UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

MAC0215: Relatório Final
O Problema da Visita de Polígonos

Gabriel Freire Ushijima

São Paulo, SP
7 de novembro de 2025

Sumário

1 Introdução

Esse projeto de pesquisa teve como objetivo explorar o Problema da Visita de Polígonos (Touring Polygons Problem - TPP), um problema de otimização geométrica que envolve encontrar o caminho mais curto que visita uma sequência de polígonos no plano. O projeto foi desenvolvido ao longo do semestre como parte da disciplina MAC0215 - Atividade Curricular em Pesquisa (2025), sendo supervisionado pelo Prof. Dr. Ernesto G. Birgin.

Nesse relatório, vamos apresentar o problema que estamos resolvendo, discutir a base bibliográfica que fundamenta nosso trabalho, detalhar as implementações realizadas, analisar os resultados obtidos e propor possíveis melhorias e trabalhos futuros. Também vamos dedicar uma seção para discutir como o tempo foi alocado ao longo do projeto, justificando o investimento de ao menos 100 horas de trabalho.

Todo o código desenvolvido para esse projeto está disponível no repositório do GitHub, onde também é possível encontrar instruções para executar os algoritmos implementados, assim como acompanhar o desenvolvimento gradual do projeto, servindo como evidência do trabalho realizado.

2 Base Bibliográfica

Esse problema não é original e foi apresentado pela primeira vez por Dror et al. (2003) [1]. No entanto, desde então não houve muitos avanços significativos na resolução do problema, e a maioria das soluções propostas ainda se baseia nas ideias apresentadas nesse artigo inicial.

Enquanto a tarefa de implementar os algoritmos propostos no artigo de Dror et al. (2003) [1] pode parecer direta, na prática, a implementação desses algoritmos apresenta desafios significativos, uma vez que o artigo em si é bastante denso e complexo, focando mais na análise teórica e demonstração matemática de corretudo do que na implementação prática, tanto que não incluem pseudocódigo, visualizações ou detalhes de implementação.

Por esse motivo, esse projeto propôs desbravar esse problema e artigo, buscando entender as ideias apresentadas e desenvolver implementações práticas dos algoritmos propostos. Dessa forma, podemos reapresentar soluções concretas para o problema, contribuindo para a literatura existente e abrindo caminho para futuras pesquisas e melhorias na área.

Além disso, também propomos variações do problema original, não cobertas na literatura atual, explorando diferentes restrições e características dos polígonos envolvidos, o que nos permite ampliar o escopo do estudo e contribuir com novas perspectivas para o campo.

3 O Problema

O Problema da Visita de Polígonos (TPP) consiste em encontrar o caminho mais curto que visita um conjunto de polígonos no plano. Esse problema é um caso específico do problema do caixeiro viajante com vizinhanças (TSPN) [2], onde o objetivo é visitar regiões genéricas no plano, que é por sua vez uma generalização do problema do caixeiro viajante (TSP) [3], onde o objetivo é visitar pontos específicos.

O enunciado formal do TPP é o seguinte: dado um ponto inicial s, um ponto final t e uma sequência de polígonos $P_1, P_2, \ldots, P_k$ no plano, encontrar o menor caminho que começa em s, termina em t e visita cada polígono P_i pelo menos uma vez, além disso os polígono devem ser tocados na ordem que são dados. Consideramos que visitar um polígono significa que o caminho pode atravessar ou simplesmente tocar na borda de cada polígono.

Para esse projeto, consideramos três variações do problema, o TPP Irrestrito, onde lidamos com polígonos convexos e disjuntos, o TPP Restrito onde os polígonos ainda devem ser convexos, mas pode ter interseção e consideramos a inclusão de uma "cerca"que restringe nosso movimento e o TPP com polígonos não convexos, onde retornamos ao TPP Irrestrito, mas consideramos que os polígonos pode ser não convexos. Vamos discutir cada um deles nas próximas seções.

3.1 TPP Irrestrito

Essa é a primeira variação do problema que abordamos, vamos considerar que recebemos um ponto inicial s, um ponto final t e uma sequência de polígonos $P_1, P_2, \ldots, P_k$ e buscamos um caminho mínimo que enconsta em cada polígono no plano, como no enunciado geral do problema. Além dessas suposições, vamos considerar que os polígonos são **convexos** e são **disjuntos**.

A figura ao lado 1 ilustra um exemplo de entrada e solução para o problema. O caminho mínimo é representado pela linha roxa, que começa no ponto s (ponto verde), termina no ponto t (ponto vermelho) e toca o triângulo, trapézio e então pentágono.

O algoritmo que implementamos é descrito no artigo [1] como tendo complexidade $O(nk \log(n/k))$, onde n é o número total de vértices dos polígonos e k é o número de polígonos. Os autores não descreveram as estruturas de dados nem detalhes que facilitassem a implementação, então tivemos que fazer diversas escolhas de implementação para conseguir a complexidade prometida.

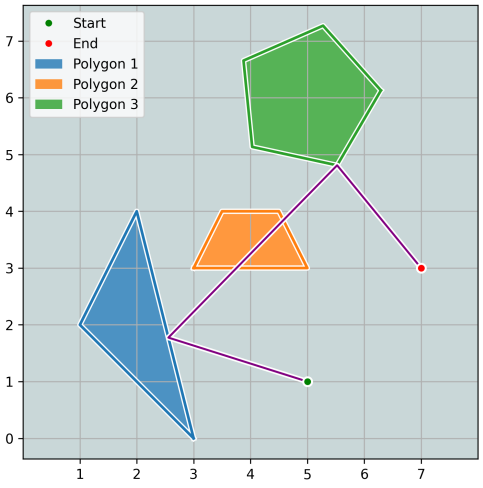

Figura 1: Caminho mínimo para um caso de 3 polígonos.

