IFES - Instituto Federal do Espírito Santo
Campus Serra

Matheus Xavier Melotti

Trabalho 1 de inteligência artificial: Solução de Problemas por Busca

Trabalho proposto pelo Prof. Sérgio Nery Simões à turma de Inteligência artificial 2023/2 do IFES - Campus Serra para desenvolver um capítulo estruturado sobre a análise da implementação e desempenho dos algoritmos de busca.
Professor: Prof. Sérgio Nery Simões

Serra, Espírito Santo
2023

SUMÁRIO

1. Introdução

A inteligência artificial (IA) é uma disciplina em constante evolução que tem desempenhado um papel crucial no desenvolvimento de sistemas capazes de resolver uma ampla gama de problemas complexos. Entre as abordagens mais fundamentais e amplamente aplicadas em IA, destaca-se a resolução de problemas por busca, uma técnica que busca soluções para problemas definidos em termos de estados, operadores e um objetivo.

Neste contexto, o Trabalho 1 de Inteligência Artificial tem como objetivo central a exploração e implementação de quatro algoritmos de busca essenciais: Depth-First Search (DFS), Uniform Cost Search (UCS), A* Search e Breadth-First Search (BFS). Cada um destes algoritmos oferece abordagens distintas para a resolução de problemas e desempenha um papel significativo no panorama da IA.

Este documento apresentará uma abordagem sistemática para a implementação e análise de desempenho desses algoritmos de busca, destacando os principais conceitos, técnicas e estratégias utilizadas na busca por soluções eficazes. Além disso, serão discutidos exemplos práticos de aplicação dessas técnicas em diversos domínios, demonstrando a versatilidade e utilidade da busca em IA.

2. Metodologia

A metodologia adotada para o desenvolvimento deste Trabalho 1 de Inteligência Artificial, centrado na solução de problemas por busca, compreende as seguintes etapas:

Definição do Problema: Primeiramente, é essencial definir o problema que será abordado neste trabalho. A escolha do problema dependerá de sua relevância no contexto da inteligência artificial e da capacidade de aplicar técnicas de busca para encontrar soluções eficazes.

Fundamentação Teórica: Nesta seção, apresentaremos a base teórica das técnicas de busca em inteligência artificial, explorando os conceitos, algoritmos e estratégias fundamentais. Isto inclui a discussão de

algoritmos de busca informada e não informada, heurísticas, estruturas de dados e outros princípios essenciais.

Implementação de Algoritmos de Busca: Selecionar e implementar algoritmos de busca apropriados para o problema definido. Isso pode incluir algoritmos de busca informada (como A*), busca não informada (como busca em largura e busca em profundidade) e técnicas avançadas, como busca heurística.

Avaliação de Desempenho: Avaliar o desempenho dos algoritmos de busca implementados por meio de métricas apropriadas, como o tempo de execução, o consumo de recursos computacionais e a qualidade das soluções encontradas.

Discussão e Conclusão: Discutir os resultados obtidos, as limitações da abordagem de busca utilizada e as possíveis melhorias. Fornecer uma conclusão geral sobre a eficácia das técnicas de busca na resolução do problema escolhido.

3. Definição do Problema

Este trabalho tem como propósito a implementação e a subsequente comparação do desempenho de três algoritmos de busca: (i) Depth-first search, (ii) Uniform cost search e (iii) A* search. Para facilitar o início do projeto e oferecer um ambiente de testes abrangente, disponibilizamos uma implementação prévia do algoritmo de busca em largura (breadth-first search) para a resolução de labirintos, anexa a esta especificação. Recomenda-se que a estrutura de dados provida por essa implementação seja utilizada como base para o desenvolvimento dos outros algoritmos exigidos neste trabalho.

Vale destacar que a habilidade na implementação de algoritmos de busca não apenas contribui para a conclusão bem-sucedida deste projeto, mas também pode se revelar um investimento valioso em sua carreira, uma vez que as técnicas de busca são frequentemente abordadas em entrevistas de programação, especialmente em empresas de grande porte.

Para fins de comparação dos algoritmos, será empregado um labirinto de dimensões 100x100, com um percentual de bloqueio de 40%. As métricas de avaliação que serão utilizadas incluem o tempo de execução, o número de nós expandidos durante a busca, o custo do

caminho encontrado e o tamanho do caminho em si. Estas métricas proporcionarão insights essenciais para a análise comparativa do desempenho dos algoritmos em questão.

