Análise Comparativa de Algoritmos de Busca na Solução de um Problema de Caminho

Lucas P. Ferreira[1]

[1]Centro de Ciências Sociais e Tecnológicas – Universidade Católica de Pelotas (UCPel)
R. Gonçalves Chaves, 373 – 96015-560 – Pelotas – RS – Brasil

lucas.ferreira@sou.ucpel.edu.br

Abstract. *A labyrinth is an intricate combination of passages or corridors, from which it is difficult to find a middle or exit path. From a computational point of view, solving this problem requires a search algorithm. Such algorithms are used when something wants to leave an origin and reach a destination efficiently. However, the use of inadequate algorithms for the problem can affect both execution time and memory consumption. In this perspective, the following analysis aims to compare the Breadth-first Search (BFS) and A* algorithms applied in an shortest path problem with different complexities. The process of comparing these algorithms was conducted by applying each algorithm to the same maze and measuring the execution time and the number of visited and stored nodes. The result of this analysis shows, therefore, which algorithm is the most suitable to be applied in the problem.*

Resumo. *Um labirinto é uma combinação intrincada de passagens ou corredores, da qual é difícil encontrar um meio ou caminho de saída. Do ponto de vista computacional, a resolução deste problema requer um algoritmo de busca. Estes algoritmos são usados quando se quer sair de uma origem e chegar até um destino de maneira eficiente. No entanto, o uso de algoritmos inadequados para o problema podem afetar tanto no tempo de execução quanto no consumo de memória. Nesta perspectiva, a seguinte análise visa comparar os algoritmos de Busca em Largura (BFS) e A* aplicados a solução de um problema de caminho com diferentes complexidades. O processo de comparação destes algoritmos foi conduzido aplicando cada algoritmo no mesmo labirinto e medindo o tempo de execução e a quantidade de nodos visitados e armazenados. O resultado desta análise demonstra, portanto, qual algoritmo é o mais adequado para ser aplicado ao problema.*

1. Introdução

Para muitos problemas do mundo real, a solução consiste em trabalhar o seu caminho através de uma sequência de decisões em que cada uma leva você mais adiante ao longo de um caminho. Este estudo, na computação, é chamado de busca de caminho e tornou-se uma área de pesquisa muito forte ao longo dos anos, principalmente pelo advento da Inteligência Artificial (IA). Nesta perspectiva, um labirinto se encaixa perfeitamente na descrição do problema e é geralmente utilizado como exemplo na aplicação dos mais diversos algoritmos de busca cega.

Labirintos fazem parte da cultura humana há milhares de anos. De acordo com a mitologia grega, o rei da ilha mediterrânea de Creta, Minos, criou um labirinto com a

ajuda de Dédalo para abrigar uma criatura mortal, o minotauro. Diz a lenda que o jovem Teseu, de Atenas, entrou no labirinto com uma espada e uma bola de cordas. Teseu matou o Minotauro e traçou seu caminho de volta desenrolando a corda até chegar ao seu ponto de partida. Assim, um labirinto pode ser associado a uma construção que leva do estado inicial para o estado final, tornando o caminho tortuoso.

Por definição, um labirinto é uma área bidimensional semelhante a uma grade, que pode assumir qualquer tamanho e seu formato é geralmente quadrado. Além disso, ele contém células, ou seja, espaços formalmente delimitados com diferentes obstáculos em diferentes quantidades. Sua complexidade é determinada pelo número de paredes (ou obstáculos), corredores, becos e entre a distância da célula inicial e a final.

A estrutura utilizada neste estudo se dá por um labirinto quadrado (que pode assumir tamanho NxN) aleatório, construído usando a sequência de decisões estocásticas tomadas sobre iterações para a criação do mesmo. Após sua criação, o objetivo da aplicação dos algoritmos é sair do estado inicial (posição x e y de onde inicia o labirinto) e chegar ao seu estado final (posição x e y de onde acaba o labirinto). O custo de caminho sempre será 1 neste caso específico, e as ações possíveis serão direita, esquerda, cima e baixo, se e somente se não tiverem obstáculos no caminho.

Para isso, serão utilizados algoritmos de busca de caminho, necessários para determinar o objetivo final o mais rápido e da melhor maneira possível. Existem diversos algoritmos que exercem essa função. Neste estudo, foi comparado o algoritmo de Busca em Largura (BFS) e o algoritmo A*. Cada um dos algoritmos tem seus próprios pontos fortes e fracos no processo de determinação do caminho mais rápido. A aplicação deles irá encontrar o melhor algoritmo para problema abordado dentre as soluções escolhidas.

