Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

BUSCA EM GRAFOS APLICADA À INTELIGÊNCIA ARTIFICIAL

Gabriel de Oliveira Guedes Nogueira

Relatório elaborado como parte do trabalho 3 da disciplina "SCC0530 - Inteligência Artificial", ofertada pelo Instituto de Ciências Matemáticas e de Computação - USP no 1º semestre de 2020. Professor responsável: **Prof. Dr. Alneu de Andrade Lopes**.

São Carlos - SP
2020

Sumário

1 Introdução

Um aspecto importante da inteligência é a solução de problemas com base em objetivos. A solução de muitos problemas (por exemplo, jogo da velha e xadrez) pode ser descrita por meio de uma sequência de ações que levam a um objetivo desejável. Cada ação altera o estado e o objetivo é encontrar a sequência de ações e estados que levam do estado inicial (início) ao estado final (objetivo). No contexto da Inteligência Artificial (IA), a sequência de etapas necessária para a solução de um problema não é conhecida a priori e deve, no geral, ser determinada por uma exploração sistemática de tentativa e erro de alternativas (Korf, 1996). Evidencia-se, assim, a importância de algoritmos eficientes de busca para a IA, principalmente aqueles aplicados a grafos, tipo abstrato de dado poderoso e bastante utilizado no campo da Inteligência Artificial para modelar diferentes classes de problemas.

Com isso em mente, o presente relatório tem por objetivo investigar e comparar a performance de diferentes algoritmos de busca em grafos aplicados a um problema tradicional de IA: a resolução de labirintos. Estes são compostos por um ponto de entrada, um ponto de saída, espaços em branco (por onde pode-se andar) e obstáculos (intransponíveis). O agente inicia no ponto de entrada (local de início da busca) e tem como tarefa encontrar o ponto de saída (final da busca). Os labirintos, que podem ser fornecidos pelo usuário ou gerados de forma aleatória, são representados por meio de grafos não-direcionados e sem peso em suas arestas. Dois algoritmos de busca cega e três algoritmos de busca informada serão analisados. São eles:

- Busca em largura (*breadth-first search*);
- Busca em profundidade (*depth-first search*);
- Best-First Search
- Busca A*;
- *Hill Climbing*.

A linguagem *Python* foi a escolhida para a implementação do trabalho. Tal escolha baseou-se na grande versatilidade e facilidade de uso da linguagem, o que possibilita ao programador um foco maior na resolução e análise teóricas do problema, preocupando-se menos com detalhes de implementação. Além de acompanhar este relatório, o código-fonte do programa também encontra-se disponível no link: https://github.com/Talendar/maze_solver.

2 Fundamentação Teórica

Nesta seção, uma breve introdução sobre os conceitos teóricos relacionados aos experimentos é feita. Como visto, é comum em IA a utilização de grafos para modelar problemas. Para tanto, é importante definir-se um espaço problema - o ambiente no qual a busca ocorrerá (Newell & Simon, 1972). O espaço problema consiste em um conjunto de estados do problema e em um conjunto de operadores que mudam o estado (Korf, 1996). Uma instância do problema é um espaço problema no qual estão definidos o estados inicial e o estado objetivo. O problema é resolvido achando-se uma sequência de operadores que mapeiam o estado inicial ao estado final.

Grafos são comumente utilizados para representar espaços problema. Os estados do espaço são representados pelos nós do grafo e os operadores pelas arestas entre os nós (Korf, 1996). As arestas podem ser direcionadas ou não-direcionadas, dependendo se os operadores correspondentes são reversíveis ou não. No caso deste trabalho, utiliza-se grafos com arestas não-direcionadas e sem peso, de forma a melhor representar um labirinto.

2.1 Busca Cega

Os algoritmos de busca mais gerais são os de busca cega ou não-informada. Eles realizam a busca por meio da força bruta, visto que não necessitam de nenhuma informação específica prévia sobre o domínio do problema (Korf, 1996). Tudo o que precisam é de uma descrição de estados, um conjunto de operadores legais, um estado inicial e um estado final alvo. Os algoritmos mais comuns de busca cega, e também usados neste trabalho, são a busca em largura (*breadth-first search* ou BFS) e a busca em profundidade (*depth-first search* ou DFS).

A busca em largura expande os nós na ordem de sua distância da raiz (nó representante do estado inicial), explorando um nível da árvore por vez até que a solução seja encontrada. Já a busca em profundidade sempre explora o próximo filho do nó mais profundo e ainda não explorado. Ambos os algoritmos podem ser implementados usando-se uma lista de nós ainda não expandidos, com a diferença de que a BFS gerencia a lista como sendo uma fila (*first-in, first-out*) e a DFS como uma pilha (*last-in, first-out*).

