UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

BRUNO PEREIRA PALMA

PRM-EB
UM NOVO ALGORITMO PARA A OTIMIZAÇÃO DE ROTAS BASEADO NA TEORIA DE APOSTAS.

Rio de Janeiro
2016

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

BRUNO PEREIRA PALMA

PRM-EB
UM NOVO ALGORITMO PARA A OTIMIZAÇÃO DE ROTAS BASEADO NA TEORIA DE APOSTAS

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Instituto Tércio Pacciti de Aplicações Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para a obtenção do título de mestre em informática.

Orientador: Josefino Cabral Melo Lima
Co-orientador: Wouter Caarls

Rio de Janeiro
2016

BRUNO PEREIRA PALMA

PRM-EB
UM NOVO ALGORITMO PARA A OTIMIZAÇÃO DE ROTAS BASEADO NA TEORIA DE APOSTAS

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Instituto Tércio Pacciti de Aplicações Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para a obtenção do título de mestre em informática.

Aprovado em**:** Rio de Janeiro, ____ de _____ de _____.

_____
Prof. Dr. Josefino Cabral Melo Lima (Orientador)

_____
Prof. Dr. Wouter Caarls (Coorientador)

_____
Profa. Dra. Priscila Machado Vieira Lima

_____
Prof. Dr. Adriano Joaquim de Oliveira Cruz

_____
Prof. Dr. Max Suell Dutra

_____
Prof. Dr. Luciano Santos Constantin Raptopoulos

À minha família.

AGRADECIMENTOS

Agradeço à minha família e a todos os meus amigos, por todo o apoio que me deram em todas as áreas da minha vida, me ajudando e me incentivando, compartilhando alegrias, ansiedades, momentos de descontração e assim me permitindo chegar até aqui.

Agradeço também aos meus orientadores, sempre bastante pacientes, me incentivando e me cobrando quando necessário, tornando assim possível a realização deste trabalho.

RESUMO

Probabilistic roadmaps (PRM) é uma das abordagens mais utilizadas em planejamento de rotas. Por não se prender em mínimos locais, o PRM é capaz de gerar diversas rotas ligando diferentes pontos do mapa. Entretanto, estas rotas encontradas nem sempre são tão eficientes, notadamente em ambientes complexos.

O objetivo desta dissertação é o de propor um algoritmo original para melhorar a técnica do PRM. Este novo algoritmo, chamado PRM-EB, propõe uma técnica de otimização baseada na teoria de apostas. Experimentos utilizando este novo algoritmo são também descritos e analisados e mostraram que o algoritmo proposto é capaz de gerar rotas com redução de até 10% em seus tamanhos.

Palavras-chaves: Planejamento de rotas, Probabilistic roadmaps, Teoria de Apostas.

ABSTRACT

Probabilistic roadmaps (PRM) are one of the most used approaches applied to route planning. As PRM is not caught in local minima, it is able to generate several routes connecting different points on the map. However, these routes are not always so efficient, especially in complex environments.

The aim of this research is to propose an original algorithm improving the technique of PRM. This new algorithm, called PRM-EB, proposes an optimization technique based on betting theory. Some experiments using this new algorithm are also described and analyzed and they have shown that the proposed algorithm is able to generate routes with a reduction of up to 10% on their sizes, especially in complex environments.

Keywords: Path planning, Probabilistic roadmaps, Betting Theory.

Lista de Figuras

Lista de Algoritmos

Sumário

1 Introdução

O problema clássico na locomoção de robôs num ambiente real é encontrar um caminho livre (*rota*) que o leve de uma determinada posição inicial (*origem*) a uma posição final (*destino*) sem colisões com obstáculos [1]. Existem métodos bem conhecidos visando resolver este problema via planejamento e geração de rotas seguras [2], tais como os "Campos Potenciais Artificiais" (APF - *Artificial Potencial Field*) [11], as "Árvores Aleatórias de Exploração Rápida" (RRT - *Rapidly-Exploring Random Trees*) [9], as "Árvores de Espaços Expansivos" (EST - *Expansive-Space Trees*) [10], o "Passeio Randômico-Adaptativo" (ARW - *Adaptative Random Walk*) [12] e o "Mapa Probabilístico" (PRM - *Probabilistic Roadmap*) [8]. Alguns métodos são exatos e outros aproximados, porém em ambientes de configurações complexas os exatos têm se mostrado geralmente inviáveis.

Em geral, o problema de planejamento de rotas está inserido em dois tipos de ambiente: um ambiente é dito *estático* se seus obstáculos são fixos (sem inclusão, remoção ou movimentação) e dito *dinâmico* caso contrário. O PRM tem sido objeto de várias pesquisas [13, 15, 16, 17] e mais dedicado a ambientes estáticos, ainda que algumas pesquisas o explorem em ambientes dinâmicos [4, 18]. Geralmente, o PRM tem se mostrado bem eficiente, fácil de implementar e aplicável a vários tipos de problemas de planejamento de rotas. Um mapa probabilístico é muito útil, por exemplo, num galpão de estoque, no qual produtos são deslocados de um local para outro. Com um mapa, os robôs transportadores poderão ter rotas no galpão que levam produtos de uma *origem* a um *destino*, sem colisões com obstáculos.

O PRM requer algumas técnicas importantes: *navegador local*, *algoritmo de busca*, *método de amostragem*, *método de escolha dos vizinhos* etc. O *navegador local* busca uma rota entre dois nós de um grafo; o *algoritmo de busca* procura a rota mais curta entre dois nós (dado um mapa probabilístico), o *método de amostragem* seleciona os nós que irão compor o mapa probabilístico e o *método de escolha dos vizinhos* define para um nó quais nós estão mais próximos dele. Um ambiente no qual um robô se encontra é o *C-space*. O espaço livre de obstáculos neste ambiente é definido como o *C-free*. O espaço ocupado pelos obstáculos é o *C-obst*. Um grafo, neste contexto, é um conjunto de nós (e arestas) dentro do *C-free*. A ideia básica do PRM é amostrar nós no ambiente, verificar se eles estão no *C-free* e tentar conectá-los aos seus vizinhos mais próximos. A *origem* e o *destino* são inseridos no grafo e um algoritmo de busca é aplicado ao mapa para determinar uma rota *origem-destino*. Algumas pesquisas buscam melhorar a forma de amostrar os nós e de selecionar os vizinhos mais próximos [3, 7, 14, 15], outras buscam escolher quais nós selecionados se conectam para formar arestas [5, 16].

Um dos principais pontos fracos do PRM é que sua rota final pode não ser tão próxima da rota ótima. Num ambiente 100X100, por exemplo, para explorar todo o mapa o PRM teria que selecionar 10.000 (100x100) nós, mas como isto não é viável ele amostra, portanto, apenas uma parte destes nós. Desta forma a maior parte dos 10.000 nós existentes no mapa não são selecionados e não podem fazer parte da melhor rota. Ou seja, poderia existir uma rota melhor do que a encontrada pelo PRM se alguns desses nós não utilizados fizessem parte da rota ótima. No entanto, usualmente pode-se considerar a rota encontrada pelo PRM boa o suficiente posto o alto custo da fase de investigação do PRM com muitos nós selecionados.

O problema de locomoção de robôs não é um problema simples, pois o ambiente pode ser um ambiente muito complexo onde a exploração rápida de todo o espaço poderia significar o uso de algoritmos com custos muito elevados. A busca pela melhor solução em ambientes com níveis altos de complexidade geralmente passa pelo uso de heurísticas que buscam soluções que, mesmo não sendo as melhores, são boas o suficiente para resolver o problema ("The best is the enemy of good"). Uma das heurísticas mais interessantes são os algoritmos evolutivos.

1.1 Objetivo

O objetivo da pesquisa que resultou nesta dissertação é o de propor uma melhor solução justamente para o ponto mais fraco do PRM. A ideia básica é a de encontrar a rota menor possível, dado um ambiente estático complexo, não imprimindo importância ao tempo de elaboração da rota. Para isto toma-se por base o mapa inteiro e não somente alguns nós selecionados na amostra, como originalmente acontece com o PRM. Esta pesquisa utiliza uma proposta recente de Demasi [19], cujo objetivo é o de propor um novo método de otimização baseado na teoria de apostas [32, 33, 34].

1.2 Escopo

Muito resumidamente, o *algoritmo de apostas* é um algoritmo evolutivo, tais como são os algoritmos genéticos [35, 37], a otimização por enxames [36] etc. Os algoritmos evolutivos em geral consideram as possíveis soluções de um problema como indivíduos e,

através de um processo de evolução destes indivíduos vão otimizando as soluções. O algoritmo de apostas, entretanto, é mais ousado, criando jogadores que fazem apostas em soluções que lhe parecem boas e esses jogadores vão evoluindo. Com jogadores cada vez melhores as apostas serão cada vez melhores e, desta forma, as soluções que serão encontradas serão também cada vez melhores.

1.3 Contribuição

O algoritmo de otimização proposto nesta dissertação, chamado PRM-EB (Probabilistic Roadmaps – Evolutionary Betting), melhorou o tamanho da rota do PRM para vários tipos de cenários, com diferentes configurações, tipos, tamanhos e quantidade de obstáculos. Em alguns cenários essas otimizações chegaram a reduzir o tamanho da rota da *origem* até o *destino* em mais de 10%. O algoritmo proposto provou assim que pode ser muito útil na otimização do planejamento de rotas, obtendo resultados promissores em diferentes tipos de cenários.

É importante ressaltar que este trabalho é, até onde conhecemos, o primeiro a utilizar um algoritmo, baseado no algoritmo de apostas, na área de robótica. Cabe salientar também que o tempo do algoritmo proposto não foi um parâmetro de avaliação. Com efeito, como é sabido, em algoritmos evolutivos a exatidão e a robusteza da solução são mais importantes do que tempo de execução do algoritmo. Estes são superados por aqueles, notadamente no que concerne à garantia de convergência para a melhor solução. Assim sendo, nesta pesquisa, achar a melhor rota foi o objetivo maior (em detrimento do tempo) e, sobretudo, através de uma aplicação original do algoritmo de apostas na área da robótica.

1.4 Organização

Esta dissertação está dividida em sete capítulos: o capítulo 2 apresenta os principais métodos utilizados por planejadores de rotas robóticas; o capítulo 3 oferece o suporte teórico e explica o funcionamento de algoritmos evolutivos e do algoritmo baseado na teoria de apostas; o capítulo 4 descreve detalhadamente o PRM, mostrando definições importantes e explicitando suas principais variantes; o capítulo 5 descreve o algoritmo de otimização proposto nesta dissertação PRM-EB; o capítulo 6 apresenta os experimentos realizados para a validação do PRM-EB bem como a análise dos dados obtidos; por fim, o capítulo 7 mostra a conclusão do trabalho, assim como apresenta sugestões para trabalhos futuros.

2 Planejamento de rotas

Neste capítulo são apresentados os principais métodos de planejamento de rotas utilizados na robótica, mostrando as principais vantagens e desvantagens de cada método, assim como ambientes nos quais ele possui melhor ou pior desempenho. Este capítulo explicita também a razão de se estar propondo um algoritmo original para planejamento de rotas e quais as vantagens deste novo algoritmo.

2.1 Campos potenciais artificiais (APF)

O método de *campos potenciais artificiais (APF – Artificial Potencial Field)* [11] foi um dos primeiros na área de planejamento de rotas de robôs, e até hoje é bastante utilizado, notadamente em algoritmos híbridos. APF utiliza apenas duas variáveis principais: a posição do robô no ambiente e o ângulo de direção. Em sua formulação mais simples a fórmula basilar do APF é:

$$U(q) = Uatr(q) + Urep(q)$$

onde q é o vetor de localização do robô, $Uatr$ a força de atração gerada pelo *destino* e $Urep$ a força de repulsão gerada pelos obstáculos. Nesta abordagem, o movimento do robô é obtido descendo o gradiente de $U(q)$ e o movimento termina quando o gradiente se anula. O ponto final é chamado ponto crítico de U.

A grande vantagem desse método é que além de fácil intuição, as rotas geradas são boas e, nas curvas, o robô passa geralmente a uma distância relativamente segura dos obstáculos. A principal desvantagem é que esta função tem que ser determinada criteriosamente para evitar ao máximo que o robô estacione em algum mínimo local, uma tarefa bastante difícil em ambientes complexos. Para que o problema do mínimo local não ocorra, é necessário que a função tenha somente um ponto de mínimo (no *destino*), o que é bastante raro em ambientes complexos [44].

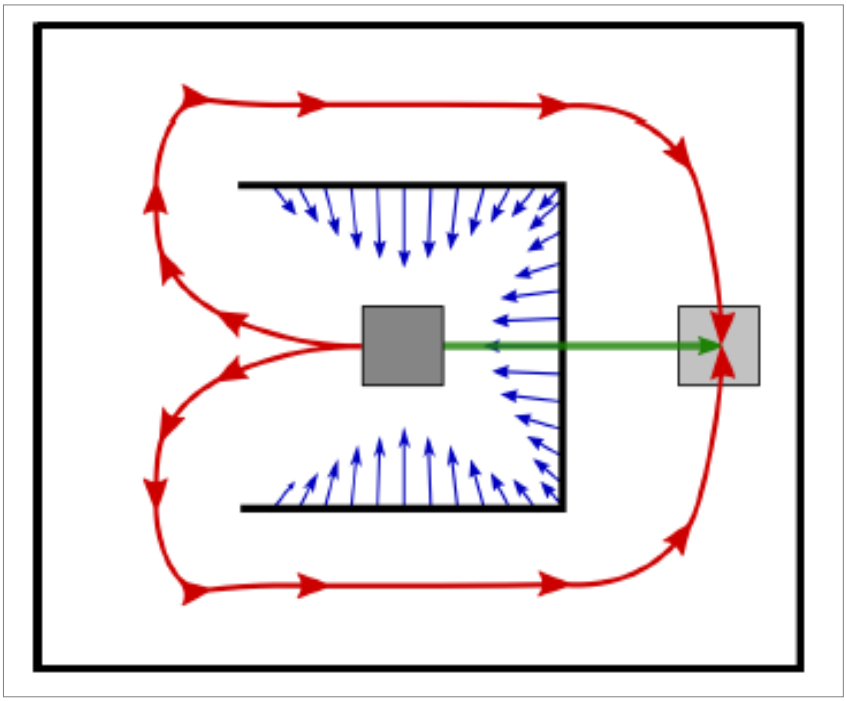

Figura 1: Robô estacionado em mínimo local. O quadrado é o robô e as linhas verdes e azuis mostram as forças potencias do campo – extraída de [7].

Na figura 1, as setas vermelhas indicam a rota que o robô intuitivamente deveria seguir. Verifica-se, que neste caso, as forças se anulam e o robô fica estacionado num local que não é o *destino*.

A função *Uatr* geralmente utilizada é o potencial cônico:

$$Uatr(q)=c*||q-qgoal||$$

onde *c* é um parâmetro de escala para o efeito atrativo e ||*q-qgoal*|| é a distância euclidiana entre o robô e o *destino*.

Algoritmo 1: APF

```
1 q (0) = start
2 i=0
3 while | AU (q (i) ) | > e do
4    q (i+1) = q (i) + a (i) * AU (q (i) )
5    i=i+1
```

O algoritmo 1 demonstra o funcionamento de um APF: inicialmente a posição do robô é a *origem* (linha 1) e então o robô vai se movendo na direção do vetor gradiente até que o vetor gradiente se aproxime muito de zero (linhas 3-5), ou seja, até o robô estacionar no *destino*.

2.2 Planejador por frente de onda (WP)

O método de *planejador por frente de onda (WP – Wavefront Planner)* [7] é simples e determinístico. Neste algoritmo, o ambiente é dividido em células através de grades. Inicialmente as células vazias são rotuladas com 0, as células com obstáculos são rotuladas com 1 e a célula *destino* é rotulada com 2 e a partir dela as células vizinhas são rotuladas com o valor do seu rótulo mais 1. O processo se repete com os vizinhos dos

vizinhos e segue adiante até que a *origem* do robô seja alcançada. A solução é encontrada partindo da configuração inicial em ordem decrescente até a célula do *destino*. (Figura 2)

12	11	10	9	10	11
11	10	9	8	1	1
10	1	1	7	1	1
9	1	1	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2

Figura 2: Exemplo de WP no mapa. A célula verde é a origem, a célula amarela é o destino e os quadrados vermelhos são os obstáculos.

A figura 2 mostra o funcionamento do planejador por frente de onda, preenchendo com número 1 as células com obstáculos, com número 2 o *destino*, com número 3 as células adjacentes ao *destino* e assim em diante até chegar na *origem*.

O WP é bastante eficiente, pois além de ser simples de compreensão e aplicação, ele evita de ficar preso num mínimo local e, ademais, sempre encontrará a rota solução (se ela existir). Além disso, a solução encontrada será sempre a solução ótima segundo o critério de Manhattan. Contudo, como as demais soluções determinísticas, é um algoritmo muito custoso e se torna um método impraticável para ambientes complexos. Além disso, este algoritmo apresenta uma desvantagem: o robô passa muitas vezes próximo aos obstáculos, o que pode ser muito perigoso devido aos erros de medições, atrito com o solo, condições climáticas etc.