O algoritmo recomendado para ser aplicado no problema abordado será considerado a partir dos diversos aspectos dos resultados entre os dois algoritmos comparados, e a partir do teste de objetivo, cuja função é a de testar se a posição atual x e y corresponde a do estado final.

2. Trabalhos Relacionados

Um trabalho conduzido por Aqsa Zafar e outros com o título “Analysis of Multiple Shortest Path Finding Algorithm in Novel Gaming Scenario” [Zafar et al. 2018] discutiu a comparação de 5 algoritmos de busca de caminho. Os cinco algoritmos utilizados para o teste foram Busca em Largura (BFS), Busca em Profundidade (DFS), algoritmo de busca melhor-primeiro, Dijkstra e A*. Apesar de utilizar esses cinco algoritmos de busca, a discussão no estudo apenas compara o algoritmo de Dijkstra e o Algoritmo A*. Na discussão, imagens do jogo Age of Empires e Civilization V são fornecidas, mas não explicadas de maneira clara sobre como foram usadas e nem como foi feita sua aplicação. O que pode ser extraído deste estudo é a atenção ao algoritmo A*, que se mostra melhor que os demais.

O segundo trabalho, intitulado “Pathfinding car racing game using dynamic pathfinding algorithm and algorithm A*” [Sazaki et al. 2017], desenvolvido por Sazaki e outros, utilizam algoritmos NPC para jogar contra um ser humano em um jogo de corrida de carros. O algoritmo de busca utilizado neste trabalho para encontrar o menor caminho foi o A*, combinado com um algoritmo de busca de caminho dinâmico afim de evitar obstáculos dinâmicos ou estáticos. Os resultados experimentais comprovaram que a

combinação de ambos os métodos podem ser implementados de maneira satisfatória em jogos de corrida, com as condições de obstáculos e estradas estáticas. Enquanto o algoritmo agia sobre a pista com obstáculos dinâmicos, a combinação de ambos os métodos foi pouco satisfatória e apenas sob certas condições se mostrou viável.

3. Metodologia

O problema abordado neste estudo é a pesquisa pela rota mais curta e mais eficiente em um labirinto. A metodologia de pesquisa utilizada foram testes comparativos empregando diferentes tabuleiros, com as mesmas posições de obstáculos, para cada um dos algoritmos testados. As variáveis observadas neste teste comparativo foram o tempo de execução, o resultado dos nodos visitados e o número total de nodos armazenados na memória. Ao final da execução de cada algoritmo, pode-se decidir qual algoritmo é mais adequado para ser implementado no problema em questão.

4. Algoritmos

Existem diversos algoritmos desenvolvidos para as mais diversas variantes de um problema. As variantes podem incluir bordas direcionadas versus não direcionadas, por exemplo, e frequentemente fazem uso de grafos. Um grafo é um número de nós e arcos que os conectam, e um grafo rotulado tem uma ou mais assinaturas anexadas a cada nó que distingue o nó de qualquer outro nó no grafo. A pesquisa de grafos é dividida em pesquisa de busca cega e em pesquisa heurística.

Às vezes, a pesquisa de busca às cegas é chamada de pesquisa uniforme, pois não tem conhecimento sobre seu domínio. A única opção que uma pesquisa cega é capaz de fazer é distinguir um estado não objetivo de um estado objetivo. A busca cega não tem preferência sobre qual estado (nó) pode ser expandido a seguir; diferentemente da pesquisa heurística. A pesquisa heurística é o estudo dos algoritmos e regras de descoberta e desenvolvimento. Heurísticas são regras práticas que podem resolver um determinado problema, mas não garantem uma solução.

Neste trabalho, serão aplicados dois algoritmos: Busca em Largura (BFS) e A*. O primeiro é do tipo busca cega e o segundo contém uma heurística.

