Projeto 1 - MC906A

Marcelo Biagi Martins
183303

Obs: para rodar o projeto: "python3 projeto1.py"

1 Resumo

O projeto 1 de Introdução à inteligência artificial teve como objetivo exercitar a capacidade de modelagem através de um problema de busca. Com o uso de ferramentas como a biblioteca AIMA de python[1], foi possível obter acesso à diversas implementações de algoritmos de busca para futuras comparações de eficiência. Com isso, o projeto se resumia à modelagem do problema e criação do universo para armazenar o caminho e os pontos iniciais e finais percorridos pelo "robô", que deveria chegar de um lugar a outro.

Na comparação de resultados, foram obtidos resultados já esperados, que comprovam a alta eficiência do algoritmo breadth search de grafos(BFS) e a péssima eficiência do mesmo algoritmo para árvores grandes. Além disso, alguns resultados inesperados com uso de heurísticas em algoritmos não tão utilizados na graduação puderam ser analisados. Somado a isso, a biblioteca matplotlib de python serviu como ferramenta para auxiliar na observação dos dados resultantes, através da plotagem do mapa após a execução. Dessa forma, dados sobre consumo de tempo e memória puderam ser comparados para posterior análise.

Sendo a área de algoritmos de busca de imensa importância para a história da ciência da computação, esse projeto serve como base para que a inteligência artificial possa ser compreendida de forma simples, com o uso de algoritmos conhecidos já conhecidos por alunos da área.

2 Introdução

O problema tratado neste projeto consistia em variar diversos parâmetros para verificar a eficiência do modelo desenvolvido. O cenário fornecido consiste em um mapa quadrado 61x61 com paredes nas quatro bordas, além de duas paredes dentro deste espaço, em (20, [0:39]) e (40, [21,60]). O "robô" tinha como objetivo chegar em um ponto final pré definido, que poderia variar, assim como o mapa e o ponto inicial. Já a modelagem buscava melhorar a o gasto de tempo e memória do robô em seu percurso. Poderiam ser testadas diversas configurações de movimentos, pontos iniciais e finais e algoritmos de busca informados e sem informação, a fim de encontrar os melhores resultados.

O modelo adotado neste relatório usou sempre o mesmo mapa com os mesmo obstáculos. O ponto inicial também ficou fixo em (10, 10, Norte). Já o ponto objetivo foi constantemente variado, juntamente com os algoritmos, para que uma comparação de eficiência justa pudesse ser feita, seguindo os mesmos parâmetros. Os algoritmos utilizados foram: breadth first tree search, depth first tree search, breadth first graph search, depth first graph search e best fit search com duas heurísticas distintas.

3 O modelo

A discretização do modelo foi feita seguindo a forma sugerida no enunciado, com três incógnitas (x, y, θ). O movimento do robô era limitado, inicialmente, em andar para a frente e girar para os oito pontos cardeais. Posteriormente, foram realizados testes utilizando o algoritmo de busca Breadth First Tree Search do AIMA e buscando comparar a eficiência de se usar oito ou quatro pontos cardeais. Os resultados obtidos estão representados nas tabelas 1 e 2.

Já se tratando do código, a criação do mapa foi feita através de cláusulas if-else, que verificavam se o robô saia da área estipulada ou batia nas paredes. Essas cláusulas citadas realizavam as verificações necessárias para

decidir as ações possíveis que o objeto poderia realizar. Eram duas disponíveis: andar e girar, sendo que o robô apenas girava no sentido horário e as direções estavam representadas por números(Norte = 1, Leste = 3, Sul = 5 e Oeste = 7). Por fim, para realizar a impressão do caminho percorrido, foi criada uma matriz para armazenar esses valores, as bordas do mapa e as paredes. Nesta matriz, o caminho percorrido era verde, os pontos analisados durante o percurso eram amarelos e as bordas e paredes eram azuis. Por fim, a biblioteca pyplot foi utilizada para imprimir o caminho encontrado.

