PLANEJAMENTO DE ROTAS

Seminário de Robótica

Bruno de Abreu Silva

SUMÁRIO

- Introdução
- Conceitos gerais
- Métodos de planejamento de rotas

DEFINIÇÃO DO PROBLEMA

- Dadas as configurações inicial e final de um robô, descobrir uma sequência de movimentos a ser executada pelo robô para que ele saia da primeira e chegue à segunda sem colidir com obstáculos.

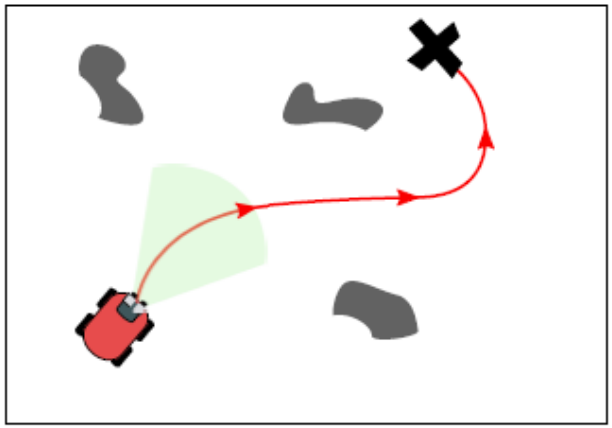


Figura 1.1: Exemplo típico de planejamento de rotas em robótica móvel. $\mathcal{C}$ é tridimensional e $\mathcal{W}$ é bidimensional.

- O planejamento de rotas geralmente ocorre em *C* e não em *W*
- A solução é dada por uma função contínua π tal que $\pi:[0,1] \rightarrow C\_livre$, em que $\pi(0) = q\_inicio$ e $\pi(1) = q\_fim$
- Dificuldade do planejamento de rotas determinístico:
  - tempo de execução que cresce exponencialmente com o número de graus de liberdade do robô.
- Assim, têm sido pesquisados métodos que aproximam *C_livre* por meio de amostras do mesmo e que são computacionalmente eficientes.

- O grande desafio é projetar planejadores que tenham:
  - Bom desempenho computacional
  - Produzam rotas com boa qualidade
  - Que não sejam dependentes do espaço de trabalho
- Métodos para planejamento de rotas foram amplamente investigados na teoria e na prática e suas aplicações extrapolam a robótica

APLICAÇÕES

- Animação de atores artificiais
- Biologia molecular
- Problemas de montagens complexas
- Planejamento de movimento para objetos flexíveis e planejamento para robôs humanóides
- Planejamento multirrobos
- Planejamento em ambientes com obstáculos que se movem

SUMÁRIO

- Introdução
- Conceitos gerais
  - Espaço de configurações
  - Representação dos objetos em *W*
    - *Representação por polígonos convexos*
    - *Modelos semi-algébricos*
    - *Grades de ocupação*
- Métodos de planejamento de rotas

ESPAÇO DE CONFIGURAÇÕES

- Permite resolver com o mesmo algoritmo diferentes problemas de planejamento
- Permite representar conceitos físicos como força e atrito
- O planejamento de rotas acontece em *C_livre*

$$\mathcal{C}_{\text{livre}} = \mathcal{C} \setminus \bigcup_{i=1}^{n} \mathcal{CO}_i = \left\{ q \in \mathcal{C} \mid \mathcal{A}(q) \cap \left(\bigcup_{i=1}^{n} \mathcal{O}_i \right) = \emptyset \right\}. \qquad (1.1)$$

REPRESENTAÇÃO DOS OBJETOS EM *W*

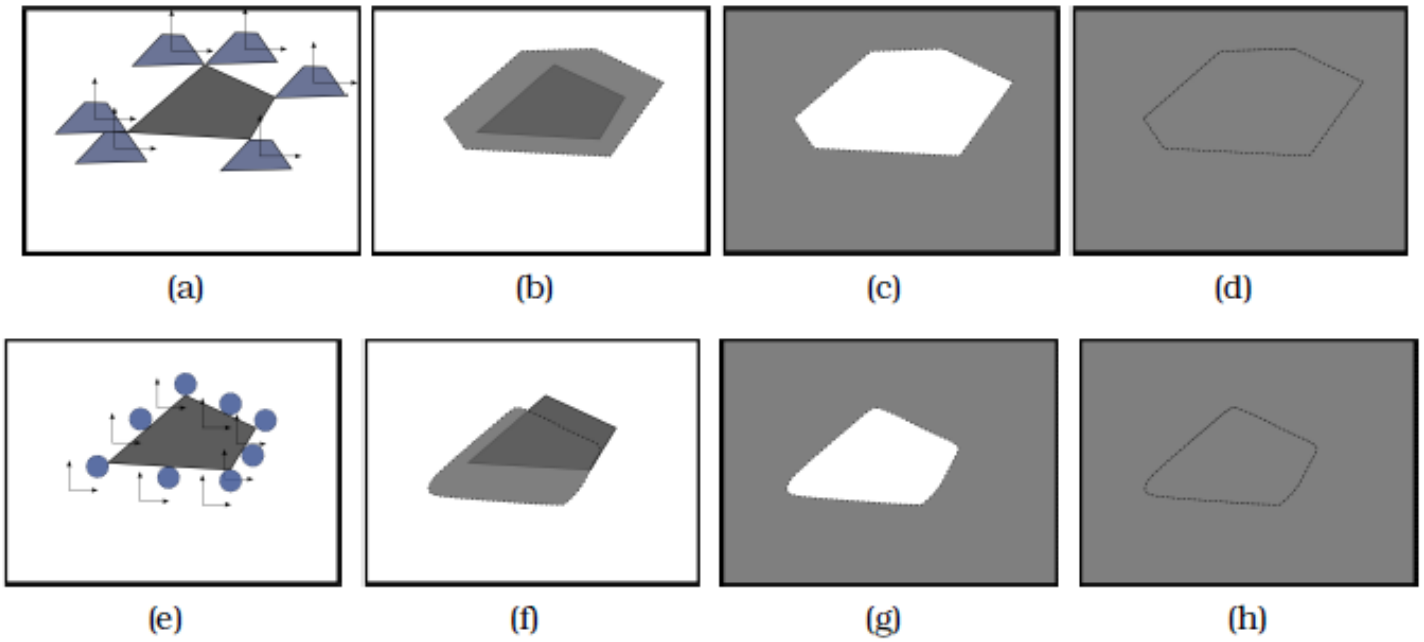
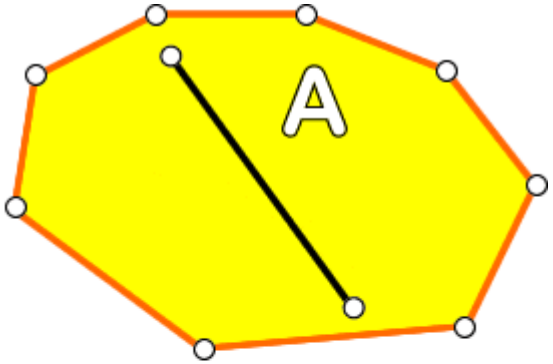
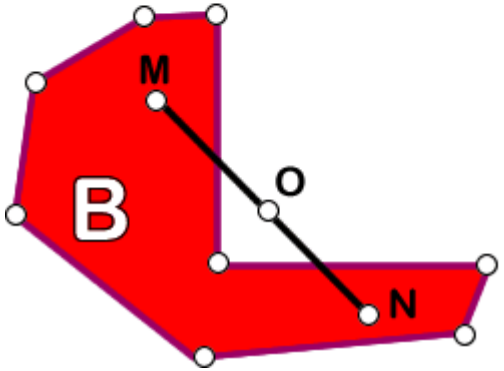


Figura 1.2: Exemplo mostrando que o espaço de configurações independe do formato do robô. (a) Mapeamento de um robô trapezoidal. (b) $\mathcal{C}_{\text{obs}}$ dentro da região tracejada. (c) $\mathcal{C}_{\text{livre}}$ hachurado. (d) $\mathcal{C}_{\text{obs}} \cup \mathcal{C}_{\text{livre}} = \mathcal{C}$. (e) Mapeamento de um robô circular. (f) $\mathcal{C}_{\text{obs}}$ dentro da região tracejada. (g) $\mathcal{C}_{\text{livre}}$ hachurado. (h) $\mathcal{C}_{\text{obs}} \cup \mathcal{C}_{\text{livre}} = \mathcal{C}$.