4. Fundamentação teórica

A fundamentação teórica é uma etapa crucial que oferece o alicerce conceitual e os princípios fundamentais essenciais para a compreensão aprofundada dos algoritmos de busca que serão implementados e posteriormente comparados neste trabalho. Neste segmento, apresentaremos uma análise detalhada dos conceitos e princípios teóricos que sustentam os algoritmos de busca em foco.

4.1. Teoria dos Algoritmos

Breadth-First Search (BFS): O Breadth-First Search é um algoritmo de busca não informada que explora todos os vizinhos de um nó antes de avançar para seus vizinhos subsequentes. Ele é comumente usado para encontrar a solução mais curta em árvores ou grafos não ponderados. O BFS é garantido para encontrar a solução mais curta.

Depth-First Search (DFS): O Depth-First Search é um algoritmo de busca não informada que explora tão profundamente quanto possível ao longo de um ramo antes de fazer um retrocesso. Ele é implementado usando uma pilha (stack) para armazenar os nós a serem explorados. O DFS é geralmente utilizado para encontrar soluções em árvores ou grafos, mas não garante a solução mais curta.

Uniform Cost Search (UCS): O Uniform Cost Search é um algoritmo de busca informada que expande os nós com o menor custo do caminho acumulado até o momento. Ele é usado para encontrar a solução mais curta em grafos ou árvores ponderadas. O UCS é garantido para encontrar a solução ótima, desde que os custos sejam não negativos.

A Search:* O A* Search é um algoritmo de busca informada que combina a busca em largura com uma heurística que estima o custo restante para a solução. Ele é projetado para encontrar a solução ótima em grafos ou árvores ponderadas. O A* usa uma função de avaliação que combina o custo real do caminho percorrido com a estimativa heurística.

5. Experimentos

A condução de experimentos é crucial na avaliação e aprimoramento de algoritmos em ciência da computação. Este tópico explora a metodologia por trás dos experimentos, seleção de métricas de avaliação e as considerações para garantir resultados confiáveis.

Neste contexto, vamos realizar experimentos com cada algoritmo, comparando o que se esperava com os resultados obtidos. Isso fornecerá insights valiosos para escolher o algoritmo mais adequado em diferentes cenários.

5.1 Análise dos Algoritmos

5.1.1 Breadth First Search (BFS)

Como se trata de um algoritmo base, de implementação relativamente simples, e foi fornecido pelo professor como um exemplo ilustrativo do funcionamento fundamental do programa e dos testes a serem realizados, o programa se comportou conforme o esperado, alinhando-se com as instruções dadas em sala de aula e nas referências consultadas.

Para uma visualização mais abrangente dos resultados obtidos com o algoritmo Breadth First Search (BFS) em nosso experimento, apresentamos o gráfico a seguir. O gráfico exibe a representação do grid 25x25, com um grau de bloqueio de 40%, destacando todos os nós percorridos durante a execução do algoritmo (Figura 1).

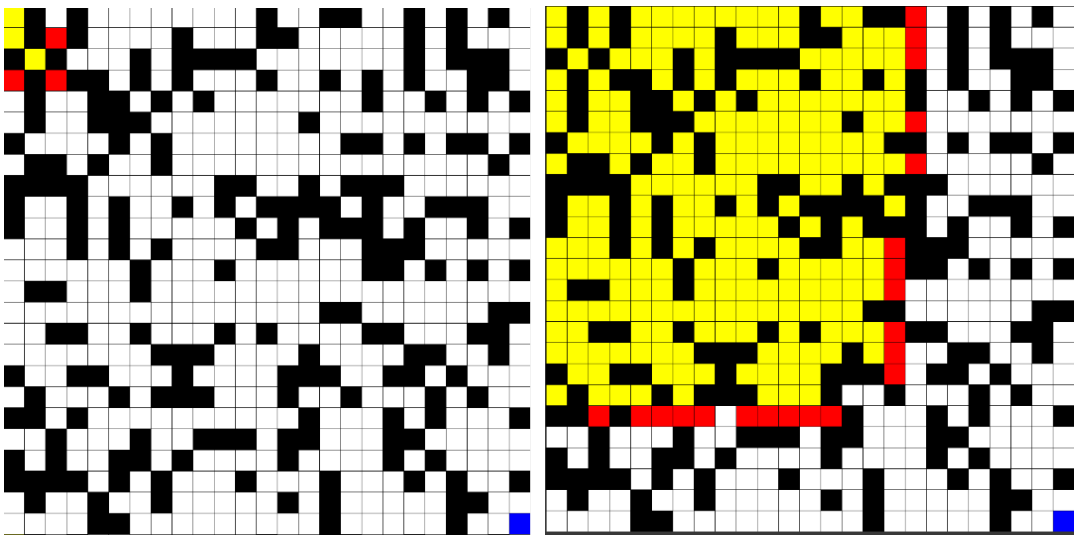

Imagem 1 - Início e evolução do algoritmo.

