UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

O Problema da Visita de Polígonos

Gabriel Freire Ushijima

São Paulo, SP
3 de novembro de 2025

1 Introdução

Este relatório descreve a implementação e os resultados obtidos na resolução do problema da visita de polígonos, usando como base o paper de Mitchell [1] que descreve algoritmos para o caso sem e com restrições. Buscamos apresentar uma abordagem mais prática e detalhada para o problema, sem um foco tão grande na análise teórica.

2 O Problema de Visita de Polígonos Irrestrito

Vamos tomar como base a implementação do algoritmo de Mitchell para o problema irrestrito que fizemos em *Python*. O código pode ser encontrado no arquivo `TouringPolygons/problem1.py`.

2.1 Definições e Notação

Considere o seguinte problema: dados dois pontos $s, t \in \mathbb{R}^2$ e uma sequência de polígonos convexos disjuntos $P_1, P_2, \ldots, P_k$, encontrar o caminho de menor comprimento que se inicia em s, termina em t e toca cada polígono P_i em pelo menos um ponto, podendo atravessá-los.

A figura ao lado ilustra um exemplo de entrada e a solução ótima para o problema para um caso com 3 polígonos. Temos s como o ponto verde, t como o ponto vermelho e os polígonos P_1, P_2 e P_3 como o triângulo azul, o trapézio laranja e o pentágono verde, respectivamente. O caminho mínimo é representado pela linha roxa.

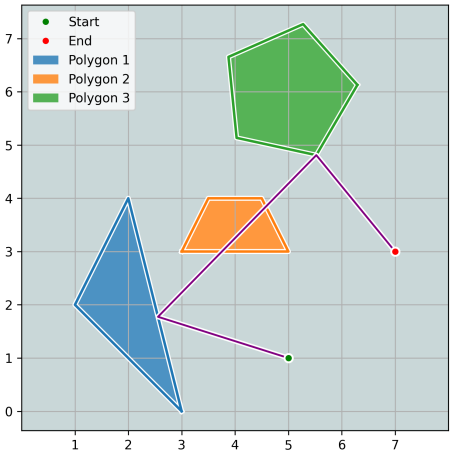

Figura 1: Caminho mínimo para um caso de 3 polígonos.

Retomando o paper, definimos um i-path até p como um caminho mínimo que começa em s, encosta em cada um dos polígonos $P_1, P_2, \ldots, P_i$ e termina em p. Note que nosso objetivo é encontrar um k-path até t. Enquanto não vamos entrar em detalhes, todo i-path é único.

A função central desse algoritmo será a função Query(p, i), que recebe um ponto p e um índice i e retorna o penúltimo ponto q do i-path até p (ou seja, o ponto imediatamente anterior a p nesse caminho). Note que a resposta do problema pode ser obtida chamando Query(t, k).

Primeiramente, vamos descrever procedimentos auxiliares que serão úteis para a implementação da função Query. Finalmente, vamos descrever como respoder consultas usando esses procedimentos. Por enquanto, vamos assumir que sabemos como respoder as consultas.

2.2 Representação dos Objetos

Antes de descrever os procedimentos auxiliares, é importante definir como representamos os objetos geométricos envolvidos no problema. Utilizamos as seguintes representações:

- **Pontos** e **Vetores**: Representados por uma classe `Vector2` que guarda um par de números reais (x, y).
- **Polígonos**: Representados por uma classe `Polygon2` que guarda uma lista ordenada de seus vértices (representados por `Vector2`) no sentido anti-horário. Um vértice arbitrário é escolhido para ser indexado como o primeiro.
- **Caminho Final**: Representado como uma lista de pontos (representados por `Vector2`) na ordem em que aparecem no caminho.
- **Regiões de Vértice**: Cada região é representadas por uma lista de pares de `Vector2`

2.3 Particionando o Plano

O primeiro passo do algoritmo é, para cada polígono P_i, criar uma partição S_i do plano. Essa partição tem 3 tipos de regiões, Regiões de Vértice, Regiões de Aresta e Regiões de Atravessa. Essa partição é usada para responder consultas, uma vez que o comportamento da função Query(p, i) depende de qual região p pertence.

2.3.1 Representado as Partições

Primeiramente, note que as regiões de vértice são todas diversos 'cones' associados à vértices do polígono P_i, mais formalmente, cada região de vértices é a região do plano delimitada por duas semi-retas que partem de um vértice v de P_i. Por esse motivo, pensamos em cada região de vértice como uma tripla (v, d_1, d_2) respresentando o vértice, a direção da primeira semi-reta e a direção da segunda semi-reta. Também é importante ressaltar que a ordem das retas importa, assim consideramos que a região é o conjunto atingido por um movimento anti-horário a partir de d_1 até d_2.