POLÍGONOS CONVEXOS

- Uma região plana é chamada de região convexa se e somente se todo o segmento de reta cujas extremidades pertencem à região só tem pontos na mesma região





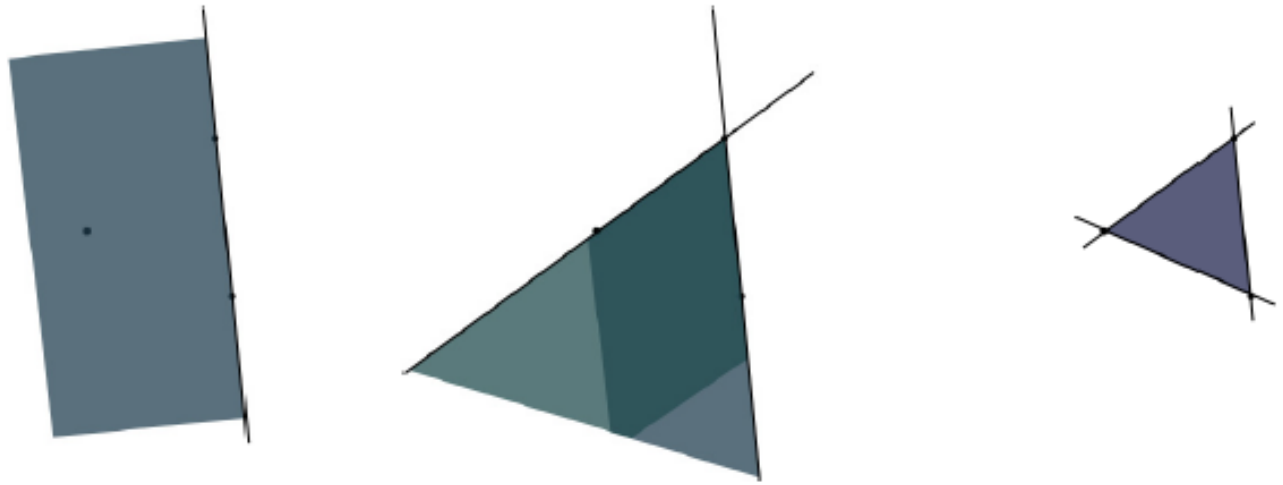


Figura 1.3: Triângulo construído pela intersecção de três semiplanos. (a) Construção do primeiro semiplano. (b) Intersecção entre dois semiplanos. (c) Triângulo resultante da intersecção entre três semiplanos.

MODELOS SEMI-ALGÉBRICOS

- Um conjunto de pontos determinado por uma única primitiva polinomial é dito algébrico. Uniões e intersecções de conjuntos algébricos formam um conjunto semi-algébrico.
- Modelos semi-algébricos são uma generalização da representação por polígonos convexos

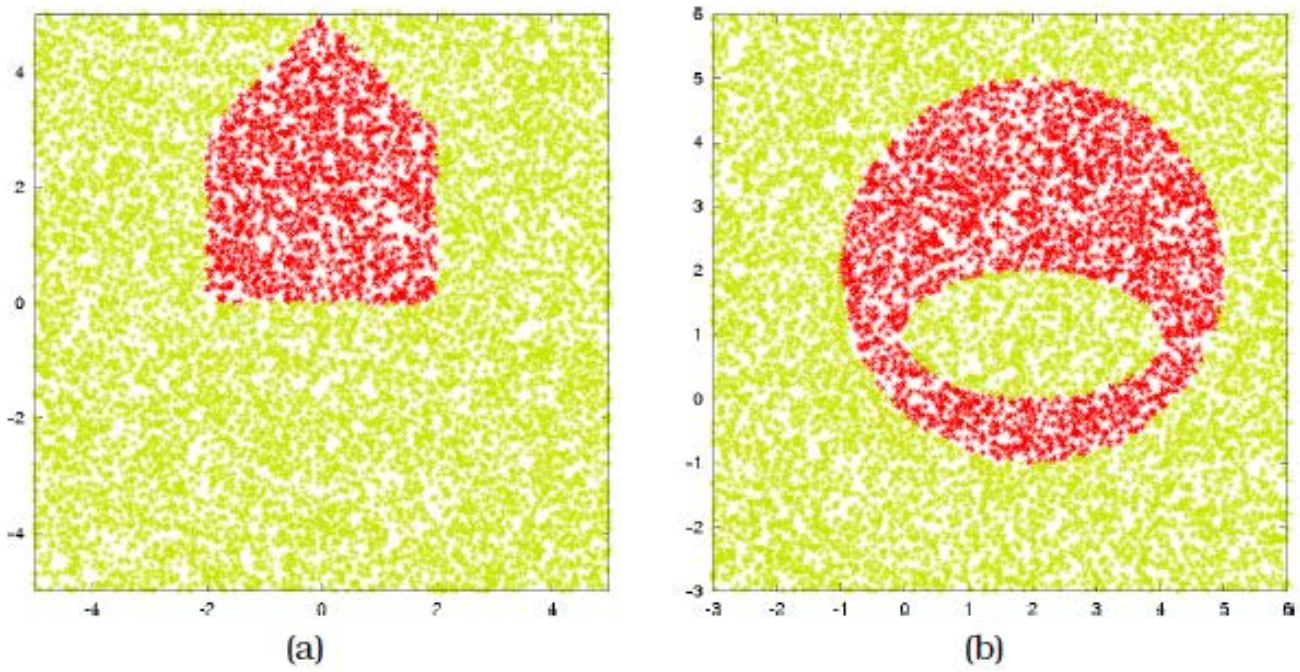


Figura 1.4: Detecção de colisão usando representação por (a) polígonos convexos e (b) por primitivas semi-algébricas.

GRADES DE OCUPAÇÃO

- O espaço é dividido em células que contém uma probabilidade de ocupação associada.
- Consegue representar as incertezas relacionadas ao mapeamento do espaço de trabalho
- *Bitmap* – útil para fins de mapeamento de rotas e pode ser obtido atribuindo 0 ou 1 para cada probabilidade das células de acordo com um limiar

(a)

(b)

Figura 1.5: (a) Grade de ocupação e (b) sua transformação em *bitmap*.

SUMÁRIO

- Introdução
- Conceitos gerais
- Métodos de planejamento de rotas
  - Campos potenciais
  - Planejamento por frente de onda
  - Decomposição celular
  - Métodos probabilísticos
  - Suavização de rotas

MÉTODOS DE PLANEJAMENTO DE ROTAS

- Podem ser divididos em:
  - Algoritmos que utilizam mapa de rotas
  - Algoritmos que fazem decomposição em células
  - Algoritmos que utilizam funções potenciais

- Podem ser divididos também em:
  - De questionamento único
  - De múltiplos questionamentos
- Por fim, também dividem-se em:
  - Determinísticos
  - Probabilísticos

CAMPOS POTENCIAIS

- A idéia básica é atribuir um potencial de atração gerado pela configuração de destino e um potencial de repulsão gerado pelos obstáculos. Assim, naturalmente o robô tende a ir de encontro à configuração desejada enquanto automaticamente desvia dos obstáculos.

