

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 sobrepõem, 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, \dots, 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 polígonos na ordem P_1, P_2, \dots, P_k se existem pontos no caminho p_1, p_2, \dots, p_k tais que $p_i \in P_i$ para $i = 1, 2, \dots, k$. 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, \dots, 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 $T_0, \dots, T_i, S_0, \dots, S_i$ 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: $p, i, (P_1, \dots, P_i), (T_1, \dots, T_i), (S_1, \dots, S_i)$

Saída : i -path até p como uma sequência de pontos em \mathbb{R}^2 .

```

1 if  $i = 0$  :
2   return  $[s, p]$ 
3  $R \leftarrow \text{LocalizaPonto}(p, S_i)$  // Localiza o ponto  $p$  no mapa de último passo  $S_i$ 
4 if  $R$  corresponde a um vértice  $v$  de  $P_i$  :
5   // Adiciona  $v$  ao final do  $(i-1)$ -path até  $v$ 
6   return  $\text{ConsultaCaminho}(v, i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1})) \oplus [p]$ 
7 else if  $R$  corresponde a uma aresta  $e$  de  $P_i$  :
8    $p' \leftarrow$  reflexão de  $p$  em relação à reta que contém  $e$ 
9    $\text{path} \leftarrow \text{ConsultaCaminho}(p', i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1}))$ 
10   $\text{path}' \leftarrow$  pontos de  $\text{path}$  que antecedem  $p$ 
11   $q' \leftarrow$  último ponto de  $\text{path}'$ 
12   $q \leftarrow$  ponto de interseção entre  $\overline{q'p'}$  e  $e$ 
13  // Adiciona  $q$  e  $p$  ao final do  $(i-1)$ -path até  $p'$  que precede  $q'$ 
14  return  $\text{path}' \oplus [q', p]$ 
15 else
16   //  $R$  corresponde a atravessar  $P_i$ 
17   return  $\text{ConsultaCaminho}(p, i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1}))$ 

```

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: $p, i, (P_1, \dots, P_i), (T_1, \dots, T_i), (S_1, \dots, S_i)$

Saída : Ponto que precede p no i -path até p .

```

1 if  $i = 0$  :
2   return  $s$ 
3  $R \leftarrow \text{LocalizaPonto}(p, S_i)$ 
4 if  $R$  corresponde a um vértice  $v$  de  $P_i$  :
5   return  $v$ 
6 else if  $R$  corresponde a uma aresta  $e$  de  $P_i$  :
7    $p' \leftarrow$  reflexão de  $p$  em relação à reta que contém  $e$ 
8    $\text{path} \leftarrow \text{ConsultaCaminho}(p', i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1}))$ 
9    $\text{path}' \leftarrow$  pontos de  $\text{path}$  que antecedem  $p$ 
10   $q' \leftarrow$  último ponto de  $\text{path}'$ 
11   $q \leftarrow$  ponto de interseção entre  $\overline{q'p'}$  e  $e$ 
12  return  $q$ 
13 else
14  return  $\text{ConsultaÚltimoSegmento}(p, i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1}))$ 

```

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

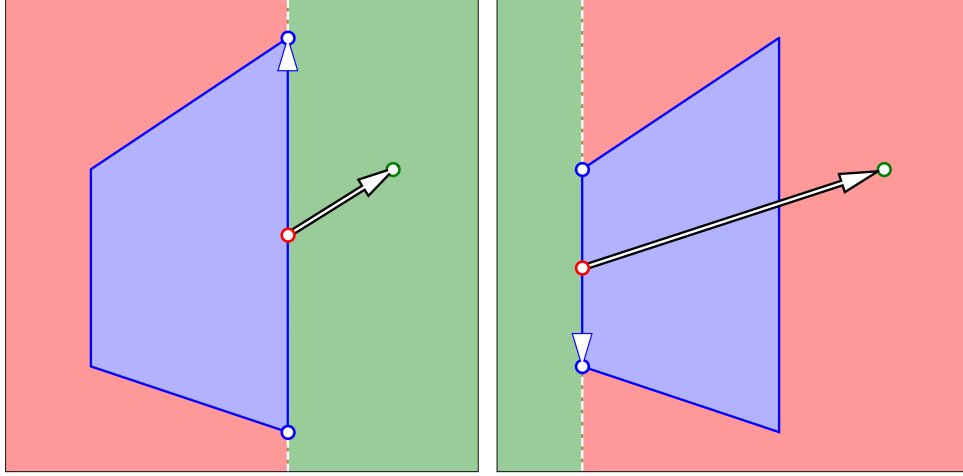


Figura 1: Verificando se uma aresta pertence à região de primeiro contato de um polígono.

