Análise de Algoritmos de Busca em Grafos KNN

Alneu A. Lopes[1], **Calvin S. Camargo**[2], **Guilherme S. Silvestre**[2]

[1]Instituto De Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Av Trabalhador São-Carlense, 400, Centro - São Carlos, SP - Brasil - Caixa-postal: 668

[2]Escola de Engenharia de São Carlos – Universidade de São Paulo (USP)

alneu@icmc.usp.br, calvinssdcamargo@usp.br, gsoares.silvestre@usp.br

1. Introdução

A palavra "grafo" é um neologismo derivado da palavra inglesa *graphem*, primeira vez usada com sentido científico pelo matemático inglês James Joseph Sylvester (1814-1897). Em sua definição matemática, grafo é um par (V, A) sendo V um conjunto arbitrário de informações, chamados de vértices, e A são os subconjuntos que compreendem dois vértices vizinhos $V^{(2)}$, estes denominados como arestas [Feofiloff et al. 2011].

Os grafo podem representar diversos conjuntos relacionados de informações: movimentos em um jogo de xadrez, posição geográfica e rotas entre cidades no mapa, árvore hereditária, entre outros. Aliás, os grafos também podem receber muitas regras que controlam sua forma e sua aplicabilidade para o campo desejado, e dentre as diversas opções, vamos estudar os grafos *KNN* e os algoritmos de análise de busca.

1.1. Grafos e Algoritmos de Busca

Grafos *KNN* ou *k-nearest neighbor graph* é um tipo específico de grafo que podem ser configurados utilizando dois parâmetros: (V, K), sendo V o número total de vértices e K a instrução de quantas arestas serão geradas de um certo nó. As arestas são construídas de forma a conectar os vértice mais próximos.

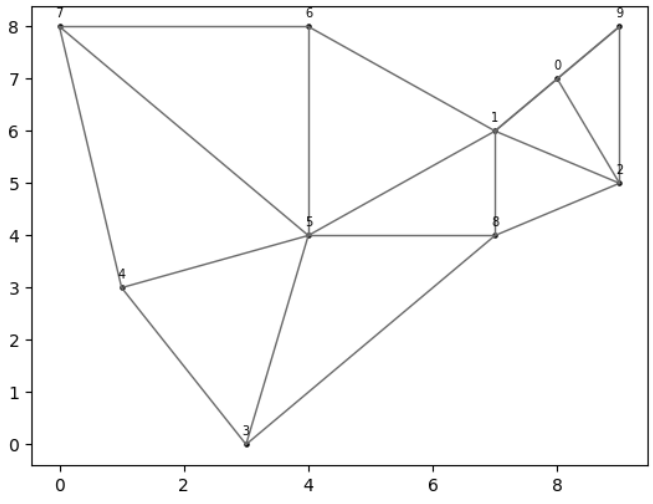


Figura 1. Grafo KNN com dez vértices e três arestas. Fonte: Autor.

Podemos estudar com eficiência métodos de algoritmos de busca em grafos. Dentre os algoritmos, temos duas categorias importantes de busca: A busca não-informada (cega), ou busca exaustiva, é um tipo de busca que procura um determinado alvo percorrendo os vértices de um grafo, sem quaisquer informações que auxiliem sua tarefa.

A busca heurística que é guiada por informações quantitativas de definem um meio de decidir seu percurso no grafo. Para isso, utilizaremos a categoria de classificação heurística de *Best First* que aplicará uma fórmula que definirá qual vértice visitar primeiro:

$$f(V) = g(V) + h(V)$$

Seja $f(V)$ função do cálculo heurístico total de um determinado vértice, $g(V)$ o valor heurístico calculado do percurso até esse vértice e $h(V)$ um valor fixo que expressa a proximidade do vértice com o vértice alvo. Então, $f(V$ representa a função a ser minimizada para alcançar o objetivo final da busca.

1.2. Busca cega ou busca exaustiva

A busca cega demonstra-se como um desafio computacional, especulando uma complexidade alta na realização da tarefa pois, de certa forma, devemos testar unitariamente diversos vértices até encontrarmos o desejado alvo.