$$U(\mathbf{q}) = U_{\text{att}}(\mathbf{q}) + U_{\text{rep}}(\mathbf{q}), \tag{1.15}$$

$$\mathbf{f} = \mathbf{f}_{\text{att}} + \mathbf{f}_{\text{rep}}. \tag{1.16}$$

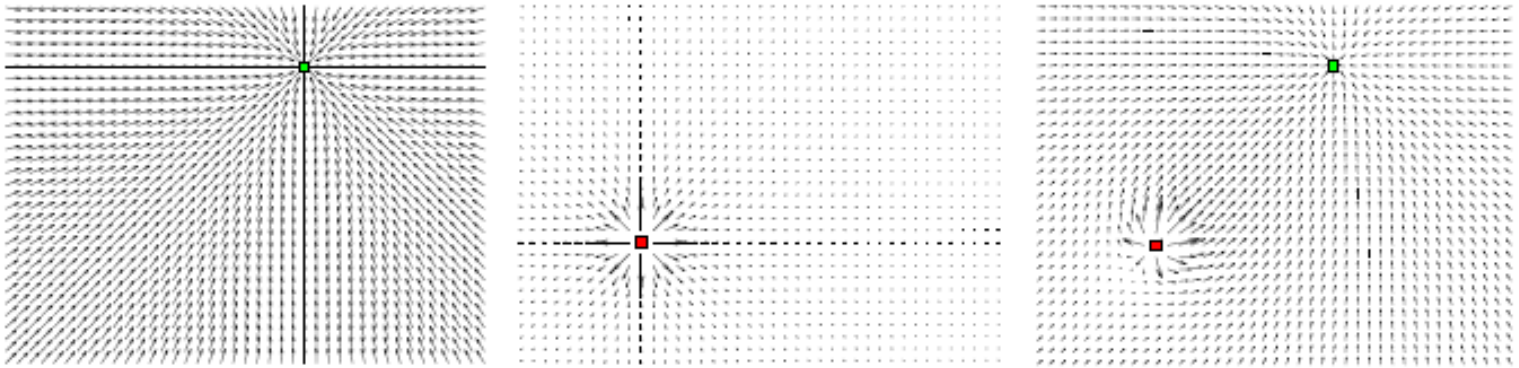


Figura 1.6: Campos potenciais: o gradiente do campo é equivalente a uma força atuando no robô. (a) Gradiente do potencial de atração. (b) Gradiente do potencial de repulsão. (c) Gradiente do potencial resultante.

Algoritmo 1 Planejador que utiliza campos potenciais artificiais.

Require: Configurações inicial $q_{início}$ e final q_{fim}

Ensure: Rota

1: $q \leftarrow q_{início}$
2: **enquanto** $\|f\| > \varepsilon$ **faï¿½a**
3: Calcula a resultante f_{rep} das forças de repulsão geradas pelos obstáculos
4: Calcula a força de atração f_{att} gerada pela configuração de destino q_{fim}
5: $f \leftarrow f_{att} + f_{rep}$
6: $q \leftarrow q + \alpha f$
7: **fim enquanto**

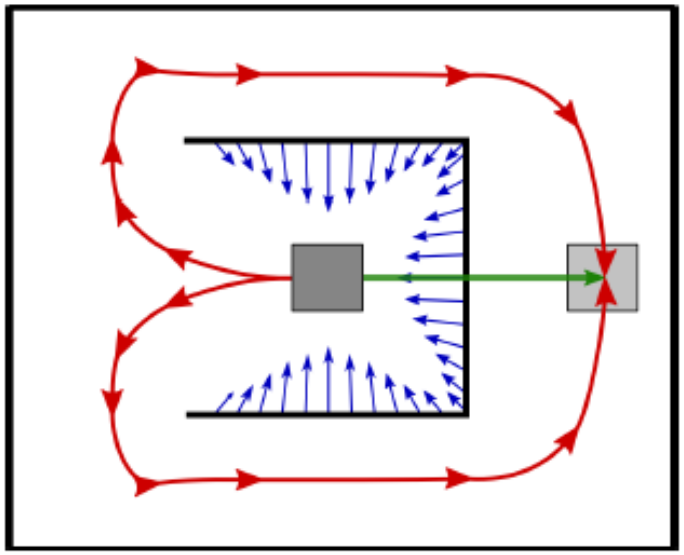
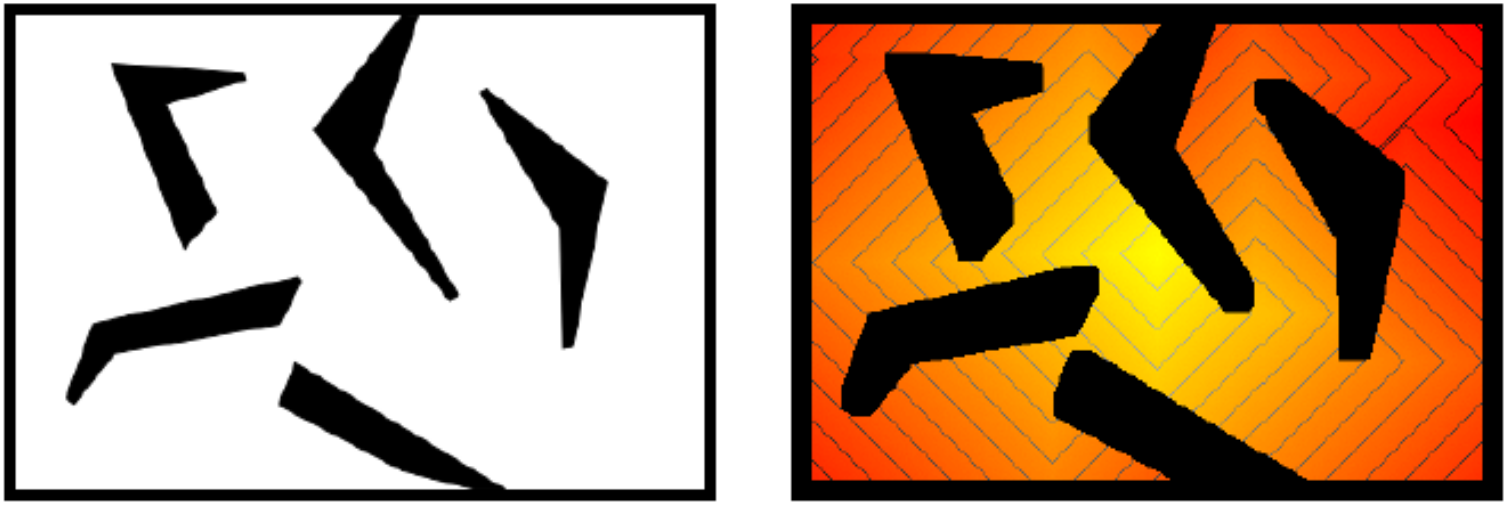


Figura 1.7: Planejador baseado em campos potenciais. A cavidade do obstáculo não-convexo constitui um mínimo local.

- Abordagens que visam eliminar o problema de mínimos locais:
  - Passeios aleatórios
  - Outras funções potenciais que resultam em um único mínimo localizado na configuração final
    - Campos eletrostáticos
    - Funções harmônicas.

PLANEJADOR POR FRENTE DE ONDA (PFO)

- Técnica simples e determinística
- Utiliza grade no espaço de configurações
- Algoritmo:
  - Rotulam-se as células ocupadas pelos obstáculos com 1 e o espaço livre em 0.
  - A configuração final é rotulada com 2 e a partir dela as células vizinhas livres são numeradas com o valor de seu rótulo mais 1, ou seja, 3.
  - O processo é repetido com todas as células vizinhas e assim por diante.
- Solução: partindo da configuração inicial e seguindo sequencia decrescente até a célula com rótulo 2



(a) (b)

Figura 1.8: Potencial gerado pela frente de onda cujo mínimo global está localizado no centro do espaço de configurações. (a) Espaço de trabalho. (b) Potencial gerado pela frente de onda em $\mathcal{C}_{\text{livre}}$.

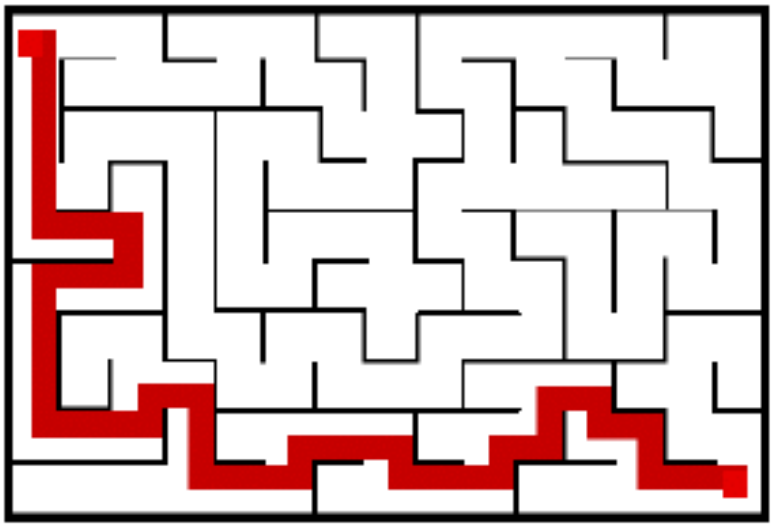


Figura 1.9: Rota gerada pelo planejador por frente de onda.