A figura 3 mostra um exemplo de uma rota encontrada pelo algoritmo WP: pode-se notar que o robô realmente passa sempre bem rente às paredes do labirinto. A linha

vermelha é a rota traçada pelo robô que vai da *origem* (no canto superior esquerdo) até o *destino* (no canto inferior direito).

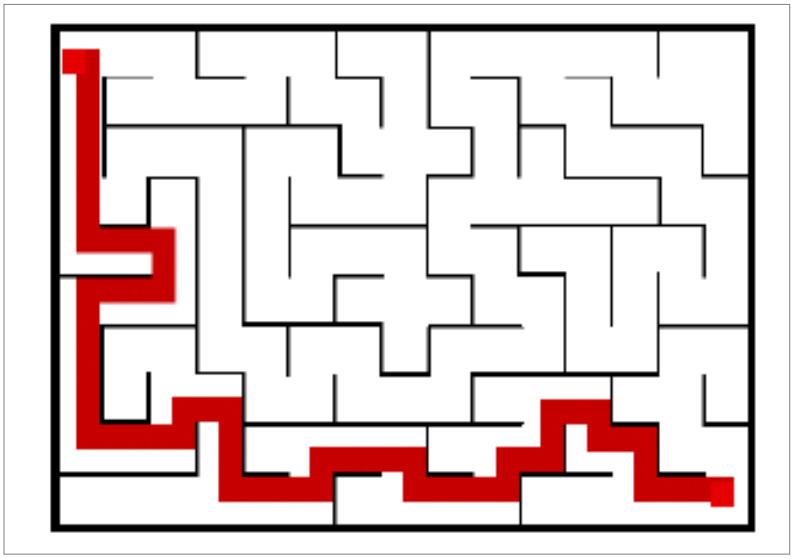

Figura 3: Rota encontrada pelo WP - extraída de [7].

2.3 Árvores aleatórias de exploração rápida (RRT)

O método *RRT* (Rapidly-Exploring Random Trees) [9] é muito utilizado em problemas de robótica que lidam com o movimento puro, sem referência às massas e forças envolvidas. Esse algoritmo é adequado para fazer buscas em ambientes de grandes dimensões, que possuam restrições provenientes dos obstáculos e restrições do movimento do robô. Uma de suas vantagens é que o algoritmo não somente leva o robô da *origem* ao *destino*, como também planeja a velocidade e aceleração do robô em sua trajetória. Isto é importante, pois um planejador convencional pode planejar uma rota impossível de ser feita

pelo robô (por exemplo, exigindo que o robô ande numa velocidade maior do que seu limite de velocidade, ou fazendo curvas mais fechadas do que o eixo do robô permite).

O algoritmo deve evitar, além das regiões ocupadas por obstáculos, as regiões em volta destes aonde são inevitáveis escapar de uma colisão (quando o robô está próximo demais de um obstáculo e não tem capacidade de desviar antes da colisão).

Dados a *origem* e o *destino* criam-se duas árvores cujos nós correspondem a estados aleatórios e que crescem da *origem* para o *destino* e do *destino* para a *origem*, respectivamente. O objetivo é fazer a fusão das árvores, resolvendo assim o problema.

Apesar de o RRT ter sido desenvolvido para planejamento no espaço de estados, ele pode ser também adaptado para buscas no espaço de configurações, nos casos em que a dinâmica não necessita ser levada em consideração para a resolução do problema.

Algoritmo 2: Palnejador de rotas RRT

```
1  Ta.init (qstart)
2  Tb.init (qgoal)
3  for k =1:K
4     gera uma configuração aleatória qrand em C de acordo com uma distribuição uniforme
5     if RRT-expand (Ta , qrand ) not TRAPPED
6        if RRT-connect (Tb , qrand ) = REACHED
7           return caminho (Ta , Tb )
8     else
9        swap (Ta , Tb)
10 return NULL
```

O objetivo do RRT é explorar uniformemente o ambiente, porém a solução é comumente encontrada bem antes do espaço ter sido completamente explorado. O algoritmo 2 mostra o funcionamento do planejador de rotas RRT. Nas linhas 1 e 2 são recebidas as configurações iniciais e finais do robô e na linha 4 uma configuração é

escolhida aleatoriamente. Na linha 5 é feita a expansão da árvore e na linha 6 é verificado se as duas árvores (com raiz na *origem* e com raiz no *destino*) podem se fundir. Se a solução ainda não foi encontrada então se troca as árvores de local (linha 9) e repete-se o procedimento com a outra árvore.

Algoritmo 3: RRT-expand

```
 1 qnear = vizinho_mais_próximo (qrand , T)
 2 if distancia (qnear , qrand ) > e
 3    Gera qnew a uma distância e de qnear no sentido qnear -> qrand
 4 else
 5    qnew = qrand
 6 if planejador_local_caminho_reto (qnear , qnew )
 7    T.add_vertex (qnew )
 8    T.add_edge (qnear , qnew )
 9    if qrand = qnew
10       return REACHED
11    else
12       return ADVANCED
13 else
14    return TRAPPED
```

O algoritmo 3 faz a expansão do RRT. O parâmetro *e* é muito importante para o algoritmo, pois caso seja muito pequeno o algoritmo vai demorar a convergir e, caso seja muito grande, muitas amostras podem ser rejeitadas. O algoritmo 4 faz a tentativa de fusão das árvores. Ambos os algoritmos (expansão e fusão) podem ser utilizados também separadamente para a resolução do problema.

Algoritmo 4: RRT-connect

```
 1 qnear = vizinho_mais_próximo (q , T)
 2 if planejador_local_guloso (qnear , q , qnew)
 3    T.add_vertex (qnew)
 4    T.add_edge (qnear , qnew)
 5    if q = qnew
 6       return REACHED
 7    else
 8       return ADVANCED
 9 else
10    return TRAPPED
```

Uma das desvantagens do RRT é que para cada nova rota requisitada o algoritmo deve recomeçar tudo do início, embora em alguns casos específicos partes destas rotas possam ser salvas para reaproveitamento em outras rotas.

2.4 Árvores em espaços expansivos (EST)

A idéia das *EST (Expansive-Space Trees)* [10] é bastante semelhante a do RRT, partindo da construção de duas árvores (uma raiz na *origem* e outra no *destino*) e tentando interligá-las.

Para expandir as árvores, uma configuração é selecionada seguindo alguma distribuição de probabilidade pré-definida e após isto é aleatoriamente escolhida alguma configuração em sua vizinhança para ser a nova configuração. A definição de tamanho desta vizinhança é muito importante, pois se esta for pequena demais o algoritmo pode demorar a convergir, porém caso seja grande demais podem ser amostradas áreas que não sejam muito relevantes. Uma distribuição de probabilidade muito utilizada é a distribuição

de probabilidade inversamente proporcional ao número de vizinhos de uma determinada configuração.

O algoritmo 5 mostra a construção das árvores. A idéia é que uma árvore cresça a partir da *origem* e outra cresça a partir do *destino*.

Algoritmo 5: Build-EST

```
1 Adiciona q0 à árvore T
2 for i=1:N
3   qrand = configuração aleatória escolhida em T com probabilidade p (qrand)
4   qnew = configuração aleatória na vizinhança de qrand , tal que dist (qnew , qerand) < d
5   if planejador_local (qrand , qnew)
6     adiciona qnew a T
7     adiciona uma borda ligando qrand a qnew
8 return T
```

O algoritmo 6 tenta a fusão entre as duas árvores, tentando ligar todos os nós de cada uma das árvores que estejam suficientemente próximos entre si. Desta forma, somente os nós próximos são passados para o navegador local, diminuindo o custo do algoritmo.

Algoritmo 6: Merge-EST

```
1 for all qa pertencente a Tstart e a Tend
2   if dist (qa , qb) < l
3     if planejador_local (qa , qb)
4       return PATH
```

Algo que também causa grande impacto na velocidade do algoritmo 6 é a escolha da distribuição de probabilidade do novo nó a ser selecionado. A escolha da distribuição de probabilidade é muito importante, pois além de afetar a velocidade do algoritmo afeta diretamente o seu desempenho. Se esta função polarizar fortemente a amostragem em zonas

pouco exploradas, pode haver problemas com mínimos locais, prejudicando a construção da árvore. Por outro lado, se a distribuição for uniforme demais, a tendência é que mais configurações sejam geradas em ambientes já bastante explorados.

Independentemente dos parâmetros utilizados, o EST costuma produzir soluções com bastante "zigue-zagues".

2.5 Passeio aleatório (ARW)

O algoritmo *ARW (Adaptative Randon Walk)* [12] é um planejador que não exige etapa de pré-processamento. As principais vantagens do ARW são a eficiência do tempo de execução e garantia de convergência para a solução (caso exista). Este algoritmo vai explorando o espaço livre até que a última amostra seja conectada ao *destino* pelo navegador local. Todas as amostras vão sendo armazenadas, formando uma cadeia, e quando a última amostra é ligada ao destino utiliza-se desta cadeia para encontrar a rota solução.

O ARW é um processo estocástico discreto, sendo um passeio aleatório partindo da *origem* e com o objetivo de alcançar o *destino*. A evolução segue equações recursivas, onde inicialmente é feito:

$$q[0]=qstart$$

e posteriormente segue-se com:

$q[k]=g*(q[k-1],v[k])$, para k=1,2,3,...

aonde, se *q[k-1]* pode se ligar a *q[k-1] + v[k]* pelo planejador local, a função *g* é dada por *q[k-1]+v[k]*, caso contrário a função *g* é *q[k-1]*.

A variável *v[k]* é gerada seguindo uma normal de média 0 e variância *E(k)*, onde *E(k)* é a matriz de covariância adaptativa do processo.

Pode-se também adotar uma abordagem bidimensional, onde dois passeios aleatórios são iniciados (um da *origem* outro do *destino*) e a rota é encontrada ou quando o passeio partindo da *origem* chega ao *destino*, ou quando o passeio partindo do *destino* chega à *origem*, ou ainda quando os dois passeios se cruzam em algum ponto (neste caso a soma dos dois passeios será a rota final). A principal desvantagem do ARW é a baixa qualidade da rota resultante.

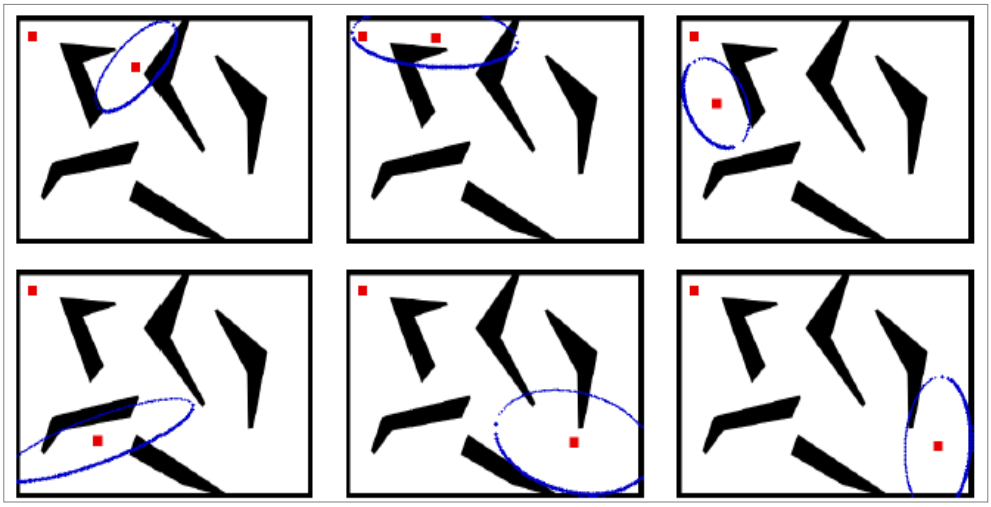

Figura 4: Adaptação da variância em diversos pontos do espaço – extraída de [7].

A figura 4 mostra a importância da estimativa iterativa da variância. Em cada etapa da rota o robô fica a distâncias diferentes dos obstáculos mais próximos. Por exemplo, no primeiro quadro (canto superior esquerdo), o robô está bem próximo a dois obstáculos, enquanto no último quadro (canto inferior direito) ele tem uma área livre de obstáculos um pouco maior ao seu redor. Sendo assim, se a variância do último quadro fosse aplicada no primeiro quadro, muitos pontos seriam gerados dentro de obstáculos e seriam descartados. Da mesma forma, se a variância do primeiro quadro fosse aplicada no último quadro, o próximo nó selecionado seria provavelmente muito próximo ao nó anterior, o que não é bom.

Algoritmo 7: ARW simples

```
 1 k = 0
 2 qk = qstart
 3 E(0) = Emin
 4 while not planejador_local (qk , qgoal)
 5     Gerar uma nova configuração aleatória vk pertencente a N (0,Ek)
 6     s = qk + vk
 7     se planejador_local (qk , s)
 8         k = k + 1
 9         qs = s
10         Atualizar a matriz de covariância Ek
11         Inserir qk na lista de configurações intermediárias
```

O algoritmo 7 mostra o funcionamento de um passeio aleatório. Nas linhas 1 e 2 são inicializada a variável *q* (localização do robô) e a matriz de covariância. Nas linhas 3-5, enquanto o navegador local não ligar a localização atual do robô com o *destino*, é gerada uma configuração aleatória. Na linha 6 verifica-se se o navegador local encontra uma rota entre a configuração atual do robô e a configuração aleatória gerada em torno dele, caso esta seja encontrada nas linhas 7-10 o robô se move para esta nova configuração, atualiza a

matriz de covariância e insere a configuração anterior na lista de configurações intermediárias.

Uma das principais vantagens do ARW é a simplicidade de implementação. Além disso, como não tem que procurar os vizinhos mais próximos, o custo de uma nova amostra é bem baixo. O ARW também se adapta a estrutura do espaço de configurações para gerar uma nova amostra, permitindo que o algoritmo extraia informações sobre a região onde o passeio aleatório se localiza e polarizando assim a amostragem no *C-free*.

A maior desvantagem do ARW é a qualidade das rotas geradas, cheias de "zigue-zagues" e voltas desnecessárias, precisando de um pós-processamento para corrigir este problema.

2.6 Breve Análise dos métodos de planejamento de rotas

Todos os métodos de planejamento de rotas apresentados possuem desvantagens. O *APF* fica preso muitas vezes em mínimos locais. O WP é um método exato, porém inviável para ambientes complexos. Os métodos que utilizam árvores (*RRT* e *EST*), têm que recomeçar todo o processo caso a *origem* ou o *destino* sejam alterados (mesmo que esta alteração seja milimétrica). O método ARW gera rotas grandes, muitas vezes com "zigue-zagues" e voltas desnecessárias.

O PRM-EB, proposto nesta dissertação, utiliza como rota inicial a rota encontrada por um PRM. O PRM não esbarra nos problemas de mínimos locais, é perfeitamente viável para ambientes complexos, aproveita a maior parte de sua estrutura caso uma nova rota precise ser encontrada (mesmo com mudanças bruscas de locais de *origem* e *destino*),

porém apresenta rotas nem sempre tão eficientes quanto às outras técnicas apresentadas. Para resolver esta deficiência do PRM, o PRM-EB utiliza a rota inicial do PRM com uma técnica baseada na teoria de apostas, que otimiza o tamanho desta rota, obtendo então uma solução mais eficiente.

3 Teorias de suporte

Os algoritmos evolutivos são os que teoricamente se baseiam na Teoria de Darwin da evolução das espécies. Seguindo a analogia da evolução, o método supõe que, assim como os animais vão sofrendo mutações e se adaptando mais ao ambiente conforme o passar do tempo, também as soluções vão se adaptando ao ambiente com o passar do tempo. Num ambiente estático, podemos dizer que esta adaptação será uma melhora para aquele ambiente.

Muitos tipos de algoritmos evolutivos já foram propostos [19, 25, 31], como o clássico algoritmo genético, algoritmos de otimização por partículas etc. O algoritmo genético, por servir de suporte teórico principal ao algoritmo original proposto nesta dissertação, será detalhado a seguir.

3.1 Algoritmos genéticos

Algoritmos Genéticos (AG) possuem uma estrutura muito eficiente e têm sido muito estudado nas últimas décadas. Eles são cada vez mais utilizados em diversos tipos de problemas. Um AG se baseia basicamente em três funções principais: seleção de indivíduos para o cruzamento, cruzamento e mutação. Se na natureza apenas os indivíduos mais adaptados tendem a propagar mais os seus genes, assim também funciona na analogia matemática. A principal diferença da natureza para o algoritmo matemático é que este faz tudo isso de uma forma muito mais simplória e mais rápida.

Cada indivíduo é uma sequência de genes definidos por determinada codificação. Uma das codificações mais comumente utilizada é a codificação binária, que codifica a solução de um problema num vetor de "zeros" ou "uns". Outras codificações podem ser utilizadas, mas nesta dissertação será utilizada a codificação binária. Cada solução do problema é um indivíduo na população.

O algoritmo funciona da seguinte forma (como exemplifica a figura 5): inicialmente algumas soluções aleatórias são geradas (obedecendo às restrições do problema) e a partir daí, as melhores soluções são escolhidas para gerarem novos indivíduos (soluções) da próxima geração. Estes indivíduos fazem o cruzamento e após a geração dos novos indivíduos, estes sofrem algumas alterações em seus genes (mutação) para então se ter definitivamente a nova geração de indivíduos. O algoritmo utiliza esta nova geração na próxima iteração e assim vai iterando até que algum critério de parada previamente definido seja satisfeito. Os critérios de parada mais utilizados são: solução boa o suficiente, melhora muito pequena na otimização (de uma geração para a posterior), ou um limite no número de iterações.