3.1 Tabelas e figuras

Tabela 1: **Breadth first tree search com 4 direções(N, S, L, O)**

(X,Y)	Tempo(s)	Nós visitados	Memória utilizada(Kb)
(10,10)	0.0002	0	0
(10,15)	0.0036	62	0
(15,10)	0.0154	190	62
(15,20)	9,8741	130828	31780
(21,15)	não chega	-	-
(15,21)	19,1561	261393	63300
(15,25)	não chega	-	-
(15,28)	não chega	-	-
(21,1)	não chega	-	-

Tabela 2: **Breadth first tree search com 8 direções(N, NE, L, SE, S, SO, O, NO)**

(X,Y)	Tempo(s)	Nós visitados	Memória utilizada(Kb)
(10,10)	0.0002	0	0
(10,15)	0.0051	62	0
(15,10)	0.0315	446	62
(15,20)	0,2814	4127	920
(21,15)	não chega	-	-
(15,21)	0,5421	8240	2080
(15,25)	9.6856	128660	31120
(15,28)	73,7961	1020820	247100
(21,1)	não chega	-	-

Observação: Para os resultados que demoravam mais do que 90 segundos, "não chega"foi adicionado à tabela.

Além das tabelas, a figura 1 exemplifica o funcionamento dos dois algoritmos. Em verde se encontra o caminho do robô até o ponto final e em amarelo estão os nós analisados para poder chegar no objetivo. O ponto inicial é (10,10, Norte) em ambos.

Para esse algoritmo em questão, o uso de oito pontos cardeais apresenta uma vantagem grande em relação ao uso de apenas quatro. No entanto, nos algoritmos seguintes e mais eficientes, essa mudança não surtiu grandes efeitos, visto que o mapa é pequeno e fixo. Portanto, o uso de apenas quatro pontos foi escolhido para facilitar a comparação dos outros parâmetros: nós visitados e memória utilizada.

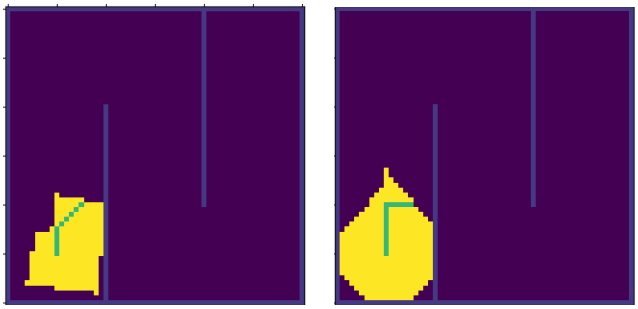

Figura 1: Breadth Tree Search com 8 e 4 pontos cardeais

4 Resultados e Discussão

Foram escolhidos quatro algoritmos para realizar a comparação de eficiência: Breadth First Graph Search, Depth First Graph Search e Best Fit Search com duas heurísticas distintas, sendo elas: H1: minimizar a distância do ponto atual até o ponto objetivo e H2: minimizar a distância entre o módulo das abcissas do ponto atual e do ponto objetivo. Segue abaixo as figuras 2, 3, 4 e 5 ilustrando o funcionamento de tais algoritmos, sendo o ponto inicial sempre (10, 10, Norte). A direção do ponto final é sempre omitida, pois os algoritmos apenas comparam as ordenadas e as abcissas para decidir se o destino foi alcançado, não importando se o robô chega ao destino com um ângulo diferente.

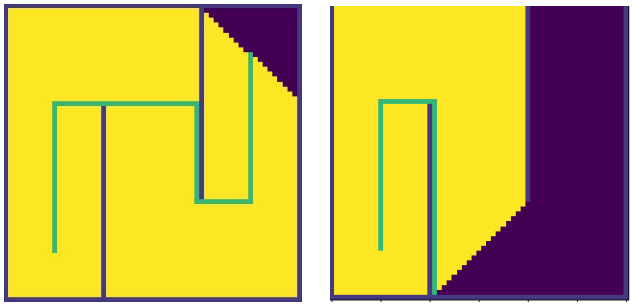

Figura 2: BFS no ponto (50,50) e no ponto (21,1)