Para esse tipo de busca, temos o algoritmo de busca de largura, que faz a sua procura testando os vértices "de nível a nível"de distância nodal do ponto de origem, é denominado um algoritmo "completo"pois pode listar todos os caminhos da origem ao vértice. Possui a vantagem de encontrar o alvo rapidamente quando próximo da origem da busca e garante fazer o caminho mais curto (em quantidade de nós), porém possui a desvantagem precisar testar quase todos os nós do grafo caso seu alvo esteja do lado oposto à origem de busca.

Na mesma categoria, vamos testar a busca de profundidade, que consiste em traçar um caminho "sem rumo"no grafo, evitando os vértices já visitados, até encontrar seu alvo. Possui a vantagem de oferecer a probabilidade de encontrar um alvo a diversos nódulos de distância, porém quase sempre traça um caminho muito longo até o alvo. É considerado um método "incompleto"pois pode falhar quando sua análise entrar em um ramo infinito de vértices.

1.3. Busca heurística

A busca heurística, como definido anteriormente, calcula os parâmetros de um vértice e escolhe, dentre as opções, o que tiver um valor heurístico mais promissor para encontrar o alvo. Para definimos o cálculo heurístico dos pontos da seguinte forma:

$$f(V) = G.g(V) + H.h(V)$$

Sendo $g(V)$ o valor heurístico calculado do percurso até esse vértice, i.e, a distância percorrida nó a nó da origem até o vértice, $h(V)$ um valor fixo que é a distância em linha reta (distância euclidiana) do vértice com o vértice alvo, adquirindo o valor de 0 quando este é o próprio alvo e G e H são constantes que irão configurar o perfil da busca.

- O primeiro desses algoritmos é o *Best First* que trata o problema de uma maneira gulosa ao analisar somente o valor heurístico fixo do ponto, desconsiderando seu caminho. Para isso, utilizamos os parâmetros $H = 1$ e $G = 0$.
- O segundo é a busca A que superestima o custo para alcançar o objetivo, configurando $H = 10$ e $G = 1$.
- O último é a busca A* que apresenta um perfil otimista em propor a encontrar o caminho mais curto em distância até o alvo, para isso fazemos $H = 1$ e $G = 1$.

2. Implementação

Foi utilizado *Python* como linguagem para o desenvolvimento do *software* que analisas as buscas. Além disso, utilizamos de recursos e bibliotecas do estado da arte para a otimização e aumento da produtividade. Isso foi feito de modo que o objeto de estudo nesse trabalho se manteve condizente com a proposta. Os códigos estão disponíveis em `https://github.com/calvinsuzuki/Busca-Trabalho2-IA`.

A geração do grafo se beneficiou do pacote *networkx*, de forma a otimizar a criação de grafos e, principalmente, o acesso aos seus elementos. A contribuição principal provém da estrutura de aceso à instâncias dos grafos, como as arestas ou os filhos de um vértice. Isso é feito a partir do tipo *dicionário* (estrutura de dados presente no python), permite esse acesso mais rápido devido à forma de tabelas de *hash* [NetworkX 2020].

Então, o grafo foi arquitetado de forma que todos os nós apresentassem ao menos três vértices. Isso pois, garantimos que houvessem as ligações que saem dos vértices em direção aos seus K vizinhos mais próximos. Outra lei implementada se baseia na possibilidade de duplicação de uma aresta com sentido contrário entre dois nós. Isso permite que um caminho possa seguir ambos os sentidos da arestas, consistindo um um ambiente conciso para os algoritmos de busca.