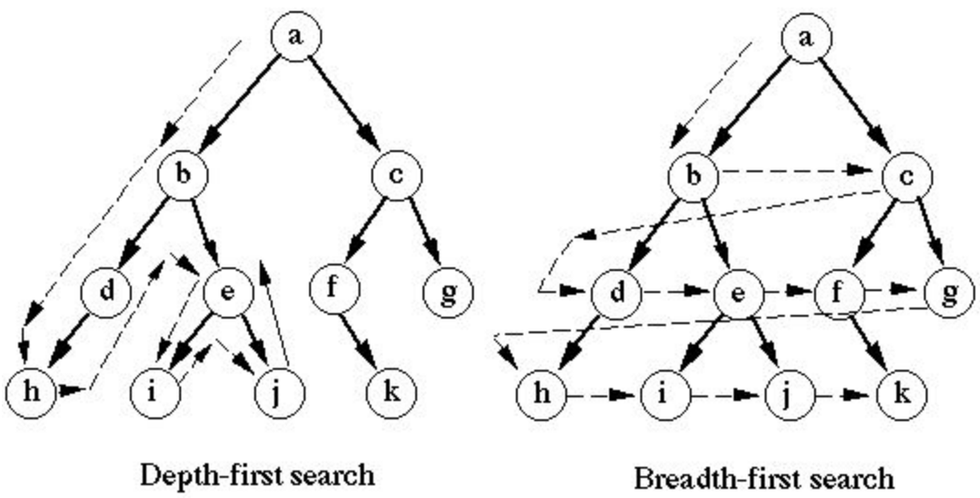

Fig. 1 - ilustração da diferença da sequência em que os nós são explorados entre a busca em profundidade (à direita) e a busca em largura (à esquerda). Fonte: http://www.cse.unsw.edu.au/~billw/Justsearch.html.

2.2 Busca Heurística/Informada

Para a resolução de problemas maiores, conhecimento específico do domínio deve ser adicionado para melhorar a eficiência da busca (Korf, 1996). Nesse contexto inserem-se os algoritmos de busca heurística ou informada, que, em muitos casos, podem achar soluções de forma mais eficiente do que estratégias de busca cega. A abordagem geral é chamada de *best-first search* (pesquisa por melhor escolha, em tradução livre), em que um nó é selecionado para expansão com base em uma função de avaliação (do inglês, *evaluation function*), *f(n)*. Esta é interpretada como sendo uma estimativa de custo, de forma que o nó com o menor custo é expandido primeiro (Russell & Norvig, 2009).

A escolha de *f* determina a estratégia de busca do algoritmo. A maior parte dos algoritmos de *best-first search* incluem, como um componente de *f*, uma função heurística, denotada por *h(n)*. Ela estima um custo otimista do menor caminho entre o estado em um nó *n* qualquer e um estado almejado (objetivo). Funções heurísticas são a maneira mais comum pela qual conhecimento adicional sobre o problema é transmitida ao algoritmo de busca (Russell & Norvig, 2009). De forma geral, se *n* é o nó objetivo, então $h(n) = 0$. Abaixo, uma breve introdução aos três algoritmos de busca heurística usados neste trabalho é feita.

2.2.1 Greedy Best-First Search

O algoritmo *Greedy Best-First Search*, por vezes chamado de busca gulosa ou simplesmente *best-first search*, tenta expandir o nó que está mais próximo ao objetivo, com o argumento de que ele provavelmente levará a uma solução mais rapidamente. Assim, o algoritmo avalia nós usando apenas a função heurística, isto é, $f(n) = h(n)$.

2.2.2 Busca A*

O algoritmo de *best-first search* mais conhecido é o chamado Busca A* (pronunciado "A-estrela"). Ele avalia nós através da combinação da função heurística $h(n)$ com uma função $g(n)$, que calcula o custo do caminho do nó de início da busca até o nó n (Russell & Norvig, 2009). Assim, $f(n) = g(n) + h(n)$ estima o custo da solução mais barata através de n. Trata-se de uma estratégia bastante poderosa: caso a função heurística $h(n)$ satisfaça certas condições, a Busca A* é tanto completa (achará uma solução, caso ela exista) quanto ótima (quando encontra uma solução, ela é a melhor possível).

2.2.2 Hill Climbing

Hill Climbing é um algoritmo de busca heurística usado para problemas de otimização matemática. Dada um conjunto de entradas e uma boa função heurística, ele tenta encontrar de forma rápida uma solução suficientemente boa, que pode não ser um máximo (ou mínimo, dependendo do problema) global. De forma geral, o *Hill Climbing* analisa os nós vizinhos do nó atual e seleciona aquele melhor otimiza o custo da solução atual. Um dos maiores problemas com essa estratégia de busca, e que se mostrou particularmente relevante no que tange a resolução de labirintos, é a possibilidade de que ele fique "preso" em um máximo/mínimo local, de forma a não chegar ao objetivo final (máximo/mínimo global).

