Projeto 1 - MC906 A

LUIZ EDUARDO CARTOLANO *

*Engenharia de Computação - Graduação
E-mail: 1183012@dac.unicamp.br

***Resumo* – O trabalho em questão visa resolver um problema no qual um robô deve ser capaz de sair de um ponto inicial e chegar ao um ponto final em um plano cartesiano, desviando de eventuais obstáculos e usando métodos de busca, supervisionados e não supervisionados para alcançar seu objetivo. Para implementar uma solução fez-se uso da linguagem *Python* e de suas bibliotecas, o que facilitou bastante as implementações feitas. O resultado final do projeto foi satisfatório, uma vez que foi possível obter tempos de execução, além de outras informações importantes sobre os algoritmos estudados.**

***Palavras-chave* – Inteligência artificial - Problemas de Busca - Robótica**

I. INTRODUÇÃO

Um problema muito comum no ramo da inteligência artificial é o de desenvolver um sistema capaz de atingir um determinado objetivo uma vez descrita seu estado e conjunto de ações possíveis. Idealmente, esse conjunto de ações é então repassado a um agente que irá solucionar o problema. Como pode ser visto com mais detalhes em [1], o design desses problemas tem acompanhado a o ramo desde seus primeiros dias, e um alto número de algoritmos tem sido desenvolvido para resolver, cada vez melhor, esse tipo de problema.

Este trabalho encontra-se organizado da seguinte forma: a seção 2 apresenta a maneira como o trabalho foi organizado a fim de solucionar o problema. A seção 3 descreve os testes que foram feitos para validar a solução adota e também as metodologias abordadas. Os resultados são apresentados na seção 4, e as conclusões são apresentadas na seção 5.

Os códigos referentes as soluções implementadas neste projeto encontram-se disponíveis em 2 e 3.

II. TRABALHO PROPOSTO

O trabalho proposto, que pode ser visto com mais detalhes em 4, solicitava a implementação de algoritmos de busca conhecidos para solucionar um problema no qual um robô deveria sair de uma posição inicial conhecida e chegar a um destino também conhecido, desviando de eventuais obstáculos no caminho. Para explicar o trabalho de maneira mais detalhada nas próximas subseções, explicando, de modo mais detalhado, o problema, as ferramentas utilizadas para resolvê-lo e a organização macro do projeto.

A. Problema

O problema proposto no enunciado, consistem em um robô que se encontra na posição (x,y,θ) de um plano cartesiano de dimensões $60x60$, como o mostrado pela Figura 1. As paredes, representas por linhas pretas no mapa são intransponíveis, logo o robô deve ser capaz de desviar delas ao longo do seu percurso.

O objetivo do trabalho é implementar dois algoritmos de busca não supervisionados e um algoritmo supervisionado (usando duas heurísticas diferentes) para permitir que o robô cumpra com seu objetivo.

B. Ferramentas utilizadas

A fim de realizar as implementações propostas utilizou-se como linguagem de programação a versão 3.7 do *Python*. Em conjunto com ela, usou-se a plataforma *Jupyter Notebook*, uma vez que com ela é possível acompanhar de maneira mais fácil os resultados dos algoritmos aplicados e também pois ela permite a execução de trechos específicos de código, o que garante maior flexibilidade no uso dos diversos algoritmos usados.

Outro detalhe importante das ferramentas utilizadas foi o uso da biblioteca *AIMA*, uma biblioteca escrita em *Python* que implementa diversas das buscas usadas no projeto. Sua documentação, instruções de instalação e uso, podem ser vistas com detalhes em 5.

C. Organização do projeto

O projeto foi estruturado de modo que ele possui quatro arquivos *.py* principais que implementam a modelagem do problema e as buscas supervisionadas e não supervisionadas, sendo estas separadas uma em cada arquivo, dois dos quatro arquivos foram usados como um ambiente que a princípio serviria como alguns testes a fim de entender melhor os códigos, contudo algumas informações interessantes foram obtidas neles, de modo que também estão sendo referenciados neste trabalho. Além destes, o projeto ainda conta com outros sete arquivos *.py* oriundos da biblioteca *AIMA*, sendo o *search.py* o principal deles para nosso escopo, visto que é o responsável por implementar os algoritmos de busca utilizados. Para garantir o funcionamento dos códigos ainda é necessário a presença do diretório *aima-data* na mesma pasta.

III. MATERIAIS E MÉTODOS

Nesta seção iremos explicar com mais detalhes a modelagem do problema, os algoritmos de busca utilizados, os testes realizados para analisar a corretude dos algoritmos aplicados e também iremos comentar sobre eventuais restrições apresentadas nas soluções.

A. Modelagem do Problema

A modelagem do problema consistiu no maior desafio presente na implementação, visto que os algoritmos de busca foram abrangidos pela biblioteca *AIMA*.

O primeiro desafio na modelagem foi a criação do *grid* no qual o robô iria andar buscando o *goal*. A fim de utilizar as funções do *AIMA* sem realizar modificações foi preciso fazer com que o mapa (como o da Figura 1) fosse estruturado como um grafo. Para tal, adotou-se a seguinte estratégia, criou-se uma estrutura de dicionário na qual cada nó representa uma chave e, associada a esta chave, temos um dicionário dos seus vizinhos. A fim de facilitar a criação do mapa, não levou-se em consideração, na sua formação, as paredes.