A solução em si se baseia na ideia de, para cada polígono P_i, particionar o plano em três tipos de região, permitindo que façamos consultas do tipo $q = \text{Query}(p, i)$, onde q é o último passo do menor caminho que parte de s, toca cada polígono $P_1, \ldots, P_i$ e chega em p. Formulando o problema dessa maneira, podemos observar que, se conseguirmos criar todas as partições para cada polígono, podemos resolver o problema facilmente, definindo $q_k = \text{Query}(t, k)$, então $q_{k-1} = \text{Query}(q_k, k-1)$, e assim por diante, até chegarmos em $q_0 = s$. Dessa forma, conseguimos reconstruir o caminho mínimo de trás para frente.

Na figura abaixo 2 é possível ver um exemplo dessas partições, para polígono criamos uma partição diferente, usando as partições anteriores. Dessa forma a primeira partição (a do triângulo) usa apenas o ponto s, a segunda partição (a do trapézio) usa a partição do triângulo e s, e a terceira partição (a do pentágono) usa a partição do triângulo, trapézio e s.

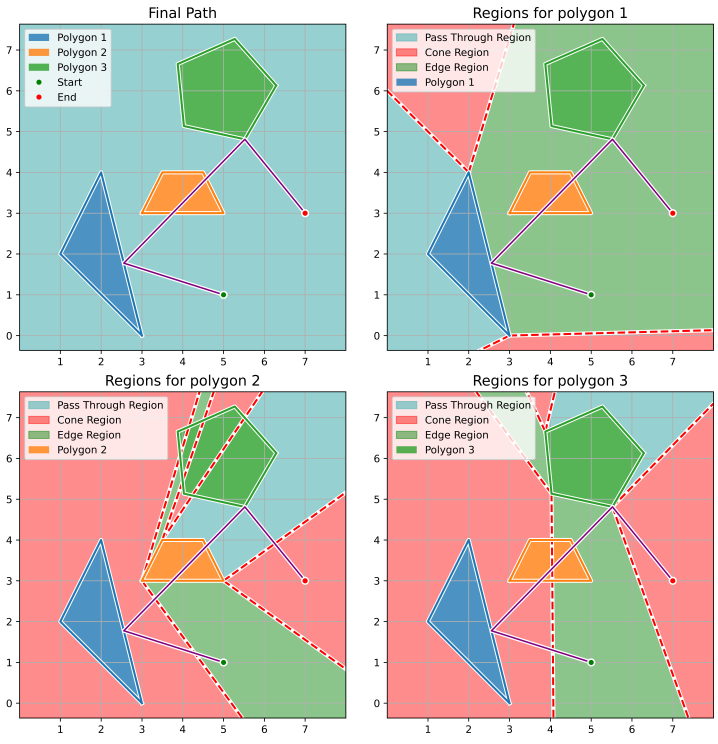

Figura 2: Caminho mínimo para um caso de 3 polígonos.

Essas regiões são de três tipos, chamdas de regiões de vértice (em vermelho), aresta (em verde) e sombra (em ciano). As regiões de vértice são associadas à um vértice específico do polígono P_i, e consistem em todos os pontos p tais que o caminho mínimo toca em P_i nesse vértice e então chega em p. As regiões de aresta são associadas a uma aresta específica do polígono P_i, e consistem em todos os pontos p tais que o caminho mínimo toca em P_i nessa aresta e então chega em p. Finalmente, as regiões de sombra são associadas ao polígono em si e consistem em todos os pontos p tais que o caminho mínimo toca em atravessa P_i para chegar em p.

A seguir vamos construir o caminho mínimo para o exemplo da figura 1, usando as partições da figura 2. Começamos com $q_3 = \text{Query}(t, 3)$, olhando na partição do pentágono, observamos que t está na região de vértice, assim, sabemos que q_3 deve ser esse vértice de P_3. Em seguida, calculamos $q_2 = \text{Query}(q_3, 2)$, olhando na partição do trapézio, observamos que q_3 está na região de sombra, assim, podemos essencialmente ignorar P_2, pois qualquer ponto mínimo até q_3 que toque P_2 deve necessariamente atravessá-lo. Finalmente, calculamos $q_1 = \text{Query}(q_3, 1)$, olhando na partição do triângulo, observamos que q_3 está na região de aresta, assim, sabemos que q_1 deve ser o ponto na aresta de P_1 que minimiza a distância de s até q_2. Com isso, conseguimos reconstruir o caminho mínimo completo.

Tendo essa ideia geral em mente, resta apenas como desafio criar essas partições e localizar pontos nelas de forma eficiente. Essencialmente, para criar a partição de P_i, devemos calcular um caminho mínimo

que toca cada polígono $P_1, \ldots, P_{i-1}$ até cada vértice de P_i e apartir da maneira que esse caminho chega nesses vértices, podemos determinar as regiões de vértice. Uma vez que temos as regiões de vértice, notamos que as regiões de aresta e sombra são as regiões restantes no plano, dessa forma, é apenas necessário saber diferenciar essas regiões. Para isso, podemos observar que se um caminho mínimo que chega em um ponto interno de uma aresta de P_i atravessa P_i, essa aresta deve pertencer à região de sombra, caso contrário, pertence à região de aresta. Com isso, conseguimos construir todas as partições necessárias.

É claro que essa explicação é bastante simplificada, e muitos detalhes de implementação foram omitidos para manter o foco na ideia geral. O artigo original [1] apresenta todas os lemas e teoremas necessários para garantir a corretude do algoritmo e a complexidade prometida, enquanto os detalhes de implementação e o código em si podem ser encontrados por completo no repositório do GitHub associado a esse projeto.