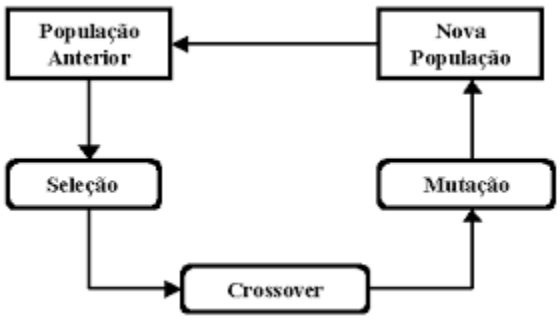

Figura 5: Ciclo de um algoritmo genético – extraído de [37].

3.1.1 Seleção

Após a formação de uma nova geração, cada um dos indivíduos gerado é testado através de uma função de avaliação, que de alguma forma avalia este indivíduo em alguma escala de utilidade (se ele é ou não um bom indivíduo para o cruzamento). A seleção é geralmente feita de alguma forma tal que todos os indivíduos tenham chances de serem selecionados para o cruzamento, porém são os melhores indivíduos que tem chances mais altas. Um dos métodos mais conhecidos para seleção é o *método da roleta* que divide uma roleta imaginária em várias partes, com fatias diferentes. Quanto melhor for avaliado um indivíduo, maior será sua fatia na roleta e assim ele terá mais chances de ser escolhido para o cruzamento.

Na figura 6, por exemplo, os indivíduos 1 e 3 possuem as maiores chances de serem selecionados para o cruzamento (pois são os indivíduos mais bem avaliados), seguidos dos indivíduos 5 e 4. O indivíduo 2 possui chances bem menores.

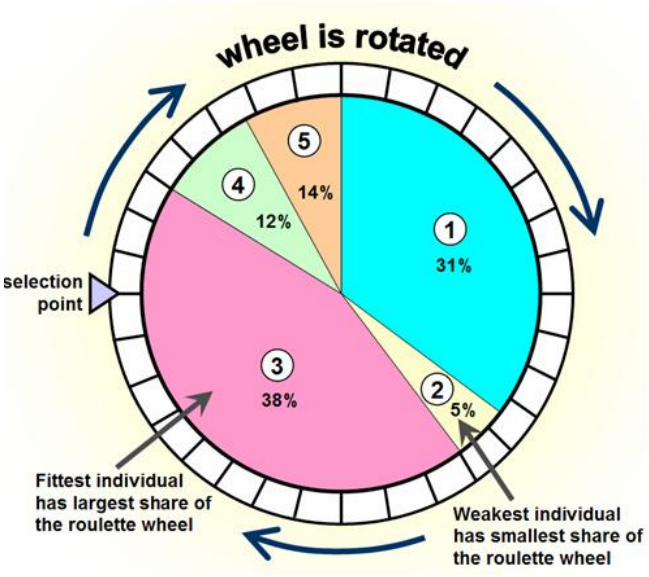

Figura 6: Método de roletas para seleção de algoritmo genético – extraída de [38].

3.1.2 Cruzamento

Para o cruzamento, utiliza-se geralmente um par de indivíduos para gerar cada novo par de indivíduos, ou seja, caso cada geração tenha vinte indivíduos, serão selecionados dez pares e cada par gerará dois indivíduos. Um indivíduo pode estar em mais de um dos dez pares (e isto muito provavelmente ocorrerá com os melhores indivíduos de uma geração), por outro lado, um indivíduo pode não estar em nenhum dos dez pares. O primeiro indivíduo gerado do cruzamento é a primeira metade do vetor de genes do primeiro indivíduo do par selecionado, com a segunda metade do vetor de genes do segundo indivíduo do par selecionado. Da mesma forma, o segundo indivíduo gerado é formado pela primeira metade do vetor de genes do segundo indivíduo do par selecionado com a segunda metade do vetor de genes do primeiro indivíduo do par selecionado. Na figura 7, o

vetor de cores quentes (roxa e amarela) é o primeiro indivíduo do par selecionado e o vetor de cores frias (verde e azul) é o segundo. O primeiro indivíduo gerado do cruzamento é a primeira metade do vetor de cores quentes com a segunda metade do vetor de cores frias, já o segundo indivíduo gerado é a primeira metade do vetor de cores frias com a segunda metade do vetor de cores quentes.

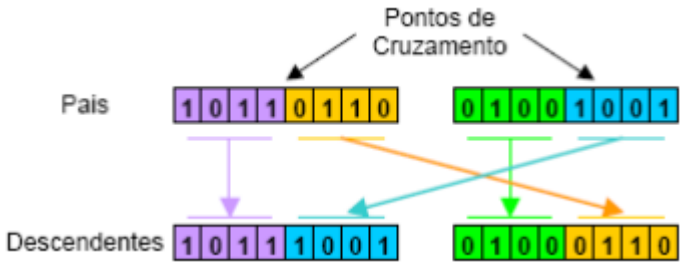

Figura 7: Exemplo do cruzamento de dois indivíduos de um algoritmos genético – extraída de [39].

3.1.3 Mutação

Após o cruzamento, os novos indivíduos sofrem a mutação, que geralmente é feita da seguinte forma: uma probabilidade bem pequena de modificação de cada gene é definida (geralmente algo menor que 5%), e cada um dos genes é aleatoriamente alterado ou não seguindo esta probabilidade. No caso da codificação binária, esta mudança é simplesmente transformar 0 em 1 ou vice-versa (Figura 8). Em casos de utilização de outras codificações esta mutação pode ser mais complexa.

Após a mutação, os indivíduos são testados pela função de avaliação, para que assim possa começar um novo ciclo de seleção, cruzamento e mutação.

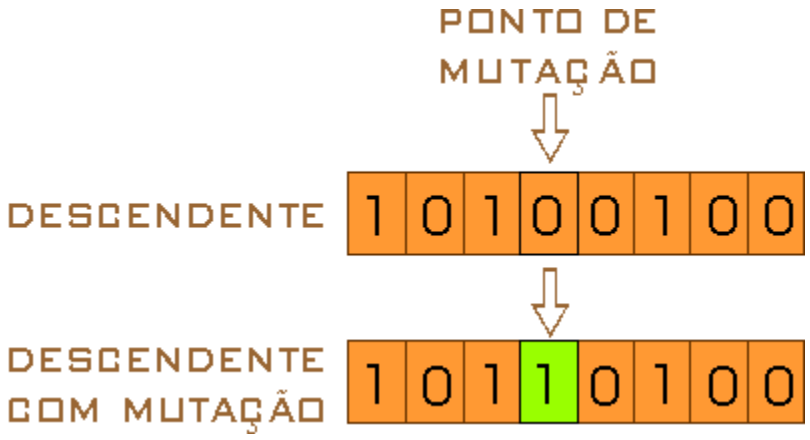

Figura 8: Exemplo de mutação de um indivíduo do algoritmos genético – extraída de [40].

A figura 8 mostra um exemplo de mutação, aonde só o quarto bit é selecionado para a mutação. Neste caso, como o quarto bit estava preenchido com 0 passou a estar preenchido com 1.

3.1.4 Vantagens e desvantagens

A técnica dos algoritmos genéticos tem como uma de suas principais vantagens a flexibilidade, o que permite de trabalhar em conjunto com outras técnicas de otimização. Além disso, pode-se formar uma infinidade de algoritmos genéticos diferentes para se resolver cada problema, deixando a escolha de número de indivíduos por geração, critérios de paradas, codificação do indivíduo, função de avaliação dos indivíduos, formas de seleção, formas de cruzamento e formas de mutação como itens que podem ser escolhidos livremente, buscando a melhor configuração para cada tipo de problema.

Uma técnica bastante utilizada nos algoritmos genéticos é a técnica do elitismo. Nem sempre o melhor indivíduo da geração nova vai ser melhor do que o melhor indivíduo da geração anterior, embora esta seja uma tendência geral. Neste caso, se não fizermos nada, podemos perder a melhor solução encontrada no meio do caminho. Para que isto não ocorra, muitas vezes o algoritmo genético utiliza o elitismo que seleciona geralmente os dois melhores indivíduos de cada geração diretamente para a próxima geração, excluindo assim algum par de nós selecionado para o cruzamento, pois, vale ressaltar, o tamanho da população do algoritmo não aumenta nem diminui enquanto as iterações vão ocorrendo.

O algoritmo genético possui alguns pontos fracos como, por exemplo, a dificuldade de se escolher uma boa função de codificação e uma boa função de avaliação. Além disso, o algoritmo (assim como os demais algoritmos evolutivos) muitas vezes converge para ótimos locais.

3.2 Algoritmo de apostas

Demasi [19] propôs um novo tipo de algoritmo evolutivo, baseado em teorias de apostas. O algoritmo de apostas é uma meta-heurística evolutiva que tem por objetivo a otimização de funções explorando técnicas de apostas. Neste algoritmo, o que evolui não é a solução (não diretamente), mas sim os apostadores. Os apostadores vão fazendo suas apostas em algumas soluções possíveis e os melhores apostadores vão sendo recompensados, enquanto os piores apostadores vão perdendo seu cacife. Desta forma, os melhores apostadores vão ficando no jogo e os piores vão saindo, dando lugar a outros

possíveis bons jogadores. Sendo assim, com apenas os melhores jogadores ficando no jogo, as soluções encontradas tendem a ficar cada vez melhores.

Em geral, no mercado de apostas, as apostas são feitas da seguinte forma: existe um evento no qual se deve apostar, e existe uma banca que oferece as cotações para cada aposta. Por exemplo, partida de futebol entre o time A e o time B, a banca X pode oferecer as seguintes cotações: vitória do time A (1.2 para 1), vitória do time B (5.7 para 1) e empate (2.9 para 1). Neste caso, se o apostador apostar R$50,00 na vitória do time A e o time A ganhar, ele recebe da banca 1.2 * R$50,00 = R$60,00 gerando um lucro de R$10,00 (já que R$50,00 foram pagos anteriormente no momento em que apostou). A cotação define quantas vezes o valor da aposta o apostador vai receber em caso de acerto. Eventos mais prováveis possuem cotações pequenas (muito próximas de 1, que é o mínimo, pois o apostador precisa receber mais do que pagou caso vença a aposta, para que faça algum sentido apostar) e eventos menos prováveis possuem cotações maiores (para incentivar a aposta).

O modelo proposto por Demasi se baseia em algumas técnicas utilizadas por apostadores reais como, por exemplo, *dutching* e a arbitragem. O *dutching* consiste em fazer duas ou mais apostas diferentes no mesmo evento (ex: numa corrida de cavalos com oito cavalos, o apostador aposta uma quantia no cavalo 2 e outra no cavalo 5, de forma que ele tenha lucro caso qualquer um dos dois venha a ganhar) e a arbitragem consiste em fazer duas ou mais apostas no mesmo evento em bancas de apostas diferentes. Por exemplo, supondo uma final de um campeonato de futebol entre o time A e o time B. A banca X oferece as seguintes cotações: time A campeão (1.2 para 1) e time B campeão (2.5 para 1), enquanto a banca Y oferece as seguintes cotações: time A campeão (1.8 para 1) e time B campeão (1.5 para 1). Neste caso, um apostador utilizando a arbitragem pode ir até a banca

X e apostar R$15,00 no título do time B, depois vai a banca Y e aposta R$20,00 no título do time A. Desta forma, terá lucro de qualquer forma, pois se o time A ganhar ele vai ter lucro de R$16,00 na banca Y e prejuízo de R$15,00 na banca X, totalizando um lucro para o jogador de R$1,00. Caso o time B seja o campeão ele vai ter um lucro de R$22,50 na banca X e um prejuízo de R$20,00 na banca Y, totalizando um lucro de R$2,50 para o jogador. Ou seja, explorando diferentes bancas o jogador consegue obter lucro sempre, independente do resultado, caso utilize a arbitragem corretamente.

3.2.1 Conceitos básicos

No contexto de apostas uma das coisas mais importantes que precisam ser definidas é como será feita a representação de cada solução. Demasi propôs que isto fosse feito de maneira muito parecida com a dos algoritmos genéticos, ou seja, cada solução será representada por um vetor de *n* bits. Sendo assim o espaço-solução a ser explorado será de 2^n.

Um dos conceitos mais importantes do algoritmo é a definição da máscara. A máscara *M* é um vetor de tamanho |*M*| (que será alterada a cada rodada de apostas) que define quais bits podem ser alterados naquela rodada. Por exemplo, supondo uma cadeia de bits de tamanho 8 com uma máscara de tamanho 3, uma máscara selecionada aleatoriamente pode ser *[1,5,7]*. Desta forma, só os bits 1, 5 e 7 poderão ser alterados (ou não) pelos apostadores nesta rodada de apostas. Neste caso a aplicação da máscara pode ser dada de 8 formas (*2^|M|*), ou seja, não modificando nenhum bit, modificando somente o primeiro, somente o segundo, somente o terceiro, somente os dois primeiros, somente o

primeiro e o terceiro, somente os dois últimos ou modificando todos os bits. Cada jogador vai escolher as aplicações de máscaras que lhe parecerem as melhores. Por default foi utilizada nesta dissertação cada aplicação de máscara *i* como (*i-1)* convertido para número binário. Por exemplo, a aplicação de máscara 1 é a representação do número 0 em binário, ou seja, *[0, 0, 0]*, sendo assim nenhum bit é alterado, já a aplicação de máscara 5 é a representação do número 4 em binário, ou seja, *[1, 0, 0]*, modificando assim somente o primeiro bit.

Na figura 9 é apresentado um exemplo no qual a máscara selecionada é *[3, 4, 6]*. Sendo assim, na respectiva iteração só podem ser feitas mudanças nestes bits e só estes são contabilizados nos cálculos dos pesos dos jogadores e da cotação. O restante da matriz é temporariamente inutilizado.

	p1	p2	p3	p4	p5	p6
Jogador1	0,80	0,85	0,05	0,30	0,60	0,50
Jogador2	0,40	0,10	0,95	0,45	0,55	0,70
Jogador3	0,20	0,05	0,45	0,40	0,90	0,75

Figura 9:Exemplo de máscara selecionada.

Os jogadores são definidos como um vetor de probabilidades de tamanho *n*, onde cada probabilidade representa a probabilidade da mudança de um bit melhorar a solução segundo determinado jogador. Ou seja, se o jogador 1 é representado pelo vetor *[0.8, 0.85, 0.05, 0.3, 0.6, 0.5]* significa que ele acha que mudar os bits 1, 2 e 5 provavelmente vai melhorar a solução (isso porque ele supõe que as probabilidades das mudanças desses bits melhorarem a solução são maiores do que 0.5), diferentemente da mudança dos demais bits. Com este vetor de probabilidades cada jogador pode calcular os pesos para cada aplicação

da máscara, ou seja, definir quais aplicações de máscaras tem mais chances de produzir boas soluções. Para calcular os pesos dos jogadores utiliza-se:

$$w_i = (\prod(p(m_j))) \times (\prod(1-p(m_k)))$$

onde *p* é o vetor de probabilidades que representa cada jogador, *mj* é um vetor com todos os bits ativados por aquela transformação da máscara e m_k o vetor com todos os bits desativados por aquela transformação. Supondo então o jogador 1 com *p = [0.8, 0.85, 0.05, 0.3, 0.6, 0.5]* e uma máscara que seja *M = [3,4,6]*, os pesos deste jogador ficariam da seguinte forma:

$$w1 = 0.05 * 0.3 * 0.5 = 0,0075$$

$$w2 = 0.05 * 0.3 * (1 - 0.5) = 0,0075$$

$$w3 = 0.05 * (1 - 0.3) * 0.5 = 0,175$$

$$w4 = 0.05 * (1 - 0.3) * (1 - 0.5) = 0,175$$

$$w5 = (1 - 0.05) * 0.3 * 0.5 = 0,1425$$

$$w6 = (1- 0.05) * 0.3 * (1 - 0.5) = 0,1425$$

$$w7 = (1 - 0.05) * (1 - 0.3) * 0.5 = 0.3325$$

$$w8 = (1 - 0.05) * (1 - 0.3) * (1 - 0.5) = 0,3325$$

Nota-se que o jogador 1 tende a apostar nas máscaras 7 e 8, por acreditar que somente a mudança do último nó (dentre os nós que estão na máscara) possa ser boa para a solução.

Após todos os jogadores calcularem seus pesos, a banca utiliza todos estes pesos para calcular as cotações das apostas e cada apostador aposta somente nas opções de aplicação de máscara nas quais a cotações lhes são favoráveis, utilizando o critério de Kelly:

$$\frac{(p'w-1)*p}{w-1}$$

O critério de Kelly foi definido pelo físico americano John Kelly, visando descobrir o valor ideal que se deve apostar em determinado evento para maximizar o lucro a longo prazo [46].

Pode-se notar que quando $p'=\frac{1}{w}$, o critério de Kelly propõe uma aposta de valor igual a 0. Caso $p'<\frac{1}{w}$, o critério de Kelly propõe um valor de aposta negativo, o que é impossível. Isto ocorre porque nunca vale a pena apostar quando $p'<\frac{1}{w}$.