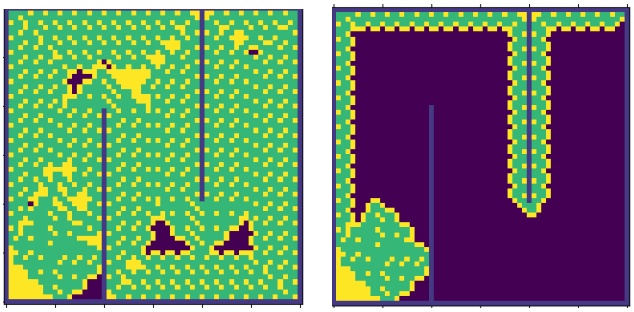

Figura 3: DFS no ponto (50,50) e no ponto (59,59)

Para visualizar melhor as diferenças, foram feitos gráficos que comparam a quantidade de tempo gasto e memória utilizada, representados na figura 6.

Além disso, foram feitos gráficos que representam como as paredes influenciam na eficiência dos algoritmos. Dado que as mesmas são fixas, foi fixado a ordenada em 30 e variou-se a abcissa para analisar diferenças na eficiência. Os gráficos estão representados na figura 7.

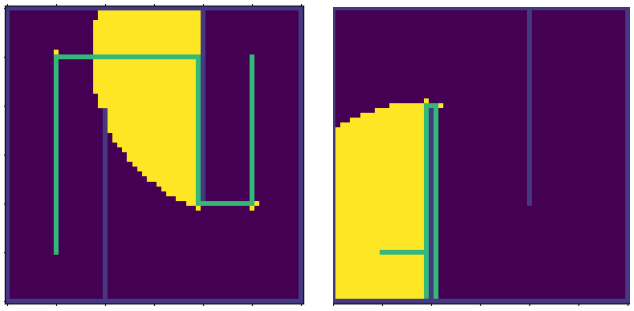

Figura 4: Best Fit com H1 no ponto (50,50) e (21,1)

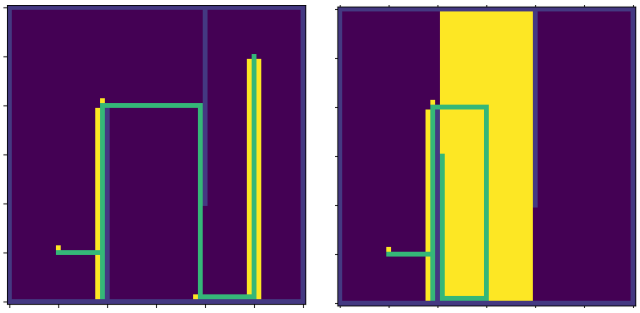

Figura 5: Best Fit com H2 no ponto (50,50) e (21,30)

