^n$, the homogenization of \mathcal{S} , denoted as $\tilde{\mathcal{S}}$, is defined as:

$$\tilde{\mathcal{S}} := \{(x, y) \in \mathbb{R}^{n+1} : y > 0, x \in y\mathcal{S}\}.$$

This transformation effectively “lifts” the set \mathcal{S} into a higher-dimensional space, where the additional dimension y allows for the representation of convex combinations of points in \mathcal{S} . The homogenized set $\tilde{\mathcal{S}}$ is convex if and only if the original set \mathcal{S} is convex.

Homogenization of Sets in Conic Form

The homogenization of a convex set \mathcal{C} described in conic form is computed very easily. In fact, for $y > 0$, we observe that

$$y\mathcal{C} = \{yx : Ax + b \in \mathcal{K}\} = \{yx : A(yx) + by \in y\mathcal{K}\} = \{x' : Ax' + by \in \mathcal{K}\},$$

and this gives us

$$\tilde{\mathcal{C}} = \{(x, y) : y > 0, Ax + by \in \mathcal{K}\}. \quad (2.3)$$

In addition, it is also easily verified that

$$\text{cl}(\tilde{\mathcal{C}}) = \{(x, y) : y \geq 0, Ax + by \in \mathcal{K}\}. \quad (2.4)$$

The expressions (2.3) and (2.4) have great practical relevance. Roughly speaking, they tell us that if we can efficiently do computations with a set \mathcal{C} described in conic form, then the same is true for (the closure of) its

homogenization $\tilde{\mathcal{C}}$.

Homogenization is particularly useful in optimization problems, as it allows for the application of convex optimization techniques to problems that may initially appear non-convex. By working in the homogenized space, one can leverage the properties of convex sets and functions to find optimal solutions more efficiently.

2.1.6 Mixed-Integer Programs

In the field of mathematical optimization, Mixed-Integer Programs (MIPs) represent a class of problems in which the decision variable vector is not uniform regarding its domain. Instead, the variables are partitioned into two distinct subsets: one constrained to integer values, and the other permitted to assume continuous real values. MIPs can be formulated as follows:

$$\text{minimize} \quad f(x, z) \tag{2.5a}$$

$$\text{subject to} \quad (x, z) \in \mathcal{C}, \tag{2.5b}$$

$$z \in \mathbb{Z}^m, \tag{2.5c}$$

where $x \in \mathbb{R}^n$ denotes the continuous variables, $z \in \mathbb{Z}^m$ denotes the integer variables, $f : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$ is the objective function, and $\mathcal{C} \subseteq \mathbb{R}^{n+m}$ represents the feasible set.

While linear programming problems are defined over convex polyhedra, allowing for the application of gradient-based or simplex methods, the imposition of integrality constraints renders the feasible set discrete and non-convex. This loss of convexity significantly increases computational complexity, typically transforming polynomial-time solvable problems into NP-hard tasks. Consequently, standard analytical methods relying on continuity and differentiability are not directly applicable.

To address MIPs, relaxation methods are frequently employed. The most

common approach is linear programming relaxation, wherein integer constraints $z_j \in \mathbb{Z}$ are replaced by weaker continuous constraints (for instance, replacing $z_j \in \{0, 1\}$ with $0 \leq z_j \leq 1$). The objective function value obtained from the relaxed problem provides a bound (specifically, a lower bound for minimization problems) on the optimal value of the original MIP. If the solution to the relaxed problem satisfies all integrality constraints, it is also the optimal solution for the MIP. However, if the relaxed solution contains fractional values for variables required to be integers, algorithms such as Branch-and-Bound are typically implemented. The Branch-and-Bound method partitions the feasible region into smaller sub-regions (branches) by introducing new constraints to eliminate fractional values, thereby creating a search tree to systematically explore the solution space.

2.2 Graph Theory and the Shortest Path Problem

2.2.1 Graph Theory Fundamentals

2.2.1.1 Basic Definitions and Concepts

Formally, a graph is defined as an ordered pair $G = (\mathcal{V}, \mathcal{E})$, comprising a set of vertices \mathcal{V} and a set of edges \mathcal{E} , where each edge connects a pair of vertices. The structural classification of the graph depends on the ordering of these pairs: unordered pairs $\{u, w\}$ denote an undirected graph, whereas ordered pairs (u, w) characterize a directed graph. In the context of optimization, the graph is typically augmented with a cost function $c : \mathcal{E} \rightarrow \mathbb{R}$, assigning a real-valued weight to each edge.

Key topological properties include the vertex degree, which counts incident edges; in directed graphs, this is distinguished into in-degree (\deg^-) and out-degree (\deg^+). Connectivity is described via paths—finite sequences of

distinct vertices and edges—where the path length is the aggregate cost of its constituent edges. A cycle is defined as a path that originates and terminates at the same vertex. Optimization problems generally operate on a search domain defined by subgraphs $H = (W, F)$, formed by subsets of vertices $W \subseteq \mathcal{V}$ and edges $F \subseteq \mathcal{E}$ that maintain incidence relationships.

2.2.1.2 Combinatorial Optimization on Graphs

Combinatorial optimization problems on graphs generally seek a substructure that minimizes an objective function subject to validity constraints. For a weighted graph $G = (\mathcal{V}, \mathcal{E})$ and a set of feasible subgraphs \mathcal{H} , the problem is to identify a subgraph $H = (W, F) \in \mathcal{H}$ that minimizes the cumulative weight of its components. This is formally expressed as:

$$\text{minimize} \quad \sum_{v \in W} c_v + \sum_{e \in F} c_e \quad (1.1a)$$

$$\text{subject to} \quad H = (W, F) \in \mathcal{H}. \quad (1.1b)$$

To utilize standard mathematical programming techniques, such as Branch and Bound, the problem is transformed into an Integer Linear Programming (ILP) formulation. This involves parameterizing the search space using incidence vectors $\mathbf{y}^H \in \{0, 1\}^{|\mathcal{V} \cup \mathcal{E}|}$, where binary variables indicate the inclusion of specific vertices or edges in the solution. Geometrically, the set of valid subgraphs corresponds to a polytope \mathcal{Y} in Euclidean space. The intersection of this polytope with the integer grid, $\mathcal{Y} \cap \{0, 1\}^{|\mathcal{V} \cup \mathcal{E}|}$, yields the incidence vectors of feasible subgraphs. Consequently, the discrete optimization problem is recast as a linear optimization over an integer domain:

$$\text{minimize} \quad \sum_{v \in \mathcal{V}} c_v y_v + \sum_{e \in \mathcal{E}} c_e y_e \quad (1.2a)$$

$$\text{subject to} \quad \mathbf{y} \in \mathcal{Y} \cap \{0,1\}^{|\mathcal{V} \cup \mathcal{E}|}. \quad (1.2b)$$

This formulation allows for the application of convex analysis, ensuring that the optimal solution to the algebraic model corresponds exactly to the optimal substructure in the original graph problem.

2.2.2 The Shortest Path Problem (SPP)

2.2.2.1 Definition and Classical Algorithms

The Shortest Path Problem (SPP) on a weighted graph $G = (\mathcal{V}, \mathcal{E})$ involves identifying a path P between a source s and a destination t such that the summation of edge weights is minimized. If P comprises a sequence of edges e_1, \dots, e_k , the objective is to minimize $\sum_{i=1}^k c_{e_i}$.

Classical approaches to solving the SPP vary based on edge weight characteristics. Dijkstra's algorithm employs a greedy strategy suitable for graphs with non-negative weights ($c_e \geq 0$). For graphs containing negative edge weights, provided no negative cycles exist, the Bellman-Ford algorithm is utilized, albeit with higher computational complexity. The Floyd-Warshall algorithm extends this to the all-pairs shortest path problem, accommodating negative edges but remaining sensitive to negative cycles.

2.2.2.2 Network Flow Formulation for the Shortest Path Problem

The Shortest Path Problem on a directed graph $G = (\mathcal{V}, \mathcal{E})$ with non-negative edge costs $c_{uv} \geq 0$ can be rigorously modeled as a Minimum Cost Flow problem involving a single unit of flow. We define binary decision

variables $x_{uv} \in \{0, 1\}$ for each edge $(u, v) \in \mathcal{E}$, where $x_{uv} = 1$ indicates the edge is part of the optimal path. The optimization model is formulated as follows:

$$\text{minimize} \quad \sum_{(u,v) \in \mathcal{E}} c_{uv} x_{uv} \quad (2.6a)$$

$$\text{subject to} \quad \sum_{v \in \mathcal{V}^+(u)} x_{uv} - \sum_{v \in \mathcal{V}^-(u)} x_{vu} = \begin{cases} 1 & \text{if } u = s \\ -1 & \text{if } u = t \\ 0 & \text{otherwise} \end{cases}, \quad \forall u \in \mathcal{V} \quad (2.6b)$$

$$\sum_{v \in \mathcal{V}^-(u)} x_{vu} \leq 1, \quad \forall u \in \mathcal{V} \setminus \{s, t\} \quad (2.6c)$$

$$x_{uv} \geq 0, \quad \forall (u, v) \in \mathcal{E} \quad (2.6d)$$

Here, $\mathcal{V}^+(u)$ and $\mathcal{V}^-(u)$ denote the sets of successors and predecessors of node u , respectively. The constraints in this formulation enforce specific structural properties. Equation (2.6b) represents flow conservation (Kirchhoff's law), ensuring that a net flow of one unit leaves the source s , enters the target t , and is conserved at all intermediate nodes. Equation (2.6c) imposes a node capacity constraint, restricting the inflow at any intermediate vertex to unity, thereby enforcing a simple path structure where no vertex is revisited. Given non-negative costs, cycles are inherently suboptimal, rendering explicit cycle-elimination constraints unnecessary.