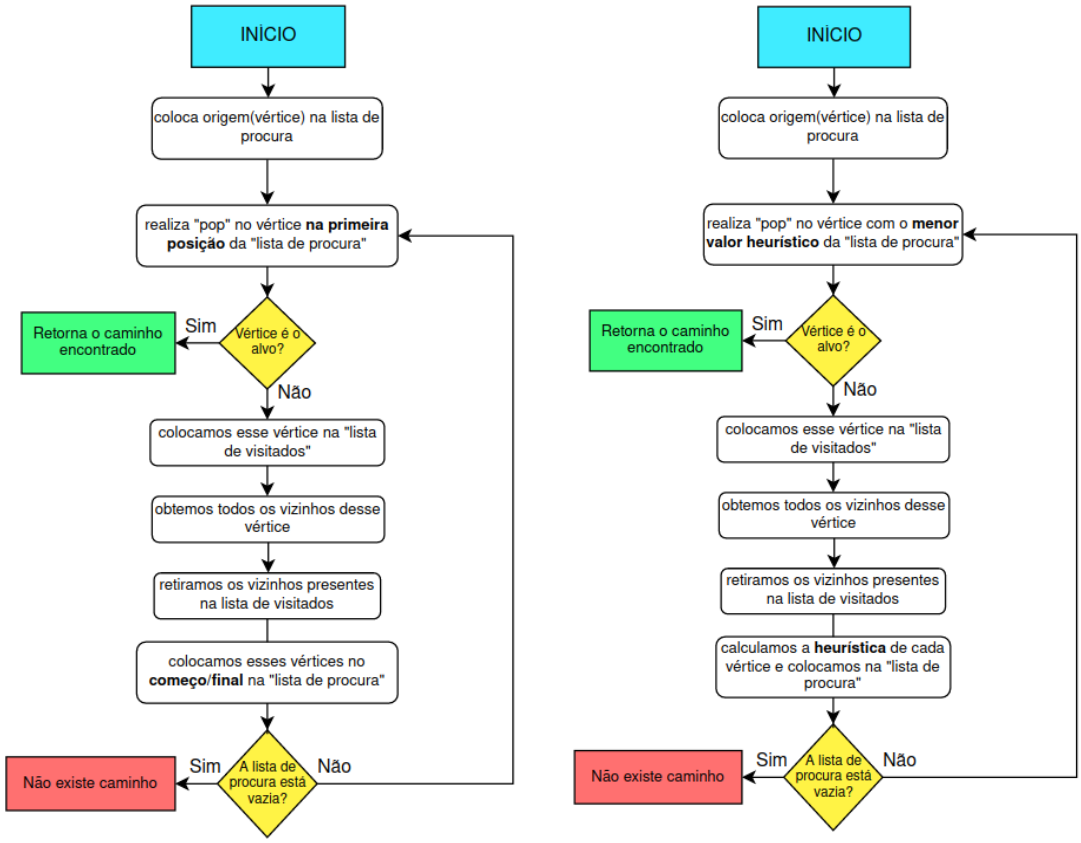
Já no que tange às implementações dos algoritmos de busca, esses foram representados pelas buscas exaustivas: Busca em Largura e Busca em Profundidade. Como também pelos métodos Heurísticos: *Best First*, *A* e *A**. As operações matriciais foram realizadas com o *numpy*, de forma a evitar a latência do *Python* em operações devidos à não tipagem das variáveis. O uso do *numpy* promove maior eficiência em operações recorrentes no método, podendo diminuir grandemente a complexidade de cada um [num].

A manufatura dos gráficos utilizou-se do pacote *matplotlib*, apresentando recursos gráficos específicos voltados para a visualização e interpretação de gráficos. Nesse campo, a visualização dos grafos gerados foram facilitadas, como também a visualização dos resultados finais que serão mostrados na secção 3. Por fim, uma concatenação animada dos caminhos processados pelos métodos pôde ser implementada (Figura 3).

O monitoramento da memória dos algoritmos também apresentou grandes desafios em sua elaboração. Em especial, a biblioteca *tracemalloc* foi capaz de prover recursos em baixo nível para a análise dos processos presentes no tempo de execução. O enfoque foi calcular a máxima memória alocada em algum instante de tempo durante a execução.

3. Resultados

Os resultados abordaram quatro entidades essenciais para a análise desses algoritmos: o tempo de processamento, a utilização de memória, o tamanho do caminho em



(a) Ao incluir os vértices, profundidade os coloca no **começo** da lista de procura, e largura no **final**.

(b) Os cálculos heurísticos *Best First* são configurados com as constantes G e H.