Ademais, as regiões de aresta são todas diversas 'faixas' associadas às arestas do polígono P_i, mais formalmente, cada região de aresta é a região do plano delimitada por duas semi-retas que partem dos extremos de uma aresta e de P_i e a prórpia aresta e. Por esse motivo, pensamos em cada região de aresta como uma quádrupla (u, v, d_1, d_2) respresentando os dois extremos u e v da aresta, a direção da primeira semi-reta e a direção da segunda semi-reta. Aqui a ordem também importa, assim consideramos que a região é o conjunto atingido por um movimento anti-horário a partir de u e d_1 até v e d_2.

Finalmente, podemos dizer que as regiões de atravessa são todas as outras regiões do plano, ou seja, o complemento das regiões de vértice e aresta. Note que há exatamente uma região de atravessa.

Do ponto de vista da implementação, como a regiões de aresta são compostas por dois vértices do polígono e duas direções já definidas nas regiões de vértice, não precisamos armazenar as regiões de aresta explicitamente, uma vez que podemos recuperá-las a partir das regiões de vértice. Assim, para representar a partição S_i, basta armazenar todas as regiões de vértice de P_i.

Para representar as regiões de vértice,

2.3.2 Construindo as Partições

Primeiramente, a partir da figura, note que nem todo vértice e nem toda aresta de P_i geram uma região na partição S_i. Dizemos que uma aresta que gera uma região é uma aresta visível, assim, nosso primeiro passo é encontrar todas as arestas visíveis de P_i.

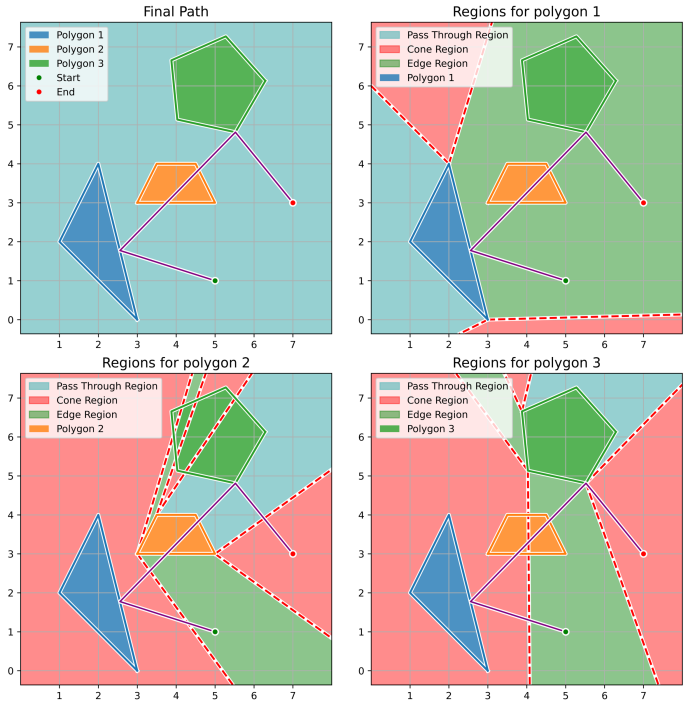

Figura 2: Partição do plano para cada poígono do exemplo anterior.

Para deteminar se uma aresta e é visível, simplesmente calculamos $p =$ Query(m, i) onde m é o ponto médio de e e verificamos se $\overline{pm}$ atravessa P_i. Se o segmento atravessa P_i, então e não é visível, caso contrário, e é visível.

Uma vez que temos todas as arestas visíveis, podemos construir as regiões de vértice e de aresta. Para cada vértice v de P_i, calculamos $p =$ Query(v, i) e consideramos a direção de $d = v - p$. Se a aresta anterior a v (no sentido anti-horário) é visível, então d_1 será a reflexão de d em relação à aresta anterior, caso contrário, d_1 será a mesma direção de d. Similarmente, se a aresta posterior a v é visível, então d_2 será a reflexão de d em relação à aresta posterior, caso contrário, d_2 será a mesma direção de d. Assim, criamos a região de vértice (v, d_1, d_2).

Dessa forma, sabemos todas as regiões de vértice. Agora para cada

aresta visível $e = (u, v)$ de P_i, temos que d_1 será a segunda direção da região de vértice associada a u e d_2 será a primeira direção da região de vértice associada a v. Assim, criamos a região de aresta (u, v, d_1, d_2).

Finalmente, não é necessário criar a região de atravessa, uma vez que sabemos que um ponto pertence a ela se ele não pertence a nenhuma outra região.