VANTAGENS DO PFO

- É de resolução completa;
- Existe só um ponto de mínimo, localizado no ponto de destino;
- A solução é ótima no sentido de ter a menor distancia de Manhattan;
- É de fácil implementação, sem estruturas de dados complexas.

DESVANTAGENS DO PFO

- por ser determinístico, o PFO pode se tornar computacionalmente impraticável para espaços de configuração de ordem mais alta;
- rotas geradas são muito próximas dos obstáculos, que pode comprometer a segurança do robô;
- o critério da distancia de Manhattan não gera rotas ótimas segundo o critério da distancia Euclidiana.

DECOMPOSIÇÃO CELULAR

- Dividem-se em duas categorias principais: decomposição exata e decomposição aproximada.
- Em ambas as categorias, o princípio básico do planejador é particionar *C_livre* em um conjunto de células *Ki* cujas regiões interiores não se sobrepõem.
- O objetivo é achar uma sequência de células adjacentes *K1 … Kn E C_livre*, tal que *K1 = Kinicio*, *Kn = Kfim*.

DECOMPOSIÇÃO CELULAR EXATA

- O espaço de configurações livre é particionado tal que a união das células resulta exatamente nele mesmo.
- O planejador resultante é considerado completo.
- Uma vez que *C_livre* foi particionado, a solução para o planejamento de rotas consiste em achar uma sequência de células adjacentes ligando *qinicio* a *qfim*.

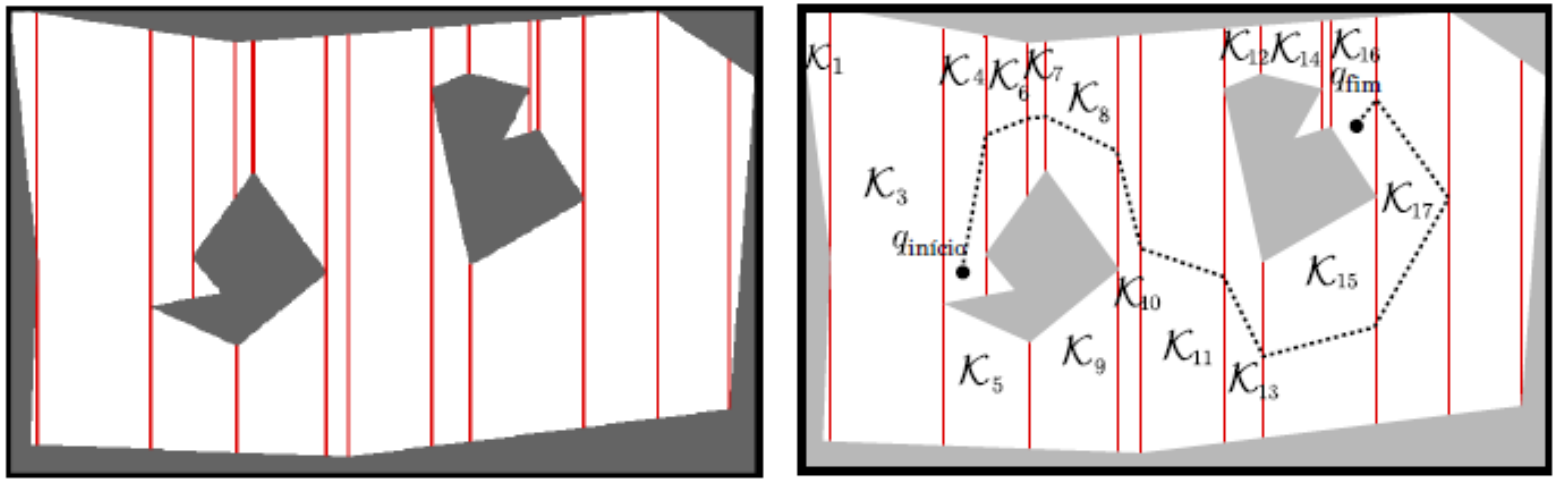


Figura 1.10: Decomposição celular exata. (a) Decomposição trapezoidal. (b) Rota resultante.

DECOMPOSIÇÃO CELULAR APROXIMADA

- Não permitem a representação exata de *C_livre*
- É mais fácil de se implementar que as versões exatas
- São de resolução completa
- O espaço de configurações é dividido sucessivamente em células denominadas *vazias*, *cheias* ou *mistas*.
- O objetivo é refinar a amostragem de células mistas até que elas sejam divididas apenas em células vazias ou cheias, ou então até uma resolução máxima.

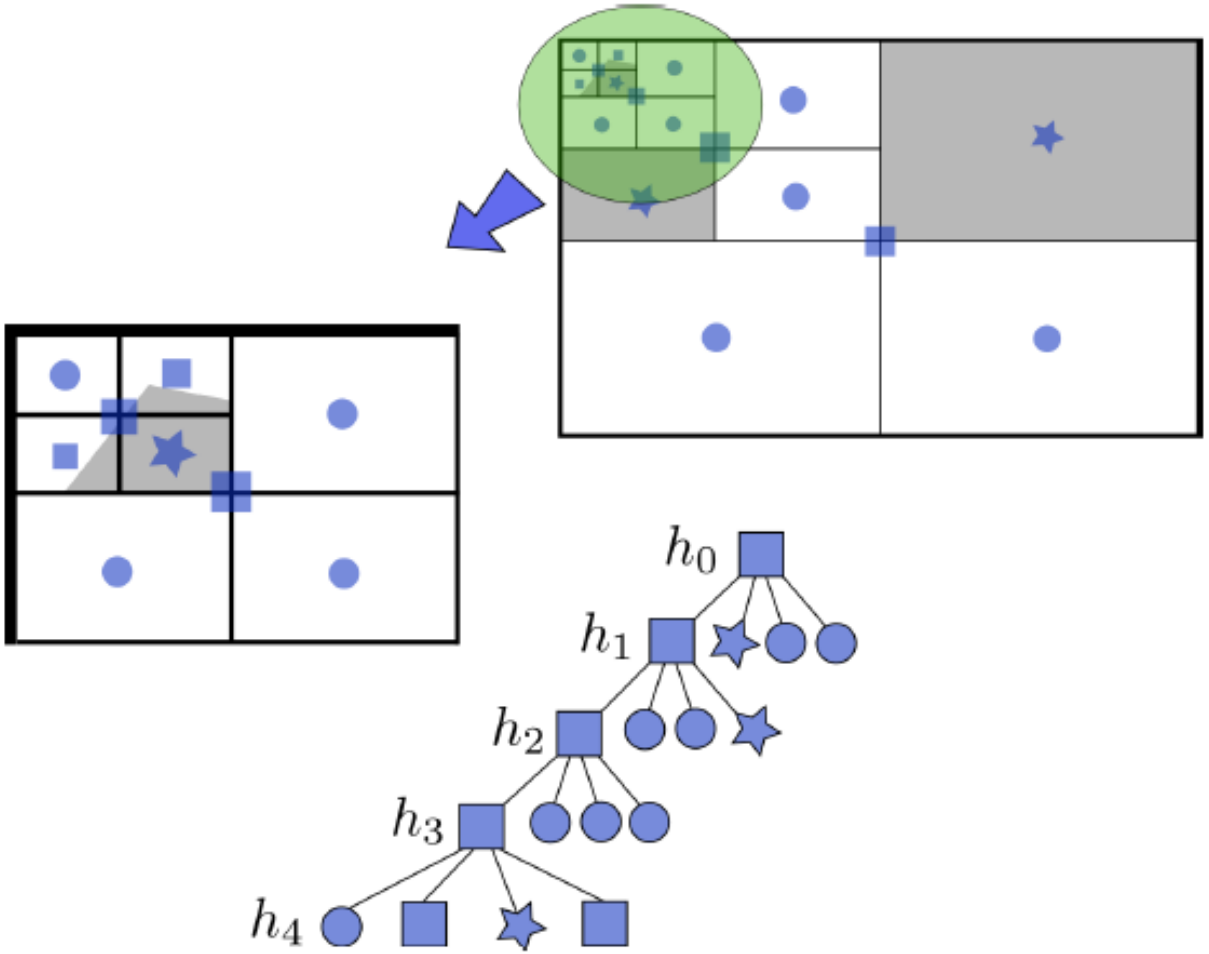


Figura 1.11: Decomposição celular aproximada para $\mathcal{C} \in \mathbb{R}^2$ e a árvore equivalente.