Figura 2. Implementação dos algoritmos de busca.

vértices e no espaço (distância euclidiana). Exceto para a análise da memória, o procedimento busca gerar pontos iniciais e finais aleatórios de forma que todos os métodos busquem o mesmo ponto a cada iteração a partir de um *set* fixo de configurações de grafos. Ou seja, para uma nova repetição, todos os métodos tem um mesmo ponto inicial e final, entretanto diferentes da iteração anterior.

O *set* de grafos apresentam vértices de 500, 5000 e 1000, todos com coordenadas instanciadas aleatoriamente. Os números de arestas que estão presentes são 3, 5 e 7, formando 9 combinações de grafos geradas. Para a análise de memória, completou-se o conjunto com três grafos com 100 vértices. A representação do grafo de 500 vértices e 3 arestas se encontra na Figura 3.

Os dados de tempo, tamanho em vértices e em distância foram coletados e armazenados. Calculou-se média e o desvio padrão para cada entidade, permitindo avaliar a variação apresentada pelos algoritmos nas condições de teste. Essas estatísticas são representadas nas Figuras 4, 5 e 6, contendo os resultados no tempo de processamento médio, número de vértices médio dos caminhos gerados e a distância média percorrida pelos algoritmos.

A coleta de dados da memória foi baseada na performance crítica dos algoritmos a partir dos pontos mais fastados do grafo. Dessa forma, buscou-se avaliar a alocação de

memória para cada método na pior situação, visando majorar as necessidades de armazenamento em um caso de aplicação. Ou seja, para todos os 9 grafos houve o mapeamento dos pontos mais afastados e a aplicação dos métodos implementados com o monitoramento de memória. O gráfico gerado é apresentado na Figura 8.

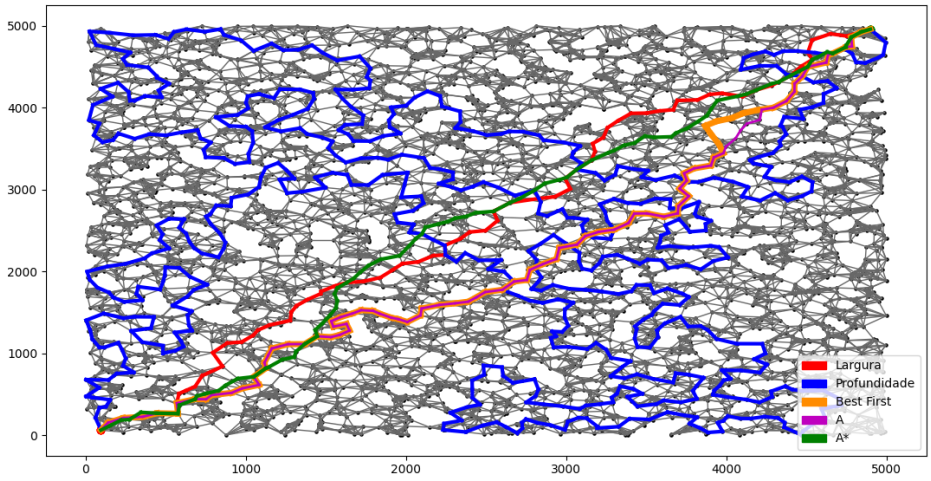


Figura 3. Comparação dos caminhos dos algoritmos no grafo de 500 vértices e 3 arestas. Fonte: Autor.

4. Discussão

Podemos analisar com os dados gerados o algoritmo que mais bem utiliza o seu tempo, o *Best First* realizou as buscas mais rapidamente (). Isso é devido a solução ”gulosa”que ele apresenta ao escolher a visita aos vértices. Olhando de uma certa perspectiva, ele se parece com uma busca de profundidade porém direcionada ao alvo, o que implica na rapidez de seu trabalho. Em contrapartida, o algoritmo mais demorado é o de profundidade (quase tanto quanto largura), pois em diversos casos ele se perde no grafo realizando uma série de ”zig-zags”, tomando muito tempo de processamento.