A significant advantage of this formulation lies in the algebraic properties of the constraint matrix derived from (2.6b). This node-arc incidence matrix exhibits Total Unimodularity (TUM). According to combinatorial optimization theory, if the constraint matrix is TUM and the right-hand side vector is integral, every basic feasible solution of the Linear Programming (LP) relaxation is guaranteed to be integral [8]. Consequently, despite the continuous non-negativity constraints in (2.6d), the problem can

be solved efficiently using standard LP algorithms to yield a valid binary path, obviating the need for computationally intensive integer programming methods.

2.3 Kinodynamic Trajectory Generation

This section outlines the mathematical formulation for generating kinodynamically feasible trajectories. Unlike geometric path planning, which solely considers collision-free configurations in the workspace, motion planning (kinodynamic) accounts for the system's differential constraints, such as velocity, acceleration, and jerk limits. We leverage Bézier curves as the trajectory parameterization scheme, which is particularly advantageous for the Graph of Convex Sets (GCS) framework due to its convexity properties.

2.3.1 Kinodynamic Formulation

Consider a robotic system whose state evolution is governed by a set of ordinary differential equations (ODEs):

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t) \in \mathcal{X}, \mathbf{u}(t) \in \mathcal{U} \quad (2.7)$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ represents the state vector (typically comprising position, velocity, and acceleration), and $\mathbf{u}(t) \in \mathbb{R}^m$ denotes the control inputs. The goal is to compute a trajectory $\sigma(t) : [0, T] \rightarrow \mathcal{X}$ that minimizes a specific cost functional while satisfying:

1. **Boundary Conditions:** The start and goal states $\mathbf{x}(0) = \mathbf{x}_{start}$ and $\mathbf{x}(T) = \mathbf{x}_{goal}$.
2. **Safety Constraints:** $\sigma(t) \in \mathcal{X}_{free}$ for all t , ensuring collision avoidance.

3. **Dynamic Limits:** Inequality constraints on higher-order derivatives to ensure physical feasibility:

$$|\dot{\sigma}(t)| \leq v_{max}, \quad |\ddot{\sigma}(t)| \leq a_{max}, \quad |\ddot{\sigma}(t)| \leq j_{max} \quad (2.8)$$

2.3.2 Smoothness and Continuity Constraints

In the GCS framework, a full trajectory is composed of multiple piecewise Bézier segments. To ensure smooth motion across the boundaries of adjacent convex sets, continuity constraints must be enforced at the transition points. Let $r_A(t)$ defined by control points $\{a_0, \dots, a_n\}$ and $r_B(t)$ defined by $\{b_0, \dots, b_n\}$ be two consecutive segments.

- **C^0 Continuity (Position):** Ensures the path is connected.

$$a_n = b_0 \quad (2.9)$$

- **C^1 Continuity (Velocity):** Ensures continuous velocity, preventing infinite acceleration spikes.

$$a_n - a_{n-1} = b_1 - b_0 \quad (2.10)$$

- **C^2 Continuity (Acceleration):** Ensures continuous force/torque application, which is crucial for minimizing mechanical wear and jerk.

$$(a_n - 2a_{n-1} + a_{n-2}) = (b_2 - 2b_1 + b_0) \quad (2.11)$$

By enforcing these equality constraints at the edges of the graph, we generate a globally smooth trajectory that respects the underlying physics of the robot.

2.3.3 Background on Bézier Curves

In order to tackle the kinodynamic planning problem numerically, it is necessary to parameterize the trajectory functions through a finite number of decision variables. To this end, we employ Bézier curves. This section recalls the definition and the basic properties of this family of curves.

A Bézier curve is constructed using Bernstein polynomials. The k -th Bernstein polynomial of degree d , with $k = 0, \dots, d$, is defined as:

$$\beta_{k,d}(s) := \binom{d}{k} s^k (1-s)^{d-k}, \quad s \in [0, 1] \quad (2.12)$$

Note that the Bernstein polynomials of degree d are nonnegative and, by the binomial theorem, they sum up to one. Therefore, for each fixed $s \in [0, 1]$, the scalars $\{\beta_{k,d}(s)\}_{k=0}^d$ can be thought of as the coefficients of a convex combination. Bézier curves are obtained using these coefficients to combine a given set of $d+1$ control points $\gamma_k \in \mathbb{R}^n$:

$$\gamma(s) := \sum_{k=0}^d \beta_{k,d}(s) \gamma_k \quad (2.13)$$

It is easily verified that Bézier curves enjoy the following properties, which are instrumental for the GCS framework:

- **Endpoint values:** The curve γ starts at the first control point and ends at the last control point:

$$\gamma(0) = \gamma_0 \quad \text{and} \quad \gamma(1) = \gamma_d \quad (2.14)$$

This property is essential for enforcing continuity constraints (C^0) when stitching multiple Bézier segments together in the graph.

- **Convex hull property:** The curve γ is entirely contained in the

convex hull of its control points:

$$\gamma(s) \in \text{conv}(\{\gamma_k\}_{k=0}^d), \quad \forall s \in [0, 1] \quad (2.15)$$

In the context of GCS, if a convex region X_v represents a safe set, guaranteeing collision avoidance is reduced to constraining all control points to lie within that set:

$$\gamma_k \in X_v, \quad \forall k \in \{0, \dots, d\} \implies \gamma(s) \in X_v \quad (2.16)$$

This allows us to enforce continuous safety constraints through a finite set of linear inclusion constraints.

- **Derivative (Hodograph):** The derivative $\dot{\gamma}$ of the curve γ is also a Bézier curve, but of degree $d - 1$, with control points defined by the difference of adjacent points:

$$\dot{\gamma}(s) = \sum_{k=0}^{d-1} \beta_{k,d-1}(s) \dot{\gamma}_k \quad (2.17)$$

where $\dot{\gamma}_k = d(\gamma_{k+1} - \gamma_k)$ for $k = 0, \dots, d - 1$. Similarly, higher-order derivatives (acceleration, jerk) can be expressed as linear combinations of the control points. Consequently, kinodynamic constraints (e.g., velocity limits $|\dot{\gamma}(s)| \leq v_{max}$) can be imposed as linear constraints on the differences between adjacent control points, preserving the convexity of the optimization problem.

- **Integral of convex function:** For a convex function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ (often representing the cost function in motion planning, such as energy or path length), the integral along the curve is bounded by the convex

combination of the function values at the control points:

$$\int_0^1 f(\gamma(s))ds \leq \frac{1}{d+1} \sum_{k=0}^d f(\gamma_k) \quad (2.18)$$

This inequality provides a convex upper bound on the integral cost, facilitating efficient optimization within the GCS mixed-integer convex programming formulation.

2.4 Convex Decomposition

This section provides an overview of convex decomposition, a fundamental technique used in computational geometry and physics simulations.

2.4.1 Định nghĩa

Phân hoạch lồi (Convex Decomposition) là quá trình chia một hình học phức tạp (ví dụ: đa giác hoặc đa diện không lồi) thành một tập hợp các **vùng con lồi** sao cho hợp của các vùng này khôi phục lại toàn bộ hình ban đầu và các vùng con không chồng lấn nhau, ngoại trừ biên chung.

Về mặt hình 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. Mục tiêu là tìm tập phân hoạch này như là số lượng các vùng lồi tối thiểu, giảm thiểu phần chồng chéo hay chất lượng của vùng lồi được tạo ra (tránh quá "mảnh" hoặc "dài").

2.4.2 Phân loại Phương pháp Phân hoạch Lồi

Trong lĩnh vực hình học tính toán và hoạch định chuyển động, các phương pháp phân hoạch lồi thường được chia thành hai loại chính: **Phân hoạch**

Lồi Chính xác (Exact Convex Decomposition – ECD) và **Phân hoạch Lồi Xấp xỉ** (Approximate Convex Decomposition – ACD). Sự khác biệt giữa hai loại này phản ánh sự đánh đổi cơ bản giữa *tính chính xác hình học* và *hiệu quả tính toán*.

2.4.2.1 Phân hoạch Lồi Chính xác (Exact Convex Decomposition – ECD)

Phân hoạch Lồi Chính xác (ECD) là quá trình chia một hình dạng không lồi thành các **thành phần con hoàn toàn lồi**, không chồng lấn, sao cho hợp của chúng tái tạo lại chính xác hình dạng ban đầu. Mục tiêu thường là tối thiểu hóa số lượng vùng lồi.