MÉTODOS PROBABILÍSTICOS

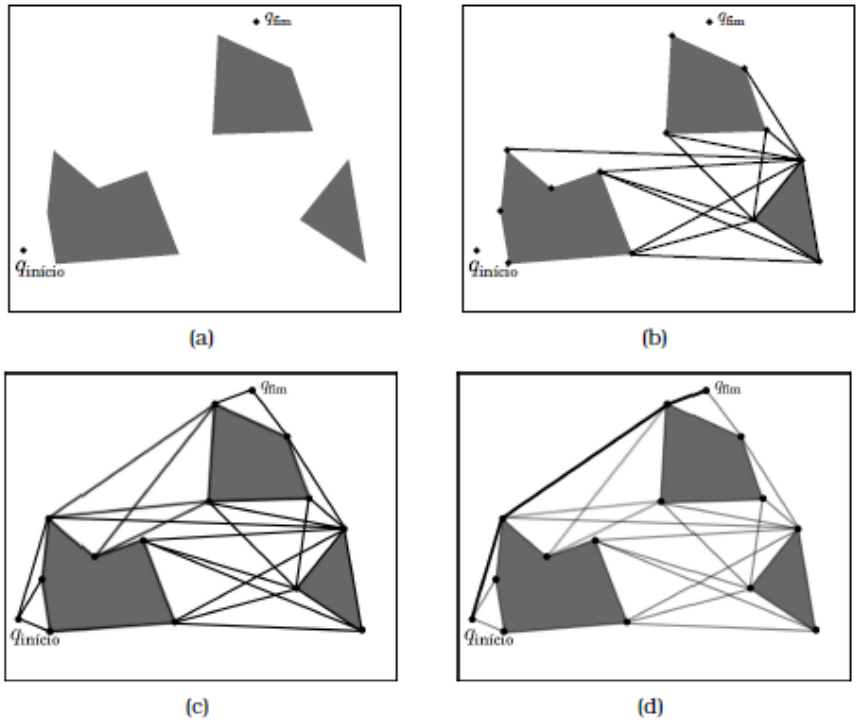


Figura 1.12: Construção do mapa de visibilidade. (a) Espaço de configurações bidimensional com os obstáculos representados por polígonos. (b) Etapa intermediária da construção do mapa de visibilidade: os vértices dos polígonos correspondem aos nós do grafo; observa-se a criação das arestas para o triângulo, sendo que todas as rotas que ligam os seus vértices aos vértices dos outros polígonos e que se encontram em $\mathcal{C}_{livre}$ são armazenadas no mapa de rotas; nota-se que as arestas dos polígonos também são armazenadas. (c) Mapa de visibilidade final. (d) Rota ligando as configurações inicial e final.

MAPA DE ROTAS PROBABILÍSTICO (PRM)

- A estrutura básica deste algoritmo é um grafo localizado em *C_livre*
- O algoritmo PRM é dividido em duas etapas: aprendizado e questionamento. A primeira consiste em expandir o grafo na região *C_livre* para poder gerar rotas rapidamente na etapa seguinte.
- Uma das premissas utilizadas no PRM padrão é que o espaço de trabalho seja estático ou que as mudanças ocorrem lentamente.

Algoritmo 2 Etapa de aprendizado do PRM

Require: max_nós, max_vizinhos e D_n
Ensure: Mapa de rotas.

```
1: i ← 0
2: N ← 0
3: E ← 0
4: enquanto i < max_nós faï¿½a
5:    q_nova ← configuração aleatória em C
6:    se q_nova ∈ C_livre então
7:       i ← i + 1
8:       Adiciona q_nova ao grafo por meio de um nó em N
9:       Seleciona max_vizinhos vizinhos q_j em torno de q_nova que estejam a
         uma distância inferior a D_n
10:      para j = 1 até max_vizinhos faï¿½a
11:         se planejador_local(q_nova, q_j) e não mesmo componente conexo então
12:            Adiciona uma aresta em E conectando o nó referente a q_nova ao nó
               referente a q_j
13:         fim se
14:      fim para
15:   fim se
16: fim enquanto
```

Algoritmo 3 planejador_local

Require: As configurações $\mathbf{q}_i, \mathbf{q}_j \in \mathcal{C}_{\text{livre}}$ e $\alpha \in (0, 1)$
Ensure: 1, se $\mathbf{q}_i$ puder ser conectado a $\mathbf{q}_j$ e 0, caso contrário.

1: $\mathbf{q} \leftarrow \mathbf{q}_i$
2: $t \leftarrow 0$
3: **enquanto** $t < 1$ **faï¿½a**
4: **se** detecção_colisão(q) **então**
5: **return** 0
6: **senão**
7: $t \leftarrow t + \alpha$
8: $\mathbf{q} \leftarrow (1 - t)\mathbf{q}_i + t\ \mathbf{q}_j$
9: **fim se**
10: **fim enquanto**
11: {Se chegou até este ponto, é porque não houve colisão.}
12: **return** 1

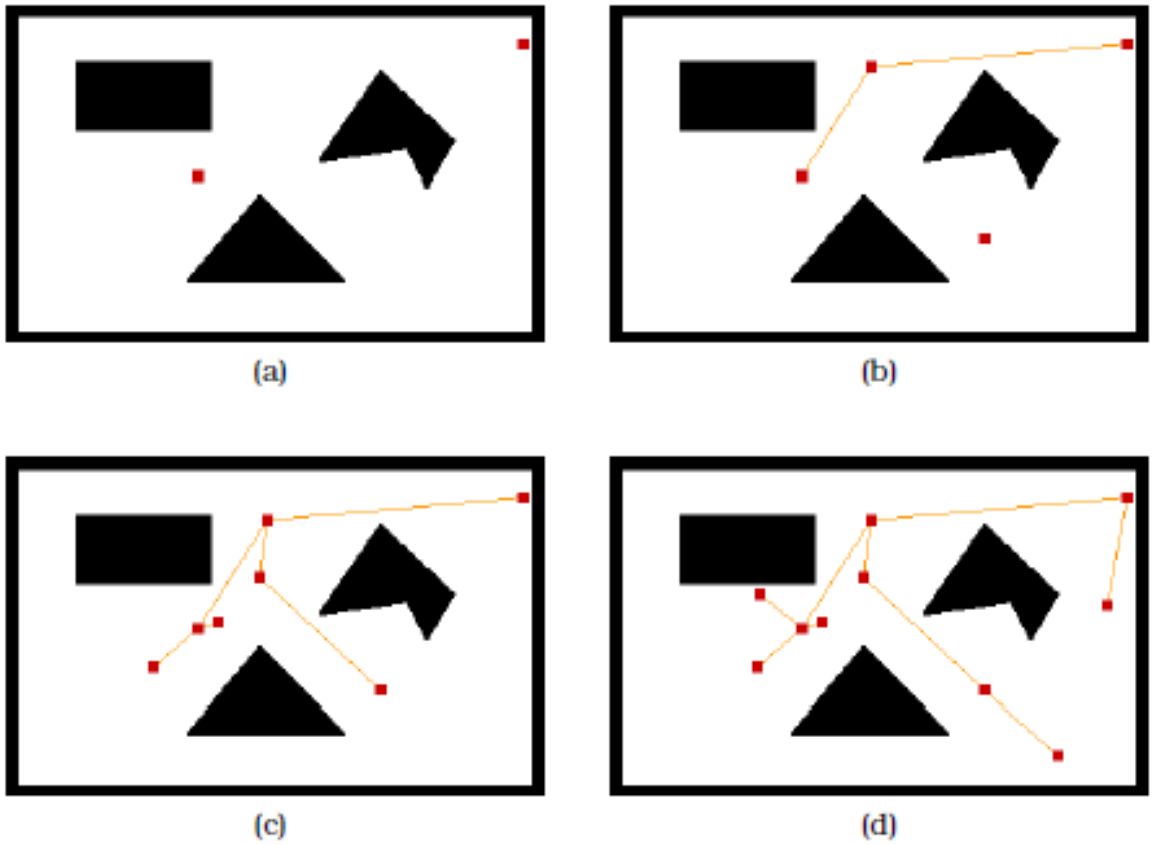


Figura 1.13: Evolução da construção do mapa de rotas no algoritmo PRM. (a) Dois nós são gerados em $\mathcal{C}_{\text{livre}}$. (b) Mais dois nós são gerados e, após a conexão com os vizinhos, dois componentes conexos são formados. (c) Mais nós são gerados e os dois componentes conexos se fundem em apenas um. (d) Mapa de rotas resultante após a geração de 10 configurações.