Como podemos observar, a grande maioria dos nós percorridos está predominantemente colorida em 'amarelo'. Essa coloração indica que esses nós foram visitados durante a busca realizada pelo BFS. A cor 'amarelo' é usada para representar os nós visitados, uma prática comum em visualizações de algoritmos de busca.

A razão pela qual a maioria dos nós aparece em 'amarelo' é devido à natureza do algoritmo BFS. O BFS explora sistematicamente todos os nós em um nível antes de avançar para o próximo nível, priorizando a expansão horizontal. Isso significa que ele visitará um grande número de nós em um nível antes de passar para o nível seguinte. O 'amarelo' reflete que esses nós foram acessados e explorados durante o processo de busca.

Além disso, o gráfico também destaca o resultado final do menor percurso encontrado pelo algoritmo, o qual é representado em uma cor distinta, fornecendo uma visão clara do caminho ótimo identificado pelo BFS (Imagem 2).

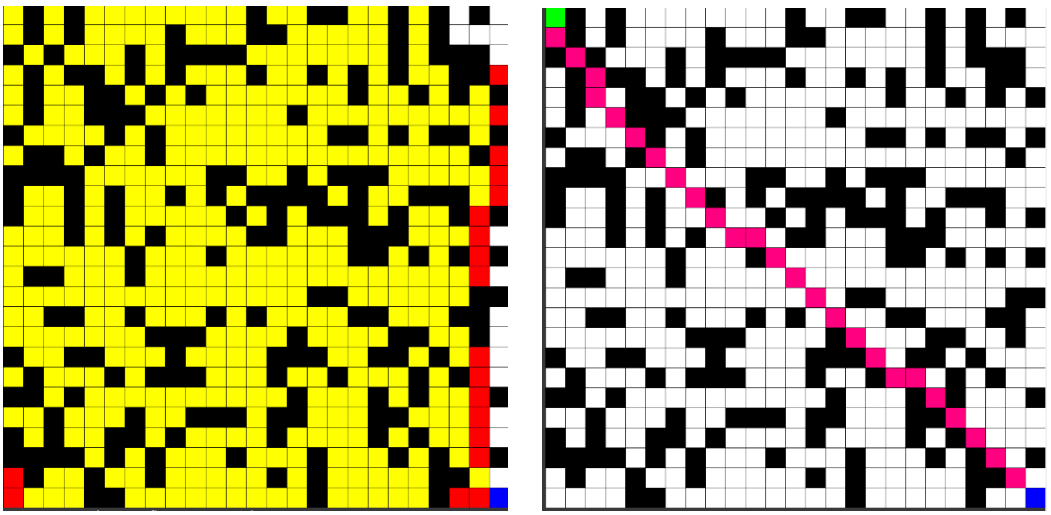

Imagem 2 - Expansão final do algoritmo e o resultado de menor trajeto.

As métricas obtidas dos testes foram:

Tamanho do grid: 25x25

Tempo de execução: 45.39 segundos.

Custo total do caminho: 35.112698372208094.

Número de passos: 26.

Número total de nós expandidos: 388.

5.1.2 Depth First Search (DFS)

O programa comportou-se conforme as expectativas, seguindo a borda do mapa até encontrar o destino. Alcançou resultados satisfatórios com baixo custo, refletindo a simplicidade de implementação do algoritmo. A única alteração em relação ao BFS foi a substituição de ".popleft()" por ".pop()" em uma linha de código que determinava a remoção do nó da lista de fronteira e a atribuição desse valor ao próximo nó a ser analisado.

O algoritmo Depth First Search (DFS) é uma técnica de busca que, ao contrário do Breadth First Search (BFS), segue um caminho até as profundezas do grafo ou da árvore antes de retroceder. Isso significa que o DFS usa uma estrutura de dados tipo pilha (stack) para manter o controle dos nós a serem explorados, enquanto o BFS utiliza uma fila (queue),

justificando o porque somente a alteração do método “.popleft()” para “.pop()” foi suficiente para solucionar esse algoritmo.