Algoritmo 3: Região de Primeiro Contato

Entrada: $s, i, (P_1, \dots, P_i), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1})$

Saída : Região de primeiro contato T_i de P_i como um conjunto de arestas de P_i .

```

1  $T_i \leftarrow \emptyset$ 
2 for cada aresta  $e$  de  $P_i$  :
3    $v^1 \leftarrow$  vértice anterior de  $e$ 
4    $v^2 \leftarrow$  vértice posterior de  $e$ 
5    $m \leftarrow (v^1 + v^2)/2$ 
6    $q \leftarrow \text{ConsultaÚltimoSegmento}(m, i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1}))$ 
7   if  $(q - m) \times (v^2 - v^1) < 0$  :
8      $\_ \leftarrow$  Adiciona  $e$  a  $T_i$ 
9 return  $T_i$ 

```

Ademais, uma vez que temos T_0, \dots, T_i , podemos calcular o mapa de último passo S_i de P_i . Para tal, notamos que S_i tem exatamente três tipos de regiões: regiões associadas a vértices de P_i , regiões associadas a arestas de P_i uma única região de atravessar P_i . Ademais, uma região associada à aresta e é delimitada pelas regiões associadas às pontas de e e pela aresta e em si. Além disso, a região de atravessar P_i é definida como o complemento das regiões associadas a vértices e arestas de P_i . Assim, para construir S_i , precisamos apenas determinar as regiões associadas a vértices de P_i .

Pelas condições de otimalidade discutidas em [3, **Local Optimality Conditions.**], sabemos que para cada vértice v de P_i que está em T_i , a região associada a v em S_i é simplesmente um cone definido pelo ponto v e um par ordenado de raios que partem de v . Podemos calcular esses raios a partir do último segmento do $(i-1)$ -path até v e sua reflexão em relação às arestas de P_i que contêm v . Entrando em mais detalhe, se e^1 e e^2 são as arestas de P_i em sentido anti-horário que contêm v e q é o ponto que precede v no $(i-1)$ -path até v , então o primeiro raio é a reflexão do vetor \vec{qv} em relação à reta que contém e^1 caso e^1 faça parte de T_i , ou simplesmente o vetor \vec{qv} caso contrário. De forma análoga, o segundo raio é a reflexão do vetor \vec{qv} em relação à reta que contém e^2 caso e^2 faça parte de T_i , ou simplesmente o vetor \vec{qv} caso contrário. Repetimos esse processo para todos os vértices de P_i e assim obtemos todas as regiões associadas a vértices de P_i em S_i . Implementamos essa ideia no Algoritmo 4.

Algoritmo 4: Mapa de Último Passo**Entrada:** $s, i, (P_1, \dots, P_i), (T_1, \dots, T_i), (S_1, \dots, S_{i-1})$ **Saída** : Mapa de último passo S_i de P_i como uma associação de vértices de T_i a pares ordenados de raios.

```

1  $S_i \leftarrow \emptyset$ 
2 for vértice  $v$  em  $P_i$  :
3    $e^1 \leftarrow$  aresta anterior de  $v$  em  $P_i$ 
4    $e^2 \leftarrow$  aresta posterior de  $v$  em  $P_i$ 
5    $q \leftarrow$  ConsultaÚltimoSegmento( $v, i-1, (P_1, \dots, P_{i-1}), (T_1, \dots, T_{i-1}), (S_1, \dots, S_{i-1})$ )
6    $d \leftarrow v - q$ 
7   if  $e^1$  pertence a  $T_i$  :
8      $r^1 \leftarrow$  reflexão de  $d$  em relação à reta que contém  $e^1$ 
9   else
10     $r^1 \leftarrow d$ 
11   if  $e^2$  pertence a  $T_i$  :
12     $r^2 \leftarrow$  reflexão de  $d$  em relação à reta que contém  $e^2$ 
13   else
14     $r^2 \leftarrow d$ 
15   Associa  $v$  a  $(r^1, r^2)$  em  $S_i$ 
16 return  $S_i$ 