Quando consideramos a menor distância separada por vértices, o menor caminho sempre é a primeira solução encontrada pela busca de largura, pois em sua definição operativa, o algoritmo testa todos os caminhos na ordem crescente em distância nodal da origem até encontrar o alvo. Novamente, o algoritmo que é contraindicado para solicitação de um caminho com menos vértices é o profundidade. Ele apresenta caminhos extremamente longos com uma razão média de três vezes mais vértices.

Para procurarmos o caminho com a menor distância euclidiana até o alvo, devemos usar o caminho encontrado pelo algoritmo $A*$, pois em seu cálculo heurístico, ele pondera igualmente a distância percorrida e a distância até o alvo, seu caminho necessariamente tenta encontrar o alvo percorrendo o mínimo de espaço possível. Assim como explicado no caso anterior, o algoritmo de profundidade também é o pior no quesito de distância percorrida, ele na média entrega caminhos ao menos quatro vezes mais longos.

É importante pontuar as diferenças entre uma busca exaustiva e uma busca heurística como o evidente impacto na questão de tempo empregado e resultado satis-

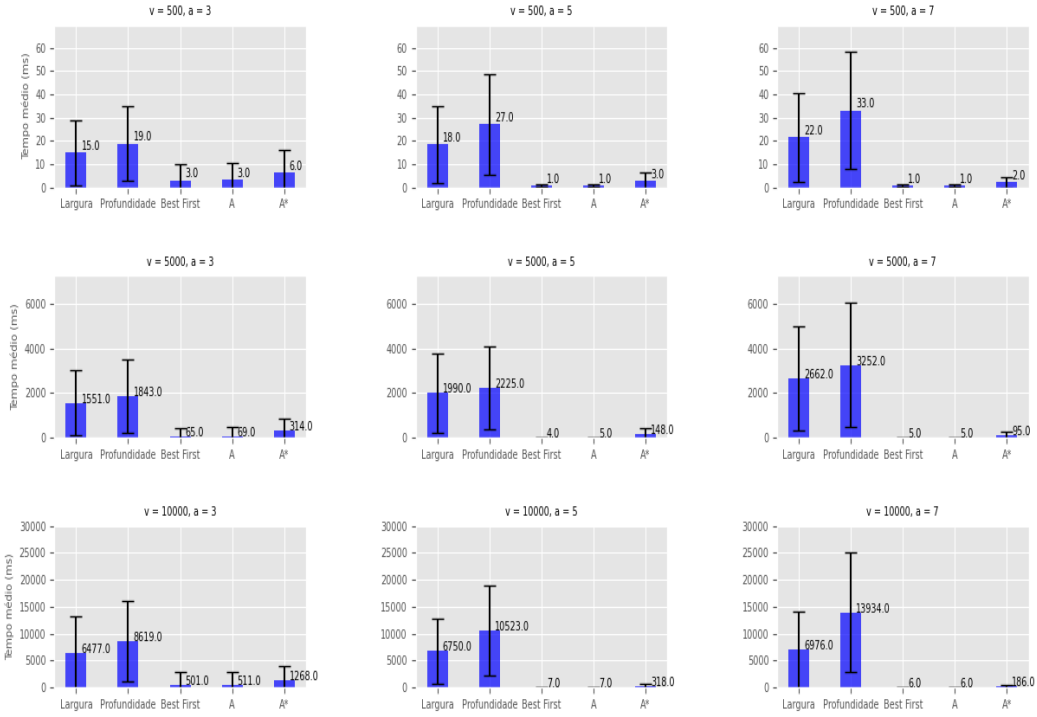


Figura 4. Tempo médio de processamento em 100 interações , variando o ponto inicial e o objetivo. Fonte: Autor.

fatório, possível gráceis ao acesso e o uso da informação sobre o alvo buscado. Além de possuir uma média melhor de tempo, a sua variância também é menor em relação às buscas cegas.

As buscas exaustivas possuem uma maior sensibilidade à escolha da origem e do alvo no grafo. Em (citar gráficos de tempo) é possível observar a grande variância presente nos dados de busca de largura e profundidade. Imaginemos os casos mais extremos da busca: o melhor é o caso que o primeiro ponto a ser analisado é o alvo, e o pior caso é o último ponto a ser analisado é o alvo. Observamos que o melhor caso pode vir a acontecer com quaisquer um dos métodos, mas o pior caso é devastador para a complexidade das buscas cegas pois elas precisam analisar todos os vértices do grafo.