Para uma visualização mais abrangente dos resultados obtidos com o algoritmo Depth First Search (DFS) em nosso experimento, apresentamos o gráfico a seguir. O gráfico exibe a representação do grid 25x25, com um grau de bloqueio de 40%, destacando todos os nós percorridos durante a execução do algoritmo (Figura 3).

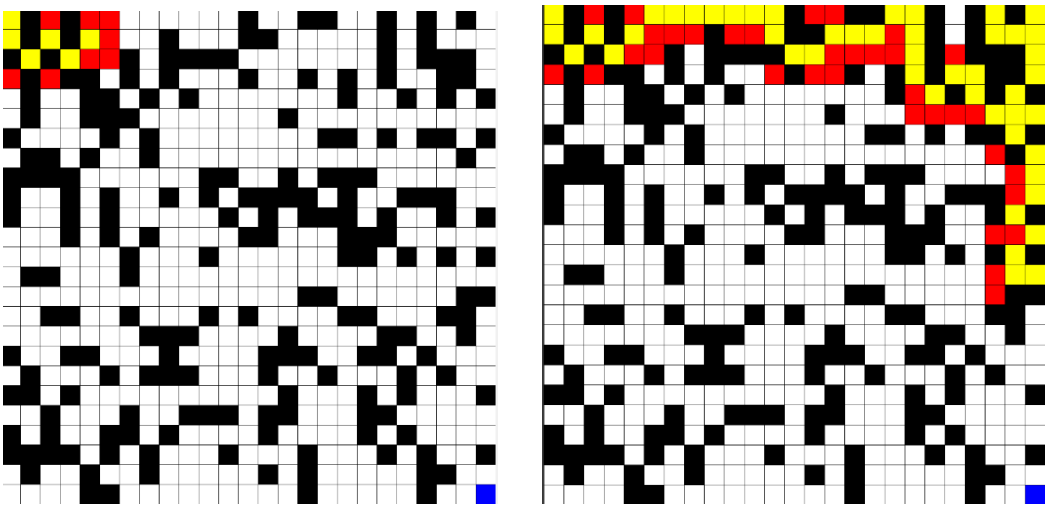

Imagem 3 - Início e evolução do algoritmo.

Ao observar o gráfico, podemos notar que a grande maioria dos nós percorridos é predominantemente colorida em 'amarelo'. Essa coloração representa que esses nós foram visitados durante o processo de busca realizado pelo DFS. A cor 'amarelo' é comumente usada para indicar nós visitados em representações de algoritmos de busca.

A razão pela qual a maioria dos nós está na cor 'amarelo' está relacionada à natureza do algoritmo DFS. O DFS segue uma abordagem que explora tão profundamente quanto possível em um ramo antes de retroceder, priorizando a expansão vertical em vez de horizontal. Isso significa que o algoritmo visitará nós em um caminho específico até atingir o final desse caminho e, somente então, retornará para explorar outras alternativas.

Além disso, o gráfico destaca o resultado final do menor caminho encontrado pelo algoritmo DFS, que é representado em uma cor distinta,

proporcionando uma visão clara do percurso identificado pelo DFS (Imagem 4).