Tuy nhiên, ECD gặp hai thách thức lớn:

- **Độ phức tạp tính toán cao:** Bài toán tìm phân hoạch lồi tối thiểu đã được chứng minh là NP-hard đối với đa giác có lỗ, nên không tồn tại thuật toán hiệu quả cho các trường hợp tổng quát.
- **Tính khả dụng hạn chế:** Ngay cả khi tính toán được, ECD thường tạo ra số lượng vùng rất lớn, khiến việc lưu trữ, xử lý và tối ưu hóa tiếp theo trở nên tốn kém.

Do đó, các phương pháp ECD chủ yếu mang tính lý thuyết, dùng làm chuẩn so sánh, trong khi hầu hết các ứng dụng thực tế trong robot và mô phỏng đều dựa vào các phương pháp xấp xỉ.

2.4.2.2 Phân hoạch Lồi Xấp xỉ (Approximate Convex Decomposition – ACD)

Trước những giới hạn của ECD, các phương pháp **Phân hoạch Lồi Xấp xỉ (ACD)** được phát triển để đạt sự cân bằng giữa độ chính xác và hiệu quả. ACD cho phép các thành phần không hoàn toàn lồi, mà chỉ “*gần lồi*” trong một mức dung sai lõm τ do người dùng xác định hoặc chỉ bao phủ

một phần không gian tự do. Triết lý cốt lõi của ACD là chấp nhận một mức độ lõm nhỏ có kiểm soát để đổi lấy:

- **Hiệu quả tính toán cao hơn:** Các thuật toán ACD, chẳng hạn như phương pháp của Lien và Amato, chạy nhanh hơn nhiều so với ECD. [9]
- **Giảm mạnh số lượng vùng:** Cho phép biểu diễn hình dạng với ít vùng lồi hơn, đơn giản hơn và dễ xử lý hơn.
- **Kiểm soát mức độ chi tiết:** Một số thuật toán ACD như phương pháp của Lien và Amato cho phép người dùng điều chỉnh tham số lõm τ để cân bằng giữa độ chính xác và số lượng vùng lồi. Hay đối với phương pháp Visibility Clique Cover (VCC), nó cho phép người dùng chọn chỉ số để thể hiện mức độ bao phủ của vùng lồi so với không gian tự do ban đầu. [10]

Chapter 3

Related works

...

3.1 Sampling-Based Methods

3.2 Discussion

In this chapter, we have seen three distinct approaches to enhance...

Chapter 4

Proposed Solution

...

4.1 Phân hoạch không gian an toàn

Thay vì tiếp cận bài toán tránh vật cản theo cách truyền thống là mô tả các ràng buộc không lồi dựa trên bề mặt vật cản, nhóm tác giả đề xuất một phương pháp chuyển đổi không gian cấu hình tự do (collision-free configuration space) thành hợp của một tập hợp hữu hạn các vùng lồi bị chặn (bounded convex regions). Trong mô hình này, các vùng lồi có thể chồng lấn lên nhau, và bài toán hoạch định chuyển động được phát biểu lại dưới dạng một chương trình quy hoạch lồi nguyên hỗn hợp (Mixed-Integer Convex Program - MICP). Cụ thể, các biến nguyên được sử dụng để gán từng đoạn quỹ đạo đa thức (polynomial trajectory segments) vào các vùng lồi an toàn cụ thể này. Phương pháp này mang lại lợi thế lớn về mặt tính toán vì số lượng biến nguyên chỉ phụ thuộc vào số lượng vùng an toàn được tạo ra (sử dụng thuật toán IRIS) thay vì phụ thuộc vào số lượng mặt của vật cản, giúp giải quyết hiệu quả các môi trường phức tạp với hàng trăm vật cản. Hơn nữa, việc ràng buộc quỹ đạo nằm trong các tập lồi đảm bảo an toàn tuyệt đối cho toàn bộ hành trình liên tục, khắc phục nhược điểm "cắt góc" (corner cutting) thường gặp ở các phương pháp chỉ kiểm tra

va chạm tại các điểm lây mảnh rời rạc.

4.2 Các Phương pháp Cốt lõi cho Phân hoạch Không gian Tự do

Để giải quyết thách thức về tính toán của phân hoạch tối ưu, một số các phương pháp đã được phát triển. Dưới đây là một số phương pháp được dùng trong bài báo cáo này:

4.2.1 Phân hoạch Lồi Xấp xỉ (ACD) của Đa giác

Phương pháp Approximate Convex Decomposition (ACD) giới thiệu một cách tiếp cận khác biệt cơ bản để quản lý sự phức tạp. Thay vì yêu cầu sự lồi hoàn hảo, nó cho phép các thành phần "lồi xấp xỉ" trong một dung sai do người dùng xác định, τ .[9] Điều này thường dẫn đến các phân hoạch có ít thành phần hơn nhiều và phù hợp hơn với các đặc điểm cấu trúc nổi bật của đối tượng.

4.2.1.1 Các Khái niệm Cơ bản trong Phân rã Lồi

Phần này trình bày các khái niệm hình học cơ bản được sử dụng trong quá trình phân rã đa giác không lồi thành các vùng lồi. Các định nghĩa này đóng vai trò nền tảng cho các phương pháp đo độ lõm và các thuật toán phân hoạch ở các phần sau.

- **Bao lồi (Convex Hull – H_P):** Bao lồi của một đa giác P là **tập hợp lồi nhỏ nhất chứa toàn bộ đa giác** đó. Trực quan, có thể hình dung bao lồi như một sợi dây chun được kéo căng bao quanh đa giác. Một đa giác P được gọi là **lồi** nếu và chỉ nếu nó trùng với bao lồi của chính nó, tức là:

$$P = H_P.$$

- **Notches:** Là các đỉnh của đa giác P không nằm trên bao lồi H_P . Về mặt hình học, đây là những đỉnh có **góc trong lớn hơn** 180° . Các Notches biểu hiện các đặc trưng lõm (concave features) trên biên của đa giác.
- **Cầu (Bridges):** Là các cạnh của bao lồi ∂H_P nối hai đỉnh không liền kề của biên ngoài ∂P_0 . Về mặt toán học, tập hợp các cầu được định nghĩa:

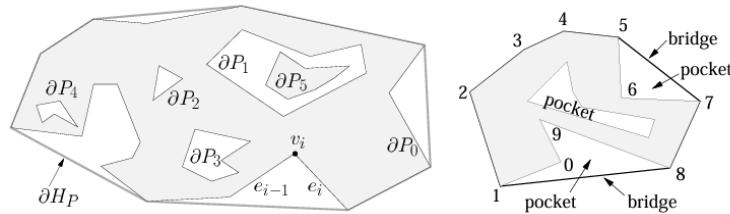
$$\text{BRIDGES}(P) = \partial H_P \setminus \partial P.$$

Mỗi cầu “bắc qua” một vùng lõm của đa giác.

- **Túi (Pockets):** Là các **chuỗi cạnh liên tiếp** của biên đa giác ∂P không thuộc về vỏ lồi ∂H_P . Về mặt toán học:

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

Mỗi túi tương ứng với một “vịnh lõm” (concave bay) trên đường biên của đa giác.



4.2.1.2 Ý tưởng và Chi tiết Thuật toán

Thuật toán là một chiến lược chia để trị đệ quy[9]:

1. Đo độ lõm: Xác định đặc điểm không lồi (một "notch" hay đỉnh lõm) quan trọng nhất trong đa giác. Đây là đỉnh có độ lõm lớn nhất. Tùy theo từng trường hợp mà ta có thể sử dụng các thước đo độ lõm khác

nhau, cách để tính những thước đo độ lõm nay sẽ được giới thiệu rõ hơn ở phần 4.2.1.3

2. Kiểm tra Dung sai: Nếu độ lõm tối đa đo được nhỏ hơn τ , quá trình kết thúc đối với thành phần đó.
3. Giải quyết Đỉnh lõm: Nếu độ lõm vượt quá τ , một đường cắt được thêm vào để giải quyết đỉnh lõm, chia đa giác thành hai thành phần mới (hoặc hợp nhất một lõi).

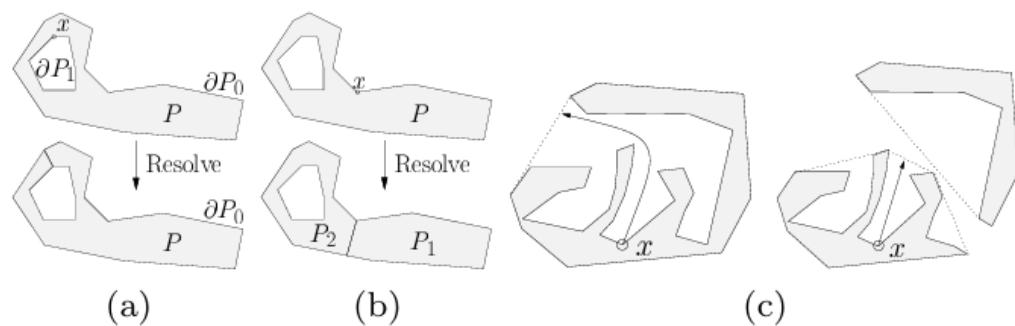


Figure 4.1: (a) Nếu $x \in \partial P_i > 0$, hàm **Resolve** **gộp** biên ∂P_i vào đa giác P_0 . (b) Nếu $x \in \partial P_0$, hàm **Resolve** **tách** đa giác P thành hai đa giác P_1 và P_2 . (c) **Độ lõm** tại điểm x thay đổi sau khi đa giác được phân rã.