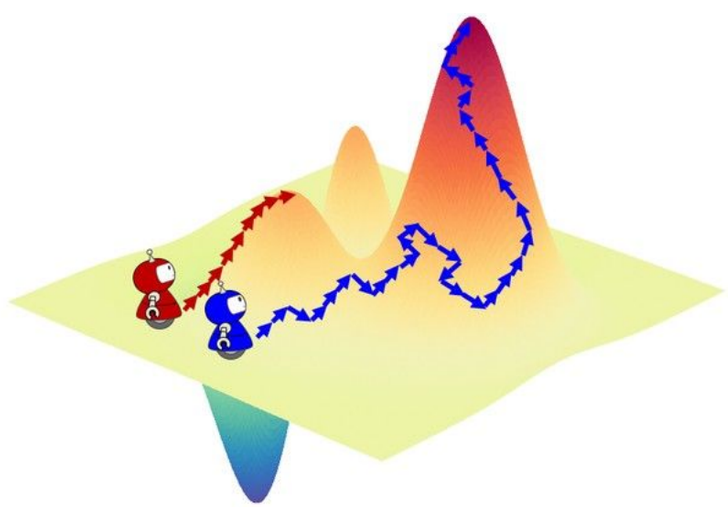

Fig. 2 - ilustração do funcionamento geral do algoritmo *Hill Climbing*. Note que, enquanto o agente azul, que iniciou a busca em um estado diferente, foi capaz de chegar ao máximo global (solução ótima), o agente vermelho ficou "preso" em um máximo local. Fonte: https://bit.ly/2z7fEW1.

2.3 Algoritmo de Prim: Geração de Labirintos Aleatórios

O algoritmo de Prim é um algoritmo guloso que encontra uma árvore geradora mínima (*minimum spanning tree* ou MST) de um grafo conexo, ponderado e não direcionado. Isso significa que o algoritmo encontra um subgrafo do grafo original no qual a soma total das arestas é minimizada e todos os vértices estão interligados. O algoritmo opera construindo a árvore um vértice por vez, adicionando, em cada passo, a conexão menos custosa possível da árvore para o outro vértice. Modificando-se o algoritmo de Prim e adicionando aleatoriedade à ele, é possível utilizá-lo para a

geração de labirintos aleatórios. Para tanto, basta fazer com que, em cada etapa, seja escolhida uma conexão aleatória ao invés da conexão de menor peso, como é feito na versão original.

3 Descrição da Implementação

Como citado, a linguagem de programação utilizada foi o *Python*. O código é dividido em quatro seções principais: implementação do TAD Grafo; implementação dos algoritmos de busca; representação do labirinto e operações relacionadas; menu principal e operações do usuário. Procurou-se detalhar diversos questões específicas da implementação na documentação do próprio código, que pode ser consultada pelo leitor.

3.1 TAD Grafo e Algoritmos de Busca

A implementação do TAD Grafo foi feita considerando-se um grafo não-direcionado e não-ponderado, mais adequado para modelar os caminhos em um labirinto do que outros tipos. A API permite, entre outros, adicionar vértices e arestas ao grafo e checar se ele contém um nó em específico. Cada nó do grafo é identificado por meio de um índice (valor inteiro) e um nome (string), ambos únicos. A implementação foi feita de maneira a permitir o uso da API para aplicações gerais, isto é, não limitadas ao problema em questão.

Seguindo o mesmo raciocínio, implementou-se, ainda, uma API contendo os algoritmos de busca utilizados no trabalho. Para tanto, foram aplicados conceitos de classes abstratas e herança, de forma a definir uma interface em comum para os diferentes algoritmos. A classe "mãe", chamada *Pathfind*, é abstrata e define as principais operações que podem ser realizadas com os algoritmos. Dela são herdeiros todas as outras classes que representam os algoritmos particulares de busca. Um detalhe relevante a ser notado é o da relação entre as classes *BestFirstSearch* e *AStar*, que representam os algoritmos homônimos. A segunda é herdeira da primeira (que, por sua vez, é filha da classe *Pathfind*), o que está em consonância direta com o fato de o algoritmo A* ser um algoritmo do tipo *best-first search* e, portanto, apresentar semelhanças significativas com ele.

3.2 Representando e Resolvendo Labirintos

Um módulo específico foi criado para representar labirintos e definir as suas operações. Ele encapsula a representação interna do labirinto (feita por meio de um grafo) e a sua resolução (por meio de algoritmos de busca), de forma a facilitar o seu uso por parte do cliente. Ao ser criado, um objeto da classe *Maze* converte uma matriz de caracteres fornecida como entrada para um grafo no qual cada nó representa uma célula andável do labirinto e cada aresta representa uma conexão entre duas células adjacentes. Nessa matriz, o símbolo "#" representa a entrada, "$" a saída, "*" uma célula andável e "-" um muro. O nome único de cada nó, usado para identificá-lo no grafo, é simplesmente a coordenada, em relação a matriz inicial de caracteres, da célula que ele representa.