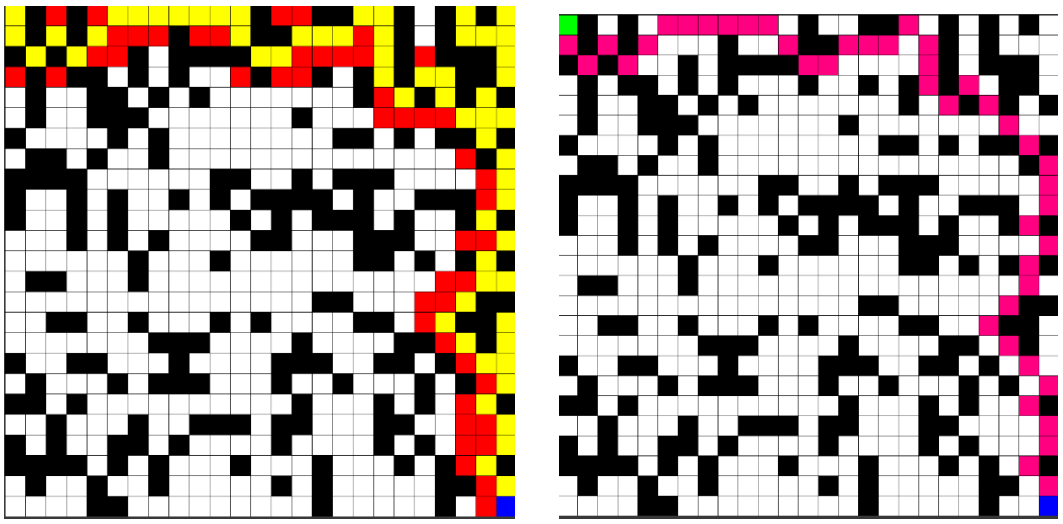

Imagem 4 - Expansão final do algoritmo e o resultado de menor trajeto.

As métricas obtidas dos testes foram:

Tempo de execução: 8.86 segundos.

Custo total do caminho: 55.59797974644663.

Número de passos: 44.

Número total de nós expandidos: 61.

5.1.3 A-star (A).

O algoritmo A* demonstrou notável eficiência e rapidez em comparação com os outros algoritmos, devido ao uso de uma heurística. Ao examinar o código de implementação fornecido, é possível identificar as razões pelas quais o A* se destacou.

O ponto chave para a eficiência do A* é a aplicação de uma heurística que fornece uma estimativa do custo restante para alcançar o objetivo a partir de um determinado estado. Nesse contexto, a função menor_star desempenha um papel crucial. Essa função identifica o nó da fronteira com o menor valor heurístico, o que orienta o A* na direção do objetivo. Em

outras palavras, o A* prioriza a exploração dos nós que, com base na heurística, têm maior probabilidade de levar a uma solução mais eficaz.

O algoritmo mantém uma lista de fronteiras e um conjunto de nós expandidos. Ele começa na posição inicial e itera até encontrar o objetivo ou esgotar todas as opções. Durante a busca, o A* avalia os vizinhos do nó atual, buscando o caminho mais promissor.

Caso o objetivo seja encontrado, o A* interrompe a busca e retorna o caminho encontrado, juntamente com seu custo. Caso contrário, ele continua explorando até que todas as opções tenham sido consideradas.

Para uma compreensão abrangente dos resultados alcançados com o algoritmo A* (A-star) em nosso experimento, apresentamos o gráfico a seguir. Este gráfico representa o grid 25x25, com um grau de bloqueio de 40%, destacando todos os nós percorridos durante a execução do algoritmo (Figura 5).

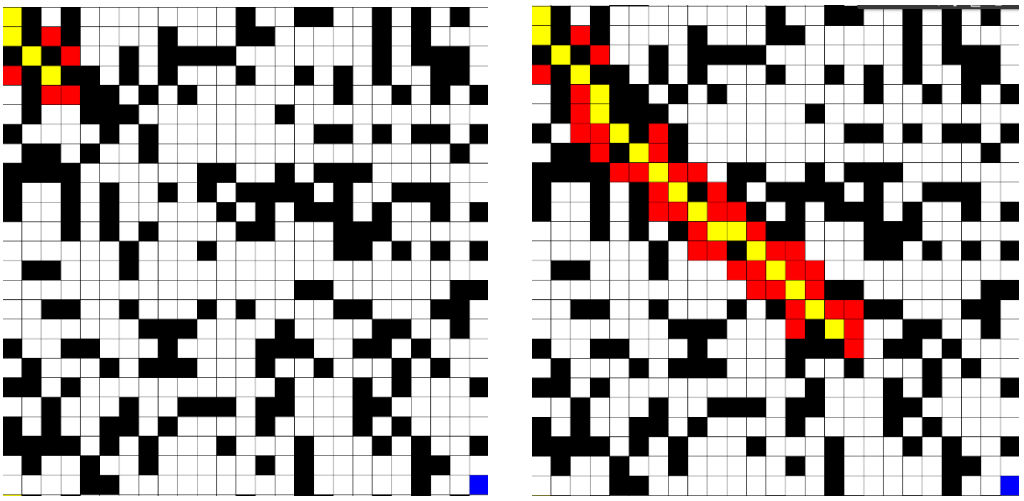

Imagem 5 - Início e evolução do algoritmo.