Feita a modelagem do *grid*, foi preciso realizar a modelagem do problema de fato, ou seja, realizar a definição do estado, objetivo, ações realizadas e assim por diante. A biblioteca usada possuia a implementação de uma classe abstrata Problema, então, foi preciso extender a mesma e sobrescrever os métodos necessários. Para definir o problema do robô proposto no enunciado, foram definidos:

1) Estado:
   Posição (x,y) e o objetivo.
2) Ações:
   Retorna uma lista das ações possíveis a partir de um estado conhecido. Nesse momento, eram consideradas as paredes.
3) Teste do estado objetivo:
   Compara se as posições (x,y) atuais são iguais as posições (x,y) do objetivo.
4) Custo do caminho:
   Para a abordagem adotada, qualquer movimentação feita possui custo um (1).
5) Heurísticas utilizadas:
   Foram utilizadas duas heurísticas na solução do problema, a primeira delas foi a de distância euclidiana, que levava em consideração a distância em linha reta entre os dois pontos (dado pela Equação 1. A segunda heurística utilizada foi a Heurística de Manhattan, ela define a menor distância possível que um carro é capaz de percorrer em uma malha urbana reticulada ortogonal, como se encontram em zonas como Manhattan. Ela leva em consideração a soma da diferença absoluta entre as coordenas x e y, sua fórmula é dada pela Equação 2. Ambas as heurísticas podem ser vistas com mais detalhes em 6.

Como mostrado pela discretização do estado, no modelo adotado, o robô consegue se movimentar em apenas 2 direções, para cima/baixo e para os lados. Tal situação foi escolhida a fim de diminuir a quantidade de variáveis que deveriam ser manuseadas ao retornar as possíveis ações do robô, facilitando assim a implementação, visto que, acreditava-se que uma discretização mais complexa não traria ganhos significativos de desempenho.

B. Algoritmos de Busca implementados

Visto que o objetivo principal do trabalho estava na modelagem do problema, e não na implementação dos algoritmos de busca, nesta seção será dada uma breve explicação do funcionamento das buscas utilizadas para que possamos discutir com maior propriedade os resultados encontrados futuramente. Na referência [7] é possível encontrar a explicação mais detalhada e provas associadas aos algoritmos.

1) BFS - Busca em Largura: A busca em largura começa seu trabalho por um vértice fornecido, que chamaremos de s, no caso deste trabalho, por exemplo, s seria o estado inicial do robô. O algoritmo então visita s, e depois todos os seus vizinhos, e então os vizinhos dos vizinhos e assim por diante, até encontrar o nó desejado.

No seu processo de busca o algoritmo numera os vértices, em sequência, na ordem na qual eles são visitados. Para tal, faz uso de uma fila.

2) DFS - Busca em Profundidade: A busca em profundidade pode ser usada para resolver uma vasta gama de problemas. Isso porque ela funciona como uma espécie de pré-processamento para a resolução de problemas concretos. A busca *DFS* ajuda na compreensão do grafo com o qual se está trabalhando, revelando sua forma e reunindo informações importantes sobre ele.

O algoritmo de busca em questão visita todos os vértices e todas as arestas do grafo numa determinada ordem e atribui um número a cada vértice: o k-ésimo vértice descoberto recebe o número k. Assim como na *BFS*, para este projeto o ponto inicial da busca é o estado inicial do robô.

3) A:* O algoritmo A* é um algoritmo para busca de caminho. Ele busca o caminho em um grafo de um vértice inicial fornecido, até um vértice final. Ele é a combinação de aproximações heurísticas como a do algoritmo *BFS* e da formalidade do Algoritmo de *Dijkstra*.

Para o uso do algoritmo em questão é necessário fornecer alguma heurística para ele. No caso do trabalho, foram usadas as heurísticas descritas na subseção de modelagem do problema.

C. Testes de Corretude e Performance

Parte fundamental do projeto são os testes que visam analisar a corretude das soluções implementadas e que nos permitem coletar insumos sobre a *performance* deles.

Para analisar a corretude desenhou-se os mapas com os caminhos percorridos pelo robô, ou seja, os nós visitados pela busca até atingir o objetivo. Os mapas plotados são como os da Figura 2. Além dos testes feitos com as posições especificadas no enunciado, plotou-se os mapas para diferentes posições iniciais, a fim de testar o mesmo sobre diferentes perspectivas.

A parte mais interessante do trabalho foram os testes realizados a fim de analisar a *performance* das soluções, visto que, foi possível executar diferentes testes, colocando o algoritmo sobre uma série de situações distintas. Para cada algoritmo implementado aplicou-se as teorias vistas em [8] e calculou-se a média, desvio padrão e intervalo de confiança da aplicação dada as situações estabelecidas no enunciado (para esse caso

executou-se o algoritmo mil vezes). Também calculou-se o custo do caminho encontrado e a relação entre a distância em linha reta entre os pontos inicial e final e o tempo gasto na execução, para esse teste variou-se o ponto inicial passando por todos os valores entre zero e quarenta e nove, com exceção das posições que representavam paredes. Coletou-se também a memória gasta por cada um dos algoritmos.

O cálculo do tempo foi feito com uso da biblioteca *time*, e a memória gasta foi observa com o auxílio da biblioteca *memory profiler*, ambas podem ser vistas em 9 e 10, respectivamente.

D. Restrições da Solução

Ainda que tenha apresentado resultados satisfatórios, a solução implementada possui algumas restrições ou problemas para caso se deseje escalar ou fazer alterações drásticas ao problema.