4.1. Busca em Largura (BFS)

Na teoria de grafos, a Busca em Largura é uma das estratégias utilizadas para realizar uma busca ou travessia em um grafo ou em uma árvore. Como o próprio nome sugere, esta abordagem envolve percorrer a árvore em largura (em vez de em profundidade). O algoritmo começa examinando todos os nós um nível (algumas vezes chamado de uma camada) abaixo do nó raiz. Se um estado objetivo for encontrado aqui, é relatado sucesso. Caso contrário, a busca prossegue pela expansão de caminhos a partir de todos os nós do nível corrente na direção do próximo nível. Desse modo, a busca continua examinando nós em um determinado nível, relatando sucesso quando um nó objetivo for encontrado e relatando falha se todos os nós tiverem sido examinados e um nó objetivo não tiver sido encontrado.

A Busca em Largura é um bom método para ser utilizado em situações nas quais a árvore pode ter caminhos muitos profundos, principalmente se o nó objetivo estiver em uma parte mais rasa da árvore. Infelizmente, ela não funciona tão bem quando o

fator de ramificação da árvore é muito alto, tais como na aplicação em jogos como Go e Xadrez. O algoritmo não é bom também em árvores onde todos os caminhos levam a um nó objetivo com caminhos de comprimentos parecidos. Em situações como esta, a Busca em Profundidade funciona melhor, pois identificaria um nó objetivo quando atingisse o final do primeiro caminho examinado.

Para encontrar a complexidade de tempo e espaço da Busca em Largura, devemos supor que os nossos nós não objetivos tenham *b* sucessores (sendo *b* é o fator de ramificação), e gerar cada sucessor pai. Essa tarefa tem complexidade assintótica constante. Em seguida, a raiz da arvore de busca gera *b* nós e consome *b* tempo. Assim, cada um dos nós *b*, com profundidade 1, gera *b* nós e consome *b* tempo, e assim por diante até que a meta de profundidade *d* seja atingida. A implementação deste algoritmo pode ser vista no apêndice A.

4.2. A*

A busca A* (chamada de A estrela) é um dos algoritmos mais famosos na identificação de caminhos. Ele é um algoritmo de busca genérica e se baseia na análise dos nós através da combinação de *g(nó)*, que é o custo para alcançar o nó, e *h(nó)*, que é o custo para ir do nó ao objetivo. Sendo assim, o algoritmo usa a seguinte notação:

$$f(nó) = g(nó) + h(nó) \tag{1}$$

F(nó) é a chamada função de avaliação baseada em caminho. Ao operar o A*, a função *f(nó)* é avaliada para nós sucessores e caminhos expandidos utilizando os nós que tenham os menores valores de f. Se o custo *h(nó)* for sempre uma subestimativa da distância de um nó a um nó objetivo, então o algoritmo A* será ótimo: é garantido encontrar o caminho mais curto até um estado objetivo.

O A* é descrito como sendo otimamente eficiente, no sentido de que, para encontrar o caminho até o nó objetivo, ele expandirá o mínimo de caminhos possível. Mais uma vez, esta propriedade depende do custo *h(nó)* ser sempre uma subestimativa. Vale observar que, ao executar o algoritmo A*, nem sempre será garantido encontrar a solução mais curta, pois os valores estimados para *h(nó)* não são todos subestimativas. Em outras palavras, a heurística que está sendo utilizada pode não ser admissível. Se for utilizada uma heurística não admissível para *h(nó)*, então o algoritmo será chamado apenas de A.

Sendo assim, podemos afirmar que o algoritmo A* só será útil quando fornecer uma subestimativa de custo verdadeiro até o objetivo, e será ótimo e completo apenas se, ao encontrar uma solução, essa seja garantidamente a melhor solução. Além disso, vale ressaltar que o algoritmo A* é análogo à Busca em Largura. De fato, a busca em largura pode ser considerada como um caso especial do A*, no qual o custo *h(nó)* é sempre 0, logo, *f(nó) = g(nó)*, onde cada caminho direto entre um nó e o seu sucessor imediato tem custo 1. A implementação deste algoritmo pode ser vista no apêndice A.

5. Resultados

Nesta análise comparativa de algoritmos de busca, são implementados os algoritmos de Busca em Largura e A* aplicados a labirintos de diferentes complexidades. Foi utilizado na comparação tabuleiros nos tamanhos 25x25, 125x125 e 525x525. Para cada tamanho,

foram utilizadas diferentes complexidades, sendo as mesmas geradas aleatoriamente divididas em níveis, sendo eles fácil, médio e difícil. Para atingir estes níveis, foram estipulados dos parâmetros que foram utilizados durante a concepção de cada tabuleiro. Além disso, para cada algoritmo foi utilizada uma representação gráfica demonstrando a solução, ou seja, a diferença entre o ponto final e o ponto inicial. O algoritmo BFS, por exemplo, mostra a solução encontrada na cor vermelha. Já o algoritmo A* mostra a solução encontrada na cor verde. Em todos os casos, a cor amarela é exibida para mostrar todos os nodos que foram visitados durante a busca pelos diferentes algoritmos.