- Número de amostras não pode ser nem muito pequeno inviabilizando o encontro da solução e nem muito grande desperdiçando recursos computacionais
- O PRM pode ser utilizado em espaços de configuração de dimensões grandes, pois uma vez que o mapa de rotas é construído, basta verificar o grafo.

ÁRVORES ALEATÓRIAS PARA EXPLORAÇÃO RÁPIDA (RRT)

- Foi desenvolvido inicialmente para planejamento kinodinâmico.
- Versão bidirecional específica do RRT para buscas no espaço de configurações: RRT-conecta.
- O objetivo é fazer com que duas árvores cresçam da configuração inicial e final do robô por meio de um processo de amostragem em *C* e expansão destas árvores em *C_livre*.

Algoritmo 4 RRT

Require: $q_{\text{início}}$ e q_{fim}
Ensure: Rota resultante

1: $\mathcal{T}_a$.inicia($q_{\text{início}}$)
2: $\mathcal{T}_b$.inicia(q_{fim})
3: **para** k = 1 **até** K **faï¿½a**
4: gera uma configuração aleatória $q_{\text{aleatória}}$ em $\mathcal{C}$ de acordo com uma distribuição uniforme
5: **se** RRT-conecta($\mathcal{T}_a$,$q_{\text{aleatória}}$) **não** BLOQUEADO **então**
6: $q_{\text{nova}} \leftarrow q_{\text{aleatória}}$ (ou a configuração que está mais perto de $q_{\text{aleatória}}$ que encontra-se em $\mathcal{C}_{\text{livre}}$)
7: **se** RRT-conecta($\mathcal{T}_b$,q_{nova}) = ALCANÇOU **então**
8: **return** rota($\mathcal{T}_a$,$\mathcal{T}_b$)
9: **fim se**
10: **senão**
11: alterna($\mathcal{T}_a$,$\mathcal{T}_b$)
12: **fim se**
13: **fim para**
14: **return** NULL

Algoritmo 5 RRT-conecta

Require: $q_{\text{aleatória}}$ e a árvore $\mathcal{T}$
Ensure: Status de q_{nova}

```
1: q_próx. ← vizinho_mais_próximo(q_aleatória, T)
2: q_nova ← planejador_local_ganancioso(q_próx., q_aleatória)
3: se q_nova não NULL então
4:    T.adiciona_nó(q_nova)
5:    T.adiciona_aresta(q_próx., q_nova)
6:    se q_aleatória = q_nova então
7:       return ALCANÇOU
8:    senão
9:       return AVANÇOU
10:   fim se
11: senão
12:   return BLOQUEADO
13: fim se
```

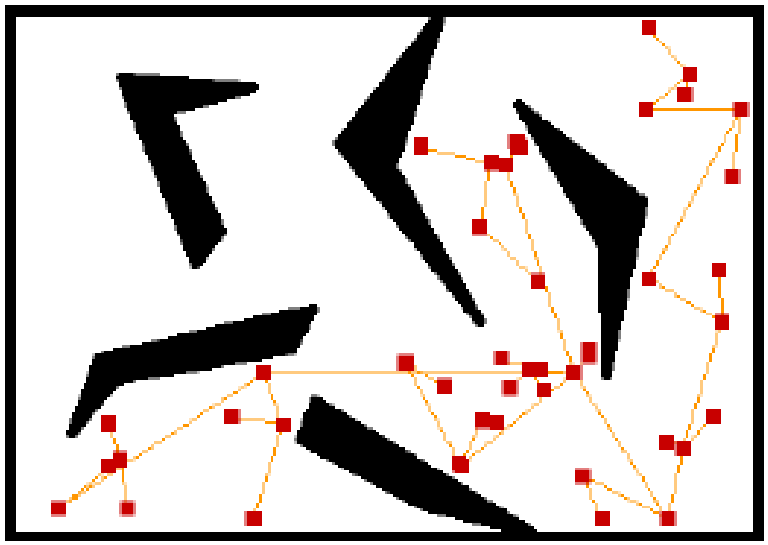


Figura 1.14: Exemplo de árvore criada pelo algoritmo RRT. Foram geradas 50 amostras.

- Como a busca é feita de forma bidirecional, a solução em geral é encontrada bem antes do espaço de busca ter sido todo explorado.
- Assim como o PRM, um problema do RRT é que a busca por vizinhos mais próximos é feita ao adicionar um novo nó na árvore de exploração.

ÁRVORES EM ESPAÇOS EXPANSIVOS (EST)

- É de questionamento único
- Assim como o RRT, visa explorar de maneira uniforme o espaço de configurações.
- Duas árvores são geradas a partir das configurações iniciais *qinicio* e *qfim* e o objetivo é que uma árvore seja alcançável pela outra, o que caracteriza a solução do problema.

Algoritmo 6 constrói-EST

Require: q_0 e N
Ensure: Árvore $\mathcal{T}$ com nó raiz q_0

1: Adiciona q_0 à árvore $\mathcal{T}$
2: **para** i = 1 **até** N **faï¿½a**
3: $q_{\text{aleatória}} \leftarrow$ conf. aleatória escolhida em $\mathcal{T}$ com probabilidade $\pi(q_{\text{aleatória}})$
4: $q_{\text{nova}} \leftarrow$ conf. aleatória na vizinhança de $q_{\text{aleatória}}$, tal que distância($q_{\text{aleatória}}, q_{\text{nova}}$) $<$ d
5: **se** planejador_local($q_{\text{aleatória}}, q_{\text{nova}}$) **então**
6: adiciona q_{nova} a $\mathcal{T}$
7: adiciona uma aresta ligando $q_{\text{aleatória}}$ a q_{nova}
8: **fim se**
9: **fim para**
10: **return** $\mathcal{T}$

Algoritmo 7 fusiona-EST

Require: $\mathcal{T}_{\text{início}}$ e $\mathcal{T}_{\text{fim}}$

1: **for all** $q_a \in \mathcal{T}_{\text{início}}$ **e** $q_b \in \mathcal{T}_{\text{fim}}$ **faï¿½a**
2: **se** dist $(q_a, q_b) < l$ **então**
3: **se** planejador_local(q_a, q_b) **então**
4: **return** ROTA
5: **fim se**
6: **fim se**
7: **fim para**