A primeira restrição está na criação do grafo que representa o mapa percorrido pelo robô. Da maneira que foi construído as paredes estão feitas de modo *hardcoding*, logo mudanças estruturais no mapa precisam ser feitas em código. Portanto, para uma próxima iteração, podemos construir um problema no qual eventuais obstáculos presentes no mapa possam ser construídos de maneira dinâmica.

Outra restrição da solução implementada está na modelagem do problema, uma vez que, da maneira feita, o robô não consegue andar na diagonal e nem possui noção de direção, isto é, para qual lado ele está virado. O custo dessa implementação, como veremos na próxima seção, foi a realização de caminhos pouco suaves. Sendo assim, em uma iteração futura é preciso concentrar esforços em uma modelagem que forneça uma gama maior de movimentos ao robô.

IV. Resultados e Discussão

Nesta seção iremos primeiro apresentar todos os resultados obtidos durante a execução da solução implementada, mapas obtidos, tabelas e gráficos gerados. Uma vez introduzidos, iremos discutir e comparar estes.

A. Resultados

O primeiro teste ao qual os algoritmos foram submetidos foi o para cálculo do consumo de memória gasto por cada uma das funções de busca. Os resultados obtidos podem ser vistos na Tabela I. Através dela é possível perceber que a *Busca em Profundidade - DFS* foi a menos custosa para o sistema, enquanto que a *Busca A*-Heurística de Manhattan*, foi a mais custosa.

Após calcular o gasto de memória, contabilizou-se o custo do caminho percorrido até o objetivo final em cada uma das buscas, obtendo os resultados observados na Tabela II. Ao contrário do observado no gasto de memória, dessa vez dois algoritmos apresentaram desempenho ótimo, foram eles a *Busca em Largura - BFS* e a *Busca A* - Heurística de Manhattan*. Enquanto que o maior custo ficou por conta da *Busca em Profundidade - DFS*.

Uma vez calculados o desempenho dos algoritmos para as situações fornecidas no enunciado, resolveu-se descobrir, e plotar, o caminho percorrido pelo robô para cada um dos algoritmos. O resultado para a *Busca em Largura - BFS* pode ser visto na Figura 2, o da *Busca em Profundidade - DFS* encontra-se na Figura 3, enquanto que para a *Busca A** com as heurísticas *Euclidiana* e de *Manhattan*, encontram-se, respectivamente, nas Figuras 4 e 5. Além do mapa no qual se destacou o caminho da solução encontrada pelo algoritmo também foi gerado um no qual foram marcados todos os nós visitados pelo mesmo (na cor verde), e o caminho da solução (na cor amarela). O resultado dessa aplicação para a *Busca em Largura - BFS* pode ser visto na Figura 6, o da *Busca em Profundidade - DFS* encontra-se na Figura 7, enquanto que para a *Busca A** com as heurísticas *Euclidiana* e de *Manhattan*, encontram-se, respectivamente, nas Figuras 8 e 9.

A última análise de desempenho a qual os algoritmos foram submetidas sob as condições padrões do enunciado, foi o cálculo do tempo de busca e a descoberta do intervalo de confiança da mesma, com confiança de noventa e cinco porcento (95%). Os resultados obtidos para média (com base na Equação 3), desvio padrão (Equação 4) e intervalo de confiança (Equação 5) estão disponíveis na Tabela III. O algoritmo mais rápido dos aplicados foi a *Busca em Largura - BFS* e o mais lento a *Busca em Profundidade - DFS*. Para esses valores também foram gerados histogramas que podem ser vistos nas Figuras 11, 12, 13 e 14.

Por fim, o último teste ao qual os algoritmos foram submetidos foi o cálculo do tempo para diferentes posições iniciais, estas foram variadas desde o ponto (5,5) até o ponto (49,49) sendo as únicas exceções os pontos que representavam as paredes. Com os resultados obtidos foram plotados gráficos nos quais o eixo x representa a distância em linha reta entre os pontos iniciais e finais, e o eixo y o tempo gasto na busca. O resultado para a *Busca em Largura - BFS* pode ser visto na Figura 15, para a *Busca em Profundidade - DFS* encontra-se na Figura 16, enquanto que para a *Busca A** com as heurísticas *Euclidiana* e de *Manhattan*, encontram-se, respectivamente, nas Figuras 17 e 18.

B. Discussão

Iremos começar nossa análise pelos resultados encontrados para o consumo de memória de cada um dos algoritmos, como visto na Seção IV-A, a *DFS* foi a menos custosa, contudo é interessante observar na Tabela I, que os consumos das buscas supervisionadas e não supervisionadas foram parecidas quando do mesmo tipo e bem discrepantes quando comparadas entre tipos diferentes. Contudo, os resultados aqui encontrados não são muito expressivos, devido a maneira como as bibliotecas do *Python* conseguem esse tipo de informação. Isto é, de acordo com 10, elas leem um arquivo que o sistema operacional gerencia no qual é informada quanta memória está alocada para o processo em questão. Contudo, medir a memória para chamada de funções é costumeiramente lento além de que em algumas situações o processo demora a desalocar a memória usada, atrapalhando a veracidade das informações coletadas, especialmente quando, como no caso deste trabalho, fez-se

uso de um notebook externo, como o *Jupyter*, para executar os códigos.