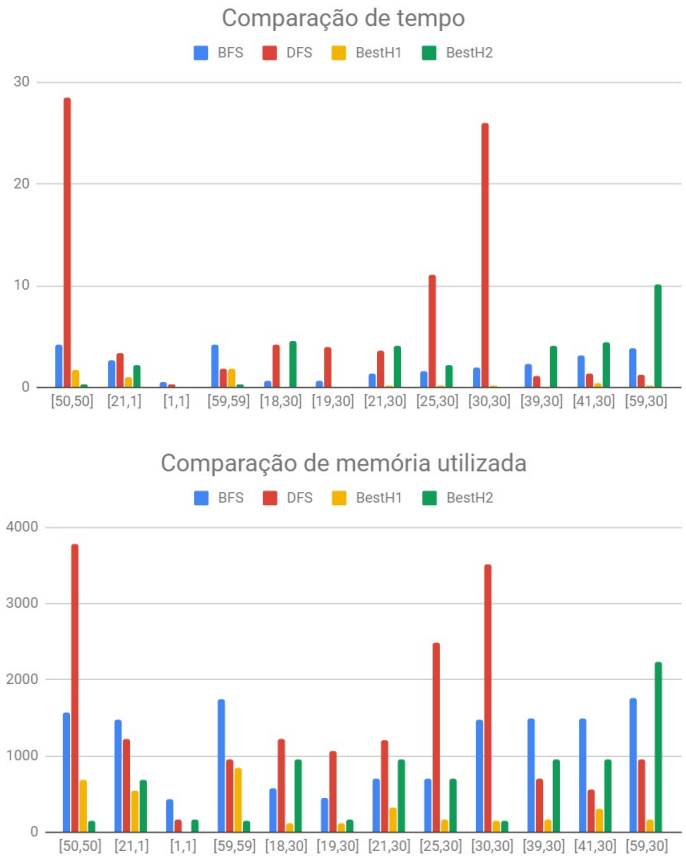

Figura 6: Gráficos de tempo e memória consumidos

Figura 7: Gráficos de análise de abcissa

4.1 Análise dos dados

Pela análise das imagens e gráficos apresentados, pode-se concluir que, de todos os algoritmos apresentados, o Breadth First Graph Search pode ser considerado o mais linear em eficiência. Pela comparação de tempo e memória, podemos ver que este é o algoritmo com menos variações bruscas de valores, mantendo sempre resultados constantes, o que o torna a melhor escolha para uma análise sem informações pré-definidas, já que ele não se baseia na distância até o objetivo e sim em encontrar o melhor caminho. Caso o mapa variasse, incluindo a posição das paredes e o tamanho geral, ou se o ponto objetivo não fosse conhecido de antemão, o BFS seria a melhor opção.

Como o valor final já é conhecido, o Best Fit, tanto com o uso da heurística 1 quanto com a 2, possui certas vantagens em relação aos outros, pois está o tempo todo calculando a distância até o ponto final. No entanto, caso as paredes possuíssem quinas, ou o mapa fosse uma espécie de labirinto, esse algoritmo teria sua eficiência muito afetada, pois pode-se analisar da figura 7 que as paredes mudam bruscamente o tempo gasto pelo best fit para encontrar o objetivo. Portanto, um mapa com mais obstáculos seria um forte motivo para preferir BFS no lugar do Best Fit. No entanto, para um mapa simples como o utilizado neste projeto, o best fit se instaura como uma forte escolha de algoritmo, principalmente se atrelado à heurística 1, que possui vantagens em relação a 2 com relação a tempo e memória consumidos.

Já com relação ao Depth First Graph Search, têm-se os resultados mais instáveis de todos. Em geral, o DFS apresenta algumas vantagens clássicas em relação ao BFS, como uso de memória e eficiência em encontrar as soluções quando elas estão longes do início(dado que DFS começa sua execução pelas folhas da árvore e não pela raiz, como ocorre com o BFS)[2]. No entanto, a solução encontrada por vezes é sub-ótima. Nos resultados apresentados, nota-se pelas imagens que o caminho realizado pelo DFS é totalmente diferente dos outros, com excesso de comparações e nós visitados. Além disso, a solução costuma ser a primeira encontrada, que pode não ser boa o suficiente dependendo do contexto em que será utilizada. Sendo assim, para o problema proposto, o DFS não apresenta resultados satisfatórios com relação algoritmos restantes.

5 Conclusões

Conclui-se, portanto, que a melhor escolha "cega"seria o Breadth First Tree Search. Este é o algoritmo que apresenta os melhores resultados quando não há informações do mapa de antemão. No entanto, visto que o caso analisado possui já um mapa fixo e com obstáculos bem definidos, tanto BFS quanto Best Fit com heurística 1 são bons o suficiente.

Quanto aos parâmetros, nota-se que as paredes influenciam muito no resultado do best fit search com as heurísticas de distância, já o BFS e o DFS se mantêm razoavelmente constantes(com exceção da anomalia apresentada pelo DFS no ponto 30, que está entre as paredes). Já quanto ao consumo de memória, nota-se uma clara vantagem de best fit search em quase todos os pontos analisados.

Por fim, com relação ao destino, caso o usuário saiba de antemão que o ponto objetivo está próximo, Best Fit com heurística 2 também é uma boa opção, pois possui grande vantagem com relação ao tempo para pontos objetivos próximos do início. Além disso, caso encontrar a solução ótima não seja necessário e tempo não seja uma prioridade ou ainda seja conhecido de antemão que o ponto final está longe do início, DFS pode ser utilizado no lugar de BFS.

Referências

[1] *AIMA python repository*, Disponível em https://github.com/aimacode/aima-python 1

[2] *Geeks for Geeks, DFS vs BFS*, Disponível em https://www.geeksforgeeks.org/bfs-vs-dfs-binary-tree/ 5