A matriz de caracteres de um labirinto pode ser tanto lida de um arquivo fornecido pelo usuário quanto gerada aleatoriamente. Há uma função específica para gerar um labirinto aleatório por meio de uma variação do algoritmo de Prim. Ela recebe como argumentos as dimensões do labirinto a ser criado e uma porcentagem de ruído. Como o grafo gerado é uma árvore (devido às particularidades do algoritmo de Prim), ou seja, não possui ciclos (só há um caminho do nó inicial ao nó objetivo), tal ruído é usado para criar ciclos e, consequentemente, mais opções de caminhos no labirinto. Ele indica, simplesmente, a porcentagem de células muro que serão escolhidas aleatoriamente para serem convertidas em células andáveis.

Há, ainda, uma função para converter a matriz de caracteres de um labirinto em uma imagem, tornando mais agradável a visualização. A altura das imagens geradas é sempre de 800px (configurável no código-fonte) e a sua largura é calculada de forma a manter a proporção com a altura. Nas imagens finais, quadrados brancos indicam células andáveis não percorrida, quadrados amarelos indicam o caminho usado pelo algoritmo, quadrados pretos indicam muros, um quadrado azul indica a entrada e um quadrado vermelho a saída do labirinto.

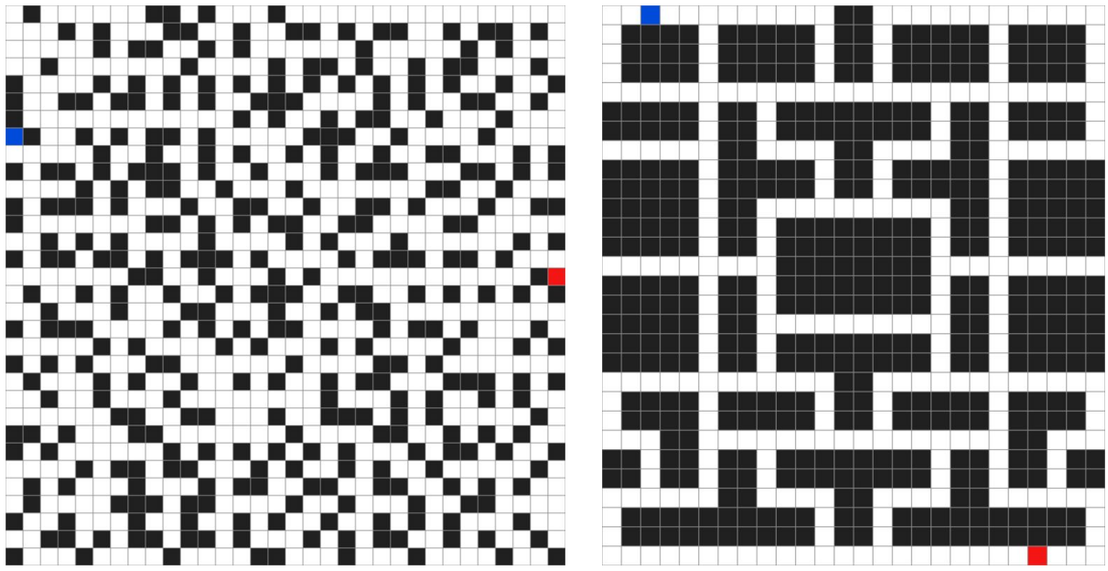

Fig. 3 - imagem de um labirinto gerado aleatoriamente (esquerda) e de um labirinto fornecido pelo usuário (direita). Ambas as imagens foram geradas pelo programa e não contém os caminhos achados pelos algoritmos.

Como heurística *h(n)* utilizada pelos algoritmos de busca informada utilizou-se a distância euclidiana entre o nó do estado atual e o nó do estado final. São consideradas as coordenadas dos nós na matriz de caracteres inicial. Como função *g(n)*, usada pela busca A*, utilizou-se o comprimento do caminho atual percorrido pelo algoritmo do nó inicial ao nó *n*. Tal função pode ser interpretada, com certas ressalvas, como sendo a distância de Manhattan entre o nó inicial e o nó *n*. Tentou-se utilizar a distância euclidiana em *g(n)*, contudo isso aumentou consideravelmente o tempo de execução do algoritmo, de forma que a ideia foi descartada.

3.3 Menu Principal e Interação com o Usuário

As interações com o usuário são feitas por meio do terminal de controle do computador. Ao ser executado, o programa apresenta um menu dando ao usuário a opção de carregar um labirinto a partir de um arquivo (operação 0), gerar um labirinto aleatório (operação 1) e sair do programa (operação 2). Quando opta-se por gerar um labirinto de forma aleatória, o usuário deve especificar as dimensões do labirinto e quantas vezes cada algoritmo de busca deverá ser executado. Para cada uma das execuções do conjunto de algoritmos de busca, um novo labirinto é criado com as dimensões especificadas. Os resultados (média dos dados obtidos nas execuções), assim como as imagens dos labirintos gerados, são salvos na pasta “out”, presente no diretório raiz do programa.