No âmbito do custo do caminho encontrado por cada uma das soluções é possível perceber que com exceção da *DFS*, os demais resultados encontrados foram consideravelmente próximos para os algoritmos. O que de certa forma, condiz com o esperado, visto que dada as condições iniciais a *DFS* seria realmente a mais desfavorecida, uma vez que ela percorreria quase todo o grafo até encontrar o nó objetivo.

As análise de tempo para as condições iniciais do especificadas no enunciado nos permitem concluir que a *Busca em Largura - BFS* foi a que apresentou melhor desempenho, possuindo um tempo médio consideravelmente menor e apresentando maior constância no tempo das buscas, o que se prova no menor desvio padrão entre todas. Além disso, é possível observar que ambas as heurísticas usadas na *Busca A** apresentaram resultados bem similares, o que provavelmente foi causado pela escolha de heurísticas até certo ponto similares, já que ambas, como visto na Seção III-A, estavam relacionadas a distância entre os pontos no plano cartesiano. Os mapas feitos com a marcação dos nós visitados por cada um dos algoritmos nos fornece insumos para completar a análise do tempo gasto por eles no seu caminho até a solução.

Um dos *outputs* que mais nos fornecem insumos para uma análise da qualidade da modelagem do problemas são os mapas com os caminhos percorridos pelo robô no seu destino até o nó objetivo. Como é possível ver nos mapas, os caminhos não são muito delicados e, na na maioria dos casos, passam a maior parte do tempo muito próximo as paredes. Além disso, é possível perceber que muitos dos movimentos foram feitos de maneira mais custosa do que poderiam devido a falta de mobilidade diagonal do robô, sendo assim, para uma iteração futura, esse é claramente um ponto que precisa ser melhorado. Os mapas também nos permite concluir que, com exceção da busca *DFS* as demais soluções são soluções ótimas, o que também era esperado pela literatura, como pode sem visto em [11].

Ainda no âmbito dos mapas, os mapas com os nós visitados, como o da Figura 9, nos trazem uma visão interessante sobre a maneira como os algoritmos executam. Através deles é possível ver por exemplo, que a *Busca A** com a heurística *Euclidiana* tem um comportamento muito parecido com *Busca em Largura - BFS*, sendo assim, podemos dizer que para o problema abordado não é uma boa escolha de eurística. Já *Busca A** usada junto a heurística de *Manhattan* mostra-se muito mais efetiva quando chegando próxima a seu destino final, logo, é uma boa escolha. Um aspecto comum a todas as buscas, por outro lado, foi a alta quantidade de nós visitados quando ainda bem longe do ponto final.

Por fim, uma das análise de tempo mais interessante, foi a feita variando a distância em linha entre os pontos inicial e final, pois ela nos permite comparar os resultados encontrados com a literatura das buscas em grafo. Ao analisar a busca *BFS*, por exemplo, é possível notar na Figura 15, que o tempo cresce quase que linearmente a medida que a distância aumenta, o que é de fato esperado, já que o algoritmo apresenta complexidade $O(b^{d+1})$ [1] [2]. Já a *DFS*, mostrada na Figura 16, o tempo se mantém quase constante, o que também faz sentido quando comparamos com o modo como o algoritmo é executado, inclusive, olhando para a literatura, sua complexidade é dada por $O(b^d)$, e, dado que o ponto final é sempre o mesmo, a altura da árvore não se altera. Outro detalhe interessante é quando comparamos a *Busca A** com cada uma das suas heurísticas, visto que elas suas curvas apresentam resultados praticamente opostos, sendo a heurística *Euclidiana* melhor para maiores distâncias e a de *Manhattan* para distâncias mais curtas.

Por fim, visando analisar a completude dos algoritmos, realizou-se alguns testes que podem ser encontrados em 3, através desses testes feitos foi possível perceber que nenhum dos algoritmos executa de maneira infinita quando está em uma situação na qual é impossível atingir o seu objetivo final. Também foi feita uma brincadeira na qual se remodelou o problema inicial de modo que o robô tivesse a mobilidade diagonal antes comentada, um dos resultados pode ser observado na Figura 10, os caminhos encontrados ainda que tenham sido menos custosos, não apresentaram a suavidade desejada, o que nos leva a observação de que uma adição importante ao modelo, visando os resultados desejados, pode ser a inclusão da noção de angulação ao robô.

Uma restrição observada no trabalho está na falta de capacidade para analisar a completude dos algoritmos, uma vez que, dada as situações do *grid* (um mapa sempre finito em suas dimensões) todos os algoritmos finalizariam sua execução em algum momento, alcançando ou não o objetivo final. Ou seja, não foi possível expor os mesmos a situações nas quais eles não terminariam, como por exemplo uma representação na qual a árvore do mapa fosse infinita.

V. Conclusões

De modo geral, o trabalho em questão apresentou resultados satisfatórios, especialmente quando levado em consideração que o objetivo deste era fornecer ao aluno conhecimentos mais aprofundados a respeito dos métodos de busca, sendo, ao fim deste, capaz de entender e analisar os algoritmos implementados de maneira não simplista e, principalmente, criar no aluno a capacidade crítica para modelar problemas e fundamentos para analisar a qualidade da modelagem e possíveis pontos de melhoras.

Dos métodos e materiais usados e vistos em detalhes na Seção III, ficam como pontos positivos do trabalho, o uso de bibliotecas externar para lidar com situações secundárias, como cálculo de tempo e *plot* dos mapas. O uso de uma biblioteca de busca externa, o que permitiu concentrar os esforços em entender a modelagem e implementação do problema e também na análise posterior dos resultados da busca. E também, a bateria de testes a qual o projeto foi submetido, pois ela permitiu a geração de dados importantes para serem

[1] b:número médio de sucessores
[2] d:altura máxima da árvore

analisados futuramente, permitindo análises mais profundas dos algoritmos.