4. Đệ quy: Quá trình được áp dụng đệ quy cho các thành phần mới.

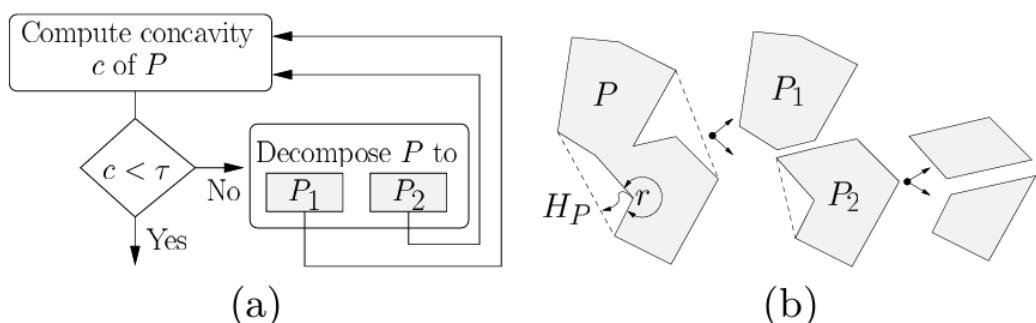


Figure 4.2: Quá trình đệ quy tiếp tục cho đến khi đạt đến giới hạn độ lõm đầu vào của người dùng τ .

4.2.1.3 Các Thước đo Độ lõm

Chất lượng của phân hoạch phụ thuộc rất nhiều vào **cách đo độ lõm** của các đỉnh trong đa giác không lồi. Các phương pháp đo độ lõm phổ biến bao gồm:

- **Độ lõm Đường thẳng (Straight-Line Concavity – SL-Concavity):** Là khoảng cách Euclid từ một đỉnh trong *túi* (pocket) đến *cầu nối* (bridge) liên kết của nó — thường là một cạnh thuộc bao lồi của đa giác. Phương pháp này rất nhanh, nhưng **không đảm bảo tính đơn điệu**, có thể khiến một số đặc trưng lõm quan trọng bị bỏ sót.



Figure 4.3: Giả sử r là điểm lõm có độ lõm lớn nhất. Sau khi giải quyết (resolve) r , độ lõm của s tăng lên. Nếu $\text{concave}(r) < \tau$, vùng s sẽ không bao giờ được xử lý, ngay cả khi $\text{concave}(s)$ lớn hơn τ .

- **Độ lõm Đường đi Ngắn nhất (Shortest Path Concavity – SP-Concavity):** Là **chiều dài đường đi ngắn nhất** từ một đỉnh trong túi đến cầu nối tương ứng, với điều kiện đường đi đó phải nằm hoàn toàn trong túi. Phương pháp này chậm hơn SL-Concavity nhưng **đảm bảo tính đơn điệu**: độ lõm của các đỉnh giảm dần sau mỗi lần phân rã, giúp đảm bảo rằng tất cả các đặc điểm lõm quan trọng cuối cùng sẽ được xử lý. (Xem chi tiết trong Phần ??.)
- **Độ lõm Lai (Hybrid Concavity – H-Concavity):** Một giải pháp trung gian thực tiễn — sử dụng SL-Concavity nhanh làm mặc định và chỉ chuyển sang SP-Concavity khi **phát hiện khả năng không đơn điệu**. Bằng cách thêm bước kiểm tra *nguy cơ tiềm ẩn* bằng cách so sánh n_β và n_i ; nếu tồn tại $n_i \cdot n_\beta < 0$, biên túi bị đảo hướng — dấu

hiệu của túi phức tạp mà SL-Concavity có thể thất bại.

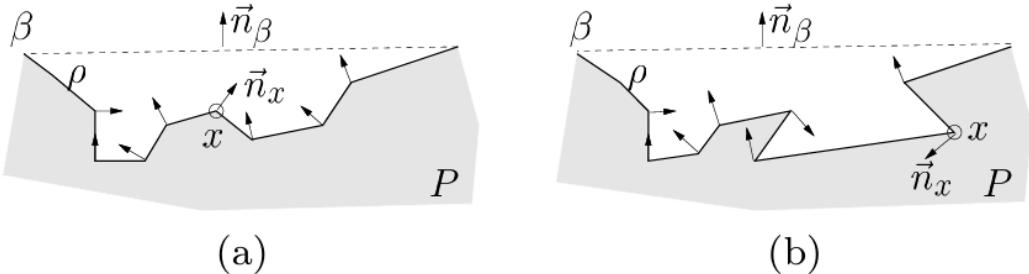


Figure 4.4: Độ lõm SL có thể xử lý được túi trong (a) một cách chính xác vì không có hướng chuẩn nào của các đỉnh trong túi ngược với hướng chuẩn của cầu. Tuy nhiên, túi trong (b) có thể dẫn đến độ lõm giảm không đơn điệu.

4.2.1.4 Đặc tính của Thuật toán

- Loại thuật toán: Đây là một phương pháp xấp xỉ theo định nghĩa. Một phân hoạch với $\tau = 0$ là một phân hoạch lồi chính xác.
- Độ phức tạp thời gian: $O(nr^2)$, trong đó n là số đỉnh và r là số đỉnh lõm.[9] Điều này nhanh hơn các thuật toán phân hoạch chính xác tối ưu cho đa giác không có lỗ.

4.2.2 Iterative Regional Inflation by Semi-Definite Programming (IRIS)

Thuật toán Iterative Regional Inflation by Semidefinite Programming (IRIS) thay vì cố gắng bao phủ toàn bộ không gian tự do cùng một lúc, nó đặt ra câu hỏi: "Với một điểm 'mầm' (seed) ban đầu, vùng lồi lớn nhất của không gian tự do mà tôi có thể tìm thấy xung quanh nó là gì?". Tức sau khi chạy xong giải thuật output của ta là một vùng lồi lớn nhất có thể của thuật toán từ 1 seed point cho trước.

4.2.2.1 Ý tưởng và Chi tiết Thuật toán

IRIS là một quy trình tối ưu hóa lặp đi lặp lại gồm 4 bước:

1. **Khởi tạo:** Thuật toán sẽ tạo ra một hình elip ban đầu là một hình cầu rất nhỏ có tâm chính là điểm mầm (seed point).

2. **Tìm Siêu phẳng Phân cách (QP):**

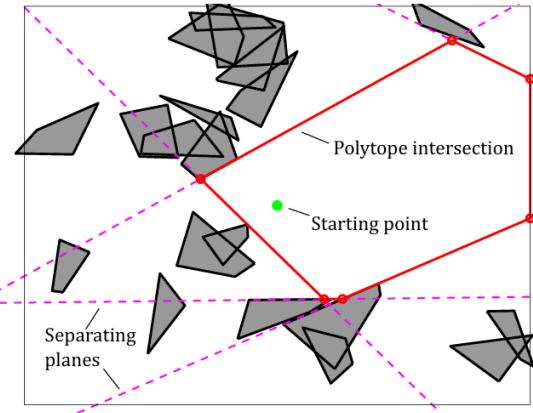


Figure 4.5: Siêu phẳng Phân cách

Đối với ellipsoid hiện tại, thuật toán tìm điểm gần nhất trên mỗi chướng ngại vật. Sau đó, nó xây dựng các siêu phẳng phân cách ellipsoid khỏi các chướng ngại vật này. Mỗi siêu phẳng tiếp tuyến với một phiên bản mở rộng của ellipsoid và đi qua điểm gần nhất trên chướng ngại vật tương ứng. Việc tìm điểm gần nhất trên mỗi chướng ngại vật lồi được xây dựng như một bài toán Quy hoạch Toàn phương (Quadratic Program - QP). Để giữ cho bài toán tối ưu hóa tiếp theo có quy mô nhỏ, các siêu phẳng thừa (những siêu phẳng không xác định biên của vùng lồi kết quả) sẽ được loại bỏ.

Bước 2.1. Tìm điểm gần nhất trên obstacle đến hình elip hiện tại

- **Biến đổi không gian (Từ không gian Ellipsoid sang không gian Quả cầu đơn vị)**

Việc tính toán khoảng cách đến một ellipsoid là phức tạp. Tuy nhiên, tính khoảng cách đến một quả cầu đơn vị tại gốc tọa độ thì lại rất đơn giản. Ý tưởng ở đây là biến đổi toàn bộ không gian sao cho ellipsoid trở thành một quả cầu đơn vị.

- Một ellipsoid E được định nghĩa là ảnh của một quả cầu đơn vị \tilde{E} qua phép biến đổi affine:

$$x = C\tilde{x} + d.$$

- Sử dụng phép biến đổi ngược:

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

để ánh xạ mọi điểm trở lại “không gian quả cầu đơn vị”.

- Trong không gian mới này, ellipsoid trở thành quả cầu đơn vị \tilde{E} tại gốc tọa độ, và mỗi chướng ngại vật l_j cũng bị biến đổi (méo đi) thành một đối tượng mới \tilde{l}_j .