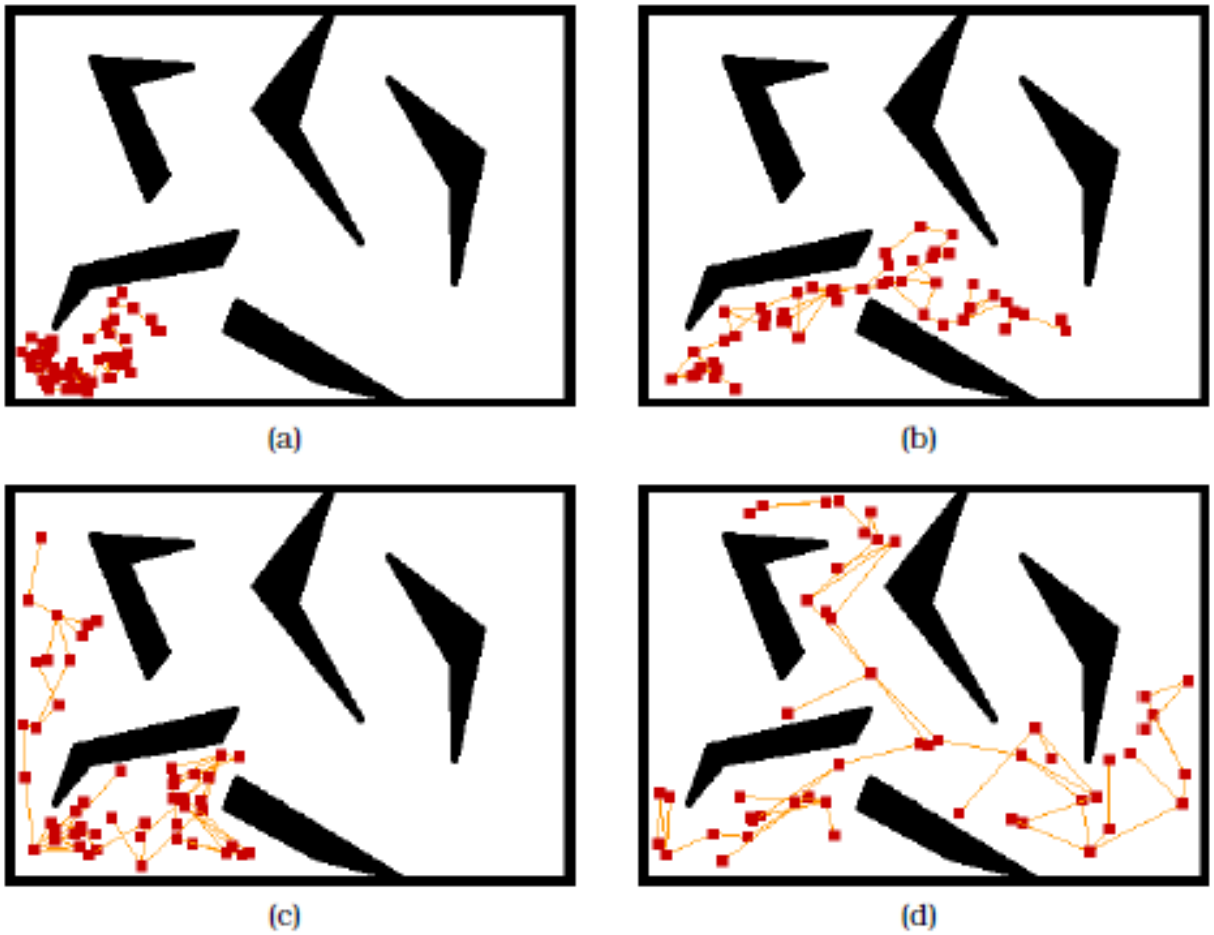


Figura 1.15: Árvore gerada pelo algoritmo EST. Foram geradas 50 amostras. (a) EST com intervalo de amostragem igual a 30. (b) EST com intervalo de amostragem igual a 60. (c) EST com intervalo de amostragem igual a 90 (d) EST com intervalo de amostragem igual a 120.

- Se a distância D for muito grande, áreas que não são relevantes para a solução podem ser amostradas. Por outro lado, se for muito pequeno, o algoritmo pode demorar para convergir.
- Outra dificuldade inerente a este algoritmo é à escolha da função PI(q). Ela deve balancear a exploração de áreas pouco densas e muito densas

PASSEIO ALEATÓRIO ADAPTATIVO (ARW)

- É de questionamento único.
- Ênfase no tempo de execução.
- Utiliza um passeio aleatório para explorar o espaço de configurações livres até que a última amostra gerada possa ser conectada ao ponto de destino por meio de um planejador local.

Algoritmo 8 ARW simples.

Require: Configurações inicial e final.

Ensure: Lista que contém a rota ligando $\mathbf{q}_{\text{início}}$ a $\mathbf{q}_{\text{fim}}$.

1: $k \leftarrow 0$
2: $\mathbf{q}_k \leftarrow \mathbf{q}_{\text{início}}$
3: Inicia a matriz de covariâncias $\mathbf{\Sigma}_0$ com $\mathbf{\Sigma}_{\text{min}}$
4: **enquanto não** planejador_local($\mathbf{q}_k$,$\mathbf{q}_{\text{fim}}$) **faï¿½a**
5: Gera uma nova configuração aleatória $\mathbf{v}_k \in N(0, \mathbf{\Sigma}_k)$
6: $\mathbf{s} \leftarrow \mathbf{q}_k + \mathbf{v}_k$
7: **se** planejador_local($\mathbf{q}_k$,$\mathbf{s}$) **então**
8: $k \leftarrow k+1$
9: $\mathbf{q}_k \leftarrow \mathbf{s}$
10: Atualiza a matriz de covariâncias $\mathbf{\Sigma}_k$ usando a Eq. (1.20)
11: Insere $\mathbf{q}_k$ na lista de configurações intermediárias
12: **fim se**
13: **fim enquanto**

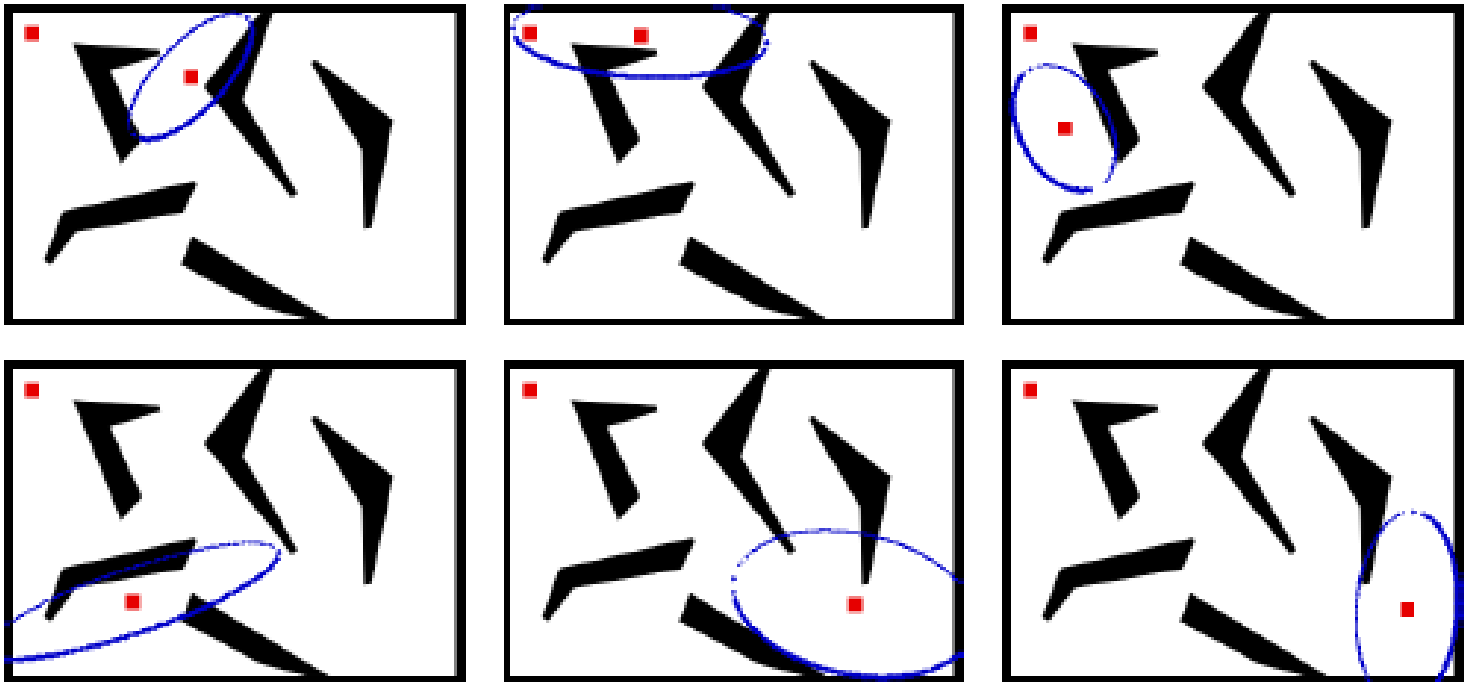


Figura 1.16: xxx

VANTAGENS DO ARW

- Sua implementação é a mais simples dentre os quatro algoritmos.
- Por não fazer busca de configurações vizinhas mais próximas, o custo de gerar uma nova amostra também é muito baixo.
- Se adapta à estrutura do espaço de configurações para gerar uma nova amostra.
- A amostragem é polarizada em *C_livre*.

DESVANTAGEM DO ARW

- Já a maior desvantagem do ARW é que a rota resultante possui uma péssima qualidade, com muitas sinuosidades e ciclos desnecessários.