Pontos fracos do trabalho, que precisam ser melhorados em iterações futuras, estão em sua maioria relacionados a simplicidade de alguns modelos adotados para a modelagem do problema. Como por exemplo, a maneira como se construiu os obstáculos e a mobilidade do robô. Além disso, outro ponto fraco identificado está mais relacionado a linguagem de programação adotada, no que diz respeito a monitoração do consumo de memória, este contudo, é um preço que vale a pena ser pago.

+————————-+

Referências

[1] J. A. Hendler, A. Tate, and M. Drummond, "Ai planning: Systems and techniques," *AI magazine*, vol. 11, no. 2, pp. 61–61, 1990. 1

[2] L. Cartolano. Implementação das buscas. [Online]. Available: https://gist.github.com/luizcartolano2/e334f48ffd761d36e7febdddd277f9ed 1

[3] ——. Testes com o robo. [Online]. Available: https://gist.github.com/luizcartolano2/dc3095b8f5930d8a2572a8be6b8f4920 1, 4

[4] E. Colombini. Enunciado projeto 1. [Online]. Available: http://www.ic.unicamp.br/~esther/teaching/2019s1/mc906/P1.pdf 1

[5] Biblioteca aima. [Online]. Available: https://github.com/aimacode/aima-python/blob/master/search.ipynb 1

[6] Teoria heurísticas. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html 2

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009. 2

[8] C. Grinstead and J. Laurie Snell, "Introduction to probability / charles m. grinstead, j. laurie snell," *SERBIULA (sistema Librum 2.0)*, 04 2019. 2

[9] Biblioteca time. [Online]. Available: https://docs.python.org/3/library/time.html 3

[10] Biblioteca memory profiler. [Online]. Available: https://pypi.org/project/memory-profiler/ 3

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. 4

Anexos

$$distancia = \sqrt{(x_{goal} - x_{state})^2 + (y_{goal} - y_{state})^2} \quad (1)$$

$$dst\_mtn = abs(x_{goal} - x_{state}) + abs(y_{goal} - y_{state}) \quad (2)$$

$$\overline{x} = \frac{\sum_{i=1}^{n} i}{n} \quad (3)$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}} \quad (4)$$

$$(\overline{x} - 1.96 \cdot \frac{\sigma}{\sqrt{n}}, \overline{x} + 1.96 \cdot \frac{\sigma}{\sqrt{n}}) \quad (5)$$

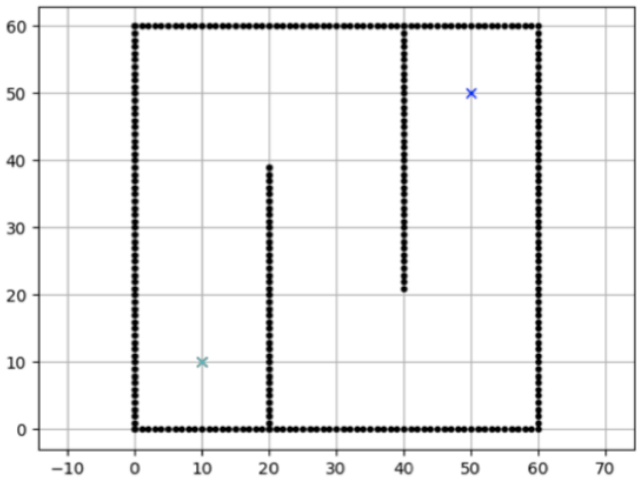

Figura 1. Mapa inicial do problema proposto.

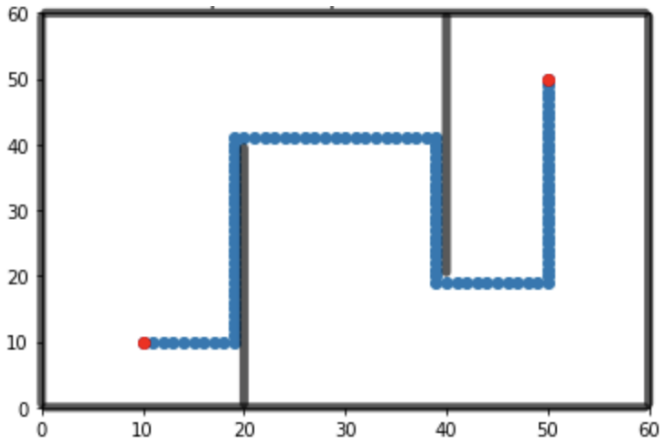

Figura 2. Caminho percorrido usando a busca BFS.

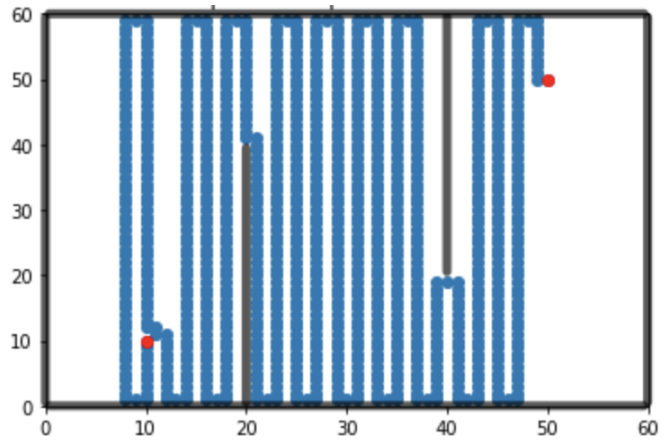

Figura 3. Caminho percorrido usando a busca DFS.

