Sistemas Inteligentes - Busca Exploratória X Explotação

Tiago Gonçalves da Silva[1], **Guilherme Aguilar de Oliveira**[1]

[1]Departamento de Informática
Universidade Tecnológica Federal do Paraná (UTFPR) – Curitiba, PR – Brazil

tiagosilva.2019@alunos.utfpr.edu.br, guilhermeoliveira.2019@alunos.utfpr.edu.br

1. Introdução

Dentro do problema estabelecido consideramos estes dois grandes subproblemas:

1. Problema de otimização de exploração por bateria e tempo pelo robô explorador;
2. Problema de salvar o maior número de vítimas no menor tempo;

No problema 1 temos os seguintes subproblemas:

a. Como fazer com que o robô não tente caminhos bloqueados (como paredes e limites do mapa) muito frequentemente?
b. Como fazer para caso o robô explore uma região desconhecida mais próxima depois de conhecer uma grande região para que ele não permaneça em um laço infinito?
c. Como saber qual a hora de voltar com base na bateria e no tempo?
d. Como achar o maior número de vítimas possível?

Já no problema 2 temos esses outros subproblemas:

a. Como salvar o maior número de vítimas possíveis com a limitação da bateria e o tempo?
b. Como achar o menor caminho entre a base e X vítimas? sendo X a quantidade de vítimas que o robô salvador consegue carregar (problema do caixeiro viajante).
c. Como otimizar o uso da bateria visto que esta leva um tempo fixo para ser recarregada aos 100% independente da carga atual?

Todos esses subproblemas conseguimos lidar nem sempre encontrando a solução ótima, porém soluções adequadas e eficientes foram encontradas.

2. Fundamentação Teórica

Para auxiliar na resolução do problema, foram utilizadas um conjunto de buscas tanto busca clássica como busca online. Dos algoritmos mais notáveis podemos citar

- busca em largura em que é possível descobrir o nó mais próximo de outros em menor número de saltos (desconsiderando os custos).
- busca em profundidade online em que foi possível explorar de maneira eficiente um ambiente desconhecido.
- busca A* em que foi utilizada uma heurística consistente e admissível para obter a solução ótima de maneira eficiente em uma região conhecida;

3. Metodologia

3.1. Modelagem Ambiente

O ambiente possui a forma de uma grade, optou-se por representá-lo dentro do programa com uma matriz de inteiros, no qual cada posição da matriz corresponde a um quadrado no labirinto, as posições da matriz do labirinto foram preenchidas recebendo os seguintes valores:

- 0 caso esteja bloqueada por barreira
- 1 caso seja um espaço vazio
- Índice das vítimas começando em 2 caso possua vítima

O robô vasculhador possui uma matriz para o labirinto com as mesmas dimensões da matriz original, porém com todas as duas posições inicializadas com o valor -1 para caracterizar um espaço desconhecido. As ações que podem ser executadas pelos robôs foram consideradas no programa como valores inteiros, sendo elas

1. Mover norte
2. Mover sul
3. Mover leste
4. Mover oeste
5. Mover nordeste
6. Mover noroeste
7. Mover sudeste
8. Mover sudoeste
9. Salvar vítima
10. Carregar carga
11. Carregar bateria
12. Ler sinais vitais da vítima

3.2. Solução escolhida - Robô Vasculhador

O robô vasculhador têm três estados: inicial, explorando e voltando. O estado inicial é o estado padrão e usa a busca em profundidade online para explorar uma região significativa do espaço de estados. A busca em profundidade utilizada foi modificada para mitigar o subproblema 1a da seguinte forma: o robô tem uma lista ordenada das ações possíveis; são priorizadas as ações no começo da lista; caso a execução de uma ação resulte em fracasso (o robô bate em uma parede), essa ação irá ir para o final da fila com a prioridade mais baixa; Caso uma ação leve a um caminho já explorado, o robô pula essa ação e prioriza as ações que o levam a caminhos desconhecidos.

O estado explorando ocorre quando todas ações do robô o levam a estados já explorados, então, a tarefa do explorador é encontrar o estado não explorado mais próximo. Para encontrar esse estado não explorado mais próximo é utilizado a busca em largura que resulta em um caminho com o menor número de saltos. Por fim se não houverem caminhos não explorados, o robô entra no estado voltando em que este volta para a base.

O estado voltando é o estado em que o robô traça e segue o menor caminho conhecido por meio do algoritmo A* de volta para a base (0,0). A cada ação, o robô calcula se sua bateria tem duração o suficiente de fazer a ação e voltar para a base pelo menor caminho, e se a bateria não for o suficiente, o robô entra no estado voltando e volta para recarregar a bateria ou concluir a exploração.

3.3. Solução escolhida - Robô Salvador

O robô salvador também possui uma matriz para o labirinto, que recebe logo após o robô vasculhador retornar de sua busca. Para o planejamento das ações de salvamento optou-se por uma busca offline, na qual o robô planeja todas as ações antes de executá-las. Para a escolha da ordem de salvamento das vítimas foi utilizada uma fila de prioridades, ordenando de maneira crescente as distâncias das vítimas até a base.

O ciclo de pensamento do robô salvador é: selecionar a vítima do topo da fila de prioridades; calcular o custo e a sequência de ações, utilizando a busca informada A*, de dois caminhos: o de ida da posição atual (inicialmente a base) para a posição dela e o de volta da posição dela para a base; caso o custo somado da ida e da volta forem inferiores ao tempo e à bateria restante: guarda as duas sequências de ações na matriz solução, marca a posição atual como a da vítima salva, decrementa o tempo com o custo da ida, repete até que a carga máxima de suprimentos seja atingida; quando o robô não conseguir salvar mais vítimas, guarda o último caminho de volta na matriz solução e reinicia o processo até acabar o tempo ou ser necessário recarregar a bateria.