Para um labirinto de tamanho 25x25, os resultados obtidos utilizando uma complexidade baixa foram os seguintes:

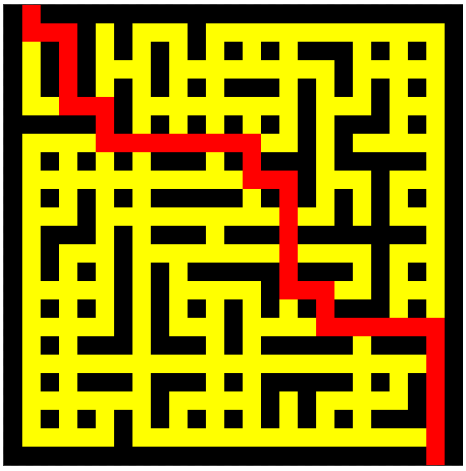

Figura 1: BFS

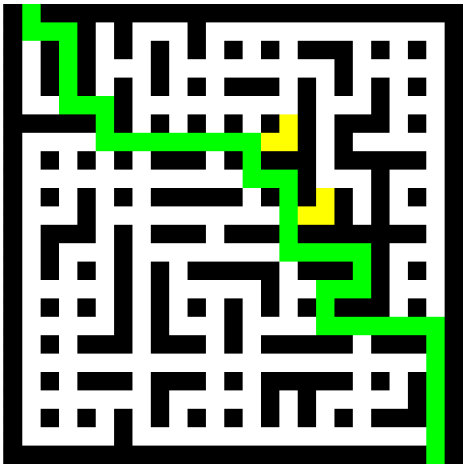

Figura 2: A*

As figuras 1 e 2 mostram os resultados computados. Os obstáculos foram organizados de maneira aleatória de modo a ter sua complexidade baixa e os resultados dos algoritmos aplicados a estes labirintos podem ser vistos na tabela 1:

Componentes	**BFS**	**A***
Nodos Visitados	346	57
Nodos na Memória	17	26
Tempo de Execução (em segundos)	0.03899	0.01100

Tabela 1: Resultados em tabuleiro de tamanho 25x25 com complexidade baixa.

Percebe-se que, para complexidades baixas, o algoritmo A* é o que visita menos nodos. Já para um labirinto de tamanho semelhante utilizando uma complexidade média, os resultados obtidos foram os seguintes:

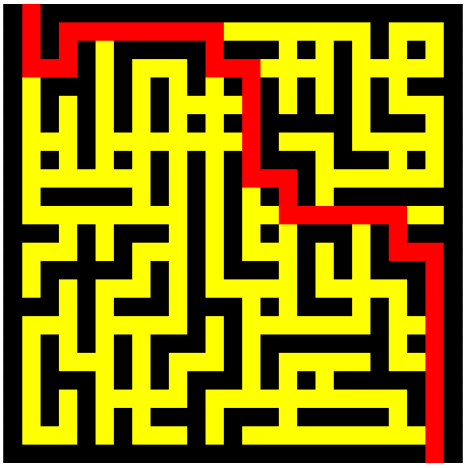

Figura 3: BFS

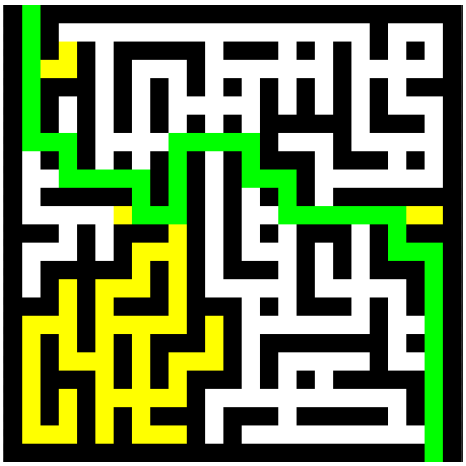

Figura 4: A*

As figuras 3 e 4 mostram os resultados computados e a tabela 2 mostra os resultados da execução destes algoritmos.