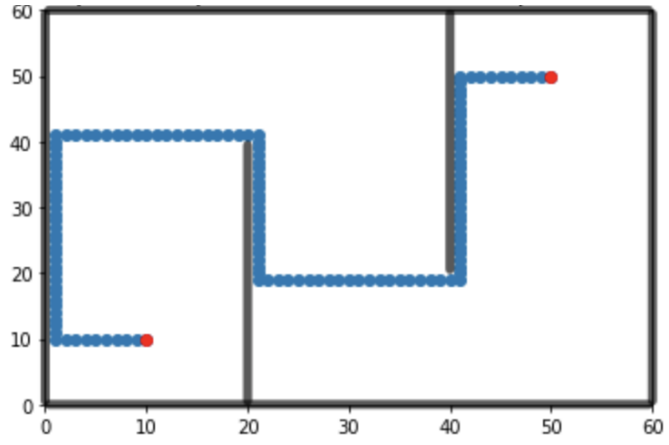

Figura 4. Caminho percorrido usando a busca A* com heurística da distância em linha reta.

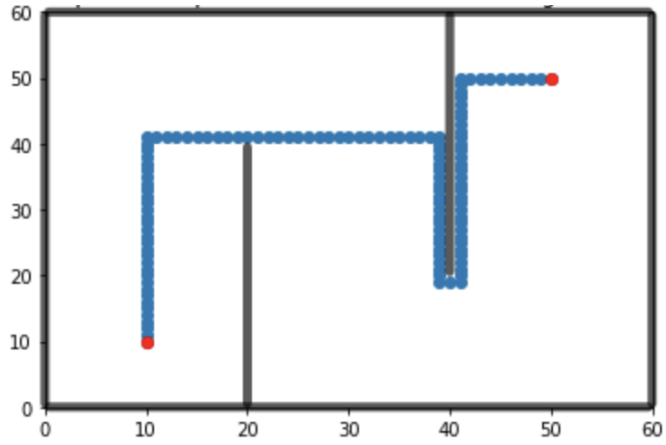

Figura 5. Caminho percorrido usando a busca A* com heurística de Manhattan.

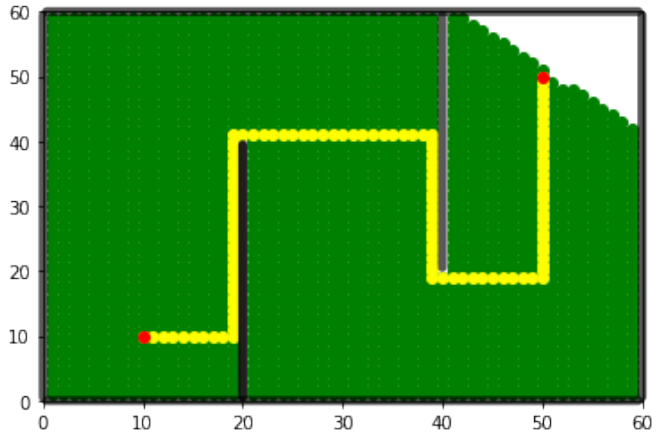

Figura 6. Caminho percorrido usando a busca BFS e marcados os nós visitados pelo algoritmos.

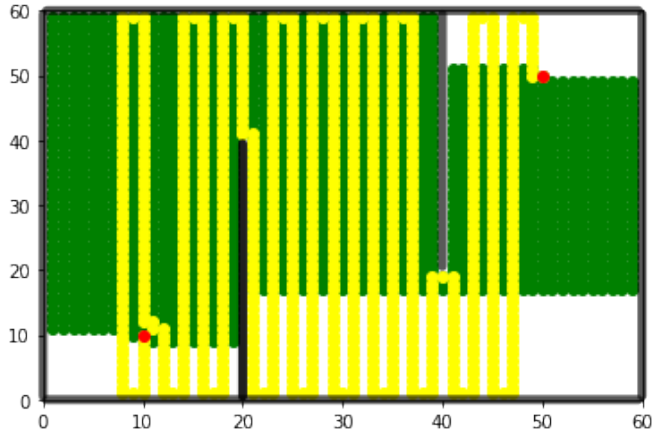

Figura 7. Caminho percorrido usando a busca DFS e marcados os nós visitados pelo algoritmos.

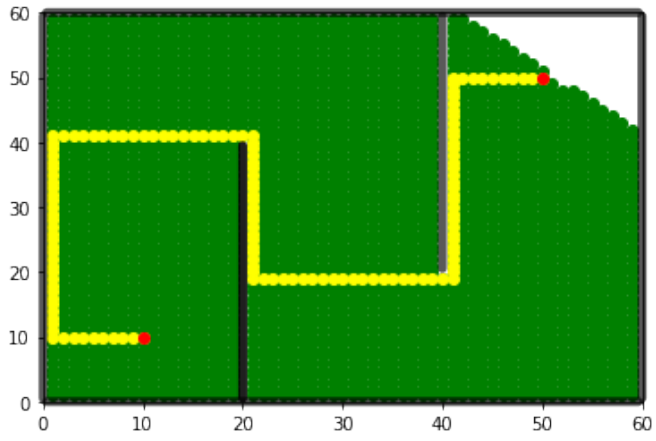

Figura 8. Caminho percorrido usando a busca A* com heurística da distância em linha reta e marcados os nós visitados pelo algoritmos.