```

Uma vez que temos T_i e S_i , desejamos implementar o algoritmo **LocalizaPonto** que localiza um ponto p no mapa de último passo S_i . Na primeira abordagem, implementamos esse procedimento de forma direta, verificando cada região de vértice, então cada região de aresta e caso o ponto não pertença à nenhuma dessas retornamos a região de atravessar P_i . Para verificar se um ponto p pertence a uma região associada a um vértice v de P_i com raios r^1 e r^2 , primeiro precisamos determinar se o ângulo de abertura entre r^1 e r^2 é menor ou maior que 180° . Se o ângulo for menor que 180° , então p pertence à região associada a v se o produto vetorial entre r^1 e \vec{vp} for não positivo e o produto vetorial entre \vec{vp} e r^2 for negativo. Caso contrário, p pertence à região associada a v se o produto vetorial entre r^1 e \vec{vp} for positivo ou o produto vetorial entre \vec{vp} e r^2 for negativo, essas ideias são ilustradas na Figura 2 e implementadas no Algoritmo 5.

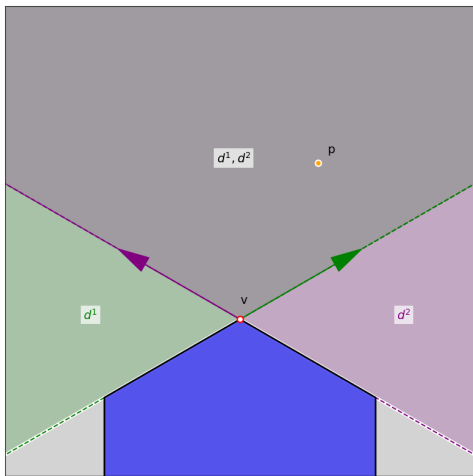
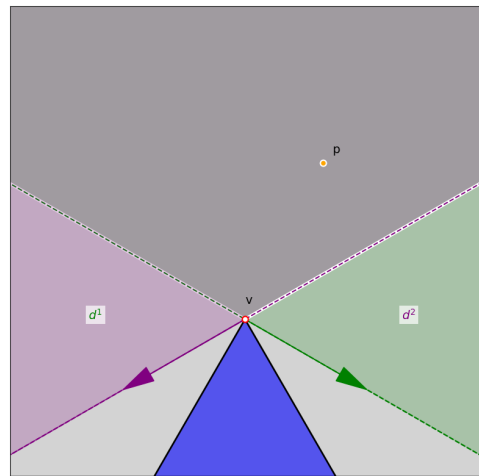
(a) Ângulo de abertura menor que 180° .(b) Ângulo de abertura maior que 180° .

Figura 2: Verificando se um ponto pertence a uma região associada a um vértice em um mapa de último passo.

Algoritmo 5: Ponto em Região de Vértice**Entrada:** p, v, r^1, r^2 **Saída** : **true** se p pertence à região delimitada por v, r^1 e r^2 , **false** caso contrário.

```

1 if  $r^1 \times r^2 \geq 0$  :
2   // Ângulo de abertura menor que  $180^\circ$ 
3   return  $(r^1 \times (p - v) > 0) \wedge (r^2 \times (p - v) < 0)$ 
4 else
5   // Ângulo de abertura maior que  $180^\circ$ 
6   return  $(r^1 \times (p - v) > 0) \vee (r^2 \times (p - v) < 0)$ 

```

Ademais, desejamos verificar se um ponto p pertence a uma região associada a uma aresta e de P_i . Para isso, consideramos que v^1 e v^2 são os vértices que definem a aresta e em sentido anti-horário e consideramos os raios r^1 como o segundo raio do vértice v^1 e r^2 como o primeiro raio do vértice v^2 em S_i . Então, p pertence à região associada a e ele estiver do lado externo da aresta e e entre os raios r^1 e r^2 , como ilustrado na Figura 3. Implementamos essa ideia no Algoritmo 6.