Componentes	**BFS**	**A***
Nodos Visitados	317	119
Nodos na Memória	14	21
Tempo de Execução (em segundos)	0.03200	0.01499

Tabela 2: Resultados em tabuleiro de tamanho 25x25 com complexidade média.

Os resultados demonstram que o algoritmo A*, de maneira semelhante a aplicação anterior, visitou menos nodos, sendo o mais vantajoso mesmo em uma complexidade um pouco maior. Para a complexidade mais alta, os resultados foram os seguintes:

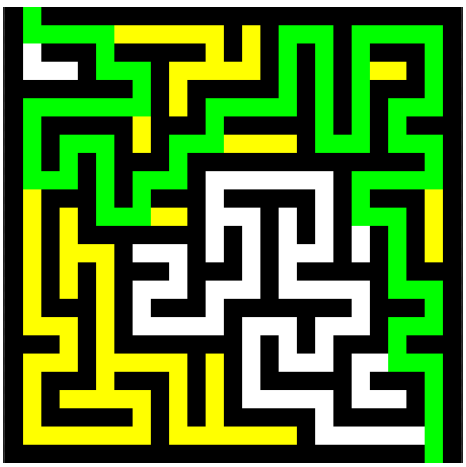

Figura 5: BFS

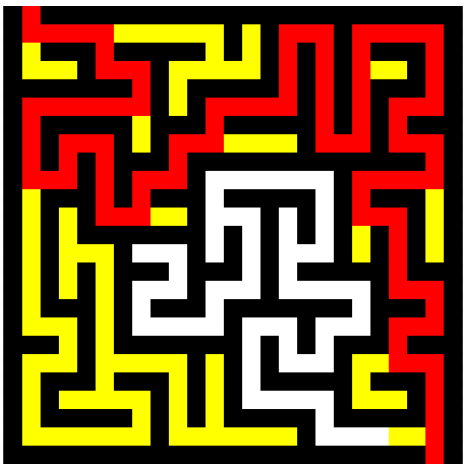

Figura 6: A*

Os resultados computados das figuras 5 e 6 podem ser visualizados na tabela 3.

Componentes	BFS	A*
Nodos Visitados	216	203
Nodos na Memória	4	6
Tempo de Execução (em segundos)	0.02102	0.02400

Tabela 3: Resultados em tabuleiro de tamanho 25x25 com complexidade alta.

Assim como os demais labirintos, o algoritmo A* se mostrou a melhor opção.

Aumentando a complexidade do problema, para um labirinto de tamanho 125x125, os resultados obtidos utilizando uma complexidade baixa foram os seguintes:

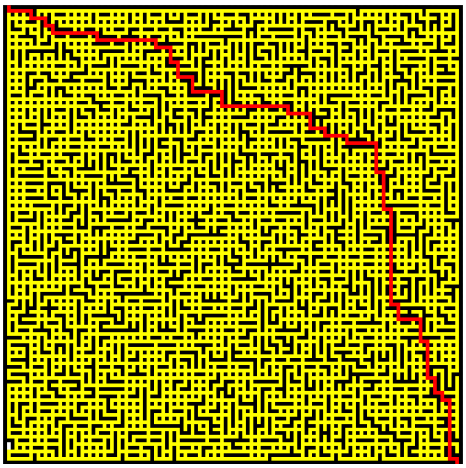

Figura 7: BFS

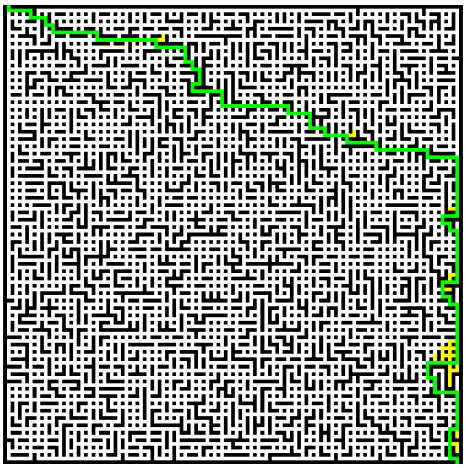

Figura 8: A*

As figuras 7 e 8 mostram os resultados computados. Assim como os demais tabuleiros, os obstáculos foram organizados de maneira aleatória e os resultados desses algoritmos são demonstrados na tabela 4.