Foi feita uma análise do uso de memória para a execução de cada algoritmo, utilizando a biblioteca *tracemalloc*, pudemos checar o uso da memória de todo e quaisquer processos envolvidos à execução do método. Observamos que dentre os algoritmos do *software*, a busca por profundidade acumulou muito mais memória que os demais métodos. Acreditamos que isso foi devido ao implementação feita em nosso programa, pois nessa análise consideramos todas as variáveis auxiliares do processo.

Referências

Numpy v1.21 manual.

Feofiloff, P., Kohayakawa, Y., and Wakabayashi, Y. (2011). Uma introdução sucinta à teoria dos grafos. pages 1–20.

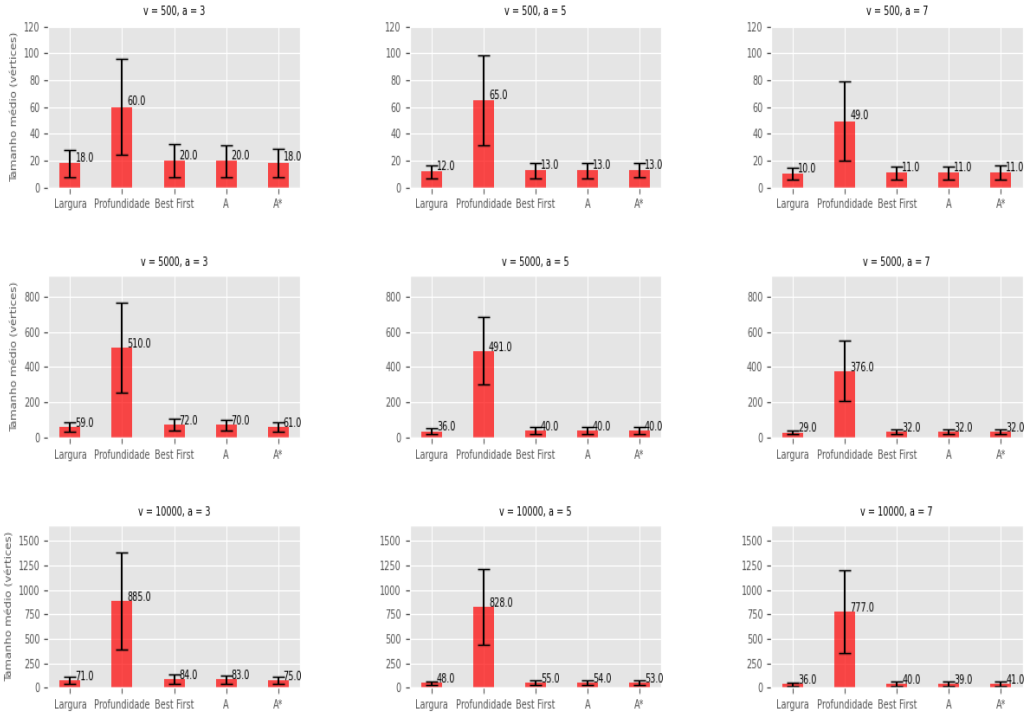
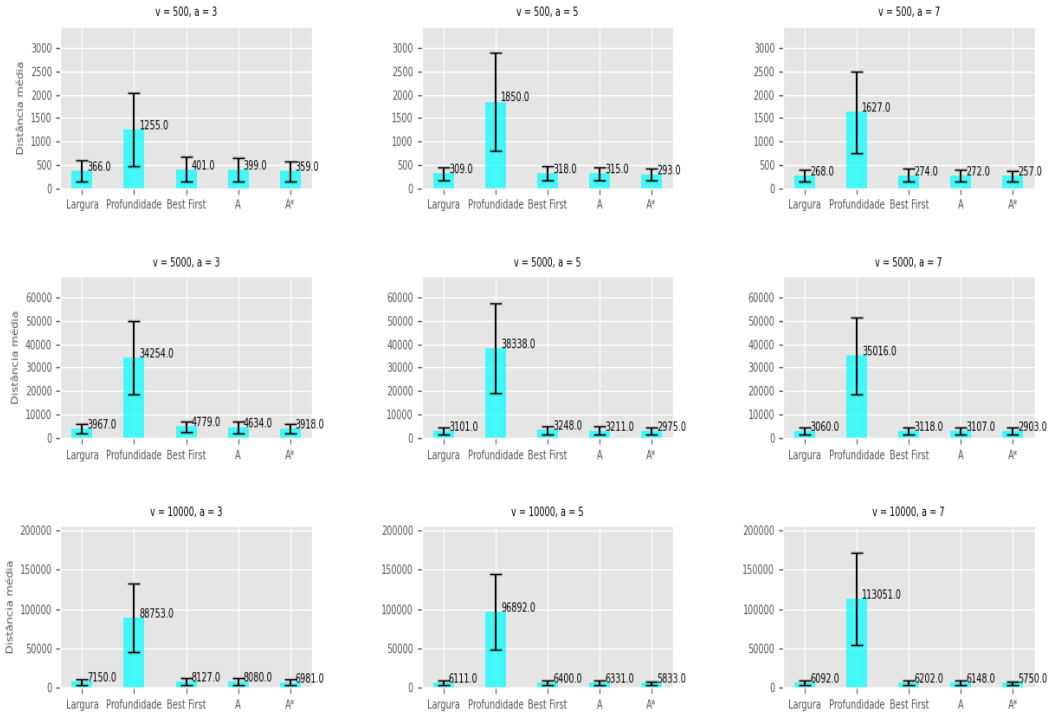


