O Problema de Visita de Polígonos*

Gabriel F. Ushijima† Ernesto G. Birgin†

7 de janeiro de 2026

Resumo

Esse artigo apresenta diversas implementações de algoritmos para o problema de visita de polígonos (Touring Polygons Problem - TPP), que consiste em encontrar o caminho mais curto que visita um conjunto de polígonos no plano. O problema em sua forma geral é NP-difícil, assim, primeiro consideramos uma versão simplificada onde os polígonos são convexos e não se sobrepõem, e apresentamos um algoritmo exato baseado em mapas de último passo. Em seguida, discutimos heurísticas para o caso geral, utilizando técnicas de programação inteira mista. Finalmente, realizamos experimentos computacionais para avaliar o desempenho dos algoritmos propostos em instâncias de diferentes tamanhos e características, comparando com solvers comerciais. Os resultados indicam que as abordagens desenvolvidas são eficazes para resolver o TPP em diversas situações práticas.

Palavras-chave: Problema de visita de polígonos, otimização geométrica, complexidade, experimentos numéricos.

1 Introdução

O Problema de Visita de Polígonos (Touring Polygons Problem - TPP) é um problema de otimização geométrica que consiste em encontrar o caminho mais curto que visita um conjunto de polígonos no plano. Esse tipo de problema tem aplicações em diversas áreas, principalmente no contexto de roteirização e planejamento de trajetórias para veículos autônomos, onde é necessário garantir que certas regiões sejam visitadas de forma eficiente, como a inspeção automatizada de armazéns [1], [2]. Dentre as diversas variações do problema, vamos considerar inicialmente uma versão simplificada onde os polígonos são convexos e não se sobrepões, apresentando três abordagens distintas para sua resolução. A primeira abordagem é uma implementação intuitiva das ideias descritas por Dror et al. (2003) [3] baseada em mapas de último passo. A segunda abordagem melhora a eficiência da primeira ao uma estratégia de busca binária para localizar os pontos nos mapas de último passo. A terceira abordagem utiliza uma estratégia de memoização para evitar cálculos desnecessários.

Em seguida, discutimos heurísticas para o caso geral do TPP, onde os polígonos podem ser côncavos. Nos baseamos principalmente na ideia de particionar os polígonos côncavos em subconjuntos convexos e aplicar as soluções desenvolvidas para o caso convexo. Além disso, exploramos técnicas de programação inteira mista para modelar o problema e utilizar solvers comerciais para encontrar soluções aproximadas, como o Gurobi, então comparamos os resultados obtidos com nossas implementações.

*Esse trabalho foi parcialmente apoiado pela agência FAPESP (processo 2025/13861-1).

†Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil (e-mail: gabriel_ushijima@ime.usp.br, egbirgin@ime.usp.br).

2 O Problema de Visita de Polígonos Convexos

Primeiramente, consideramos o TPP no caso de polígonos convexos e não sobrepostos. Seja $s, t \in \mathbb{R}^2$ pontos no plano e $P_1, \ldots, P_k$ uma sequência de polígonos convexos disjuntos, dados por uma sequência de vértices ordenados em sentido anti-horário. O objetivo é encontrar um caminho de comprimento mínimo que começa em s, termina em t e visita cada polígono P_i em ordem pelo menos uma vez. Dizemos que o caminho visita os polígnos na ordem $P_1, P_2, \ldots, P_k$ se existem pontos no caminho $p_1, p_2, \ldots, p_k$ tais que $p_i \in P_i$ para $i = 1, 2, \ldots, k$. Vamos implementar as ideias apresentadas por Dror et al. (2003) [3] para resolver esse problema de forma exata, assumindo que o leitor já leu o artigo mencionado, uma vez que fazemos uso direto de suas definições e resultados. O código completo está disponível no repositório GitHub.

Retomando as definições do artigo, dizemos que um i-path até p é um caminho de comprimento mínimo que começa em s, termina em p e visita os polígonos $P_1, P_2, \ldots, P_i$ em ordem, consideramos que já sabemos que todo i-path é a união de segmentos de reta e é único. A região de primeiro contato T_i de um polígono P_i é o conjunto de pontos $p \in \partial P_i$ tais que o segmento entre P_{i-1} e P_i do i-path até p intersecta P_i apenas em p. O mapa de último passo S_i de um polígono P_i é uma partição de $\mathbb{R}^2$ em regiões R tais que o último segmento do i-path até qualquer ponto $p \in R$ parte de um mesmo vértice de P_i ou de uma mesma aresta de P_i ou atravessa P_i.