3.2.2 O Método

O algoritmo de apostas pode ser realizado em duas fases, ainda que a segunda fase seja opcional. Na primeira fase os apostadores vão apostando nas aplicações de máscaras e os melhores jogadores vão sobrevivendo, enquanto os piores vão sendo eliminados. Quando um jogador é eliminado outro jogador é criado e inserido no lugar deste. Este novo jogador pode ser criado aleatoriamente ou baseado no melhor jogador do jogo até o

momento. A segunda fase é executada com um número pequeno de melhores jogadores da primeira fase e ao invés de utilizar as máscaras para fazer transformações na solução, as transformações são feitas diretamente através do vetor de probabilidades de cada jogador. Com efeito, este jogador provavelmente já deve ter um vetor de probabilidades muito bom (caso contrário ele não estaria entre os melhores jogadores finais)

O fluxograma da figura 10 ilustra o funcionamento da primeira fase do algoritmo de apostas. Os passos são:

1. Selecionar solução aleatória inicial
2. O critério de parada foi satisfeito?
3. Fim
4. Selecionar máscara aleatória
5. Calcular os pesos dos jogadores para cada escolha
6. Calcular as apostas dos jogadores e efetivá-las
7. Aplicar transformações à melhor solução e atualizar seu valor se pertinente
8. Pagar os prêmios das apostas vencedoras dos jogadores
9. Remover jogadores que quebram, substituindo-os por novos.

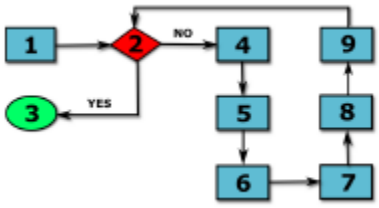

Figura 10: Fluxograma do Algoritmo de apostas – extraída de [19].

O algoritmo 8 mostra o funcionamento da primeira fase do algoritmo de apostas: inicialmente é gerada uma solução aleatória, uma máscara para cada iteração é selecionada (linha 2), e então os jogadores calculam seus respectivos pesos (linhas 3-7). A banca calcula a cotação geral de cada aposta (linhas 8-10) e cada jogador faz suas apostas (linhas 11-18). As novas soluções, aquelas em que os jogadores apostaram, são avaliadas (linhas 19-26). O jogador que apostou na solução vencedora (melhor avaliada) recebe sua recompensa (linhas 27-30) e caso algum jogador esteja com o cacife zerado é substituído por um novo jogador (linhas 31-35). A melhor solução encontrada substitui a melhor solução atual, caso seja efetivamente melhor (linhas 36-38), e o algoritmo repete o processo até que um critério de parada seja satisfeito.

Algoritmo 8: Primeira fase do algoritmo de apostas

```
 1 for ni=1:número de iterações
 2   Seleciona máscara aleatória
 3   for i=1:número de jogadores
 4     for j=1:tamanho da máscara
 5       Calcula pesos w (i,j)
 6     end
 7   end
 8   for j=1:tamanho da máscara
 9    Calcula cotação da banca w (j)
10   end
11   for i=1:número de jogadores
12     for j=1:tamanho da máscara
13       if w (j) > w (i,j)
14         Calcula va_ck (valor de aposta segundo critério de Kelly)
15         cacife (i) = cacife (i) - va_ck
16       end
17     end
18   end
19   melhor_caminho = 0
20   for m=1:tamanho da máscara
21     Avalia caminho m (caminho_atual após aplicação da máscara m)
22     if caminho m melhor que melhor_camiho
23       melhor_caminho = caminho m
24       melhor_indice = m
25     end
26   end
27   for i=1:número de jogadores
28     if jogador i apostou na máscara m
29       cacife (i) = cacife (i) + va_ck * w(m)
30     end
31     if cacife (i) = 0
32       Remove jogador i do jogo
33       Insere novo jogador no jogo
34     end
35   end
36   if melhor_camiho é melhor que caminho_autal
37     caminho_atual = melhor_caminho
38   end
39 end
```

O algoritmo 9 mostra o funcionamento da segunda fase do algoritmo de apostas: cada bit da solução é alterado ou não, seguindo o vetor de probabilidades dos jogadores sobreviventes da primeira fase (linha 5) e novas soluções (cada uma gerada por um jogador) são avaliadas (linha 7). A melhor das novas soluções substitui a solução anterior

(caso seja efetivamente melhor) e o algoritmo segue iterando até que um critério de parada seja satisfeito.

Algoritmo 9: Segunda fase do algoritmo de apostas

```
1 for nj =1:número de iterações da segunda fase
2    for i =1:número de jogadores
3    caminho = caminho_aual
4       for t =1:tamanho do caminho
5          Modifique o bit t de caminho com probabilidade p(i(t))
6       end
7    Avalia se o novo caminho é melhor que o antigo
8    end
9 end
```

4 Probabilistic roadmaps (PRM)

Neste capítulo são apresentados alguns dos fundamentos básicos do PRM, que são de suma importância para a elaboração do algoritmo PRM-EB.

4.1 Definições

Dado um *C-free* de um ambiente contendo um robô, um PRM conceitualmente consiste em relacionar (amostrando) pontos no *C-free* que serão os futuros nós do grafo (figura 11). Selecionado um nó, os nós mais próximos a ele são definidos como os *vizinhos mais próximos*. O *número de dimensões* do *C-free* é definido como número de *graus de liberdade do robô*. O *grafo* gerado é definido como o conjunto de nós e arestas que formam o *mapa probabilístico* para o ambiente. Cada rota entre dois nós é uma *aresta*, mas esta só é colocada no grafo se a rota que liga os dois nós não atravessa obstáculos. Este teste de aresta livre é definido como *detector de colisões*, feito pelo *navegador local*, que gera as arestas. O mapa probabilístico, obtido após o término da amostragem de nós e da inclusão de arestas, é definido como *mapa resultante* (ou "o PRM").

Dados dois nós $\boldsymbol{n_i}$ e $\boldsymbol{n_j}$, diz-se que eles pertencem ao *mesmo componente* se e somente se existe pelo menos uma rota entre eles via uma aresta direta ou passando por outros nós. Um grupo de nós é dito *independente* se e somente se não existe aresta o ligando a outro grupo. O *número de componentes de um mapa* é definido como o número de grupos de nós que são independentes. O mapa da figura 11, por exemplo, tem três componentes: o primeiro

possui apenas o nó *(25,10)*; o segundo os nós *(14,78) e (8,79)*; e o terceiro os outros nós. Os nós *(4,17) e (86,19)*, ainda que distantes entre eles (sem aresta os ligando diretamente), pertencem ao mesmo componente porque existe rota entre eles (passando por outros nós) [43].

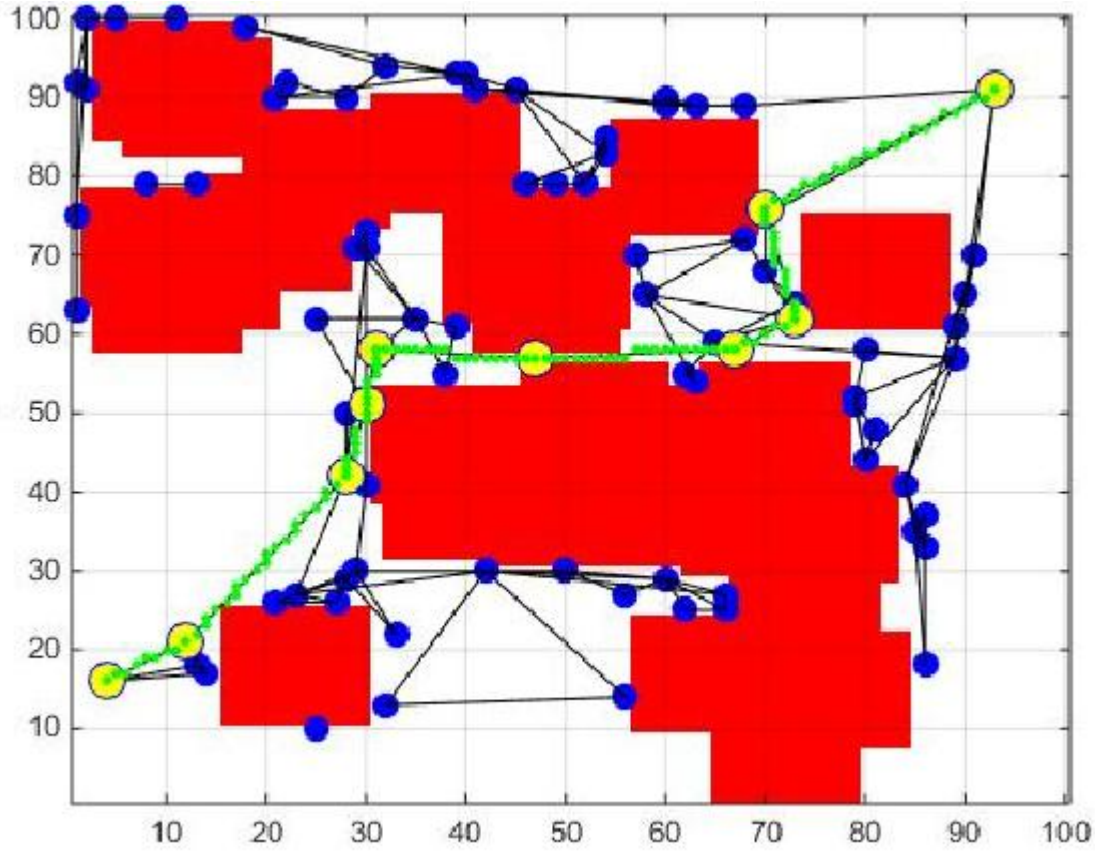

Figura 11: PRM: ponto azul ou amarelo é um nó; quadrados vermelhos são obstáculos e a linha verde é a rota para o robô.

4.2 Estado da arte - PRM

Existem vários métodos propostos para geração de mapas no PRM e eles possuem diversas características e métricas. A comparação entre esses métodos é bastante árdua, pois requer múltiplos cenários. Faz-se necessário estabelecer quais métricas devem ser utilizadas e analisar detalhadamente os resultados. Um teste promissor para comparar dois métodos deve evitar um grande número de variáveis correlacionadas e buscar medir a eficiência e completude de cada método usando também mapas simples. O mapa mais simples é aquele gerado com apenas duas dimensões num ambiente estático. Em [6] é detalhado um aprimoramento de amostragem de nós; em [15] melhorias para a escolha dos vizinhos; em [5] a redução do mapa; em [4] uma aplicação com obstáculos dinâmicos. Rantanem [17] mostra melhorias na amostragem de nós; Salzman et al. [13] e Dobson e Bekris [16] buscam reduzir o tamanho do mapa; Kallmann e Mataric [18] estudaram PRM em ambientes dinâmicos.

Em [15], Rantanem e Juhola propõem uma escolha de vizinhos via o método LSH (*Locality-Sensitive Hashing*) que busca os vizinhos mais próximos de cada nó na fase de construção do mapa (acelerando a construção, notadamente em ambientes com muitas dimensões, sem perder muito a qualidade do mapa). Para cada nó $\boldsymbol{n_i}$ selecionado é calculado um valor através de uma função hash (previamente definida) que agrupa nós por proximidade. Quando um novo nó é selecionado, calcula-se o valor e os nós selecionados como vizinhos são os nós do mesmo grupo dele na tabela hash. Este método é rápido, porém não garante que encontrará o vizinho realmente mais próximo. O principal problema é conseguir uma boa função hash.

Em [13], Salzman et al. verificaram que PRM utilizados em ambientes com muitas dimensões costumam ser muito densos (muitos nós e arestas), e propuseram a redução do mapa minimizando a perda de qualidade. Para fazer esta redução eles utilizaram o *Roadmap Sparsification by Edge Contraction* (RSEC) cuja técnica é a de ir "contraindo" o mapa enquanto analisa os vértices e arestas. Essa "contração" se passa através da seleção de uma aresta (***v'***,***v''***) que passa a inexistir (assim como ***v'*** e ***v''***) via a geração de um novo nó (***v'''***), de tal forma que ele possa se conectar com todos os nós que tinham conexão com os nós ***v'*** e ***v''***, diminuindo o número de nós e de arestas. Salzman et al. afirmam que para alguns cenários a contração do mapa pode ultrapassar 97% com um custo de degradação < 4%.

Rantanem e Juhola [4] propõem um método para ambientes dinâmicos, no qual é interessante ter um mapa com muitos nós (se um nó ou rota for bloqueado por um obstáculo móvel, existirá outra opção). Mas muitos nós num mapa implica maior custo. A ideia principal deste método é decidir, para cada nó selecionado, se ele deve ser adicionado ou não ao mapa. Estes autores dividem os nós em três classes:

- *Classe 1* - Nós que podem se conectar com apenas um componente: não são considerados úteis, pois exploram uma área já explorada, não contribuindo para conexões entre componentes;
- *Classe 2* - Nós que se conectam a dois ou mais componentes: são úteis, pois podem conectar componentes não conectados (gerando uma rota entre eles).
- *Classe 3* - Nós que não se conectam a nenhum componente: são úteis, pois exploram uma área ainda não explorada.

Se um novo nó pertence às classes *2* ou *3* ele é incluído no mapa, caso contrário seleciona-se outro nó. O mapa resultante é gerado e então os obstáculos móveis são adicionados; os nós e arestas dentro dos obstáculos móveis são desconsiderados. Apesar deste custo, o PRM ainda apresenta bons resultados.

4.3 Algoritmo básico do PRM

O algoritmo básico do PRM elabora o mapa em duas etapas: *construção* e *investigação*. Na *construção*, novos nós do *C-free* são selecionados aleatoriamente e conectados aos vizinhos mais próximos, criando arestas. Este processo se repete até que se atinja um número pré-determinado de nós. Na *investigação*, a *origem* e o *destino* são incluídos no mapa e a rota mais curta é encontrada. O algoritmo básico do PRM é flexível, pois permite utilizar diferentes métodos para diferentes partes do algoritmo, tais como diferentes formas de amostragem e diferentes métodos de seleções de vizinho. A ideia básica do PRM é escolher uma coleção de nós do *C-free* e formar um grafo. Pares de nós são selecionados e o navegador busca rotas entre eles, e quando acha, uma aresta é adicionada ao grafo. O navegador local deve ser eficiente por ser muito utilizado. Depois do mapa resultante montado, ele pode ser utilizado para responder perguntas da *investigação*. Para achar uma rota entre *origem* e *destino* estes dois nós são adicionados ao grafo pelo navegador local e então o algoritmo busca uma rota livre entre eles. A eficiência da *investigação* depende da qualidade do mapa gerado. Na maioria das versões de algoritmos PRM o nó amostrado não se conecta a todos os seus vizinhos [4, 13, 16] e nem sempre todos os nós selecionados são adicionados ao mapa [5] (se todos os vizinhos fossem

conectados o custo do algoritmo seria imenso). Ademais, o fato de se criar um grafo mais complexo não diminuiria muito o tamanho da rota encontrada. Entretanto, em casos extremos (onde é mais importante o tamanho da rota gerada do que o tempo de execução) um grafo mais complexo é útil, pois dá condições de encontrar uma rota menor.

Algoritmo 10: Algoritmo básico do PRM

```
 1 N = { }
 2 E = { }
 3 repeat
 4   q = configuração aleatória do C-espaço
 5   if q não está dentro de um obstáculo
 6     add q to N
 7     escolhe sub-conjunto Nq de N, candidatos a vizinhos de q
 8     for all q' em Nq ordenado de forma crescente por D (q , q')
 9       if planejador local pode conectar q com q'
10         add (q , q') to E
11 end
```

O algoritmo 10 mostra o funcionamento básico do PRM. As variáveis N (nós no *C-free*) e E (arestas) são inicializadas (linhas *1-2*), um nó aleatório é selecionado no *C-espaço* (linha *4*) e é verificado se este nó está no *C-free* (linha *5*). O novo nó é adicionado ao grafo (linha 6) e os vizinhos são selecionados (linha 7), os vizinhos são ordenados por ordem de proximidade do novo nó (linha 8) e este e os vizinhos passam pelo detector de colisões e, havendo rota livre, é adicionada uma aresta entre eles (linha 10). Os comandos (da seleção de um novo nó até a verificação de todos os vizinhos) são repetidos até que o grafo esteja com número de nós e arestas suficientes (quando um dado limite **ε** de número de nós ou arestas a serem adicionados é atingido a amostragem para).

4.4 Métodos de amostragem

Uma das principais escolhas que deve ser feita ao utilizar o PRM é como amostrar mais eficientemente os nós que farão parte do grafo. A técnica de amostragem deve ser bem escolhida, para que com poucos nós selecionados, a cobertura do mapa seja suficiente para encontrar uma boa rota, pois utilizar muitos nós torna o PRM inviável para muitos ambientes, devido ao alto custo do processo de inclusão de novos nós e arestas no grafo. A maneira mais simples é amostrar os nós aleatoriamente, porém outros métodos mais eficientes de amostragem foram propostos em [3, 14].