Bước 2.2: Tìm điểm gần nhất bằng Quy hoạch Toàn phương (QP)

Bây giờ, bài toán tìm điểm trên chướng ngại vật l_j gần ellipsoid E nhất tương đương với việc tìm điểm trên chướng ngại vật đã biến đổi \tilde{l}_j gần gốc tọa độ nhất.

Bài toán này được xây dựng dưới dạng một bài toán *Quy hoạch Toàn phương* (Quadratic Program - QP):

$$\begin{aligned} & \underset{\tilde{x} \in \mathbb{R}^n, w \in \mathbb{R}^m}{\text{minimize}} \quad \|\tilde{x}\|^2 \\ & \text{subject to} \quad \begin{cases} [\tilde{v}_{j,1} \ \tilde{v}_{j,2} \ \cdots \ \tilde{v}_{j,m}] w = \tilde{x}, \\ \sum_{i=1}^m w_i = 1, \\ w_i \geq 0, \quad i = 1, \dots, m. \end{cases} \end{aligned}$$

Về bản chất, ta đang tìm một điểm \tilde{x} là **tổ hợp lồi** của các đỉnh đã biến đổi $\tilde{v}_{j,k}$ sao cho khoảng cách của \tilde{x} đến gốc tọa độ là nhỏ nhất.

Bước 2.3. Xác định mặt phẳng tiếp tuyến của ellipsoid tại điểm tiếp xúc

Sau khi tìm được điểm x^* gần nhất, ta xây dựng mặt phẳng tiếp tuyến với ellipsoid tại điểm đó. Ellipsoid được mô tả bằng biểu thức nghịch đảo:

$$E = \{x \mid (x - d)^\top C^{-1} C^{-\top} (x - d) \leq 1\}.$$

Vector pháp tuyến của ellipsoid tại x^* được xác định bởi gradient:

$$a_j = \nabla_x \left[(x - d)^\top C^{-1} C^{-\top} (x - d) \right]_{x=x^*} = 2C^{-1} C^{-\top} (x^* - d).$$

Vì mặt phẳng tiếp tuyến đi qua x^* , ta có:

$$b_j = a_j^\top x^*.$$

Khi đó, phương trình mặt phẳng tiếp tuyến là:

$$a_j^\top x = b_j,$$

được thêm vào như một **ràng buộc tuyến tính** trong tập khả thi của thuật toán IRIS. Mặt phẳng này đảm bảo ellipsoid mở rộng tối đa nhưng không giao với chướng ngại vật.

Bước 2.4. Lặp lại với tất cả các obstacles trong không gian. Ta sẽ có được một tập các siêu phẳng để tạo thành một vùng lồi đa giác.

3. **Tìm Ellipsoid Nội tiếp Cực đại (SDP):** Giao của các nửa không gian được xác định bởi các siêu phẳng tạo thành một đa diện lồi (polytope). Thuật toán sau đó tìm ellipsoid có thể tích lớn nhất có thể nội tiếp trong đa diện này.

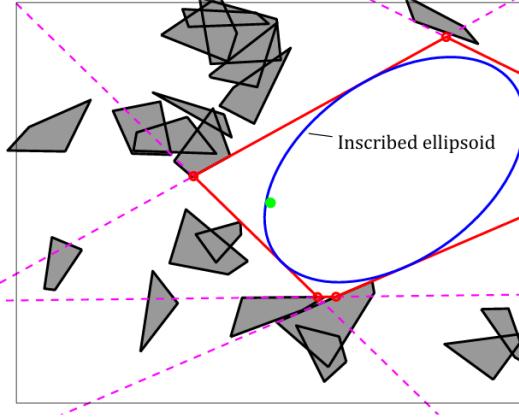


Figure 4.6: Ellipsoid Nội tiếp Cực đại

Bài toán này tìm **ellipsoid có thể tích lớn nhất** nằm gọn trong một đa diện lồi P :

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}. \quad (4.1)$$

Ellipsoid được định nghĩa là \mathcal{E} , với d là tâm và ma trận C xác định hình dạng:

$$\mathcal{E} = \{C\tilde{x} + d \mid \|\tilde{x}\|_2 \leq 1\}. \quad (4.2)$$

Thể tích của ellipsoid tỉ lệ với $\det(C)$.

Để tìm ellipsoid lớn nhất nội tiếp trong P , ta giải bài toán tối ưu lồi sau:

$$\begin{aligned} & \underset{C, d}{\text{maximize}} \quad \log \det C \\ & \text{subject to} \quad \|a_i^\top C\|_2 + a_i^\top d \leq b_i, \quad \forall i = 1, \dots, N, \\ & \quad C \succ 0. \end{aligned} \quad (4.3)$$

Mục tiêu: Cực đại hóa $\log \det C$ để tối đa hóa thể tích ellipsoid.

Ràng buộc: Đảm bảo mọi điểm của ellipsoid đều nằm trong đa diện

P .

4. **Lắp lại:** Ellipsoid mới, lớn hơn này được sử dụng làm điểm khởi đầu cho vòng lắp tiếp theo. Quá trình này hội tụ khi thể tích của ellipsoid ngừng tăng một cách đáng kể, trả về một đa diện lồi lớn và ellipsoid nội tiếp của nó.

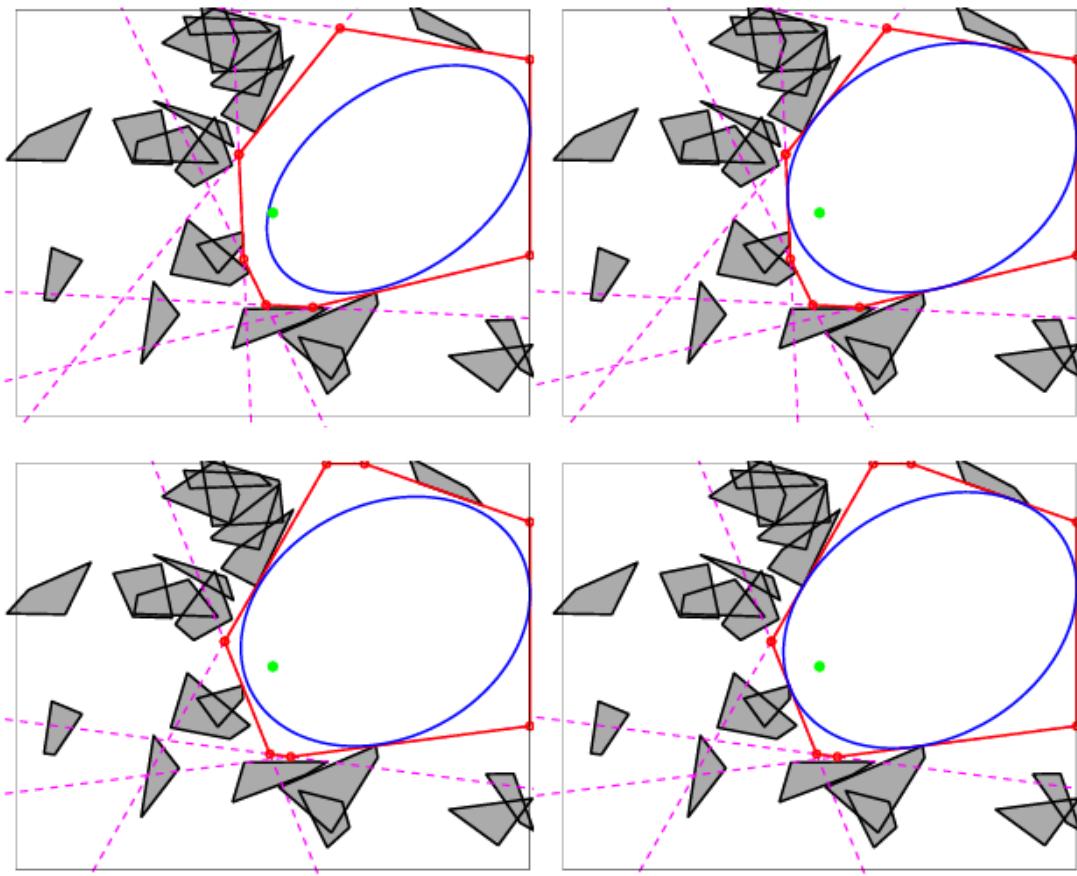


Figure 4.7: Lắp lại

4.2.2.2 Đặc tính của Thuật toán

- Loại thuật toán: IRIS là một phương pháp xấp xỉ để bao phủ toàn bộ không gian tự do. Một lần chạy duy nhất chỉ tạo ra một vùng lồi. Để có được một lớp phủ hoàn chỉnh, cần phải chạy thuật toán nhiều lần với các điểm mầm khác nhau.
- Độ phức tạp thời gian: IRIS thường hội tụ qua 4-8 vòng lặp. Thời gian tính toán của mỗi vòng lặp bằng thời gian tìm các mặt phẳng phân cách cộng với thời gian tính toán ellipses nội tiếp. Trong khi thời gian giải ellipses (SDP) gần như không đổi (constant) nhờ vào việc loại bỏ các siêu phẳng thừa thì thời gian tính các siêu phẳng phân cách lại tăng tuyến tính theo số lượng các obstacles.