Componentes	BFS	A*
Nodos Visitados	9485	332
Nodos na Memória	106	141
Tempo de Execução (em segundos)	0.78199	0.03599

Tabela 4: Resultados em tabuleiro de tamanho 125x125 com complexidade baixa.

Nota-se que o algoritmo A* foi o que visitou menos nodos e teve um maior tempo de execução, se comparado aos demais. Em complexidades mais baixas, o BFS tende a visitar mais nodos, uma vez que as passagens são maiores e a possibilidade de seguir novos caminhos também são. Sendo assim, a chance de chegar ao destino aumenta, assim como os nodos visitados.

Já para um tabuleiro com complexidade média, também do mesmo tamanho, os resultados obtidos foram os seguintes:

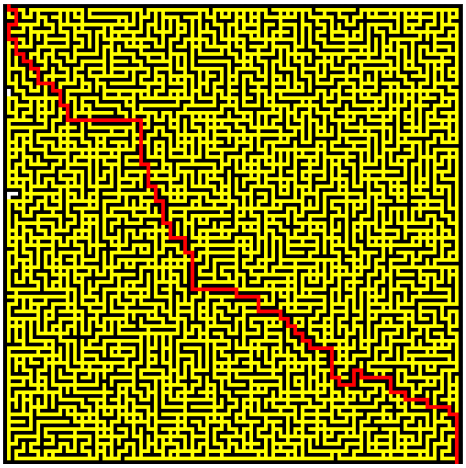

Figura 9: BFS

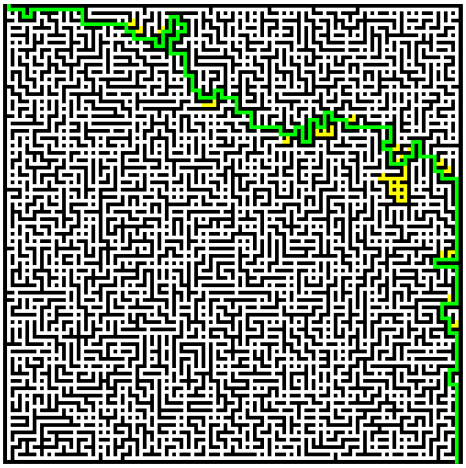

Figura 10: A*

Os resultados computados pelos gráficos podem ser vistos na tabela 5.

Componentes	**BFS**	**A***
Nodos Visitados	8609	425
Nodos na Memória	80	99
Tempo de Execução (em segundos)	0.70099	0.04799

Tabela 5: Resultados em tabuleiro de tamanho 125x125 com complexidade média.

Dos dados, temos as mesmas deduções se comparados com os resultados da execução dos algoritmos com uma complexidade baixa.

Para uma complexidade ainda maior, os resultados obtidos foram:

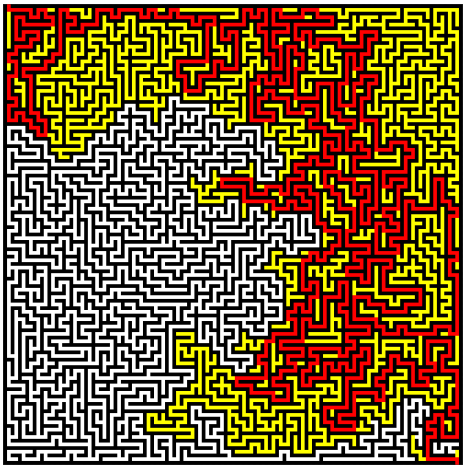

Figura 11: BFS

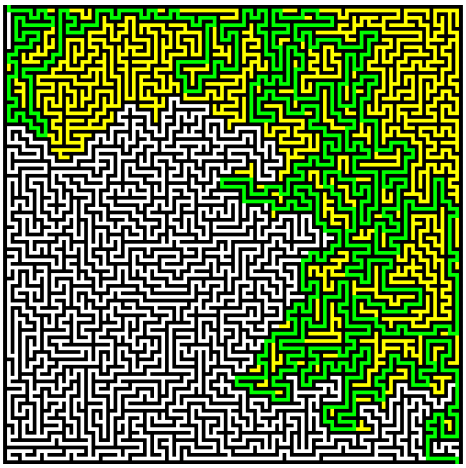

Figura 12: A*

Dos dados computados, pode-se afirmar que o algoritmo A* foi o melhor algoritmo se comparado em média. Estes dados podem ser conferidos na tabela abaixo.