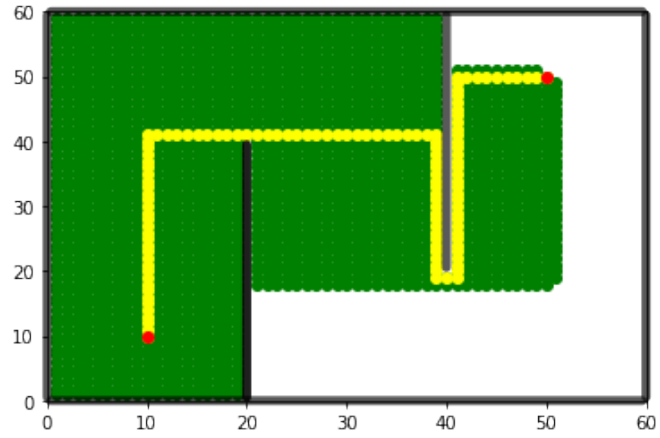

Figura 9. Caminho percorrido usando a busca A* com heurística de Manhattan e marcados os nós visitados pelo algoritmos.

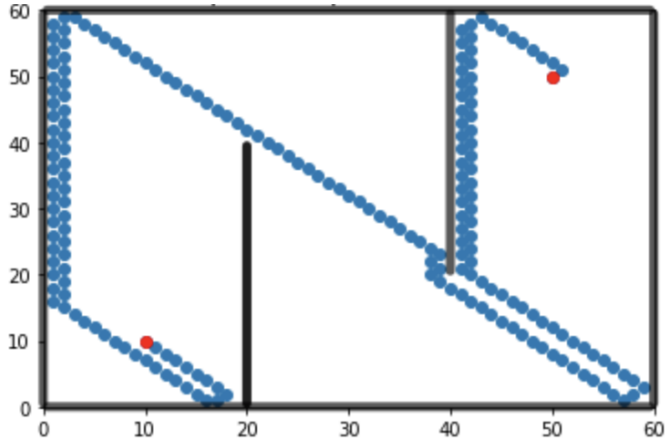

Figura 10. Caminho percorrido usando a busca DFS com mobilidade diagonal.

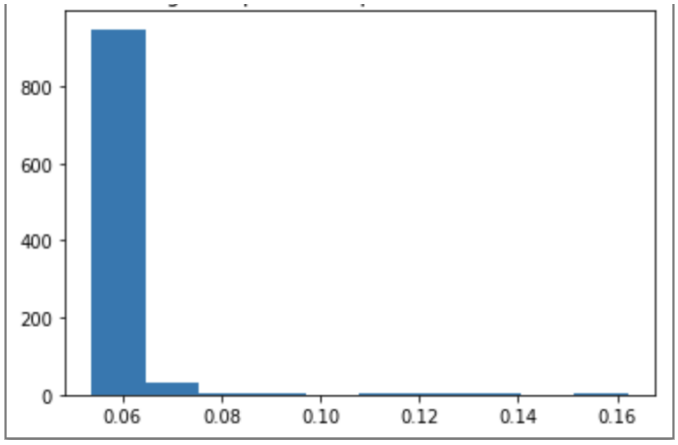

Figura 11. Histograma dos tempos calculados para a busca BFS.

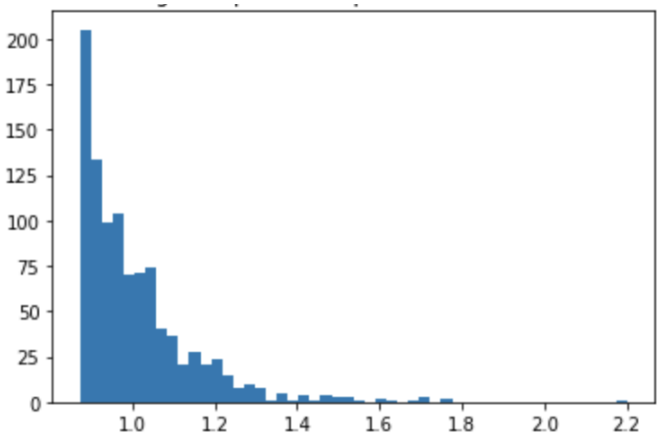

Figura 12. Histograma dos tempos calculados para a busca DFS.

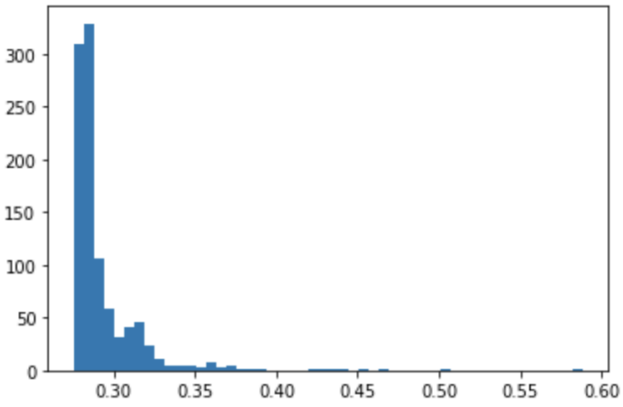

Figura 13. Histograma dos tempos calculados para a busca A* com heurística da distância em linha reta.