2.4 Respondendo Consultas

Agora que sabemos como construir as partições S_i, podemos descrever como usá-las para responder consultas Query(p, i). O procedimento utilizado é recursivo, assim, primeiro definimos o caso base, que é Query$(p, 0) = s$, uma vez que um 0-path não precisa tocar em nenhum polígono, assim, o menor caminho é uma linha reta de s até p.

Para $i > 0$, primeiramente determinamos a região R de S_i que contém p. Descrevemos acima como representar cada região e a partir dela é fácil determinar à qual um ponto p pertence. Também é importante ressaltar que p sempre pertence a exatamente uma região. Agora vamos analisar os 3 casos possíveis para R:

- Região de Vértice: Seja $R = (v, d_1, d_2)$. Nesse caso, o i-path até p é deve passar pelo vértice v, tocando o polígono P_i. Assim, temos que Query$(p, i) = v$.
- Região de Aresta: Seja $R = (u, v, d_1, d_2)$. Nesse caso, o i-path até p deve passar por algum ponto q da aresta $e = (u, v)$, tocando o polígono P_i. Para calcular q, primeiro determinamos $q' =$ Query$(p', i - 1)$, onde p' é a reflexão de p em relação à aresta e. Agora, dizemos que q é a interseção entre $\overline{q'p'}$ e a aresta e. Finalmente, respondemos Query$(p, i) = q$.
- Região de Atravessa: Seja R a região de atravessa. Nesse caso, o i-path até p automaticamente atravessa o polígono P_i em algum ponto. Portanto, podemos simplesmente responder Query$(p, i) =$ Query$(p, i - 1)$.

2.5 Calculando o Caminho Mínimo

Uma vez que sabemos como responder consultas, calcular o caminho final é muito simples. Basta calcular $q_k =$ Query(t, k), então $q_{k-1} =$ Query$(q_k, k - 1)$ e assim por diante, até $q_0 = s$. Assim, o caminho mínimo é simplesmente a sequência $s, q_1, \ldots, q_k, t$.

3 O Problema de Visita de Polígonos Geral

Vamos tomar como base a implementação do algoritmo de Mitchell para o problema restrito que fizemos em *Python*. O código pode ser encontrado no arquivo `TouringPolygons/problem2.py`.

3.1 Definições e Notação

O problema segue de forma similar ao anterior, mas agora também recebemos como entrada 'cercas' $F_0, \ldots, F_k$ tais que para todo $0 \leq i \leq k$ vale que o polígono P_i e P_{i+1} estão contidos em F_i, para tal, consideramos $P_0 = \{s\}$ e $P_{k+1} = \{t\}$. Nosso objetivo é encontrar o caminho de menor comprimento que se inicia em s, termina em t, toca cada polígono P_i em pelo menos um ponto e nunca sai da cerca F_i no seu caminho entre P_i e P_{i+1}.

Dizemos que um caminho π de a até b respeita as cercas $F_i, \ldots, F_j$ se π toca todos os polígonos $P_{i+1}, \ldots, P_j$ e para cada $i \leq l < j$, o trecho de π entre P_l e P_{l+1} está contido em F_l. Ademais, definimos um i-path até p como um caminho mínimo de s até p que respeita as cercas $F_0, \ldots, F_i$. Note que nosso objetivo é encontrar um k-path até t.

A função central desse algoritmo continua sendo Query, no entanto, dessa vez vamos adicionar um novo parâmetro, assim, a função Query(p, i, j) recebe um ponto p e dois índices i e j e retorna o penúltimo ponto q do menor caminho até p que parte de s, toca todos os polígonos $P_1, \ldots, P_i$ e respeita as cercas $F_0, \ldots, F_j$.

Primeiramente, é essencial que $i \leq j$, ademais, se $i = j$ então Query(p, i, j) é simplesmente o penúltimo ponto do i-path até p. Adicionamos o parâmetro j para o caso em que queremos um

i-path, mas p está fora da cerca F_i um caso que aparece naturalmente na recursão do algoritmo. Por enquanto, vamos assumir que sabemos como respoder as consultas.

3.2 Caminhos Restritos

Outra função extremamente importante para a implementação desse algoritmo é a função Fenced(p_1, p_2, i, j) que retorna o menor caminho de p_1 até p_2 que respeita as cercas $F_i, \ldots, F_j$. Note que se $i = j$, então Fenced(p_1, p_2, i, j) é simplesmente o segmento $\overline{p_1 p_2}$ se ele estiver contido em F_i e não existe caso contrário. Assim, vamos assumir que $i < j$.

Referências

[1] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell, *Touring a sequence of polygons*, STOC '03 (2003), 473 –482.