Componentes	BFS	A*
Nodos Visitados	4502	3897
Nodos na Memória	9	17
Tempo de Execução (em segundos)	0.36498	0.38502

Tabela 6: Resultados em tabuleiro de tamanho 125x125 com complexidade alta.

Aumentando ainda mais o tamanho do labirinto, agora para 525x525, os resultados obtidos utilizando uma complexidade baixa foram os seguintes:

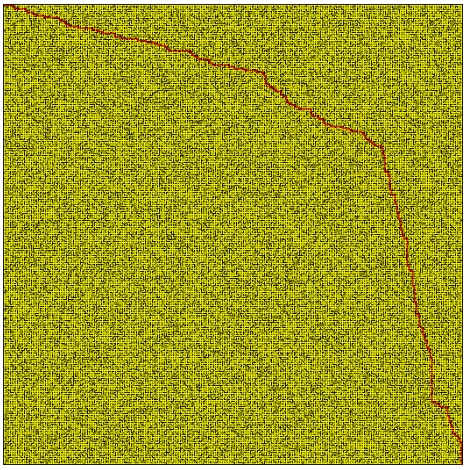

Figura 13: BFS

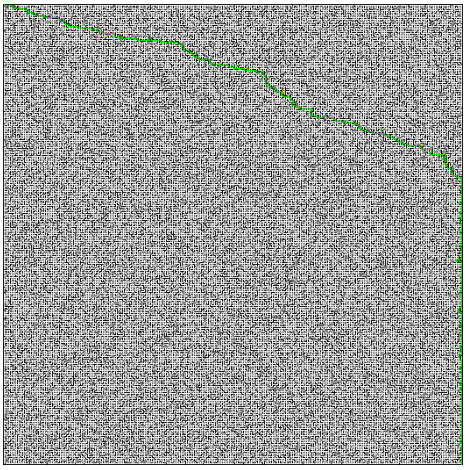

Figura 14: A*

As figuras 13 e 14 demonstram os resultados computados. Os resultados foram organizados de maneira aleatória e os resultados desses algoritmos podem ser vistos na tabela abaixo:

Componentes	BFS	A*
Nodos Visitados	170149	1215
Nodos na Memória	431	574
Tempo de Execução (em segundos)	16.0171	0.17898

Tabela 7: Resultados em tabuleiro de tamanho 525x525 com complexidade baixa.

Mais uma vez, o algoritmo A* demonstra ser a melhor opção, indiferente do tamanho do tabuleiro. Já para um labirinto de mesmo tamanho e complexidade média, os resultados obtidos foram:

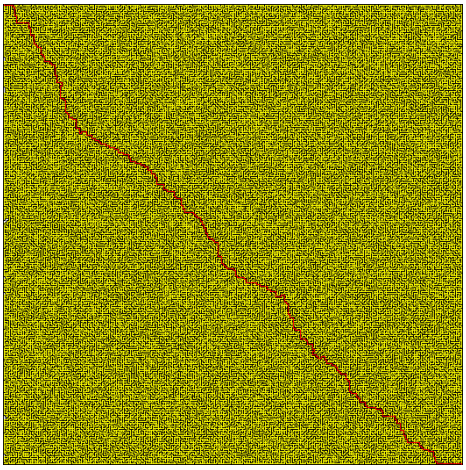

Figura 15: BFS

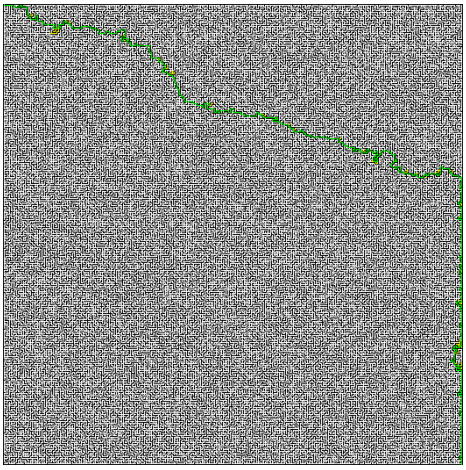

Figura 16: A*

As figuras 14 e 15 mostram... Deles, as estatísticas de execução de cada aplicativo são mostradas novamente na tabela abaixo.