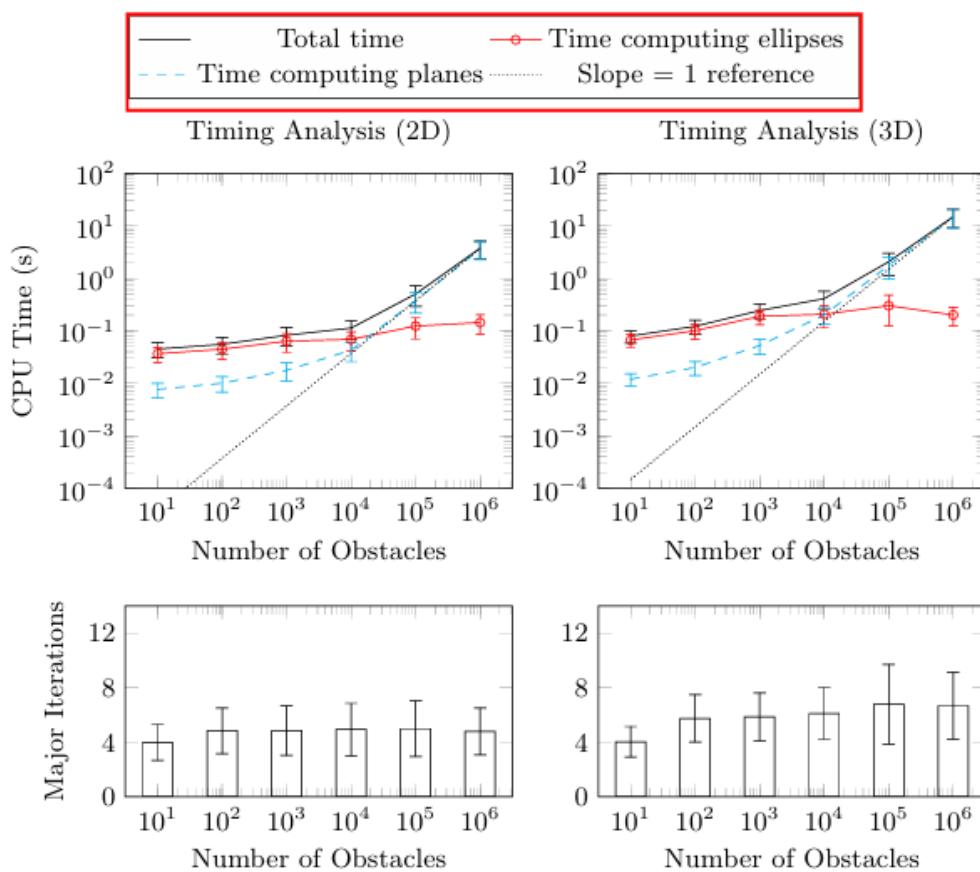


Figure 4.8: Độ phức tạp thời gian

- Xử lý Vật cản không lồi:

- Chướng ngại vật trong không gian work space: Thuật toán IRIS ban đầu giả định các chướng ngại vật là lồi, đây cũng là một nhược điểm lớn nhất của thuật toán này. Để đảm bảo các obstacles là lồi, ta thường phải tiến xử lý bằng cách bao lồi các chướng ngại vật không lồi[11]. Ngoài ra cách tiếp cận tiêu chuẩn cho các chướng ngại vật không lồi là phân hoạch chúng thành một tập hợp các mảnh lồi trước khi chạy IRIS (tức tăng số lượng obstacles của không gian lên). Điều này khả thi vì IRIS có khả năng mở rộng hiệu quả với số lượng lớn chướng ngại vật.
- Chướng ngại vật trong không gian cấu hình (Configuration Space): Đối với các robot có động học phức tạp, nơi các chướng ngại vật trong không gian cấu hình được định nghĩa một cách ẩn ý và không lồi, các phần mở rộng như IRIS-NP và C-IRIS được sử dụng. Đặc biệt, bài báo "Motion Planning around Obstacles with Convex Optimization" sử dụng một triển khai có tên là IrisIn-ConfigurationSpace trong thư viện Drake cho ví dụ về cánh tay robot.[1] Phần mở rộng này sử dụng lập trình phi tuyến (non-linear programming) để xử lý hình học phức tạp của không gian cấu hình.

4.2.3 Gieo mầm Tự động qua Lớp phủ Clique Khả kiến (VCC)

Thuật toán Visibility Clique Cover (VCC) giải quyết trực tiếp một hạn chế chính của IRIS: nhu cầu về các điểm mầm tốt.⁵ Việc lấy mầm ngẫu nhiên là không hiệu quả. VCC tự động hóa quá trình này bằng cách trước tiên tìm hiểu cấu trúc toàn cục của không gian tự do và sau đó đặt các điểm mầm một cách chiến lược vào các khu vực có khả năng mở rộng lớn.

4.2.3.1 Ý tưởng và Chi tiết Thuật toán

Quy trình VCC bao gồm bốn bước chính [10]:

1. Lấy mẫu & Xây dựng Đồ thị Khả kiến (Visibility Graphs): Thuật toán lấy mẫu n điểm ngẫu nhiên trong không gian cấu hình tự do (C_{free}). Các điểm này tạo thành các đỉnh của một đồ thị khả kiến (visibility graph)¹. Một cạnh tồn tại giữa hai đỉnh nếu đoạn thẳng nối hai điểm tương ứng hoàn toàn nằm trong không gian tự do (tức là không va chạm).

Visibility Graph -

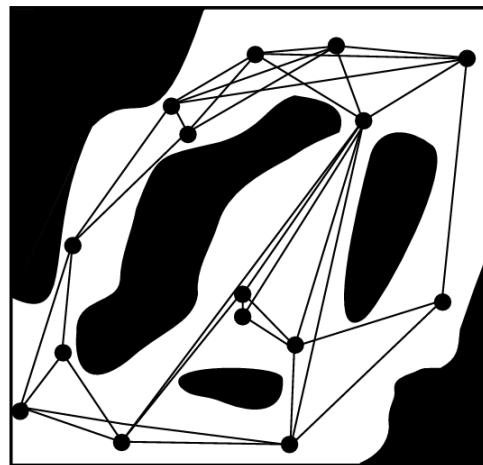


Figure 4.9: Lấy mẫu & Xây dựng Đồ thị Khả kiến (Visibility Graphs)

2. Tìm Lớp phủ Clique: Thuật toán tìm một tập hợp các clique lớn (đồ thị con đầy đủ) trong đồ thị khả kiến. Ý tưởng cốt lõi là một tập hợp các điểm mà tất cả đều "nhìn thấy" nhau có khả năng nằm trong cùng một vùng lồi². Điều này được thực hiện một cách tham lam bằng cách giải lặp đi lặp lại bài toán MAXCLIQUE (tìm clique lớn nhất), được xây dựng dưới dạng một bài toán Integer Linear Programming - ILP.

¹ *Visibility graph* là đồ thị trong đó các đỉnh biểu diễn vị trí (hoặc đỉnh chướng ngại vật), và có cạnh nối giữa hai đỉnh nếu đoạn thẳng nối chúng nằm hoàn toàn trong không gian tự do.

² Nếu hai điểm có thể nhìn thấy nhau, điều đó ngụ ý rằng con đường thẳng nhất giữa chúng là an toàn. Đây là khái niệm cơ bản để hiểu và xác định các vùng lồi trong.

► Clique Cover -

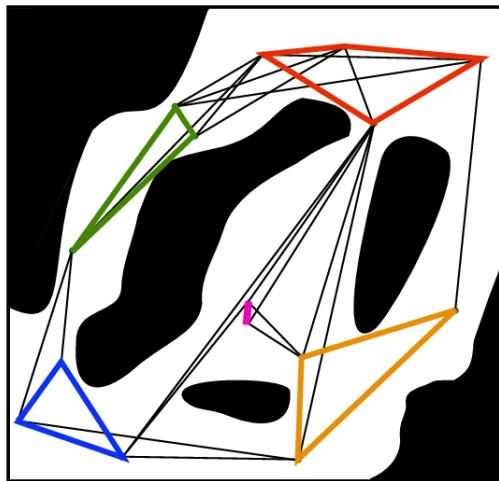


Figure 4.10: Tìm Lớp phủ Clique

3. Khởi tạo Ellipsoid: Đối với mỗi clique được tìm thấy, thuật toán tính toán ellipsoid có thể tích nhỏ nhất bao quanh tất cả các điểm trong clique đó. Ellipsoid này cung cấp một tâm (điểm mầm) và các trục chính (hình dạng ban đầu) cho bước lấp đầy tiếp theo.