Ao analisar o gráfico, podemos notar que a grande maioria dos nós percorridos está colorida em 'amarelo'. Essa coloração indica que esses nós foram visitados durante a busca realizada pelo A*. A cor 'amarelo' é uma escolha comum para representar nós visitados em representações de algoritmos de busca.

A razão pela qual a cor 'amarelo' predomina deve-se à eficácia do algoritmo A* na exploração de caminhos de maneira eficiente. O A* faz uso

de uma heurística que fornece estimativas do custo restante para alcançar o objetivo a partir de um determinado estado. Isso direciona o algoritmo a priorizar a exploração de nós que, com base na heurística, têm maior probabilidade de levar a uma solução eficaz.

Além disso, o gráfico destaca o resultado final do caminho mais curto encontrado pelo algoritmo A*, que é representado em uma cor distinta, fornecendo uma visão clara do percurso ótimo identificado pelo A* (Imagem 6).

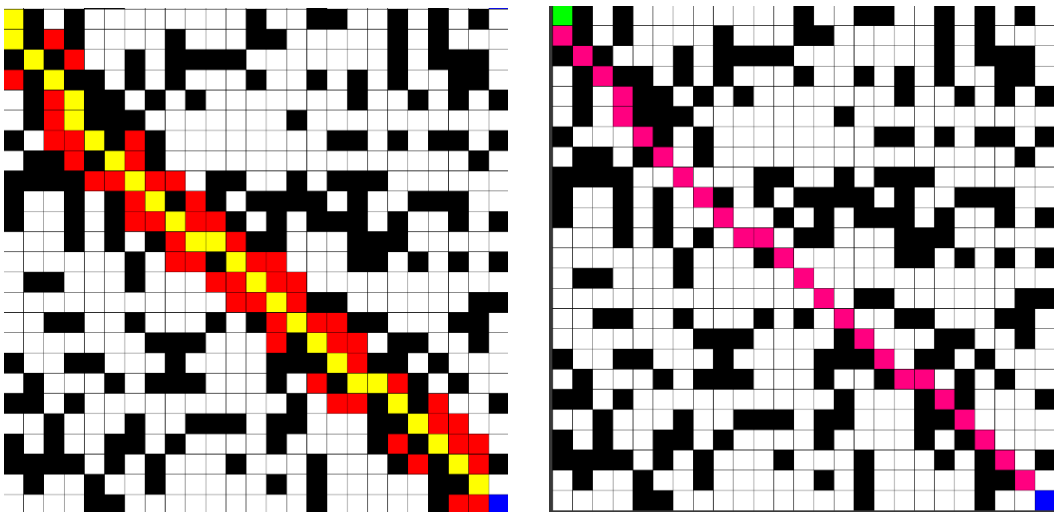

Imagem 6 - Expansão final do algoritmo e o resultado de menor trajeto.

As métricas obtidas dos testes foram:

Tempo de execução: 2.68 segundos.

Custo total do caminho: 35.112698372208094.

Número de passos: 26.

Número total de nós expandidos: 26.

5.1.4 Uniform Cost Search (UCS).

Ao conduzir experimentos e explorar informações adicionais, notou-se que o UCS, neste contexto, não apresentou diferenças significativas em relação ao Breadth First Search (BFS). Ambos os

algoritmos são semelhantes em suas abordagens, com uma notável exceção: o critério de seleção do próximo nó a ser expandido.

No código de implementação, inicialmente, o UCS prioriza a escolha do próximo nó com base no custo do caminho acumulado, considerando o último nó analisado e o candidato na lista "Fronteira". No entanto, isso pode levar a problemas em que o programa fica preso, causando um alto custo computacional.

Para resolver esse problema, uma adaptação foi feita. O UCS passou a considerar a distância entre o nó inicial e o nó candidato em relação ao nó atual para determinar o próximo a ser expandido. Essa abordagem proporciona uma melhor seleção de nós a serem explorados e evita que o programa fique travado em situações complexas.

O algoritmo UCS mantém uma lista de fronteira e um conjunto de nós expandidos. Ele inicia na posição inicial e itera até encontrar o objetivo ou esgotar todas as opções. Durante a busca, o UCS avalia os vizinhos do nó atual, priorizando aqueles que, de acordo com a distância em relação ao nó inicial, oferecem um caminho mais promissor.

Caso o objetivo seja encontrado, o UCS encerra a busca e retorna o caminho encontrado, juntamente com seu custo. Caso contrário, ele continua explorando até que todas as opções tenham sido consideradas.