Como discutido em [3, **Lemma 3.**], se tivermos $T_0, \ldots, T_i$, $S_0, \ldots, S_i$ e soubermos localizar um ponto p em S_i, então podemos calcular o i-path até p de forma eficiente. Se p pertence à região R de S_i e essa região está associada a um vértice v, então o último segmento do i-path até p é o segmento $\overline{vp}$ e o restante do caminho é o $(i-1)$-path até v. Se R está associada a uma aresta e, então refletimos p em relação à reta que contém e, chamando esse ponto de p', então calculamos o $(i-1)$-path até p', considerando-o apenas até o ponto que seu último segmento intersecta a aresta e e conectamos o final desse caminho com o ponto p, sendo esse o i-path. Por fim, se R corresponde a atravessar P_i, então o i-path até p é igual ao $(i-1)$-path até p. Implementamos essa ideia no Algoritmo 1, assumindo que temos um algoritmo auxiliar chamado `LocalizaPonto` que localiza o ponto p no mapa de último passo S_i, vamos discutir diferentes implementações para esse procedimento posteriormente.

Algoritmo 1: Consulta Caminho

Entrada: p, i, $(P_1, \ldots, P_i)$, $(T_1, \ldots, T_i)$, $(S_1, \ldots, S_i)$
Saída : i-path até p como uma sequência de pontos em $\mathbb{R}^2$.

if $i = 0$ **:**
  return $[s, p]$
$R \leftarrow$ LocalizaPonto(p, S_i) // Localiza o ponto p no mapa de último passo S_i
if R corresponde a um vértice v de P_i **:**
  // Adiciona v ao final do $(i-1)$-path até v
  return ConsultaCaminho$(v, i-1, (P_1, \ldots, P_{i-1}), (T_1, \ldots, T_{i-1}), (S_1, \ldots, S_{i-1})) \oplus [p]$
else if R corresponde a uma aresta e de P_i **:**
  $p' \leftarrow$ reflexão de p em relação à reta que contém e
  path $\leftarrow$ ConsultaCaminho$(p', i-1, (P_1, \ldots, P_{i-1}), (T_1, \ldots, T_{i-1}), (S_1, \ldots, S_{i-1}))$
  path$' \leftarrow$ pontos de path que antecedem p
  $q' \leftarrow$ último ponto de path$'$
  $q \leftarrow$ ponto de interseção entre $\overline{q'p'}$ e e
  // Adiciona q e p ao final do $(i-1)$-path até p' que precede q'
  return path$' \oplus [q', p]$
else
  // R corresponde a atravessar P_i
  return ConsultaCaminho$(p, i-1, (P_1, \ldots, P_{i-1}), (T_1, \ldots, T_{i-1}), (S_1, \ldots, S_{i-1}))$

O Algoritmo 1 permite que calculemos o i-path até qualquer ponto p de forma eficiente, podendo ser utilizado para calcular o k-path até t, que é exatamente a solução do problema. No entanto, vamos utilizar esse algoritmo em muitos outros momentos, com a diferença de que estamos apenas interessados no último segmento do i-path até p, ademais, o caso em que um ponto está em uma região associada a um vértice de P_i permite que determinemos o último segmento diretamente, sem precisar calcular o restante do caminho. Assim, implementamos o Algoritmo **??**, que dado um ponto p e um índice i, retorna o ponto inicial do último segmento do i-path até p.

Algoritmo 2: Consulta Último Passo

Entrada: p, i, $(P_1, \ldots, P_i)$, $(T_1, \ldots, T_i)$, $(S_1, \ldots, S_i)$
Saída : Ponto que precede p no i-path até p.

if $i = 0$ **:**
  return s
$R \leftarrow$ LocalizaPonto(p, S_i)
if R corresponde a um vértice v de P_i **:**
  return v
else if R corresponde a uma aresta e de P_i **:**
  $p' \leftarrow$ reflexão de p em relação à reta que contém e
  path $\leftarrow$ ConsultaCaminho$(p', i-1, (P_1, \ldots, P_{i-1}), (T_1, \ldots, T_{i-1}), (S_1, \ldots, S_{i-1}))$
  path$' \leftarrow$ pontos de path que antecedem p
  $q' \leftarrow$ último ponto de path$'$
  $q \leftarrow$ ponto de interseção entre $\overline{q'p'}$ e e
  return q
else
  return ConsultaÚltimoSegmento$(p, i-1, (P_1, \ldots, P_{i-1}), (T_1, \ldots, T_{i-1}), (S_1, \ldots, S_{i-1}))$

Esses dois algoritmos serão utilizados em todas as implementações que discutiremos a seguir, a diferença entre elas estará na implementação do algoritmo `LocalizaPonto` e na forma como calculamos as regiões de primeiro contato T_i e os mapas de último passo S_i para cada polígono P_i.