► Ellipsoids -

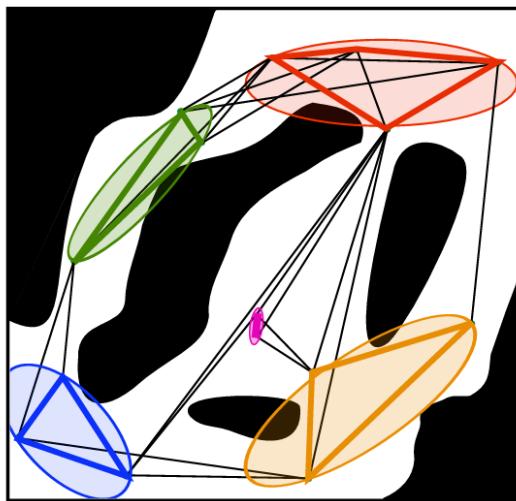


Figure 4.11: Khởi tạo Ellipsoid

4. Lấp đầy thành Đa diện: Tâm và hình dạng của mỗi ellipsoid được sử dụng để khởi tạo một vòng lặp duy nhất của thuật toán lấp đầy tương

tự IRIS. Điều này hiệu quả hơn IRIS tiêu chuẩn vì ellipsoid ban đầu đã được "thông báo" về hình học cục bộ, thường loại bỏ sự cần thiết của nhiều vòng lặp tốn kém.⁵

► Inflat Polytopes -

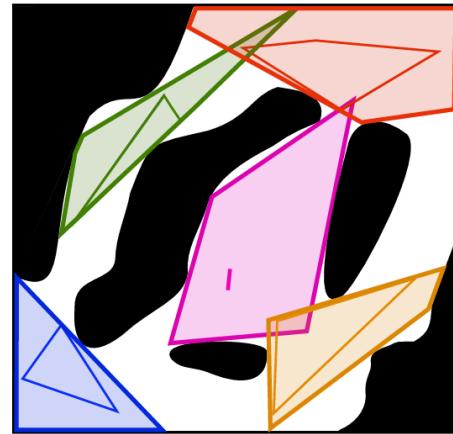


Figure 4.12: Lắp đầy thành Đa diện

5. Lắp lại và Hội tụ: Quá trình này được lắp lại cho đến khi đạt được độ phủ đủ bằng cách lấy mẫu mới từ không gian trống còn lại và lắp lại các bước trước đó.

→ ← Repeated Iterations →

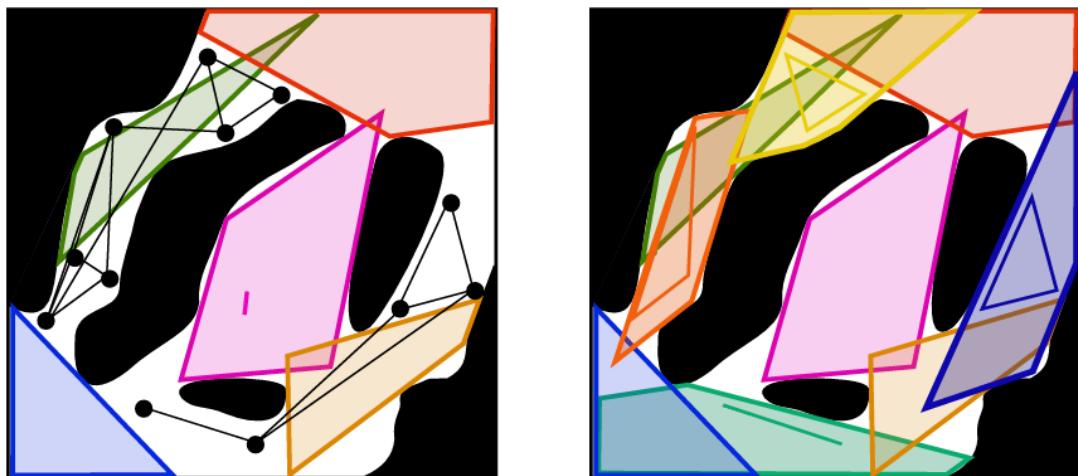


Figure 4.13: Lắp lại

4.2.3.2 Đặc tính và Tính tối ưu

- Loại thuật toán: VCC là một phương pháp xấp xỉ. Chất lượng của lớp phủ được đo bằng tỷ lệ phần trăm thể tích của C_{free} được bao phủ bởi hợp của các đa diện kết quả.⁵
- Mục tiêu Tối ưu: Thuật toán nhằm tìm một lớp phủ lồi xấp xỉ α (α -approximate convex cover) với số lượng vùng tối thiểu. Nó sử dụng bài toán lớp phủ clique tối thiểu (minimum clique cover) như một bài toán tương tự rồi rắc cho bài toán lớp phủ lồi tối thiểu.[10]
- Phương pháp: Đây là một cách tiếp cận lai ghép giữa hình học và ILP.
- Xử lý Không gian không lồi: VCC không phụ thuộc vào hình dạng của chướng ngại vật vì nó dựa vào một bộ kiểm tra va chạm chung để kiểm tra khả kiến và sau đó sử dụng một thuật toán lấp đầy cơ bản (như IRIS-NP) có thể xử lý các không gian cầu hình không lồi.[10]

4.3 Graphs of convex sets

The Graph of Convex Sets (GCS) framework provides a unified methodology for solving hybrid discrete-continuous optimization problems, effectively bridging combinatorial graph search with continuous convex programming. Unlike classical graph theory, GCS generalizes the Shortest Path Problem (SPP) by associating continuous geometric variables directly with the graph topology.

4.3.1 Shortest Path Formulation in Graph of Convex Sets

A GCS is defined as a directed graph $G = (V, E)$ where each vertex $v \in V$ is associated with a continuous decision variable x_v confined within a non-

empty, compact convex set $X_v \subset \mathbb{R}^n$. In the context of motion planning, these sets typically represent pre-computed collision-free regions in the configuration space. Transitions between vertices are governed by edges $e = (u, v) \in E$, which impose auxiliary convex constraints $(x_u, x_v) \in X_e$ and are weighted by a convex cost function $\ell_e(x_u, x_v)$. This function is required to be proper, closed, and convex, explicitly coupling the continuous variables x_u and x_v to model transition metrics such as Euclidean distance or energy expenditure. Note that, despite its name, we do not assume ℓ_e to be a valid metric, and properties like symmetry or the triangle inequality are not required to hold [12].

While the classical network flow formulation effectively solves the SPP on discrete graphs with static edge costs, the GCS framework generalizes this by introducing continuous decision variables coupled with the graph topology. In GCS, the cost of traversing an edge $e = (u, v)$ is no longer a constant scalar c_{uv} , but a convex function $\ell_e(x_u, x_v)$ dependent on the continuous states $x_u \in X_u$ and $x_v \in X_v$ at the incident vertices. The objective of the SPP in GCS is to identify a path p from a source vertex s to a target vertex t that minimizes the aggregate cost of traversing the edges along the path, while simultaneously selecting appropriate continuous variables at each vertex. Formally, this problem can be expressed as:

$$\min_{p, \{x_v\}_{v \in p}} \sum_{(u, v) \in E_p} \ell_{(u, v)}(x_u, x_v) \quad (4.4)$$

$$\text{s.t. } p \in \mathcal{P}, \quad (4.5)$$

$$x_v \in X_v, \quad \forall v \in p, \quad (4.6)$$

$$(x_u, x_v) \in X_e, \quad \forall e = (u, v) \in E_p \quad (4.7)$$

where \mathcal{P} denotes the set of all feasible paths from s to t , and E_p represents the edges included in the path p .

4.3.2 Complexity Analysis

The computational complexity of the SPP in GCS differs fundamentally from that of the classical Shortest Path Problem on discrete graphs.

In classical graph theory, the SPP is generally NP-hard (e.g., the Longest Path problem or variants involving negative cycles). However, it becomes efficiently solvable in polynomial time under two critical conditions:

1. The graph is **acyclic** (allowing for topological sort based solutions).
2. All edge and vertex costs are **non-negative** (solvable via Dijkstra's algorithm).

Proposition 1. *In stark contrast, the Shortest Path Problem in a Graph of Convex Sets remains **NP-hard** even when both of the aforementioned conditions—acyclicity and non-negative costs—are satisfied.*

This hardness is established via a reduction from the **3SAT** problem (Boolean satisfiability with clauses of three literals), which is known to be NP-complete. As demonstrated in Theorem 9.1 of [13], one can construct a layered, acyclic GCS instance with non-negative costs where identifying a valid path is equivalent to finding a satisfying assignment for a 3SAT formula. This implies that merely determining the *feasibility* of an SPP in GCS is NP-hard.

Alternative proofs of NP-hardness have also been provided for cases involving low-dimensional convex sets (1D and 2D), though these instances typically rely on graphs containing cycles [13].

4.3.3 Detailed Formulation of Vertices and Edges

In the specific context of trajectory optimization, the components of the GCS are rigorously defined to enforce both geometric safety and kinematic feasibility.

Each vertex $v \in V$ corresponds to a specific convex, collision-free region Q_v within the Configuration Space (C-space). The continuous decision variable $x_v \in \mathbb{R}^{n_v}$ associated with vertex v represents a local trajectory segment or a state configuration. It is crucial to note that the dimension n_v is local to the vertex, allowing for variable dimensionality across the graph.