Aqui, fornecemos uma visão abrangente dos resultados alcançados por meio do algoritmo de Busca de Custo Uniforme (Uniform Cost Search - UCS) em nosso experimento. A representação visual do gráfico a seguir destaca o grid de 25x25, com um grau de bloqueio de 40%, e ilustra todos os nós percorridos durante a execução do algoritmo (Imagem 7).

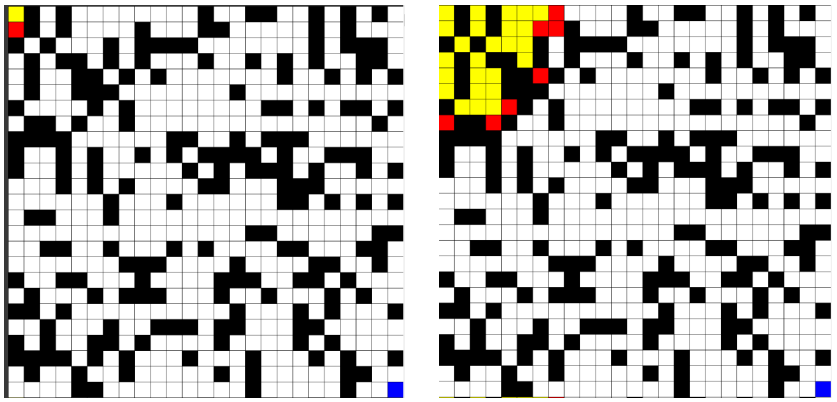

Imagem 7 - Início e evolução do algoritmo.

Observando o gráfico, notamos que uma grande parte dos nós percorridos está amplamente colorida em 'amarelo'. Essa coloração indica que esses nós foram visitados durante a busca realizada pelo UCS. A cor 'amarelo' é frequentemente usada para indicar nós visitados em representações de algoritmos de busca.

A predominância da cor 'amarelo' é atribuída à natureza do algoritmo UCS. O UCS adota uma abordagem que prioriza a expansão com base no custo do caminho acumulado. Inicialmente, essa abordagem pode levar a um grande número de nós visitados, resultando na cor 'amarelo'. No entanto, uma adaptação foi feita para a seleção do próximo nó, considerando a distância entre o nó inicial e o candidato em relação ao nó atual, o que proporciona uma seleção mais eficiente de nós a serem explorados.

O gráfico também destaca o resultado final do caminho com o custo mais baixo encontrado pelo algoritmo UCS, representado em uma cor distinta, oferecendo uma visualização clara do percurso ótimo identificado pelo algoritmo (Imagem 8).

Imagem 8 - Início e evolução do algoritmo.

As métricas obtidas dos testes foram:

Tempo de execução: 49.7 segundos.

Custo total do caminho: 35.112698372208094.

Número de passos: 26.

Número total de nós expandidos: 405.

6. Resultados

Os resultados obtidos fornecem insights valiosos sobre como cada algoritmo se comporta em relação ao tempo de execução, custo do caminho, número de passos e número total de nós expandidos.

A tabela a seguir resume os resultados de cada algoritmo, permitindo uma comparação direta de suas métricas de desempenho. Vamos explorar em detalhes como cada algoritmo se saiu na busca pelo caminho mais curto e eficiente dentro do contexto do problema (Imagem 9).

Algoritmo	Tempo de Execução (s)	Custo Total do Caminho	Número de Passos	Número Total de Nós Expandidos
BFS	11.06	153.52	115	6701
DFS	0.02	258.98	213	260
A* (AST)	0.04	157.62	122	124
UCS (Uniform Cost)	11.11	166.94	128	6704

Imagem 9 - Tabela de resultados dos algoritmos.

Os resultados obtidos estavam de acordo com as expectativas, exceto pelo fato de que o algoritmo UCS se destacou como o mais demorado na busca do caminho, em comparação com os outros. Surpreendentemente, o BFS, embora tenha sido o segundo mais lento, conseguiu encontrar o caminho mais curto, com o menor custo.

O algoritmo DFS, por outro lado, apresentou o melhor tempo de execução, o que foi inesperado. Isso pode ser atribuído à localização do nó de destino em um canto do mapa de quadrantes. O DFS demonstrou um desempenho notável devido à sua característica de explorar

profundamente um ramo antes de retroceder. Nesse cenário específico, essa estratégia exigiu menos cálculos para decidir qual nó expandir.