4 Resultados e Discussão

4.1 Labirinto Base (fornecido pelo usuário)

Dois labirintos criados manualmente foram testados. Um deles foi o fornecido nas especificações do trabalho e o outro foi criado para testar o caso em que não há um caminho da entrada à saída

do labirinto. Aqui será analisado apenas os resultados referentes ao labirinto válido, haja em vista a sua maior relevância.

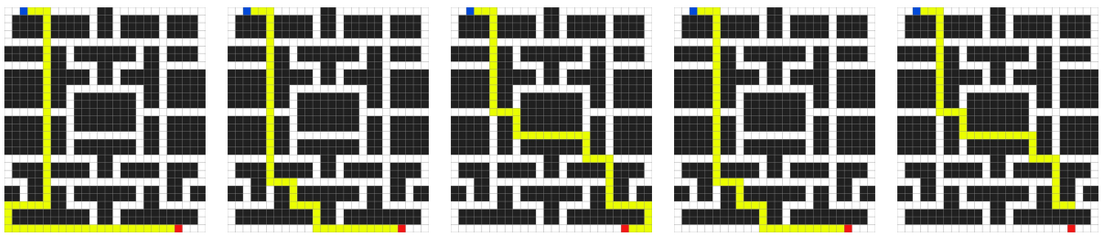

Fig. 4 - imagens geradas a partir da resolução do labirinto base por diferentes algoritmos de busca. Foi necessário reduzi-las em tamanho para que coubessem na página. As imagens originais podem ser consultadas junto ao código-fonte do programa.

Como pode-se ver através da figura 4, apenas o algoritmo de busca por *Hill Climbing* não foi capaz de encontrar a saída. Isto já era, de certa forma, esperado, haja em vista que a chance dele ficar preso em um mínimo local, impedindo-o de chegar a saída, como ocorreu neste caso, é alta. A tabela 1 mostra que o algoritmo que mais rapidamente encontrou a saída foi o de busca em profundidade (DFS), o que já era esperado, haja em vista o tamanho do grafo representativo do labirinto (pequeno).

Algoritmo	Tempo de busca	Achou a saída?	Comprimento do caminho
DFS	0.147 ms	Sim	58
BFS	0.51 ms	Sim	48
Best-First Search	0.74 ms	Sim	54
A*	2.43 ms	Sim	48
Hill Climbing	0.36 ms	Não	46

Tab. 1 - resultados da performance dos algoritmos de busca aplicados ao labirinto disponibilizado na especificação do trabalho. Note que o *Hill Climbing* não encontrou a saída, o que deve ser levado em consideração na comparação entre os algoritmos.

Nota-se que a performance dos algoritmos de busca cega foi melhor do que a dos algoritmos de busca informada. Isso já era esperado, pois trata-se de um espaço problema muito pequeno, de forma que o cálculo da heurística mais atrapalha do que ajuda, em termos de tempo de execução, os algoritmos que fazem uso dela. Os menores caminhos até a saída foram encontrados pela BFS e pela busca A*. Tal resultado também era esperado, pois, como visto, ambos possuem a garantia de encontrar uma solução ótima (neste caso, o caminho mais curto até a saída).

4.2 Labirintos Aleatórios

Nesta seção será feita a exposição e análise dos resultados dos algoritmos de busca aplicados a resolução de labirintos de diferentes tamanhos e gerados aleatoriamente. Para cada um dos tamanhos, os algoritmos são executados cinco vezes em diferentes labirintos. Isto significa que em cada uma das cinco etapas um labirinto do tamanho especificado é gerado e acha-se a sua solução usando-se cada um dos algoritmos. No final, obtém-se a média dos dados obtidos, estimando-se a performance de cada algoritmo naquele cenário.

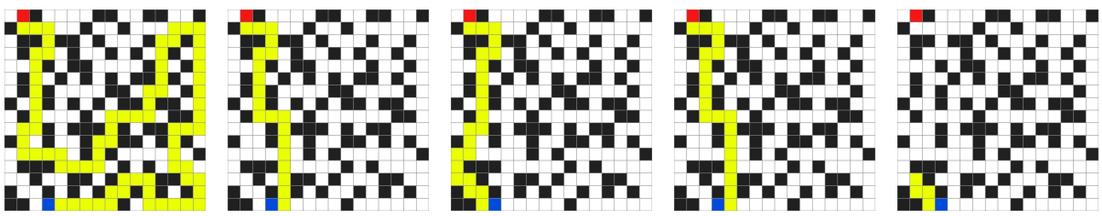

Fig. 5 - resultados dos algoritmos de busca em um dos cinco labirintos **16x16** gerados aleatoriamente. Imagens reduzidas para que coubessem na página. As imagens originais podem ser consultadas junto ao código-fonte.