Componentes	**BFS**	**A***
Nodos Visitados	154226	1604
Nodos na Memória	366	438
Tempo de Execução (em segundos)	13.5030	0.22418

Tabela 8: Resultados em tabuleiro de tamanho 525x525 com complexidade média.

Por fim, para o mesmo tamanho de labirinto, utilizando desta vez uma complexidade maior, os resultados obtidos foram os seguintes:

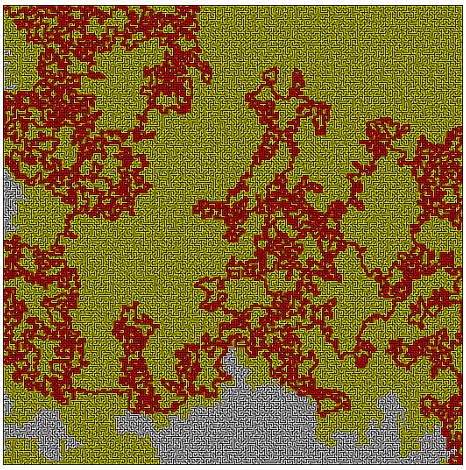

Figura 17: BFS

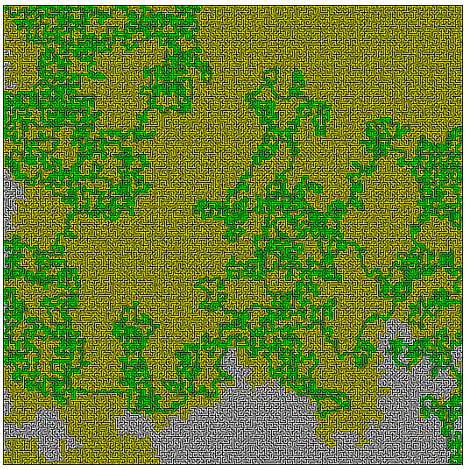

Figura 18: A*

Para os gráficos gerados, os resultados foram os seguintes:

Componentes	BFS	A*
Nodos Visitados	124282	120833
Nodos na Memória	24	67
Tempo de Execução (em segundos)	11.1930	11.7847

Tabela 9: Resultados em tabuleiro de tamanho 525x525 com complexidade alta.

Com essa análise, conclui-se que o algoritmo A* é avaliado como a melhor opção a ser aplicada no problema discutido. Isso se dá pelo fato do algoritmo não utilizar uma busca cega, resultando em uma solução mais otimizada. Já o algoritmo BFS mostra que, independente do tamanho do tabuleiro, seus resultados são praticamente idênticos, tendo divergências apenas em seu tempo de execução.

6. Considerações Finais

O conceito básico de um algoritmo de busca é somente uma pequena peça de um quebra cabeças maior que é a IA. O grande problema é como usar os algoritmos para resolver problemas difíceis. O algoritmo A*, um dos mais populares aplicados a problemas de busca, foi o algoritmo que mostrou os melhores resultados, se comparado aos demais neste problema.

A performance geral do algoritmo BFS foi satisfatória e se mostrou capaz de encontrar o menor caminho sempre entre os dois pontos na imagem.

O uso do algoritmo correto, portanto, pode fazer com que o desempenho em termos de poder computacional, uso de memória e tempo sejam reduzidos, tornando a aplicação mais robusta e estável.

Referências

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., and STEIN, C. (2012). *Algoritmos: Teoria e Prática 3ª Edição*. Elsevier Editora Ltda, Rio de Janeiro, RJ.

NORVIG, P. and RUSSEL, S. (2013). *Inteligência Artificial 3ª Edição*. Elsevier Editora Ltda, Rio de Janeiro, RJ.

Sazaki, Y., Primanita, A., and Syahroyni, M. (2017). Pathfinding car racing game using dynamic pathfinding algorithm and algorithm a. pages 164–169.

Zafar, A., Agrawal, K. K., and Anil Kumar, W. C. (2018). Analysis of multiple shortest path finding algorithm in novel gaming scenario. In Singh, R., Choudhury, S., and Gehlot, A., editors, *Intelligent Communication, Control and Devices*, pages 1267–1274, Singapore. Springer Singapore.

Apêndice A Código Fonte

O código fonte desenvolvido durante a concepção deste estudo está disponível em *www.github.com/lucaspeferreira*.