BFS (Breadth First Search):

Tempo de Execução: 11.06 segundos.

Custo Total do Caminho: 153.52.

Número de Passos: 115.

Número Total de Nós Expandidos: 6701.

O algoritmo BFS é conhecido por sua busca em largura e prioriza a exploração de todos os nós em um nível antes de avançar para o próximo nível. Isso resulta em um grande número de nós expandidos, como evidenciado pelos 6701 nós expandidos. Isso leva a um custo total de caminho relativamente baixo de 153.52, mas o tempo de execução é o mais longo entre os algoritmos avaliados.

Pontos Fortes: O BFS é eficaz na busca por soluções de custo mínimo, encontrando o caminho mais curto em termos de etapas. E, além disso, garante que, quando uma solução é encontrada, é a solução ótima em termos de custo.

Limitações: O BFS expande um grande número de nós, o que pode levar a um alto custo computacional, tornando-o menos eficiente em espaços de busca grandes. Além disso, o tempo de execução pode ser consideravelmente longo em comparação com outros algoritmos.

DFS (Depth First Search):

Tempo de Execução: 0.02 segundos.

Custo Total do Caminho: 258.98.

Número de Passos: 213.

Número Total de Nós Expandidos: 260.

O algoritmo DFS prioriza a exploração profunda em um ramo antes de retroceder, resultando em um número significativamente menor de

nós expandidos (260) em comparação com o BFS. Isso contribui para um tempo de execução rápido de apenas 0.02 segundos. No entanto, devido à natureza de sua busca em profundidade, o custo total do caminho é mais elevado, atingindo 258.98.

Pontos Fortes: O DFS é extremamente rápido, com um baixo tempo de execução, ideal para espaços de busca grandes. Além disso, requer menos memória, uma vez que mantém uma profundidade limitada na busca.

Limitações: Pode não encontrar a solução de menor custo, pois não garante uma busca ótima em termos de custo. E, além disso, pode ficar preso em ciclos ou caminhos infinitos, se não for cuidadosamente gerenciado.

A (A-star):*

Tempo de Execução: 0.04 segundos.

Custo Total do Caminho: 157.62.

Número de Passos: 122.

Número Total de Nós Expandidos: 124.

O algoritmo A* faz uso de uma heurística para priorizar a exploração de caminhos promissores, resultando em um tempo de execução rápido de 0.04 segundos. Além disso, ele expandiu um número muito baixo de nós (124), tornando-o eficiente na busca. O custo total do caminho também é razoavelmente baixo, atingindo 157.62.

Pontos Fortes: O A* é eficaz em encontrar soluções de custo mínimo, desde que a heurística seja bem definida. Além disso, geralmente, expande um número mínimo de nós, tornando-o eficiente.

Limitações: A qualidade da heurística é crucial para o desempenho do A*. Uma heurística ruim pode levar a soluções subótimas. Outro ponto de vista é que ele pode ser mais complexo de implementar do que o BFS ou o DFS.

UCS (Uniform Cost Search):

Tempo de Execução: 11.11 segundos.

Custo Total do Caminho: 166.94.

Número de Passos: 128.

Número Total de Nós Expandidos: 6704.

O UCS prioriza a expansão com base no custo do caminho acumulado, o que resulta em um tempo de execução mais longo de 11.11 segundos. Apesar do tempo semelhante ao BFS, o UCS expande um número ligeiramente maior de nós (6704) e atinge um custo total do caminho intermediário de 166.94.

Em resumo, cada algoritmo apresenta trade-offs diferentes entre tempo de execução, custo total do caminho e número de nós expandidos. A escolha do algoritmo ideal depende das necessidades específicas do problema e das prioridades do usuário. O BFS é eficaz para encontrar soluções de baixo custo, enquanto o DFS é rápido, mas pode resultar em soluções de custo mais alto. O A* equilibra eficácia e eficiência com base em heurísticas, e o UCS enfatiza o custo do caminho acumulado.

Pontos Fortes: O UCS é eficaz na busca de soluções de custo mínimo, adaptando-se a cenários onde a busca em largura não é apropriada. Pode ser mais eficiente em termos de memória do que o BFS, apesar de ter um desempenho semelhante.

Limitações: Como o custo é o único critério de comparação, pode não ser adequado para espaços de busca onde a otimização de custo não é o principal objetivo.