16x16

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	0.082 ms	5/5	40
BFS	0.140 ms	5/5	23
Best-First Search	0.270 ms	5/5	24
A*	0.559 ms	5/5	23
Hill Climbing	0.054 ms	0/5	5

Tab. 2 - resultado médio da performance dos algoritmos de busca aplicados a labirintos aleatórios de tamanho **16x16**.

Os algoritmos foram testados em 7 tamanhos diferentes de labirintos: 16x16, 32x32, 64x64, 128x128, 256x256, 512x512 e 1024x1024. As tabelas contendo os resultados obtidos para cada um dos tamanhos, tal como as imagens (apenas para os quatro tamanhos menores) do caminho encontrado estão dispostas nesta seção em ordem crescente de tamanho. A análise dos dados permite certas conclusões. Em relação a busca em profundidade, é possível perceber claramente a sua boa performance em labirintos pequenos (e, em certo grau, também nos labirintos maiores) no que diz respeito ao tempo de execução do algoritmo. Em contrapartida, a DFS acha caminhos muito mais longos que as demais estratégias de busca, muito distantes, na maioria dos casos, da solução ótima. Já a busca em largura, apesar de ter um tempo de execução consideravelmente maior, sempre acha o menor caminho até o objetivo, o que era esperado, haja em vista a garantia do algoritmo em encontrar uma solução ótima.

32x32

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	0.309 ms	5/5	129
BFS	0.529 ms	5/5	49
Best-First Search	0.738 ms	5/5	50
A*	2.287 ms	5/5	49
Hill Climbing	0.084 ms	0/5	9

Tab. 3 - resultado médio da performance dos algoritmos de busca aplicados a labirintos aleatórios de tamanho **32x32**.

Fig. 6 - resultados dos algoritmos de busca em um dos cinco labirintos **32x32** gerados aleatoriamente. Imagens reduzidas para que coubessem na página. As imagens originais podem ser consultadas junto ao código-fonte.

A busca gulosa *Best-First*, por sua vez, foi, sem dúvidas, a estratégia de busca que apresentou maior performance geral. Apesar de apresentar um tempo de execução maior do que a DFS para labirintos pequenos, foi capaz de superá-la por uma larga margem quando tratando de espaços problema maiores. Mesmo não sendo capaz de achar a solução ótima na maioria das vezes, encontrou caminhos relativamente curtos. Isso evidencia a qualidade da heurística utilizada (distância euclidiana) e demonstra o poder da busca informada.

A busca A* apresentou os resultados mais distantes do esperado. Esperava-se que tal algoritmo fosse o de melhor performance, o que não se revelou verdade. Apesar de, como esperado, ter encontrado sempre os menores caminhos, a busca A* apresentou o tempo de execução mais alto em todos os cenários. Há duas explicações possíveis prováveis para o ocorrido: houve uma falha na implementação do algoritmo ou a função *g* escolhida mostrou inadequada ao problema. Revisei o código do algoritmo diversas vezes e não fui capaz de encontrar erros e nem trechos que permitissem maior otimização. Além disso, a classe *AStar*, que implementa o algoritmo, herda a maior parte de suas funções da classe *BestFirstSearch*, que, ao que tudo indica, está funcionando corretamente. O mais provável é que a função *g*, apesar de possibilitar ao algoritmo achar o menor caminho, acrescenta um *overhead* muito grande à execução do algoritmo. Possivelmente, para problemas mais complexos, onde poder-se-ia escolher uma função de avaliação mais relevante, a busca A* teria apresentado resultados melhores.

Por fim, a busca por *Hill Climbing*, em sua forma clássica, mostrou-se extremamente inadequada para lidar com a resolução de labirintos. Por não possuir um mecanismo de *backtracking*, o algoritmo frequentemente parava a sua execução ao atingir um ponto de mínimo local, de forma que em nenhum dos casos analisados ele foi capaz de encontrar a saída do labirinto. Trata-se de um algoritmo poderoso, mas cuja a aplicação ao problema deste trabalho não mostrou-se adequada. Ele é mais indicado para lidar com problemas de otimização matemática mais diretos.

64x64

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	0.924 ms	5/5	350
BFS	2.163 ms	5/5	101
Best-First Search	2.100 ms	5/5	117
A*	10.612 ms	5/5	101
Hill Climbing	0.108 ms	0/5	12

Tab. 4 - resultado médio da performance dos algoritmos de busca aplicados a labirintos aleatórios de tamanho **64x64**.