4.4.1 Amostragem aleatória

Na amostragem aleatória, uma amostra é criada com valores aleatórios para cada um dos *graus de liberdade do robô* e selecionada se estiver no *C-free*. Num tabuleiro 8x8 (figura 12), por exemplo, para um robô ir de (3,5) para (5,3), deve-se selecionar aleatoriamente um valor v_1 para *x* e v_2 para *y* e o nó amostrado seria (v_1,v_2).

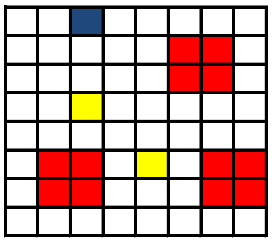

Figura 12: Os obstáculos estão em vermelho; em amarelo a origem e o destino; em azul o ponto selecionado (v1=3, v2=8).

4.4.2 Amostragem por células

Na amostragem por células, são selecionadas células dentro de células até que o número de pontos seja satisfeito. Inicialmente o mapa inteiro é uma única célula. É amostrado o primeiro ponto e esta célula é divida em subcélulas. É amostrado um ponto em cada subcélula e elas são novamente divididas e é selecionado um novo ponto e assim por diante. Na figura 12, por exemplo, seria selecionado um nó aleatório e então o tabuleiro seria dividido em células (figura 13 - esquerda) e então se selecionaria um nó aleatório em cada célula e esta se dividiria em subcélulas (figura 13 - direita). Este procedimento se repete até que o número predefinido de nós seja atingido.

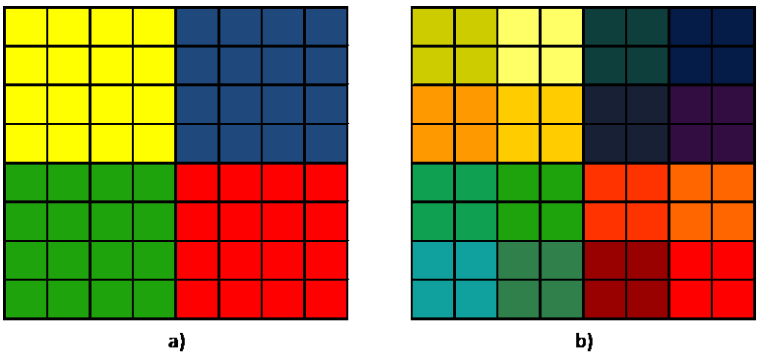

Figura 13: Amostragem por células.

4.4.3 Amostragem baseada em obstáculos

Na amostragem baseada em obstáculos, a ideia é aumentar o número de amostras selecionadas próximas aos obstáculos. É escolhido um nó aleatoriamente e se ele estiver no *C-free* é descartado, mas se estiver em algum obstáculo é adicionado ao grafo: é

selecionada uma direção aleatória e o nó move-se nesta direção em passos pequenos até que esteja no *C-free* e então é selecionado. Estas amostras provavelmente vão ficar perto dos limites dos obstáculos e existirá, pois, uma amostragem maior em áreas mais complexas. Este método foi o utilizado na figura 11 e resultou em vários nós próximos aos obstáculos.

4.5 Métodos de seleção de vizinhos

Algo de extrema relevância no PRM é a forma pela qual os vizinhos de cada nó são selecionados. Após cada nó $\boldsymbol{n_i}$ ser selecionado, ele é incluído no mapa e conectado aos demais, criando as arestas. Porém, para criar uma aresta entre um par de nós é necessário que a rota entre eles esteja livre de obstáculos. Como o detector de colisão é bastante custoso, não é bom testar a conexão de $\boldsymbol{n_i}$ com todos os demais nós e, portanto, é necessário otimizar a solução dos vizinhos a serem testados.

4.5.1 Vizinhos mais próximos

A forma mais clássica de se selecionar os vizinhos é selecionar os mais próximos e conectá-los ao novo nó. Os nós mais próximos têm mais chances de se conectar entre si e as verificações de colisões são menos custosas. Se muitos nós da vizinhança estão num mesmo componente (e não há outro componente na vizinhança) pode-se tentar conectar todos os nós ao novo nó, mas pode ser aceitável que o novo nó se conecte a apenas um número máximo $\boldsymbol{\eta}$ de vizinhos. Por exemplo, supondo $\boldsymbol{\eta}$ =2, o número máximo de conexões é 2 (cada novo nó pode se conectar a apenas dois nós já presentes no mapa). Na figura 11 se

o novo nó *(55,10)* é selecionado então a conexão inicial será com o vizinho mais próximo em distância euclidiana *(55,14)*, depois tentará se conectar a (55,28), depois a (62,26) e assim por diante, até que encontre dois vizinhos que possam se conectar a ele (neste caso, *(55,14)* e *(55,28)* já se conectariam ao novo nó e não se precisaria de mais nós).

4.5.2 Componentes

A técnica de seleção de vizinhos por componentes consiste em tentar conectar o novo nó ao nó mais próximo de cada componente, dentro de uma distância definida. A ideia é conectar um mesmo nó a diferentes componentes. Na figura 11, se o novo nó *(10,80)* for selecionado então ele vai tentar se conectar a *(8,79), (1,75) e (25,10)* que estão, nesta ordem, nos diferentes componentes mais próximos. Assim, o mapa evita a construção de arestas pouco úteis, pois se dois nós estão num mesmo componente, já existe uma rota entre eles.

4.5.3 N-Componentes

A técnica de seleção de vizinhos n-componentes assemelha-se àquela de *vizinhos mais próximos*, limitando, porém a conexão do novo nó a um número máximo de vizinhos por componente ($\boldsymbol{\eta_v}$), o que resulta num número pequeno de arestas. A ideia é que com muitos componentes é custoso criar muitas arestas num mesmo componente: é melhor explorar então novos componentes. Na figura 11, com $\boldsymbol{\eta_v} = 2$, se o nó *(45,30)* for

adicionado, ele vai tentar se conectar a *(42,31) e (49,31)* e depois a *(25,10)* e então a *(8,79) e (1,75)*.

4.6 Algoritmo A*

Após a seleção dos nós, o PRM utiliza o algoritmo A* para encontrar a melhor rota entre a *origem* e o *destino*, considerando somente os nós amostrados no PRM. Considerando a distância de cada par de nós do grafo do PRM como o custo daquela aresta, o A* encontrará a rota de menor custo.

O algoritmo A* procura a rota através de três funções básicas:

- g [x] => Custo da rota entre a *origem* e o ponto x.
- h [x] => Custo estimado da rota entre o ponto x e o *destino*.
- f [x] => A soma das funções anteriores (g[x] + h[x])

Além destas três funções, o A* também trabalha com duas listas de nós:

- Nós abertos: Nós que ainda serão analisados pelo A*.
- Nós fechados: Nós que não serão mais analisados pelo A*.

O A* então começa inserindo a *origem* na lista de nós abertos. A cada passo seleciona-se *v* como o nó que tem melhor *f [x]*, entre os presentes na lista de nós abertos. Selecionam-se todos os vizinhos *u* de *v* e calcula-se o novo *f [x]* para cada *u*, que será a soma do caminho percorrido da *origem* até *v (g[v])* com o custo da aresta (*u*,*v*), mais a função *h* no ponto *u (h[u])*.

Caso *u* já esteja listado como nó fechado ou nó aberto, como uma *f [x]* menor do que o novo *f [x]* calculado então ele é descartado, pois já existe uma rota mais curta que

chega a esse nó, e aquela é a rota que deve ser analisada. Caso contrário a função *g [u]* recebe o valor do caminho percorrido até o ponto *v (g[v])* mais o custo do caminho percorrido entre o ponto *v* e *u (c(u,v))*. Além disso, se *u* está entre os nós fechados ou entre os nós abertos ele é removido desta lista e inserido na lista de nós abertos com sua nova função *f[x]*. Após testar todos os vinhos de *v*, este é inserido na lista de nós fechados.

O algoritmo 11 mostra o funcionamento do A*: Algumas variáveis são iniciadas (linhas 1-11), verifica se a solução já foi encontrada (linhas 12-16) ou segue com a busca caso necessário (linhas 17-35). No final o algoritmo retorna uma mensagem de sucesso ou fracasso (linhas 36-39).

Algoritmo 11: Algoritmo A*

```
1  g [s] = 0
2  f [s] = g [s] + d (s)
3  pais [ ] = vazio
4  abertos [ ] = vazio
5  fechados [ ] = vazio
6  achou = falso
7  Inclui s em abertos
8  while (abertos não está vazio) e (achou <> falso)
9    v = melhor vértice de abertos
10   if v = goal
11     achou = verdadeiro
12   end
13   for cada u vizinho de v
14     novof = g [v] + c (v,u) + d (u)
15     if (u está em abertos ou em fechados) e (novof >= f [u])
16       pula para o próximo vizinho
17     else
18       pai [u] = v
19       g [u] = g [v] + c(u,v)
20       f [u] = novo_f
21       if u está em fechados
22         Remove u de fechados
23       if u está em abertos
24         Remove u de abertos
25       Inclui u em abertos
26     end
27   end
28   Inclui v em fechados
29 end
```

5 O algoritmo de otimização (PRM-EB)

Nesta dissertação é proposto um novo algoritmo, denominado PRM-EB (*Probabilistic Roadmaps - Evolutionary Betting*) que concatena a construção do mapa gerado pelo PRM, o algoritmo de busca A* e o algoritmo de apostas proposto por Demasi [19]. A otimização por algoritmo de apostasbusca solucionar um dos principais problemas do PRM tradicional, que é a geração de rotas nem sempre tão próximas da rota ótima.

O PRM recebe como entradas o mapa do ambiente (somente com a localização dos obstáculos), a *origem* e o *destino*. Na primeira fase do PRM, são gerados nós aleatórios no *C-free* e na segunda fase é utilizado o A* para encontrar a melhor rota possível (utilizando somente os pontos amostrados na primeira fase do PRM) ligando a *origem* ao *destino*.

Nesta dissertação o algoritmo de apostas recebe como entrada a rota inicial (gerada pelo A* como no PRM clássico). O algoritmo de apostas também é dividido em duas etapas. Na primeira etapa, os jogadores vão fazendo suas apostas e os melhores jogadores vão permanecendo no jogo e apostando a cada rodada, enquanto os piores jogadores vão perdendo seus cacifes e saindo do jogo. Na segunda etapa, a melhor rota encontrada na primeira etapa é otimizada se baseando em palpites somente dos melhores jogadores sobreviventes, gerando então a rota final otimizada.

Resumidamente, o algoritmo proposto neste trabalho consiste de 4 etapas, conforme demonstrado no fluxograma da Figura 14.

- Primeira etapa: Corresponde à primeira fase do PRM tradicional, que é a construção do mapa.

- Segunda etapa: Corresponde à segunda fase do PRM tradicional, que é a investigação do mapa feita pelo A*.
- Terceira etapa: Corresponde à primeira fase do algoritmo de apostas.
- Quarta etapa: Corresponde à segunda fase do algoritmo de apostas.

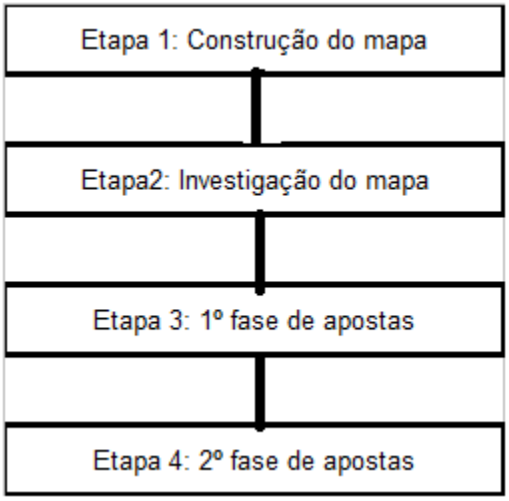

Figura 14: Etapas do PRM-EB.

5.1 Objetivo do PRM-EB

O PRM tradicional é um algoritmo muito útil, entre outras coisas porque não apresenta problemas com ótimos locais. Ao mesmo tempo em que não produz rotas tão eficientes quanto outros métodos, o PRM abre espaço para que seja utilizado conjuntamente com muitas outras técnicas.

O principal objetivo do PRM-EB é aproveitar a simplicidade e a elasticidade do PRM, não somente para traçar uma rota mais eficiente do que o PRM tradicional, mas

também para traçar rotas mais eficientes do que outros métodos, inclusive os descritos neste trabalho. Assim sendo, este trabalho propõe um novo algoritmo que é, baseado no PRM mas utilizando uma otimização baseada no algoritmo de apostas. Foi utilizada, adicionalmente, uma mutação baseada em técnicas de *shortcutting*, para acelerar a otimização.

5.2 O PRM-EB

Inicialmente o PRM gera um mapa inicial e busca através do algoritmo A* uma rota inicial para um robô. Verifica-se por quantos nós (*n*) este robô passa até que chegue ao *destino* e então é gerado um vetor-solução de tamanho 2x*n*, onde a primeira linha deste vetor é preenchida com as coordenadas *x* dos nós que estão na rota encontrada pelo A* e a segunda linha é preenchida com as coordenadas *y* dos nós que estão nesta mesma rota. Desta forma, o algoritmo PRM-EB vai gerar uma rota que passa sempre pela mesma quantidade de nós que o caminho encontrado pelo A*. Entretanto o tamanho geral da rota gerada pelo PRM-EB será menor, pois embora a quantidade de nós em ambas as rotas geradas (tanto pelo A*, como pelo PRM-EB) sejam as mesmas, o tamanho das arestas que ligam estes nós será menor na rota gerada pelo PRM-EB.

Seleciona-se como máscara um número *m* de índices do vetor-solução e estes índices poderão (ou não) ser alterados por cada jogador. Os jogadores calculam seus pesos e apostam em suas aplicações de máscaras favoritas (aquelas que apresentam melhor custo-benefício aos jogadores em termos de cotação) segundo o critério de Kelly [46],

respeitando sempre um valor mínimo de aposta e, obviamente, nunca podendo apostar mais do que o seu cacife atual.

A cada iteração do método (uma rodada de apostas) uma máscara diferente é selecionada. Isto é de extrema importância para o funcionamento do algoritmo, pois caso a mesma máscara seja selecionada sempre, os apostadores só vão fazer apostas alterando alguns bits específicos da solução e não vão conseguir otimizá-la por completo. Por isto a cada rodada de apostas uma nova máscara é selecionada aleatoriamente.

Caso um índice do vetor-solução seja selecionado para ser alterado, a coordenada é alterada da seguinte forma: gera-se o ponto médio (que é a média simples da coordenada no índice *(i-1)* com o índice *(i+1))* e gera-se uma coordenada provisória naquele exato ponto.

Alguns testes são feitos para verificar se esta coordenada provisória não está dentro de nenhum obstáculo e se possui rotas livres de obstáculos para as coordenadas de índices *(i-1)* e *(i+1)*. Caso passe nos testes, esta será a nova coordenada do índice *i*, caso contrário será escolhida aleatoriamente uma nova coordenada que esteja no raio de distância euclidiana $r<=1$ do ponto médio e os mesmos testes são feitos (verificar se esta coordenada provisória não está dentro de nenhum obstáculo e se possui rotas livres de obstáculos para as coordenadas de índices *(i-1)* e *(i+1)).* Caso a nova coordenada provisória passe no teste ela é oficializada como a nova coordenada. Caso contrário repete-se o processo escolhendo aleatoriamente uma nova coordenada que esteja no raio de distância euclidiana $r <=2$. E este processo se repete, até encontrar uma nova coordenada provisória que seja aprovada nos testes ou então até que o *r* chegue a 10. Caso *r* chegue a 10 e ainda sim nenhuma coordenada seja aprovada nos testes, a mutação não pode ocorrer e o índice *i* continuará com sua coordenada original.

A ideia principal de se utilizar o ponto médio é tentar cortar caminhos, como exemplifica a figura 15. Desta forma não se utiliza uma mutação aleatória qualquer, mas sim uma mutação que direciona o algoritmo a gerar caminhos mais retilíneos, fazendo com que o algoritmo possa convergir para a rota ótima mais rapidamente.

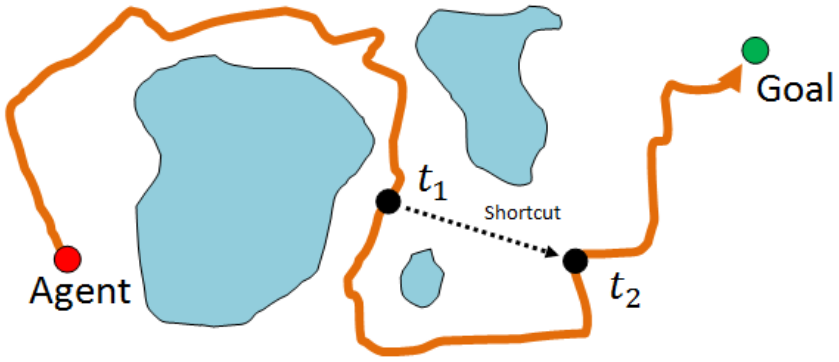

Figura 15: Possível corte de caminho utilizando ponto médio entre dois pontos – extraída de [41].

As iterações vão ocorrendo e a cada passo o vetor-solução é sempre a melhor solução encontrada na rodada anterior. Jogadores saem do jogo quando perdem todo o seu cacife e são substituídos por novos jogadores. Os jogadores vão participando de rodadas de apostas até que se alcance um número suficiente de iterações e então o vetor-solução final já será a rota otimizada.