SUAVIZAÇÃO DE ROTAS

- Dividir para conquistar
- Utilização de atalhos e remoção de ciclos

DIVIDIR PARA CONQUISTAR

Algoritmo 9 dividir_para_conquistar

Require: $q_{\text{início}}$, q_{fim} e a lista contendo a rota a ser suavizada
Ensure: Lista que contém a rota suavizada

1: **se** $(q_{\text{início}} = q_{\text{fim}})$ **ou** $(q_{\text{início}+1} = q_{\text{fim}})$ **então**
2: **return** {A rota não pode mais ser dividida, então interrompe esta execução.}
3: **senão se** planejador_local($q_{\text{início}}, q_{\text{fim}}$) **então**
4: remove_nós_intervalo($q_{\text{início}+1}, q_{\text{fim}-1}$)
5: **senão**
6: {Faz chamadas recursivas ao algoritmo dividir_para_conquistar}
7: dividir_para_conquistar($q_{\text{início}}, q_{\frac{\text{início}+\text{fim}}{2}}$)
8: dividir_para_conquistar($q_{\frac{\text{início}+\text{fim}}{2}+1}, q_{\text{fim}}$)
9: **fim se**
10: **return** Rota suavizada

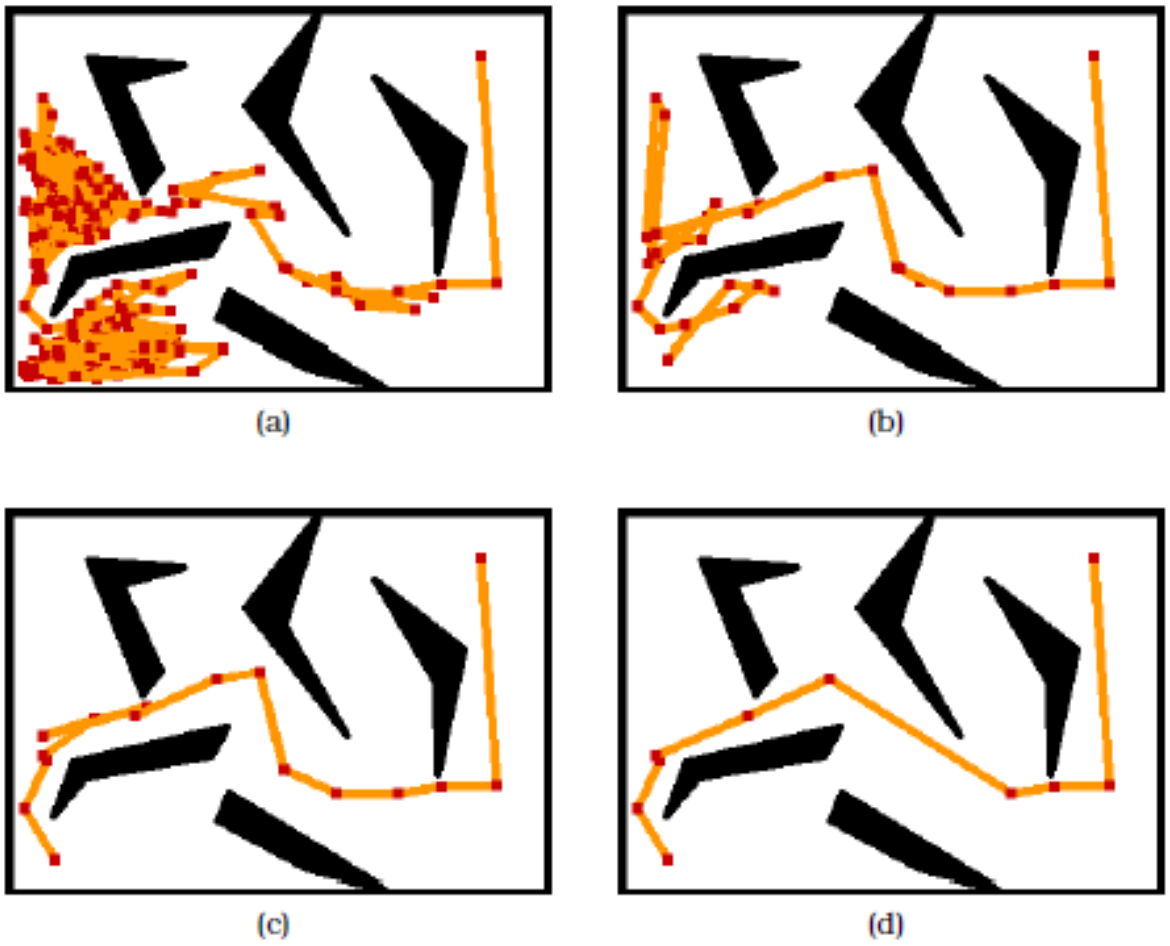


Figura 1.17: Algumas chamadas ao algoritmo dividir_para_conquistar. (a) Rota original. (b) Primeira chamada. (c) Segunda chamada. (d) Quarta chamada.

UTILIZAÇÃO DE ATALHOS E REMOÇÃO DE CICLOS

Algoritmo 10 remoção_de_ciclos

Require: $q_{\text{início}}$, q_{fim} e lista com a rota a ser suavizada
Ensure: Lista que contém a rota suavizada

1: **para** $q_i \leftarrow q_{\text{início}}$ **até** $q_{\text{fim}-1}$ **faï¿½a**
2: **para** $q_j \leftarrow q_{\text{fim}}$ **até** q_{i+1} **faï¿½a**
3: **se** planejador_local(q_i, q_j) **então**
4: remove_nós_intervalo(q_{i+1},q_{j-1})
5: $q_i \leftarrow q_j$
6: Interrompe o laço mais interno da linha 2
7: **fim se**
8: **fim para**
9: **fim para**

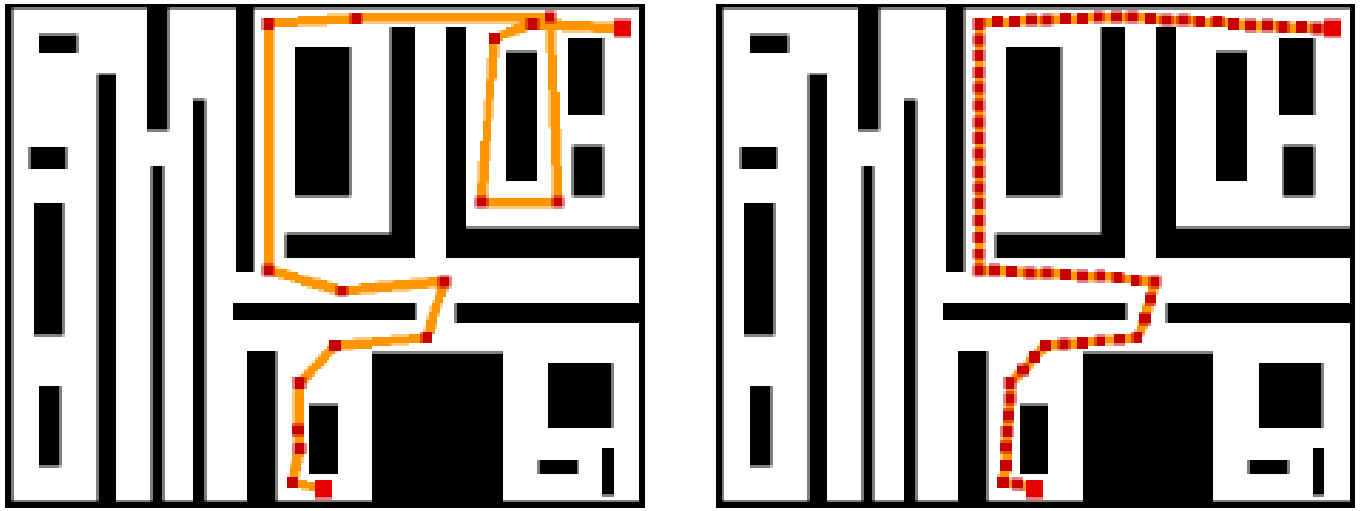


Figura 1.18: Remoção de ciclos na rota. (a) Antes da remoção dos ciclos. (b) Depois da remoção dos ciclos.