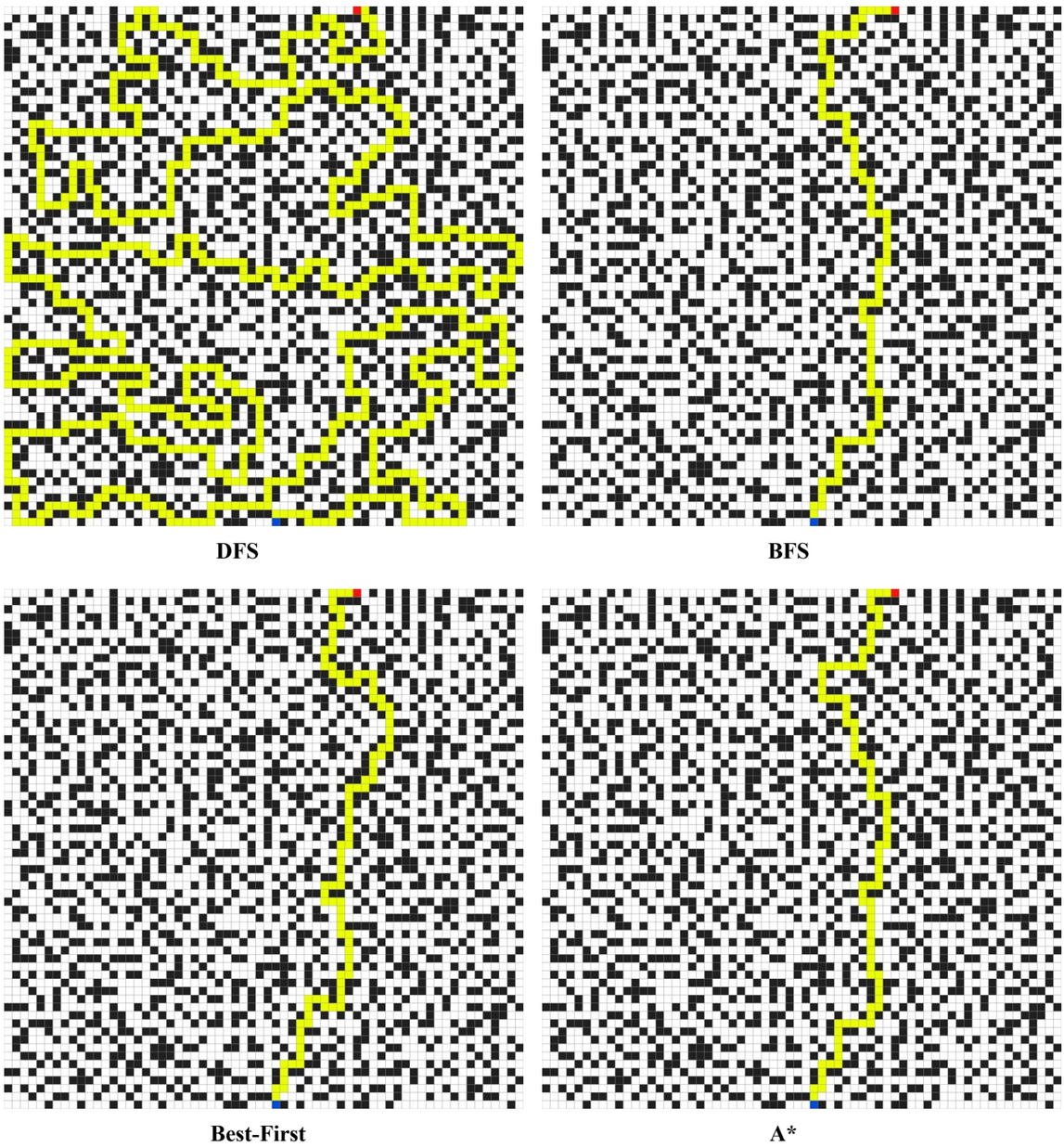

Fig. 7 - resultados dos algoritmos de busca (com exceção do *Hill Climbing*) em um dos cinco labirintos **64x64** gerados aleatoriamente. Imagens reduzidas para que coubessem na página. As imagens originais podem ser consultadas junto ao código-fonte.

128x128

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	4.016 ms	5/5	1755
BFS	8.300 ms	5/5	187
Best-First Search	2.820 ms	5/5	226
A*	37.421 ms	5/5	187

Tab. 5 - resultado médio da performance dos algoritmos (com exceção do *Hill Climbing,* removido devido a sua pouca relevância) de busca aplicados a labirintos aleatórios de tamanho **128x128**.