Os novos jogadores que vão entrando ao longo do jogo não são gerados aleatoriamente como os jogadores gerados no início do algoritmo. Os novos jogadores são criados com base no melhor jogador presente no jogo no momento de sua criação. [19]

A rota gerada pelo PRM-EB nunca vai ser pior do que a rota gerada pelo PRM, pois se por acaso, em algum momento, a solução piore após uma rodada de apostas, esta nova solução é descartada e o vetor-solução volta ser o vetor-solução da rodada anterior. Desta forma, os apostadores sempre vão se basear na melhor solução encontrada até o momento para fazer as alterações seguintes. Sendo assim, no pior dos casos, a rota gerada pelo PRM-EB vai ser igual à rota gerada pelo A*, nunca gerando uma rota maior.

O algoritmo 12 mostra o algoritmo PRM-EB, proposto nesta dissertação. A função *g [x]*, que mede a distância percorrida da *origem* até o ponto x é inicializada com 0 (linha 1), a função *f [x]* é inicializada, sendo a soma de *g [x]* com *d [x], que é a* distância entre o ponto x e o *goal* (linha 2). A *origem* do algoritmo é o *start* (*s*). Algumas variáveis auxiliares são inicializadas (linhas 3-6) e o *start* é incluído no grupo de nós abertos (linha 7). O algoritmo procura todas as rotas possíveis como acontece no A* (linhas 8-29). A rota que minimiza a função *f [x]* é então escolhida e começam as rodadas de aposta, que visam melhorar esta rota. Uma máscara aleatória é selecionada para cada rodada (linha 31) e os jogadores calculam o peso de cada permutação da máscara naquela rodada (linhas 32-36). A banca calcula suas cotações para a mesma rodada (linhas 37-39), e finalmente os jogadores fazem suas apostas (linhas 40-47). Verifica-se qual a permutação de máscara é a vencedora, ou seja, a que mais otimiza o PRM naquela rodada (linhas 48-55). Os apostadores vencedores são recompensados por seus acertos conforme a proporção do valor de cada aposta, aumentando seus cacifes (linhas 56-59) e os jogadores que perdem todo o seu cacife são substituídos por jogadores novos, que são construídos baseados no melhor jogador presente no jogo naquele momento (linhas 60-63). A melhor rota da primeira fase da otimização é então guardada (linhas 65-67) e parte-se para a segunda fase da otimização. Na segunda fase da otimização, a rota vai sendo alterada de acordo com o vetor de

probabilidades dos jogadores sobreviventes (linhas 69-77) e assim obtém a rota otimizada final. A rota otimizada final será a rota que o robô utilizará para atravessar o cenário e chegar da *origem* até o *destino* sem nenhuma colisão.

Com efeito, pode-se gastar bastante tempo para encontrar uma rota menor, mas, uma vez essa rota encontrada ela poderá ser utilizada inúmeras vezes pelo robô naquela configuração e, portanto, o tempo gasto pelo robô será muito menor, compensando amplamente o tempo resultante gasto para achar a melhor rota.

É mister salientar que existem diversos algoritmos de otimização já propostos, e muitos deles já foram utilizados em planejadores de rotas para robôs. Entretanto, como os problemas nesta área podem ser muito diferentes uns dos outros, com diferentes propriedades e restrições específicas, uma nova e original alternativa de otimização vai corroborar para as possíveis soluções de rotas para robôs em ambientes reais.

Para validar o algoritmo proposto nesta pesquisa e analisar a utilidade e robustez do mesmo em comparação a outras otimizações existentes na literatura, como os algoritmos genéticos, por exemplo, foi realizada uma bateria de testes, para diferentes tipos de cenários, com diferentes taxas de ocupação, diferentes tipos de obstáculos etc.

Algoritmo 12: PRM-EB

```
1  g [s] = 0
2  f [s] = g [s] + d (s)
3  pais [ ] = vazio
4  abertos [ ] = vazio
5  fechados [ ] = vazio
6  achou = falso
7  Inclui s em abertos
8  while (abertos não está vazio) e (achou <> falso)
9    v = melhor vértice de abertos
10   if v = goal
11     achou = verdadeiro
12   end
13   for cada u vizinho de v
14     novof = g [v] + c (v,u) + d (u)
15     if (u está em abertos ou em fechados) e (novof >= f [u])
16       pula para o próximo vizinho
17     else
18       pai [u] = v
19       g [u] = g [v] + c(u,v)
20       f [u] = novo_f
21       if u está em fechados
22         Remove u de fechados
23       if u está em abertos
24         Remove u de abertos
25       Inclui u em abertos
26     end
27   end
28   Inclui v em fechados
29 end
30 for ni=1:número de iterações
31   Seleciona máscara aleatória
32   for i=1:número de jogadores
33     for j=1:tamanho da máscara
34       Calcula pesos w (i,j)
35     end
36   end
37   for j=1:tamanho da máscara
38    Calcula cotação da banca w (j)
39   end
40   for i=1:número de jogadores
41     for j=1:tamanho da máscara
42       if w (j) > w (i,j)
43         Calcula va_ck (valor de aposta segundo critério de Kelly)
44         cacife (i) = cacife (i) - va_ck
45       end
46     end
47   end
```

```
48   melhor_caminho = 0
49   for m =1:tamanho da máscara
50     Avalia caminho m (caminho_atual após aplicação da máscara m )
51     if caminho m melhor que melhor_camiho
52       melhor_caminho = caminho m
53       melhor_indice = m
54     end
55   end
56   for i =1:número de jogadores
57     if jogador i apostou na máscara m
58       cacife (i ) = cacife (i ) + va_ck * w(m )
59     end
60     if cacife (i ) = 0
61       Remove jogador i do jogo
62       Insere novo jogador no jogo
63     end
64   end
65   if melhor_camiho é melhor que caminho_autal
66     caminho_atual = melhor_caminho
67   end
68 end
69 for nj =1:número de iterações da segunda fase
70   for i =1:número de jogadores
71   caminho = caminho_aual
72     for t =1:tamanho do caminho
73       Modifique o bit t de caminho com probabilidade p(i(t))
74     end
75   Avalia se o novo caminho é melhor que o antigo
76   end
77 end
```

6 Experimentos

Os experimentos foram feitos usando o Matlab R2014b em ambientes de duas dimensões (100x100) com diferentes quantidades, tamanhos e formatos de obstáculos. A *origem* é sempre colocada no ponto (1,1) e o *destino* no ponto (100,100).

Numa primeira análise, foi testada a otimização do PRM para vinte diferentes tipos de cenários, sendo feitos 100 testes para cada um deles, totalizando 2.000 testes. Foram utilizados desde cenários bem simples até cenários bem mais complicados, como os cenários com obstáculos em forma de "V" (que costumam apresentar grande dificuldade à maioria dos algoritmos de planejamento de rotas).

- Obst-Q (cenários com obstáculos quadrados): 4, 8, 12 ou 16 obstáculos quadrados.
- Obst-R (cenários com obstáculos retangulares): 4, 8, 12 ou 16 obstáculos retangulares.
- Obst-C (cenários com obstáculos circulares): 4, 8, 12 ou 16 obstáculos circulares.
- Obst-V (cenários com obstáculos em forma de "V"): 4, 8, 12 ou 16 obstáculos em forma de "V".
- Obst-M (cenários mistos): 1, 2, 3 ou 4 de cada um dos obstáculos citados acima, formando então cenários com 4, 8, 12 ou 16 obstáculos, assim como os demais cenários.

Cada obstáculo (independentemente de forma geométrica) possui uma área equivalente a 1% da área do mapa. Ou seja, nos cenários mais simples (4 obstáculos) a taxa de ocupação é de 4%. E nos cenários mais complexos (16 obstáculos) a taxa de ocupação é

de 16%. Não foram realizados testes com percentuais maiores de ocupação, pois o PRM não costuma encontrar rotas com frequência para estes tipos de cenários.

Para efeitos de comparação, foram feitos testes também com algoritmos genéticos, utilizados em outras propostas com PRM [31, 29, 42]. Um resultado promissor para a proposta aqui feita foi o fato de que o PRM-EB se mostrou mais eficiente do que o algoritmo genético na maioria das vezes.

Em cada teste com o PRM-EB foram executadas 16 iterações da primeira fase do algoritmo de apostas e 16 iterações da segunda fase do algoritmo de apostas, totalizando assim 144 indivíduos avaliados por teste (sendo 128 na primeira fase e 16 na segunda fase). A quantidade de apostadores foi mantida em 5 durante toda a execução e a máscara utilizada foi de tamanho 3.

Em cada teste com o algoritmo genético foram executadas 18 iterações, totalizando 144 indivíduos avaliados por teste (número de avaliações iguais as do PRM-EB, para que a comparação fosse justa). O tamanho da população utilizada foi de 8 indivíduos e a taxa de mutação foi de 0,2. Utilizou-se o elitismo somente com o melhor indivíduo, ou seja, somente o melhor indivíduo de uma geração era levado automaticamente para a geração seguinte. A codificação de cada indivíduo foi feita da mesma forma do que a do vetor-solução do PRM-EB.

A figura 16 mostra um caso aonde o PRM-EB reduziu a rota gerada pelo PRM em 21,78%. Os pontos amarelos são os nós selecionados como fazendo parte da melhor rota gerada pelo PRM, juntamente com as linhas verdes claras, que formam a rota percorrida. Os pontos pretos mostram os pontos finais resultantes da otimização através do PRM-EB, assim como as linhas verdes escuras. Pode-se ver que a rota é bem mais retilínea e por isto

a redução do tamanho deste é consideravelmente grande, principalmente nas partes inicial e final do trajeto.

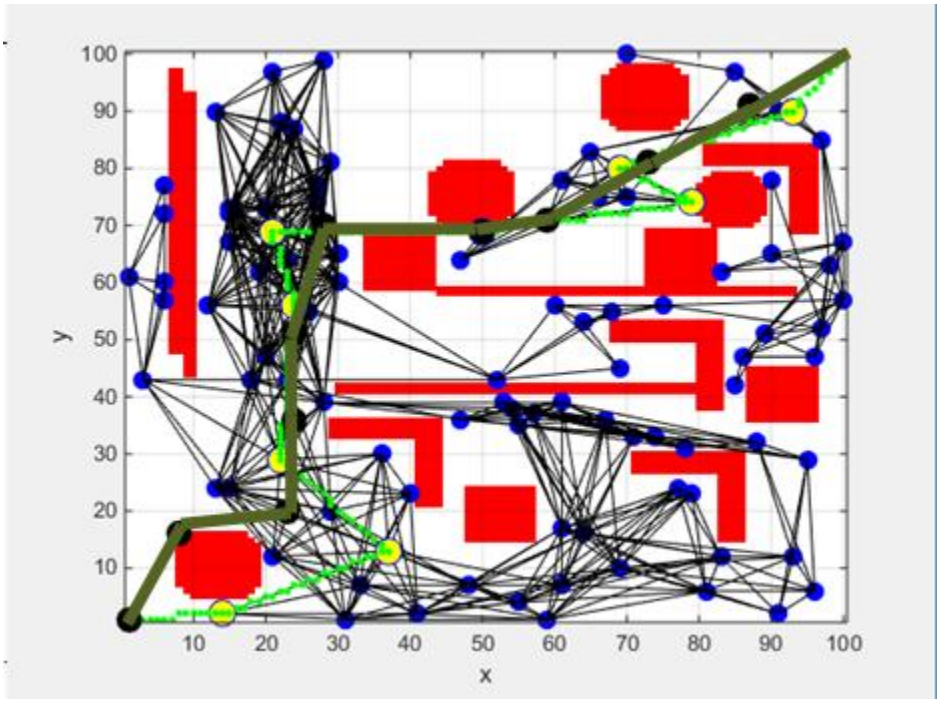

Figura 16: Exemplo de rota otimizada pelo algoritmo de apostas.

6.1 Análise dos resultados

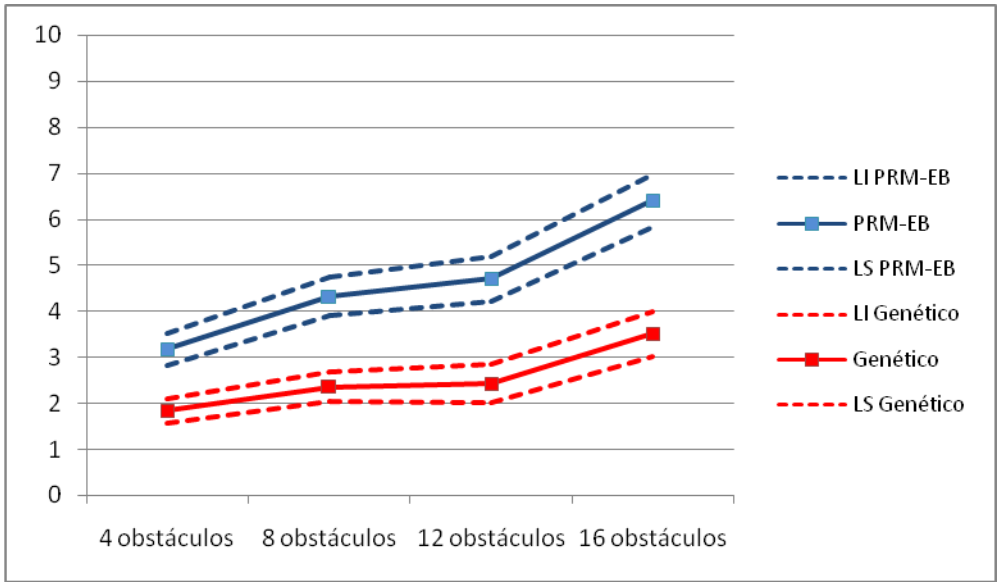

Figura 17: Percentual de melhora do PRM após utilizar o PRM-EB - Obst-M.

A figura 17 mostra o percentual de melhora da rota do PRM-EB em comparação com o PRM clássico, assim como seus respectivos intervalos de confiança (alfa=5%), nos cenários mistos (com todos os tipos de obstáculos citados). Nota-se que conforme aumenta o número de obstáculos no ambiente, aumenta também o percentual de melhora do PRM-EB, porém esse aumento é cada vez menor. A mesma coisa acontece com a otimização por algoritmos genéticos.

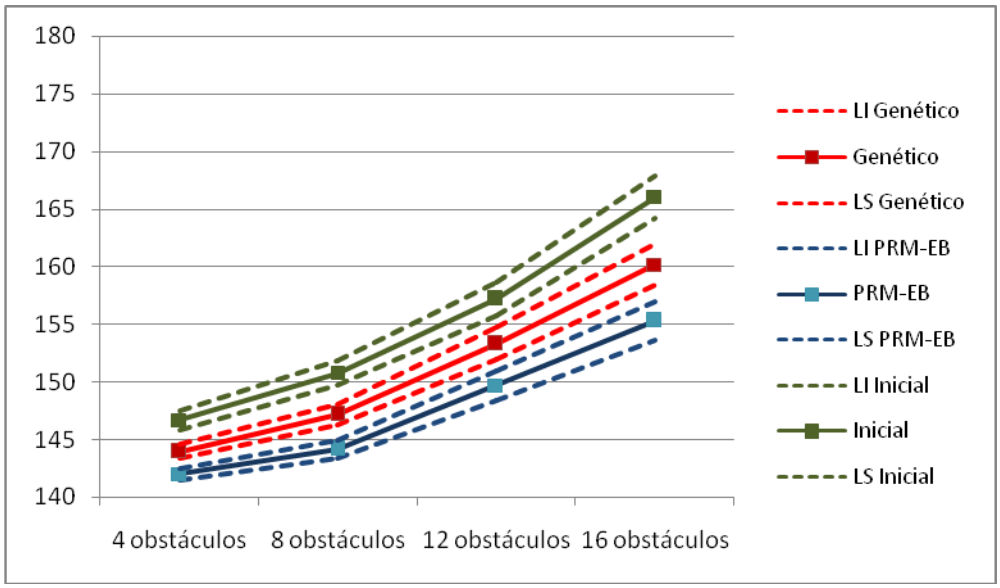

Figura 18: Média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB - Obst-M.

A figura 18 mostra a média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB, assim como seus respectivos intervalos de confiança (alfa=5%), para os cenários com obstáculos mistos. Nota-se que as rotas após a otimização são estatisticamente menores que as rotas antes da otimização, independente do número de obstáculos. São também estatisticamente menores do que a otimização por algoritmos genéticos em todos os cenários, independente da quantidade de obstáculos no cenário.