3.4. Buscas escolhidas

A busca em profundidade online para o robô vasculhador foi adequada pois, já que o robô só consegue perceber o ponto que está, esta busca permite um backtracking menor do que a busca em largura. Já a busca em largura offline foi utilizada para achar o ponto desconhecido com menores saltos do robô vasculhador.

Para as buscas offline do problema optou-se por usar a busca informada A*, com a heurística sendo a distância euclidiana entre a posição analisada e a posição objetivo. O motivo da escolha desta busca é pelo fator de maior dificuldade para o salvamento das vítimas ser o custo das ações, tanto em tempo quanto em bateria e o espaço de estados ser pequeno, pelo motivo de ela retornar uma solução com o caminho de menor ações para a posição objetivo, sendo as vítimas ou a base.

No algoritmo A*, a heurística utilizada é a distância euclidiana entre o ponto objetivo e o ponto de fronteira. Essa heurística é admissível pois se temos o custo dos movimentos horizontal e vertical igual a 1, a heurística também retornaria custo 1. Já para o custo das diagonais sendo igual a 1.5 na heurística esse custo valeria $\sqrt{2} \approx 1.41$ satisfazendo a condição de admissível $h(n) \leq h^*(n)$. A distância euclidiana é consistente pois a própria definição de distância implica em desigualdade.

4. Resultados e análise

Os resultados foram bem positivos, pois é possível explorar aproximadamente n pontos do espaço gastando $2n$ de bateria. Porém o algoritmo consome uma quantidade de memória considerável sendo no mínimo $O(n^2)$ tal que para espaços de estados muito grandes estoura a memória.

5. Conclusões

Consideramos que o trabalho atingiu o objetivo que era adequar as buscas online e offline para melhor resolver problemas de exploração e explotação. Consideramos que a próxima melhoria a ser feita para esta solução seja a implementação de um algoritmo de busca

gulosa, pois a única busca offline implementada foi a A*, que pode explodir quando o espaço de estados for muito grande. Com isso o robô poderia escolher qual busca offline fazer com base no tamanho do espaço de estados, aumentando a robustez do programa.

Outro ponto a ser explorado é a escolha da ordem de salvamento das vítimas pelo robô salvador, que atualmente as escolhe com base na menor distância entre elas e a base, outras formas de escolha seriam: escolha por sinais vitais, escolhas por distância entre vítimas ou escolha por algoritmo mais complexo como problema da mochila.

6. Referências bibliográficas

RUSSELL, Stuart J.; NORVIG, Peter. Inteligência artificial. Rio de Janeiro, RJ: Elsevier, 2004. 1021 p. ISBN 9788535211771.

7. Apêndice

Para executar o código basta executar no diretório que está o código os seguintes comandos no linux com o compilador g++ instalado:

$ make

$ ./robo

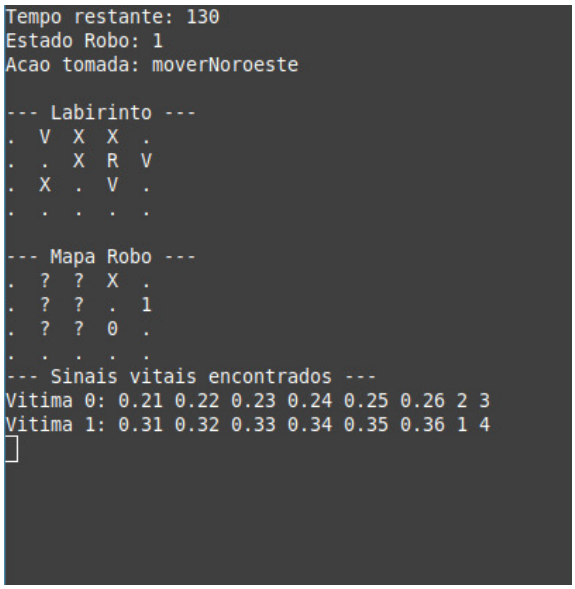

Figure 1. Tela do robô vasculhador

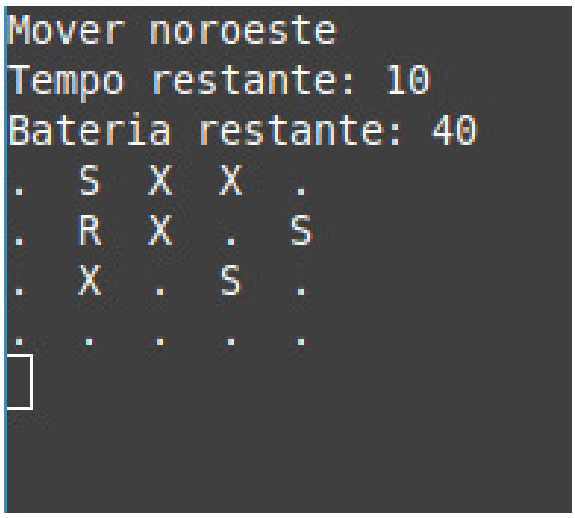

Figure 2. Planejamento do robô salvador

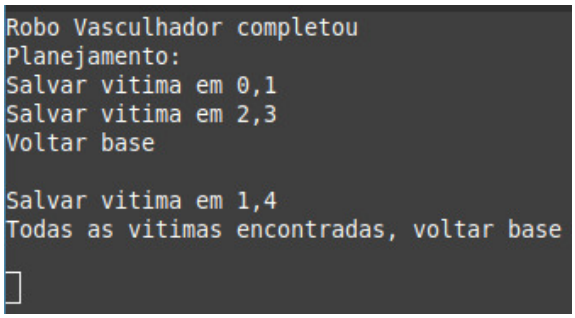

Figure 3. Tela do robô salvador