2.1 Primeira Abordagem

Primeiramente, vamos calcular a região de primeiro contato T_i de um polígono P_i, sabemos que essa região é a união de arestas de P_i, sendo delimitada por vértices de P_i [3, **Lemma 2.**], assim, apenas precisamos determinar quais arestas de P_i pertencem a T_i. Para isso, para cada aresta de P_i, definimos m como o ponto médio dessa aresta e então calculamos o i-path até m, definindo q como o ponto que antecede m nesse caminho. Se o segmento entre q e m intersecta P_i apenas em m, então, por definição, m pertence a T_i e como T_i é a união de arestas de P_i, toda a aresta que contém m pertence a T_i. Repetimos esse processo para todas as arestas de P_i e assim obtemos T_i.

Agora, para determinar se o segmento entre q e m intersecta P_i apenas em m, simplesmente verificamos se o vetor $\overrightarrow{qm}$ está do lado externo da aresta de P_i que contém m, ou seja, se o produto escalar entre $\overrightarrow{qm}$ e o vetor normal à aresta de P_i que contém m é positivo, isso é equivalente a dizer que o produto vetorial entre $\overrightarrow{qm}$ e o vetor que conecta os dois vértices da aresta de P_i que contém m é negativo. Ilustramos essa ideia na Figura 1. Também implementamos isso no Algoritmo 3

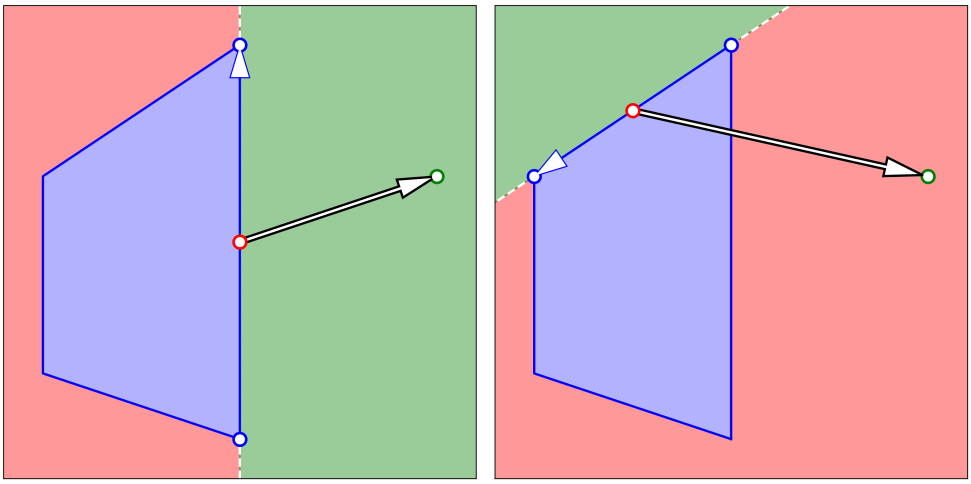

Figura 1: Região de primeiro contato T_i de um polígono P_i.

Algoritmo 3: Região de Primeiro Contato

Entrada: s, i, $(P_1, \ldots, P_i)$, $(T_1, \ldots, T_{i-1})$, $(S_1, \ldots, S_{i-1})$
Saída **:** Região de primeiro contato T_i de P_i como um conjunto de arestas de P_i.

$T_i \leftarrow \emptyset$
for cada aresta e de P_i **:**
    $v^1 \leftarrow$ vértice anterior de e
    $v^2 \leftarrow$ vértice posterior de e
    $m \leftarrow (v^1 + v^2)/2$
    $q \leftarrow$ ConsultaÚltimoSegmento$(m, i-1, (P_1, \ldots, P_{i-1}), (T_1, \ldots, T_{i-1}), (S_1, \ldots, S_{i-1}))$
    if $\overrightarrow{qm} \times \overrightarrow{v^1 v^2} < 0$ **:**
        Adiciona e a T_i
return T_i

Referências

[1] J. Liu e H. Liu, "Research on Path Optimization Method for Warehouse Inspection Robot," *Applied Artificial Intelligence*, v. 37, n. 1, 2023. endereço: https://www.tandfonline.com/doi/full/10.1080/08839514.2023.2254048#abstract.

[2] K. J. Obermeyer, "Path Planning for a UAV Performing Reconnaissance of Static Ground Targets in Terrain," *GNC-31: Intelligent Control in Aerospace Applications*, 2009. endereço: https://doi.org/10.2514/6.2009-5888.

[3] M. Dror, A. Efrat, A. Lubiw e J. S. B. Mitchell, "Touring a sequence of polygons," em *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, sér. STOC '03, San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 473–482, ISBN: 1581136749. DOI: 10.1145/780542.780612. endereço: https://doi.org/10.1145/780542.780612.