The variable x_v is constrained to lie within a compact convex set $\mathcal{X}_v \subset \mathbb{R}^{n_v}$. For trajectory generation using piecewise polynomial parameterizations (e.g., Bézier curves), \mathcal{X}_v is explicitly constructed to ensure the curve lies within the safe region Q_v and satisfies dynamic limits. The defining constraints for \mathcal{X}_v are typically formulated as:

$$r_{i,k} \in Q_i, \quad k = 0, \dots, d, \quad (4.8)$$

$$\dot{h}_{i,k} \geq \dot{h}_{\min} = 10^{-6}, \quad k = 0, \dots, d-1, \quad (4.9)$$

$$\dot{r}_{i,k} \in \dot{h}_{i,k} D, \quad k = 0, \dots, d-1, \quad (4.10)$$

$$h_{i,0} \geq 0, \quad h_{i,d} \leq T_{\max}. \quad (4.11)$$

Here, (4.8) enforces geometric containment of the spatial control points $r_{i,k}$ within the safe region. Equation (4.9) ensures strict monotonicity of the time-scaling function to preserve causality. Equation (4.10) couples spatial and temporal derivatives to satisfy velocity bounds defined by the set D , and (4.11) imposes limits on the trajectory duration.

An edge $e = (u, v) \in E$ defines the transition between two vertices. This transition is governed by a *Coupling Constraint Set* $\mathcal{X}_e \subseteq \mathbb{R}^{n_u+n_v}$. This set imposes constraints on the concatenated vector (x_u, x_v) , residing in the Cartesian product of the individual variable spaces. Mathematically, \mathcal{X}_e defines which pairs of configurations (x_u, x_v) are compatible, ensuring continuity (e.g., C^0, C^1 , or higher-order continuity) between trajectory segments.

Associated with each edge is a *Coupling Cost Function* $f_e : \mathbb{R}^{n_u+n_v} \rightarrow \mathbb{R}$. This is a scalar-valued convex function that assigns a cost based on the simultaneous selection of x_u and x_v , modeling transition metrics such as path length or energy consumption.

4.3.4 Cost Function Specifications

The choice of the edge cost function $\ell_e(x_u, x_v)$ dictates the physical interpretation of the optimal path. We examine two primary classes of cost functions relevant to this study: geometric path length and dynamic trajectory costs.

For purely geometric problems, such as finding the shortest Euclidean path through regions, the cost is typically defined as the L_2 -norm or its square. The squared Euclidean distance,

$$\ell_e(x_u, x_v) = \|x_v - x_u\|_2^2,$$

is a quadratic function that fits naturally into Quadratic Programming (QP) solvers. Alternatively, the standard L_2 -norm

$$\ell_e(x_u, x_v) = \|x_v - x_u\|_2$$

can be modeled using Second-Order Cone Programming (SOCP), preserving convexity.

In the context of motion planning using Bézier curves, the continuous variable x_v encapsulates both the control points and the time duration of the trajectory segment. Here, the cost function represents a trade-off between execution time and control effort (smoothness). A common formulation minimizes the path energy (integral of squared acceleration) normalized by time. For a transition between vertices u and v with duration T_e , the cost

takes the form

$$\ell_e(x_u, x_v) = \frac{1}{T_e} \int_0^{T_e} \|\ddot{q}(t)\|^2 dt$$

When discretized or expressed via Bézier coefficients, this function essentially behaves as a perspective function of the form quadratic-over-linear (x^2/y), which is convex regarding both the control points and the duration T_e .

4.3.5 Mixed-Integer Convex Programming Formulation via Perspective Relaxation

To transform the intractable MINLP into a solvable Mixed-Integer Convex Program (MICP), we employ a tight convex relaxation technique known as perspective relaxation. This approach isolates the non-convexity by lifting the continuous variables into a higher-dimensional space and replacing the bilinear constraints with linear flow conservation constraints on these lifted variables.

For each edge $e = (u, v)$, instead of using global variables x_u and x_v , we introduce local proxy variables $z_{e,u}$ and $z_{e,v}$ with $z_{e,u}, z_{e,v} \in \mathbb{R}^n$. These variables represent the contribution of the state at vertices u and v specifically to the traversal of edge e . If the edge is not active ($y_e = 0$), these contributions are forced to zero.

$$z_{e,u} \in y_e X_u \quad \text{and} \quad z_{e,v} \in y_e X_v$$

This implies:

$$\begin{cases} z_{e,u} \in X_u & \text{if } y_e = 1 \\ z_{e,u} = 0 & \text{if } y_e = 0 \end{cases}$$

This logic is formalized using the perspective function $\tilde{\ell}_e((z_{e,u}, z_{e,v}), y_e)$, which is the perspective of the original cost function ℓ_e . A key property of

convex analysis is that if ℓ_e is convex, its perspective $\tilde{\ell}_e$ is jointly convex in all arguments, enabling the use of efficient convex solvers.

$$\tilde{\ell}_e(z_{e,u}, z_{e,v}, y_e) = \begin{cases} y_e \ell_e \left(\frac{z_{e,u}}{y_e}, \frac{z_{e,v}}{y_e} \right) & \text{if } y_e > 0 \\ 0 & \text{if } y_e = 0 \text{ and } z_{e,u} = z_{e,v} = 0 \\ +\infty & \text{otherwise} \end{cases}$$

Crucially, rather than enforcing explicit equality constraints like $z_{e,u} = y_e x_u$ which are non-convex, we extends the standard flow conservation law (2.6b) to the continuous variables. It mandates that for any node u (excluding source and target), the sum of continuous states entering the node via incoming edges must equal the sum of continuous states leaving the node via outgoing edges.

The resulting MICP formulation for the Shortest Path Problem in GCS is defined as follows:

$$\text{minimize}_{e=(u,v) \in \mathcal{E}} \tilde{\ell}_e((z_{e,u}, z_{e,v}), y_e) \quad (4.12a)$$

$$\text{subject to} \quad \sum_{e \in \mathcal{E}^+(u)} y_e - \sum_{e \in \mathcal{E}^-(u)} y_e = \delta_u, \quad \forall u \in \mathcal{V} \quad (4.12b)$$

$$\sum_{e \in \mathcal{E}^+(u)} z_{e,u} - \sum_{e \in \mathcal{E}^-(u)} z_{e,v} = \begin{cases} x_s & \text{if } u = s \\ -x_t & \text{if } u = t \\ 0 & \text{otherwise} \end{cases}, \quad \forall u \in \mathcal{V} \quad (4.12c)$$

$$(z_{e,u}, z_{e,v}) \in y_e \mathcal{X}_e, \quad \forall e \in \mathcal{E} \quad (4.12d)$$

$$z_{e,u} \in y_e X_u, \quad z_{e,v} \in y_e X_v, \quad \forall e = (u, v) \in \mathcal{E} \quad (4.12e)$$

$$y_e \in \{0, 1\}, \quad \forall e \in \mathcal{E} \quad (4.12f)$$

In this model, equation (4.12b) ensures topological connectivity (where δ_u is 1 for source, -1 for target, and 0 otherwise). Equation (4.12c) ensuring that the continuous trajectory is seamless across the graph. Equations (4.12d) and (4.12e) enforce the geometric containment within scaled convex sets, ensuring that if $y_e = 0$, the associated continuous variables are constrained to the origin, incurring zero cost. This formulation provides a computationally efficient tight relaxation , allowing the solver to find near-global optimal trajectories that satisfy both the discrete graph structure and the complex continuous constraints of motion planning.

4.4 Proposed solution

4.5 Experiment setup

4.5.1 Implementation details

4.6 Results and analysis

Chapter 5

Conclusion

5.1 Summary

5.2 Future Work

Future research will focus on...

- ...
- ...

References

- [1] R. Deits and R. Tedrake, “Motion planning around obstacles with convex optimization,” *arXiv preprint arXiv:2205.04422*, 2022.
- [2] Boston Dynamics, *Leaps, bounds, and backflips*, <https://bostondynamics.com/blog/leaps-bounds-and-backflips/>, Accessed: November 29, 2025, 2021.
- [3] Amazon Robotics, *Amazon.com announces first quarter results*, <https://ir.aboutamazon.com/news-release/news-release-details/2023/Amazon.com-Announces-First-Quarter-Results/default.aspx>, Accessed: November 29, 2025, 2023.
- [4] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, “Fast direct multiple shooting algorithms for optimal robot control,” in *Fast Motions in Biomechanics and Robotics*, Springer, 2006, pp. 65–93.
- [5] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Iowa State University, Tech. Rep. TR 98-11, 1998.
- [6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [7] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004, ISBN: 0521833787.

- [8] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993, ISBN: 0-13-617549-X.
- [9] J. M. Lien and N. M. Amato, “Approximate convex decomposition of polygons,” *Computational Geometry: Theory and Applications*, 2006.
- [10] R. Deits and R. Tedrake, “Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs,” *arXiv preprint arXiv:2310.02875*, 2023.
- [11] R. Deits and R. Tedrake, “Computing large convex regions of obstacle-free space through semidefinite programming,” *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.
- [12] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake, “Shortest paths in graphs of convex sets,” *arXiv preprint arXiv:2101.11565*, 2023.
- [13] T. Marcucci, R. Tedrake, and P. A. Parrilo, “Graphs of convex sets with applications to optimal control and motion planning,” *MIT Department of Electrical Engineering and Computer Science, PhD Thesis*, 2024.

Appendix A

• • •

A.1 ...

A.2 ...