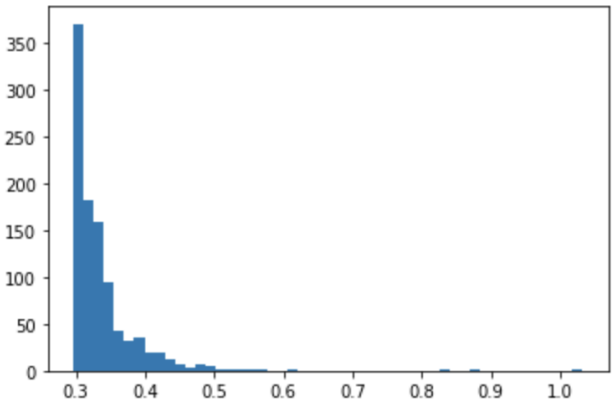

Figura 14. Histograma dos tempos calculados para a busca A* com heurística de Manhattan.

Algoritmo	Gasto de Memória (MB)
BFS	32.92
DFS	32.63
A* - Heurística Euclidiana	98.26
A* - Heurística Manhattan	98.36

Tabela I
MEMÓRIA GASTA POR CADA ALGORITMO.

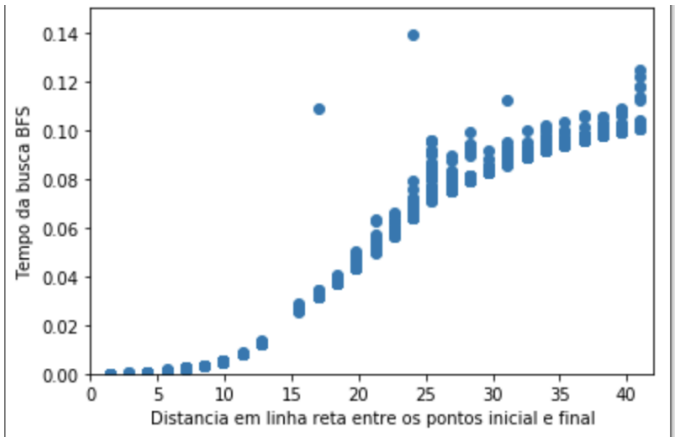

Figura 15. Projeção entre distância em linha reta e o tempo gasto para a busca BFS.

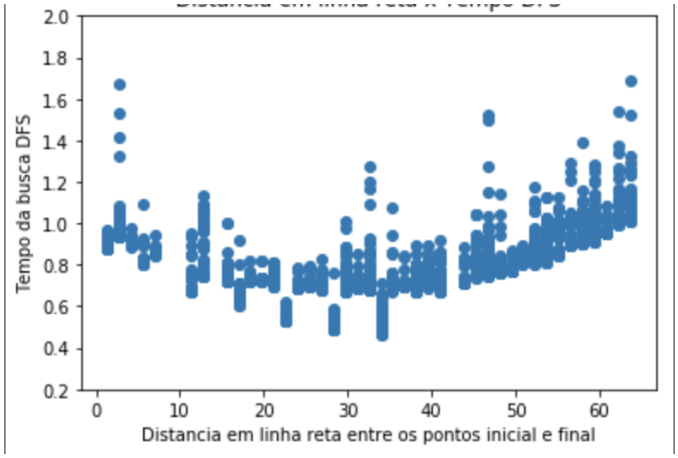

Figura 16. Projeção entre distância em linha reta e o tempo gasto para a busca DFS.

Figura 17. Projeção entre distância em linha reta e o tempo gasto para a busca A* com heurística da distância em linha reta.

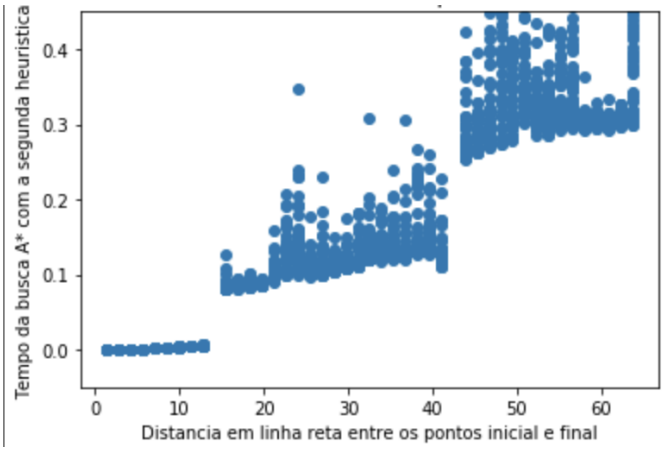

Figura 18. Projeção entre distância em linha reta e o tempo gasto para a busca A* com heurística de Manhattan.

Algoritmo	Custo do Caminho
BFS	124
DFS	1084
A* - Heurística Euclidiana	142
A* - Heurística Manhattan	124

Tabela II
TABELA COM OS CUSTOS DOS CAMINHOS DE CADA ALGORITMO.

Algoritmo	Média	Desvio	Intervalo de Confiança
BFS	0.057	0.009	(0.056,0.058)
DFS	1.010	0.144	(1.001,1.018)
A* - Heurística Euclidiana	0.293	0.025	(0.291,0.294)
A* - Heurística Manhattan	0.336	0.055	(0.332,0.339)

Tabela III
TABELA COM OS TEMPOS MÉDIOS DE EXECUÇÃO DE CADA ALGORITMO.