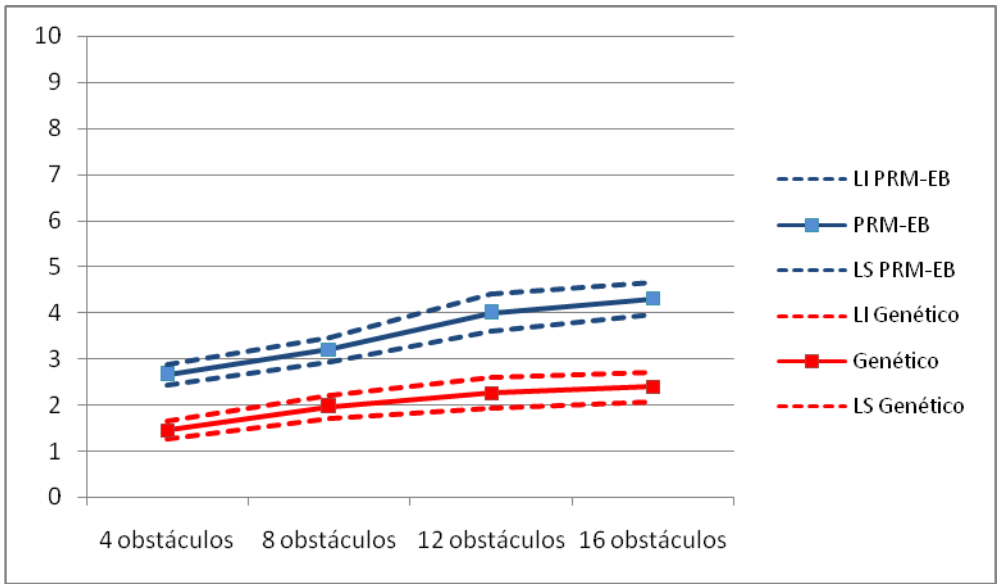

Figura 19: Percentual de melhora do PRM após utilizar o PRM-EB - Obst-Q.

A figura 19 mostra o percentual de melhora da rota do PRM-EB em comparação com o PRM clássico, assim como seus respectivos intervalos de confiança (alfa=5%), nos cenários com obstáculos quadrados. Nota-se que o comportamento/tendência é bastante similar ao dos cenários Obst-M, porém com taxas de percentual de melhora bem menores, tanto para o PRM-EB quanto para a otimização por algoritmos genéticos. Essa taxa de otimização ser menor para ambientes com obstáculos quadrados se justifica pela facilidade do PRM em lidar com este tipo de ambiente. O PRM puro já constrói caminhos bastante razoáveis para este tipo de cenário, restando pouca coisa para ser otimizada, seja pelo algoritmo PRM-EB ou pelo algoritmo genético.

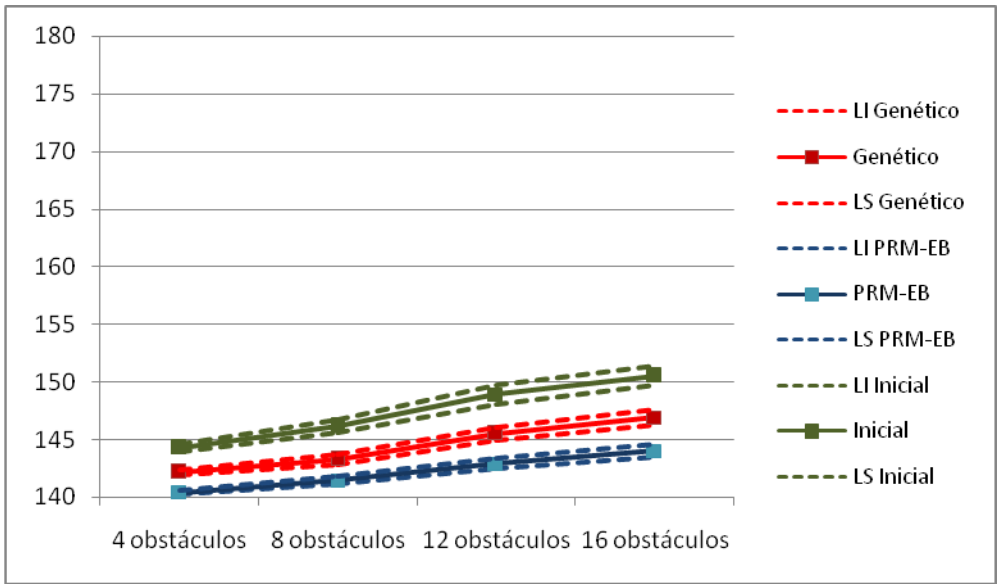

Figura 20: Média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB - Obst-Q.

A figura 20 mostra a média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB, assim como seus respectivos intervalos de confiança (alfa=5%), para os cenários com obstáculos quadrados. O comportamento é novamente similar ao Obst-M, porém com rotas bem menores em ambientes somente com obstáculos quadrados, devido à simplicidade dos obstáculos, que facilitam o robô a encontrar uma rota menor.

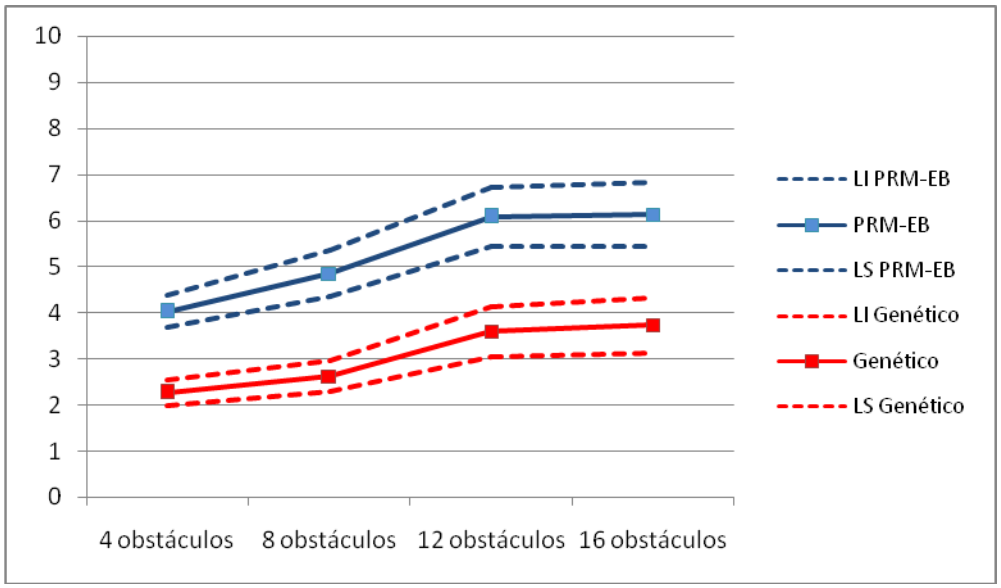

Figura 21: Percentual de melhora do PRM após utilizar o PRM-EB - Obst-R.

A figura 21 mostra o percentual de melhora da rota do PRM-EB em comparação com o PRM clássico, assim como seus respectivos intervalos de confiança (alfa=5%), nos cenários com obstáculos retangulares. A taxa de otimização do Obst-R cresce mais rápido do que para cenários Obst-M e Obst-Q. Vale ressaltar que em todos os casos a diferença de percentuais de melhora do algoritmo PRM-EB com o algoritmo genético é estatisticamente significativa.

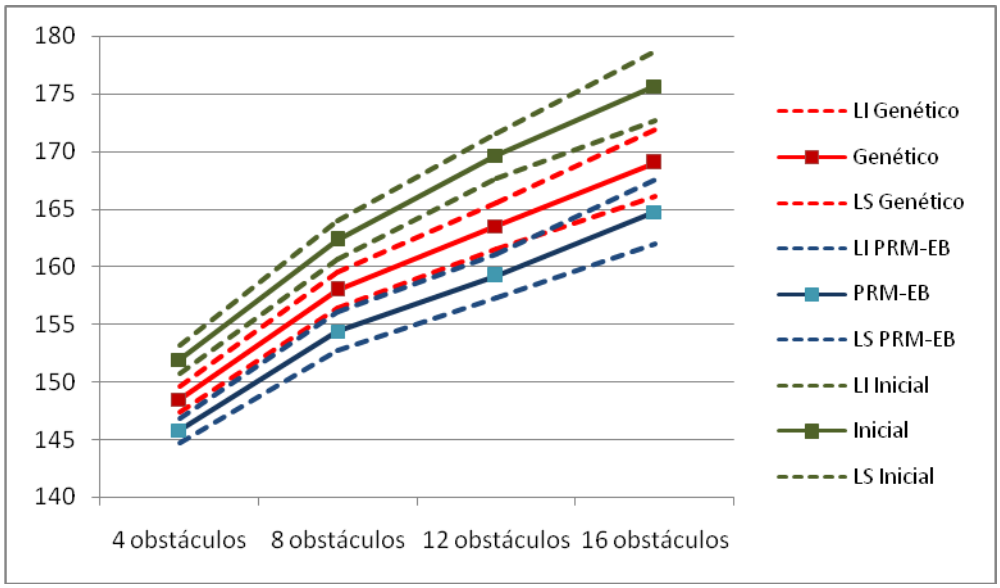

Figura 22: Média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB - Obst-R.

A figura 22 mostra a média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB, assim como seus respectivos intervalos de confiança (alfa=5%), para os cenários com obstáculos retangulares. As rotas geradas nos cenários Obst-R são as maiores entre todos os tipos de cenários testados. Isto ocorre porque como existem alguns ambientes compridos no eixo vertical e outros compridos no eixo horizontal o robô tem que fazer geralmente um caminho em formato de "V" ao invés de se aproximar de uma linha reta diagonal. Nos testes para este tipo de cenário, quando a taxa de ocupação é mais alta não podemos ver diferenças estatisticamente significativas entre as otimizações por algoritmos genéticos e o PRM-EB, embora o PRM-EB apresente melhores resultados nos testes efetuados.

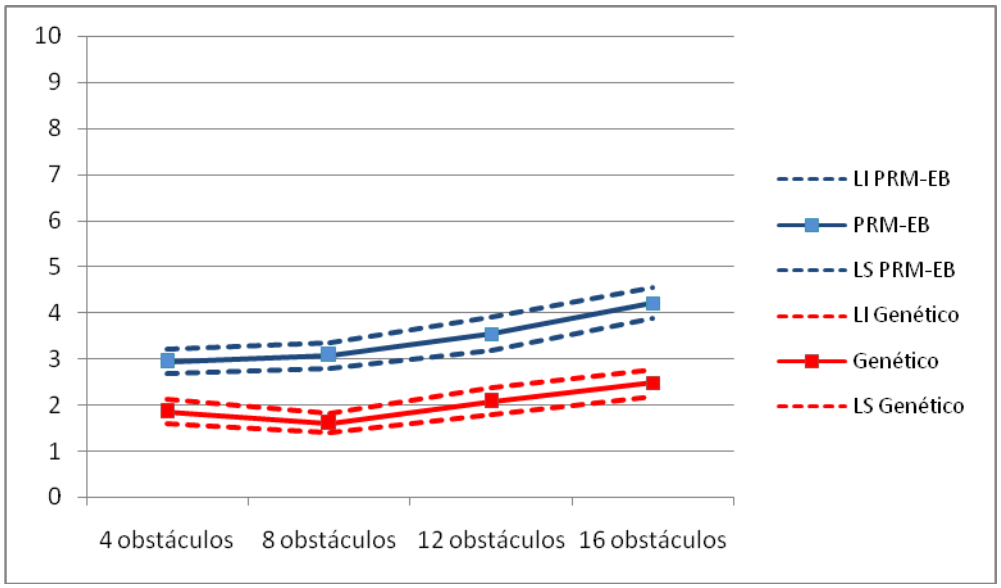

Figura 23: Percentual de melhora do PRM após utilizar o PRM-EB - Obst-C.

A figura 23 mostra o percentual de melhora da rota do PRM-EB em comparação com o PRM clássico, assim como seus respectivos intervalos de confiança (alfa=5%), nos cenários com obstáculos circulares. A taxa de otimização para cenários Obst-C mostra um padrão de crescimento da taxa de otimização parecido com o de cenários Obst-Q. A diferença é que a taxa de otimização para cenários Obst-C tem valores um pouco menores, devido à facilidade um pouco maior do cenário.

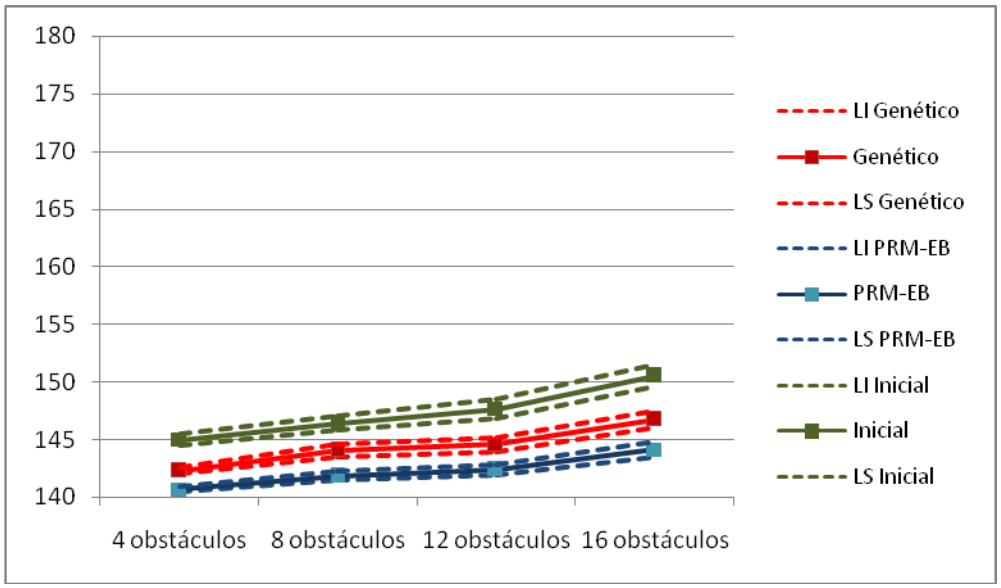

Figura 24: Média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB - Obst-C.

A figura 24 mostra a média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB, assim como seus respectivos intervalos de confiança (alfa=5%), para os cenários com obstáculos circulares. O tamanho das rotas são bastante similares ao tamanho das rotas de cenários Obst-Q. Esta semelhança provavelmente se deve ao fato de um círculo ter um formato mais próximo de um quadrado do que dos demais tipos de obstáculos.

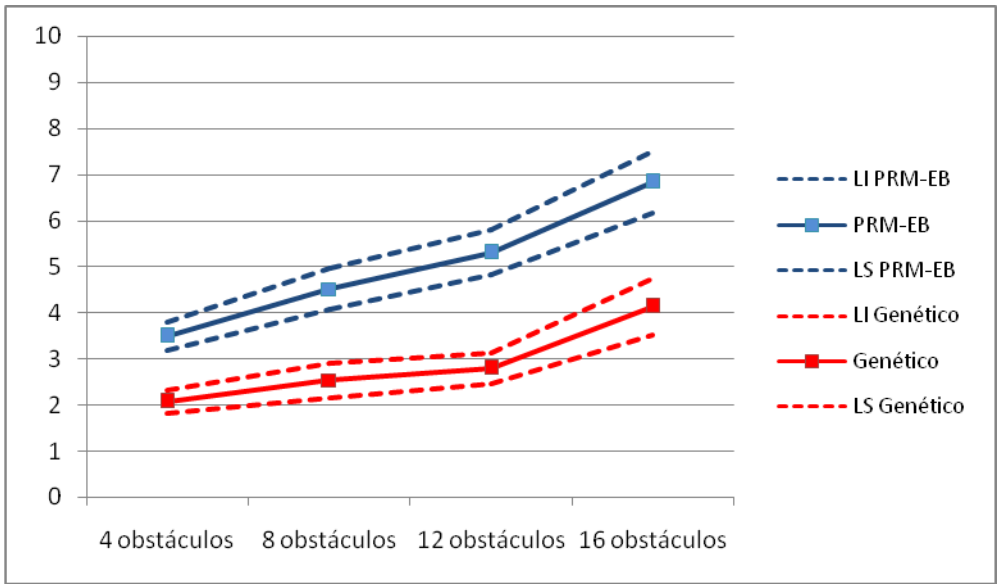

Figura 25: Percentual de melhora do PRM após utilizar o PRM-EB - Obst-V.

A figura 25 mostra o percentual de melhora da rota do PRM-EB em comparação com o PRM clássico, assim como seus respectivos intervalos de confiança (alfa=5%), nos cenários com obstáculos em forma de "V". Este é o cenário que tem a maior taxa de otimização entre os cenários testados e isto ocorre devido à complexidade do cenário Obst-V. Este tipo de cenário é um dos cenários mais difíceis para a grande maioria dos planejadores de rotas (APF, WP, RRT, EST, ARW, etc.) e não somente para o PRM. É interessante ressaltar que a otimização PRM-EB apresentou em geral melhores resultados para os cenários mais difíceis, o que corrobora a sua utilidade.