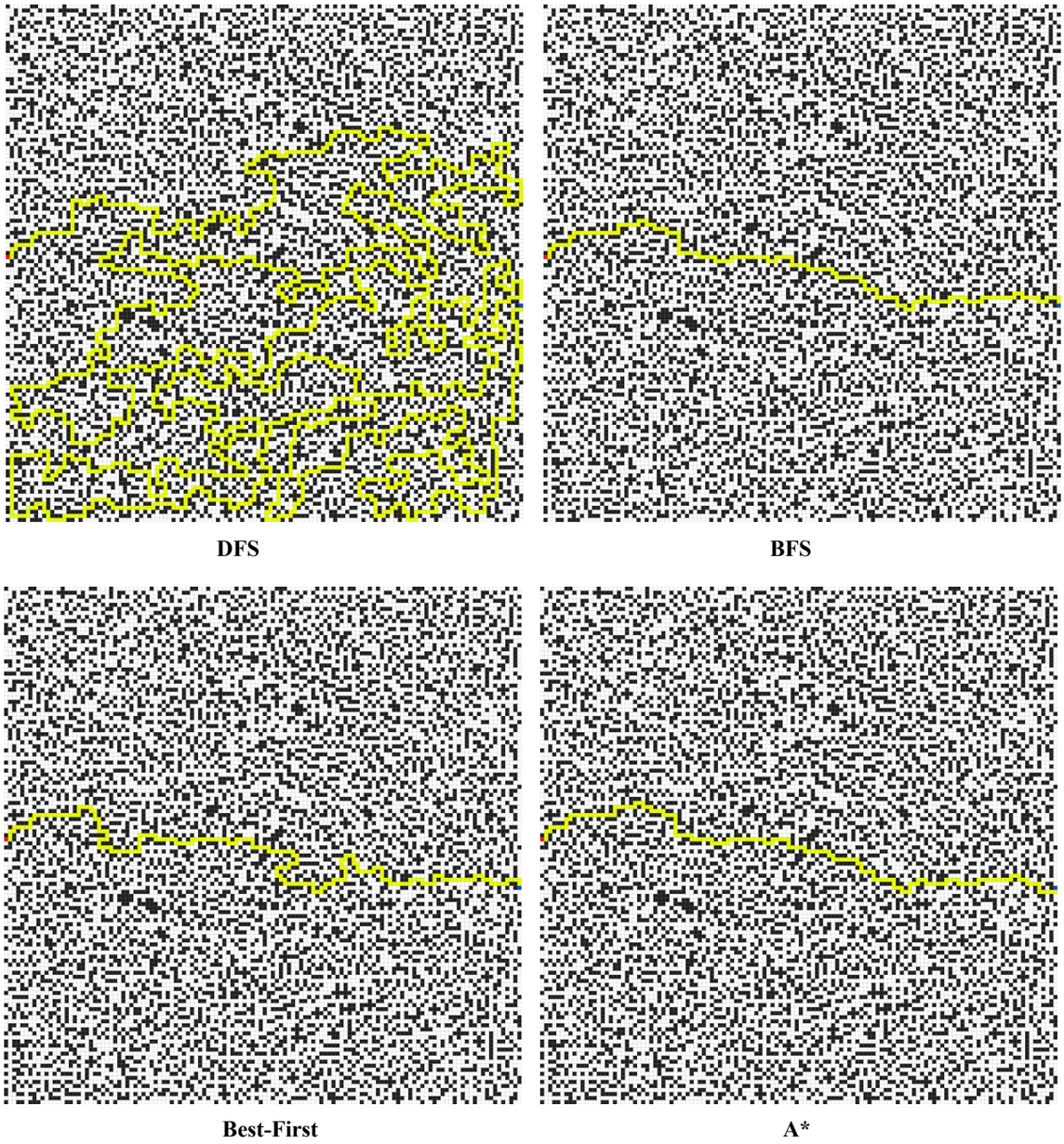

Fig. 8 - resultados dos algoritmos de busca (com exceção do *Hill Climbing*) em um dos cinco labirintos **128x128** gerados aleatoriamente. Imagens reduzidas para que coubessem na página. As imagens originais podem ser consultadas junto ao código-fonte.

256x256

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	15.585 ms	5/5	6557
BFS	41.336 ms	5/5	420
Best-First Search	7.253 ms	5/5	508
A*	301.16 ms	5/5	420

Tab. 6 - resultado médio da performance dos algoritmos (com exceção do *Hill Climbing,* removido devido a sua pouca relevância) de busca aplicados a labirintos aleatórios de tamanho **256x256**.

512x512

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	81.41 ms	5/5	25848
BFS	162.53 ms	5/5	756
Best-First Search	19.42 ms	5/5	911
A*	1570.89 ms	5/5	756

Tab. 7 - resultado médio da performance dos algoritmos (com exceção do *Hill Climbing,* removido devido a sua pouca relevância) de busca aplicados a labirintos aleatórios de tamanho **512x512**.

1024x1024

Algoritmo	Tempo de busca médio	Achou a saída?	Comprimento médio dos caminhos
DFS	166.77 ms	5/5	61448
BFS	741.08 ms	5/5	1642
Best-First Search	58.55 ms	5/5	2003
A*	13479.43 ms	5/5	1642

Tab. 8 - resultado médio da performance dos algoritmos (com exceção do *Hill Climbing,* removido devido a sua pouca relevância) de busca aplicados a labirintos aleatórios de tamanho **1024x1024**.

5 Referências

Korf, R. E. (1996). Artificial Intelligence Search Algorithms. In Algorithms and Theory of Computation Handbook. CRC Press.

Newell, A. & Simon, H.A. (1972). Human Problem Solving. Prentice-Hall, Englewood Cliffs, N.J., 1972.

Russell, S.; Norvig, P. (2009). Artificial intelligence: a modern approach (3rd ed.). Prentice Hall Press, USA.