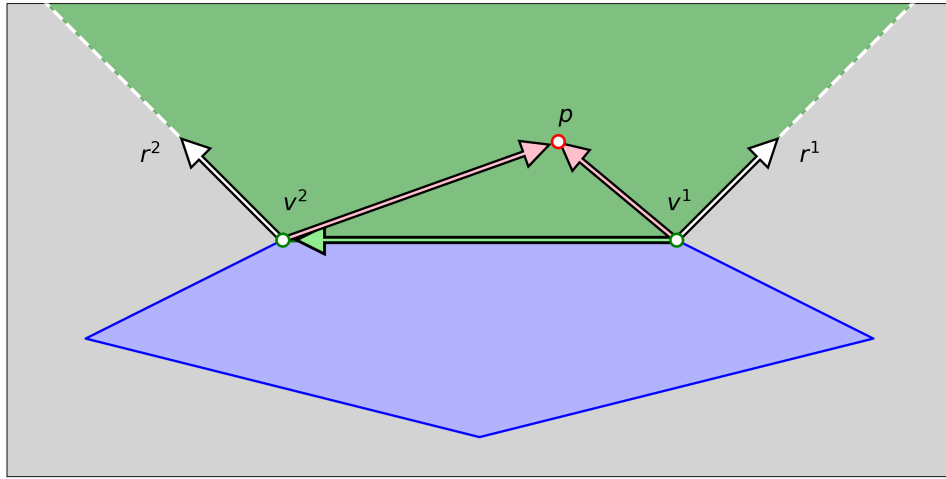


Figura 3: Verificando se um ponto pertence a uma região de aresta.

Algoritmo 6: Ponto em Região de Aresta**Entrada:** p, v^1, v^2, r^1, r^2 **Saída** : **true** se p pertence à região delimitada por $e = \overline{v^1 v^2}$, r^1 e r^2 , **false** caso contrário.

```

1 return  $r^1 \times (p - v^1) > 0 \wedge r^2 \times (p - v^2) < 0 \wedge (p - v^1) \times (v^2 - v^1) < 0$ 

```

Uma vez que conseguimos verificar se um ponto pertence a uma região associada a um vértice ou a uma aresta de P_i , podemos implementar o algoritmo **LocalizaPonto** de forma direta, como mostrado no Algoritmo 7.

Algoritmo 7: Localiza Ponto - Primeira Abordagem

Entrada: p, S_i, P_i
Saída : Região de S_i que contém p .

```

1 for cada vértice  $v$  de  $P_i$  :
2    $(r^1, r^2) \leftarrow$  raios associados a  $v$  em  $S_i$ 
3   if PontoEmRegiãoDeVértice( $p, v, r^1, r^2$ ) :
4     return região associada a  $v$ 
5 for cada aresta  $e = \overline{v^1 v^2}$  de  $P_i$  :
6    $r^1 \leftarrow$  segundo raio associado a  $v^1$  em  $S_i$ 
7    $r^2 \leftarrow$  primeiro raio associado a  $v^2$  em  $S_i$ 
8   if PontoEmRegiãoDeAresta( $p, v^1, v^2, r^1, r^2$ ) :
9     return região associada a  $e$ 
10 return região de atravessar  $P_i$ 

```

O Algoritmo 7 verifica todas as regiões de vértices e arestas de P_i para localizar o ponto p , assim, sua complexidade é $O(|P_i|)$, onde $|P_i|$ é o número de vértices de P_i . Como esse procedimento será chamado diversas vezes durante a execução dos Algoritmos 1 e 2, essa implementação pode ser ineficiente para polígonos com muitos vértices. Portanto, discutimos duas outras abordagens para implementar o algoritmo **LocalizaPonto** de forma mais eficiente, usando estratégias de busca binária e memoização.

2.2 Segunda Abordagem

Na segunda abordagem, vamos alterar apenas a implementação do algoritmo **LocalizaPonto**, utilizando uma estratégia de busca binária para localizar o ponto p no mapa de último passo S_i . O parte principal dessa abordagem é notar que se v^1 e v^2 são dois vértices de P_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>.