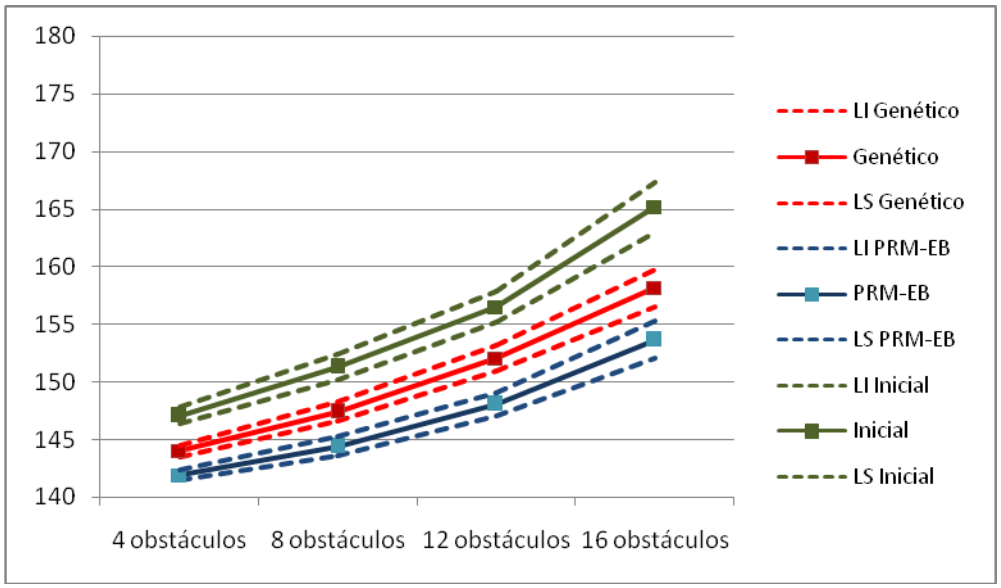

Figura 26: Média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB - Obst-V.

A figura 26 mostra a média do tamanho das rotas geradas, antes e depois de utilizar o PRM-EB, assim como seus respectivos intervalos de confiança (alfa=5%), para os cenários com obstáculos em forma de "V". Este é o tipo de cenário que gera a terceira maior média de tamanho de rota, perdendo somente para os cenários Obst-R e Obst-M, porém cabe ressaltar que diferentemente da análise do Obst-R, esta análise nos mostra que as diferenças entre os dois tipos de otimização testados (genético e PRM-EB) são estatisticamente significativas em todas as taxas de ocupação do mapa por obstáculos.

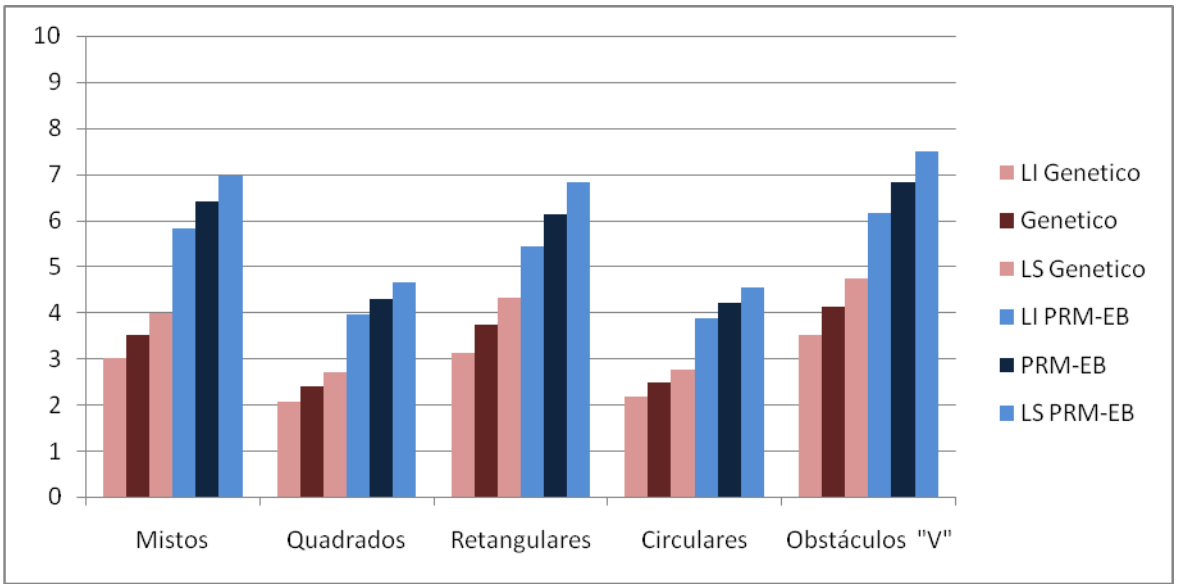

Figura 27:Taxas de otimização: Método de otimização X Tipo de cenário.

A figura 27 mostra as taxas de otimização para todos os tipos de cenários que foram gerados com 16 obstáculos, juntamente com seus respectivos intervalos de confiança (alfa=5%), cruzando cada método de otimização testado com cada tipo de cenário. Em todos os cenários, o PRM-EB tem um percentual significativamente melhor do que a otimização por algoritmos genéticos. Verifica-se também que os cenários nos quais as otimizações surtem mais efeitos são os cenários Obst-M, Obst-R e Obst-V. Ou seja, o PRM-EB é ainda mais útil quando se utiliza ambientes com obstáculos mais complexos.

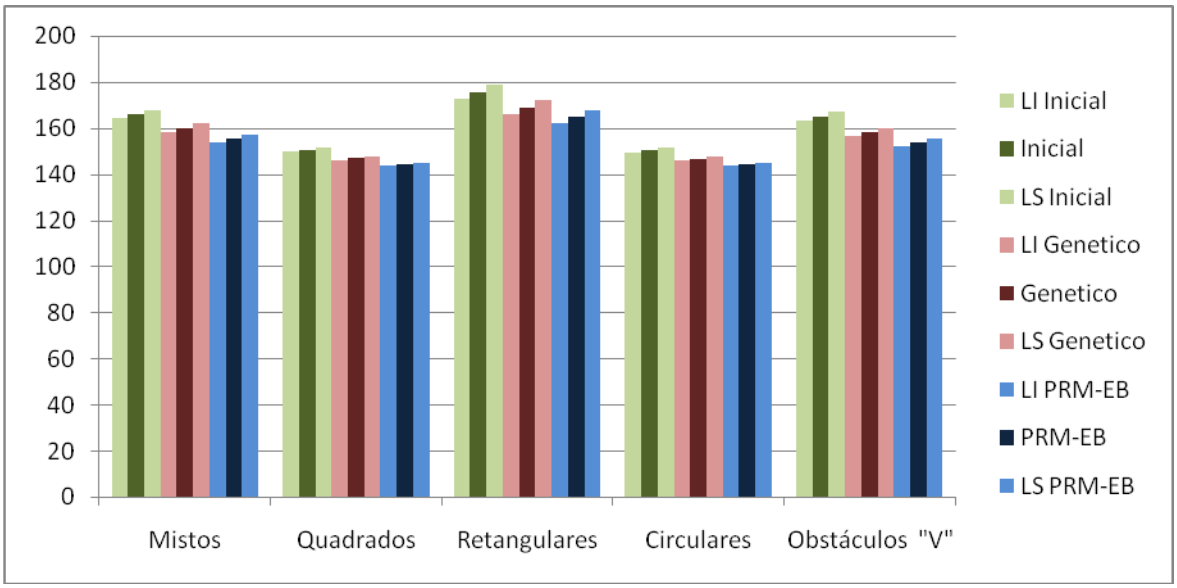

Figura 28:Tamanho médio das rotas geradas: Método de otimização X Tipo de cenário.

A figura 28 mostra os tamanhos médios das rotas geradas pelos métodos testados (PRM, PRM com otimização genética e o PRM-EB) para todos os tipos de cenários que foram gerados com 16 obstáculos, juntamente com seus respectivos intervalos de confiança (alfa=5%). As maiores rotas são geradas em ambientes Obst-R, Obst-V e Obst-M. O PRM-EB produz rotas estatisticamente menores do que as rotas geradas pelas otimizações genéticas em todos os casos (excetos nos cenários Obst-R, aonde essa diferenção não é estatisticamente significativa).

7 Conclusão

O PRM-EB (*Probabilistic Roadmaps - Evolutionary Betting*), algoritmo proposto nesta dissertação, foi elaborado baseando-se no algoritmo clássico de PRM e na teoria de apostas, acrescido de algumas otimizações locais (como, por exemplo, o uso de pontos médios para encurtamento de rotas). O objetivo de se propor um novo algoritmo para a otimização de PRM foca precisamente na mais conhecida deficiência do PRM que é a de gerar rotas nem sempre tão próximas da rota ótima. Ainda que o PRM-EB gere uma rota que passa pela mesma quantidade de nós que o caminho encontrado pelo A*, os experimentos mostraram que o tamanho total da rota gerada é normalmente menor.

Os experimentos efetuados também mostraram resultados muito significativos notadamente em ambientes complexos, nos quais o PRM clássico demonstra um desempenho acanhado. Os obstáculos em forma de V, por exemplo, que são conhecidos por acrescer bastante dificuldade na geração de rotas livres de obstáculos sem que um robô fique preso em mínimos locais, foram particularmente experimentados e, precisamente neste contexto, o PRM-EB mostrou soluções bastante eficientes. Portanto, o PRM-EB demonstrou maiores percentuais de redução das rotas justamente nos cenários mais difíceis para o PRM, o que sugere que o PRM e o algoritmo de apostas são efetivamente métodos complementares.

O tempo de execução no PRM-EB não foi motivo de preocupação nas experimentações. Com efeito, assim como acontece com algoritmos evolutivos, a precisão e convergência sobrepujam a premência de economizar tempo. Ademais, no exemplo de um chão de fábrica (aonde um robô transportador fará a mesma rota inúmeras vezes) a

economia de tempo que o robô terá, ao final do dia de trabalho, tendo escolhido sempre a menor rota, certamente é mais significativa do que o tempo de execução gasto para achar a melhor rota pelo PRM-EB.

7.1 Trabalhos futuros

Alguns trabalhos futuros poderiam aperfeiçoar ainda mais o algoritmo proposto nesta dissertação, tais como:

1. Testar o PRM-EB em ambientes dinâmicos;
2. Inserir otimizações pontuais no PRM-EB (a otimização por pontos médios, por exemplo, é demasiadamente simples e poderia ser substituída por métodos mais elaborados);
3. Testar o PRM-EB em ambientes estáticos com taxa de ocupação de obstáculos acima de 50% ou superior;
4. Testar o PRM-EB em ambientes maiores e também em ambientes com mais dimensões.

Referências bibliográficas

1. R. S. Ortigoza et all. Wheeled Mobile Robots: A Review. *IEEE Latin America Transactions*, vol. 10, nº6, 2012, pp. 2209-2217.
2. L. C. Neto and A. H. Hirakawa. An Approach by Straight Line Segment Adaptive Techniques in Robot Navigation. *IEEE Latin America Transactions*, vol 12, nº7, 2014, pp. 1292-1297.
3. R. Geraerts e M. Overmars. A Comparative Study of Probabilistic Roadmap Planners. *Algorithmic Foundations of Robotics V*, 2004, pp. 43-57.
4. M. Rantanen e M. Juhola. Using probabilistic roadmaps in changing environments. *Computer Animation and Virtual Worlds*, vol. 25, nº1, 2014, pp. 17-31.
5. M. Rantanen e M. Juhola. A configuration Deactivation Algorithm for Boosting Porbabilistic Roadmap Planing of robots. *International Journey of Automation and Computing*, vol. 9, nº2, 2012, pp. 155-164.
6. S. Kumar e S. Chakravorts. Adaptative sampling for generalized probabilistic roadmaps. *Journal of Control Theory and Applications*, vol. 10, nº1, 2012, pp. 1-10.
7. B. Adorno. Planejamento probabilístico de rotas no espaço de configuração e sua aplicação em robótica móvel, 2008, Dissertação (Engenharia), Universidade de Brasília.
8. L. Kavraki et all. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEE Transactions on Robotics and Automation*, vol. 12, nº4, 1996, pp. 565-580.
9. S. La Valle e J. Kuffner Jr. Randomized Kinodynamic Planning. *The International Journal of Robotics*, vol. 20, nº5, 2001, pp. 378-400.

10. D. Hsu, J.C. Latombe e R. Motwani. Path Planning in Expansive Configuration Spaces. *Robotics and Automation*, vol. 3, 1997, pp. 2719-2726.
11. O. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. *IEEE International Conference on Robotic Research*, 1985, pp. 500-505.
12. S. Carmpim e G. Pillonetto. Motion planning using adaptative random walks. *IEEE Transactions on Robotics*, vol. 21, nº1, 2005, pp. 129-136.
13. O. Salzman et all. Sparsification of motion-planning roadmaps by edge contraction. *The International Journal of Robotics Research*, vol. 33, nº14, 2014, pp. 1711-1725.
14. Y. Li et all.. K-order surrounding roadmaps path planner for robot path planning. *Journal of Intelligent & Robotic System*, vol. 75, nº3-4, 2014, pp. 493-516.
15. M. Rantanen e M. Juhola. Speeding up probabilistic roadmap planners with locality-sensitive hashing. *Robotica*, vol. 33, nº7, 2015, pp. 1491-1506.
16. A. Dobson e K. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *Journal International of Robotics Research*, vol.33, nº1, 2014, pp. 18-47.
17. M. Rantanem. A conectivity-Based Method for Enhancing Sampling in Probabilistic Roadmap Planners. *Journal of Intelligence and Robotic Systems*, vol. 64, 2011, pp. 161-178.
18. M. Kallmann e M. Mataric. Motion Planning Using Dynamic Roadmaps. *International Conference on Robotics & Automation*, Abr 2014.
19. P. Demasi. Heurísticas Baseadas em Apostas para Problema de Otimização Combinatória. Tese de Ciência da Computação, Universidade Federal do Rio de Janeiro, 2015.
20. P. Corke. Robotics Toolbox for Matlab.

21. L. Ma et all.. Merging grid maps of different resolutions by scalling registration. *Robotica*, Mar 2015.
22. B. Raveh et all.. A Litte More, a Lot Better: Improving Path Quality by a Path-Merging Algortithm. *IEEE Transactions on Robotics*, vol. 2, 2011, pp. 365-371.
23. Y. Li e K.E. Bekris, Learning approximate cost-to-go metrics to improve sampling-based motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4196-4201.
24. R. Luna et all. Anytime solution optimization for sampling-based motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 5068-5074.
25. C. E. Thomaz et all.. Mobile robot path planning using genetic algorithms. *International Work-Conference on Artificial and Natural Networks (IWANN'99)*, 1999, vol.1, pp. 671-679.
26. R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. *IEEE International Conference on Robotics and Atuomation*, 2000, vol.1, pp. 521-528.
27. S. Karaman e E. Frazzoli. Sampling-based algotithms for optimal motion planning. International Journey of Robotics Research, vol.7, 2011, pp. 846-894.
28. J. D. Marble e K. E. Bekris. Towards small asymptotically near-optimal roadmaps. *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 2557-2562.
29. F. Yuan et all.. A hybrid sampling strategy with optimized Probabilistic Roadmap Method. *International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2015, pp. 2298-2302.

30. S. Jeanita Bassani. Estudo comparativo entre algoritmo A* e busca em largura para planejamento de caminho de personagens em jogos do tipo e pacman, 2005, Bacharelado (Ciências da Computação), Universidade Regional de Blumenau.
31. N. Achour, A. Lakhdari e F. Ferguene. Motion Planning for Car-Like Robots Using Hierarchical Genetic Algorithms. *Proceedings of the World Congress of Engineering (WCE)*, 2013.
32. J. Moore. Complete Book of Sports Betting: a new, no-nonsense approach to sports gambling. Lyle Stuart, 1996.
33. E. Thorp. The Mathematics of Gambling. Gambling Times Publishing, 1984.
34. A. Wilson. The Casino Gamber's guide. Harpercollins, 1970.
35. M. Mitchell. An introduction to Genetic Algorithms, 1996.
36. F. A. Breve, Otmização por Enxame de Partículas (PSO) e Otimização por Colônias de Formigas (ASO) aplicadas ao Problema do Caixeiro Viajante (TSP), Bacharelado, Universidade de São Pulo, 2007.
37. D. E. Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning. Addison-Wesley, 1989.
38. Engineering Design Centre, New Castle University. Roullete Wheel Selection. Disponível em: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php. Acesso em 14/04/2016.
39. D. Filitto, Algoritmos Genéticos: Uma visão explanatória. *Saber acadêmico*, 2008, nº6, pp. 136-143.
40. Projero ISIS, Algoritmos genéticos. Disponível em: http://www.oocities.org/igoryepes/visualizar2.htm. Acesso em: 14/04/2016.

41. Gamasutra, Overview of Motion planning. Disponível em: http://www.gamasutra.com/blogs/MattKlingensmith/20130907/199787/Overview_of_Motion_Planning. Acesso em: 14/04/2016.
42. A. Lakhdari e N. Achour. Probabilistic roadmaps and hierarchical genetic algorithms for optimal motion planning. *Science and Information Conference (SAI)*, 2014. pp. 221-225.
43. B. Palma, C. Lima, W. Caarls e D. Vettorazzi. Which Probabilisitc Roadmap Method Should Be Used by a Robot in an Actual Environment? An Analysis of the Main Methods through Smulations. *Revista IEEE América Latina*, April 2016, vol. 14.
44. R. Lôbo. Otimizações no FIRN: um Framework para Navegação Inteligente de Robôs, 2012. Dissertação (Mestrado em Informática), Universidade Federal do Rio de Janeiro.
45. P. Moratori. Análise de estabilidade e robustez de controladores nebulosos: aplicação ao controle de trajetória de robôs, 2006. Dissertação (Mestrado em Informática), Universidade Federal do Rio de Janeiro.
46. E. Piotrowski e M. Schroeder. Kelly criterion revisited: optimal bets, University of Byaltistok, Department of Theoretical Physics, 2006. Departamental Working Papers 24.