Figura 5. Tamanho médio em vértices calculados em 100 iterações, variando o ponto inicial e o objetivo. O desvio padrão da busca em profundidade é significante, o que implica a sua instabilidade nesse quesito. Fonte: Autor.

NetworkX (2020). Networkx documentation. Accessed = 2021-07-07.



(1,1)

Figura 6. Tamanho médio em distância euclidiana para 100 interações. Observa-se que a busca em profundidade tem maior tamanho médio, enquanto a busca A* apresenta a distância ótima até o objetivo. Fonte: Autor.

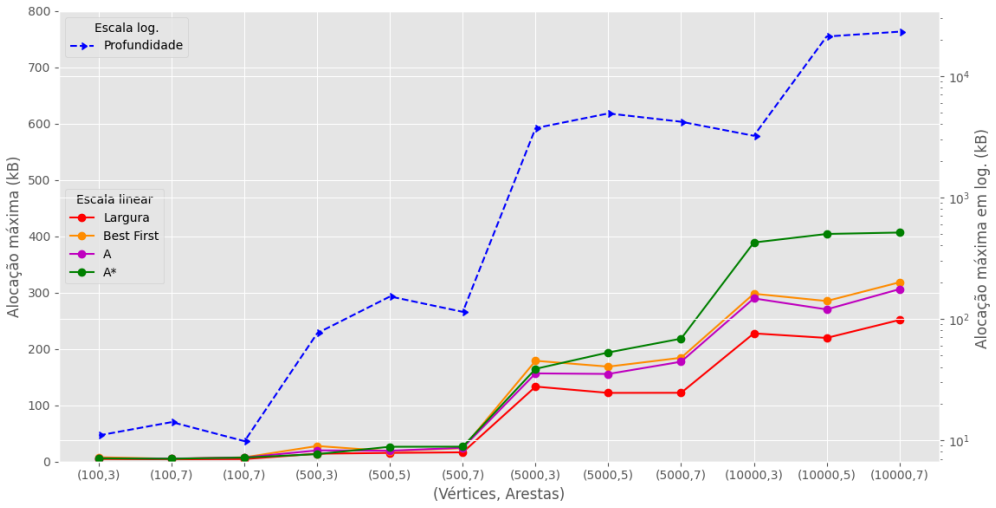


Figura 7. Alocação de memória total de todos os processos filhos chamados no tempo de execução de cada método. Observa-se que a escala para a busca em profundidade é logarítmica. Fonte: Autor.

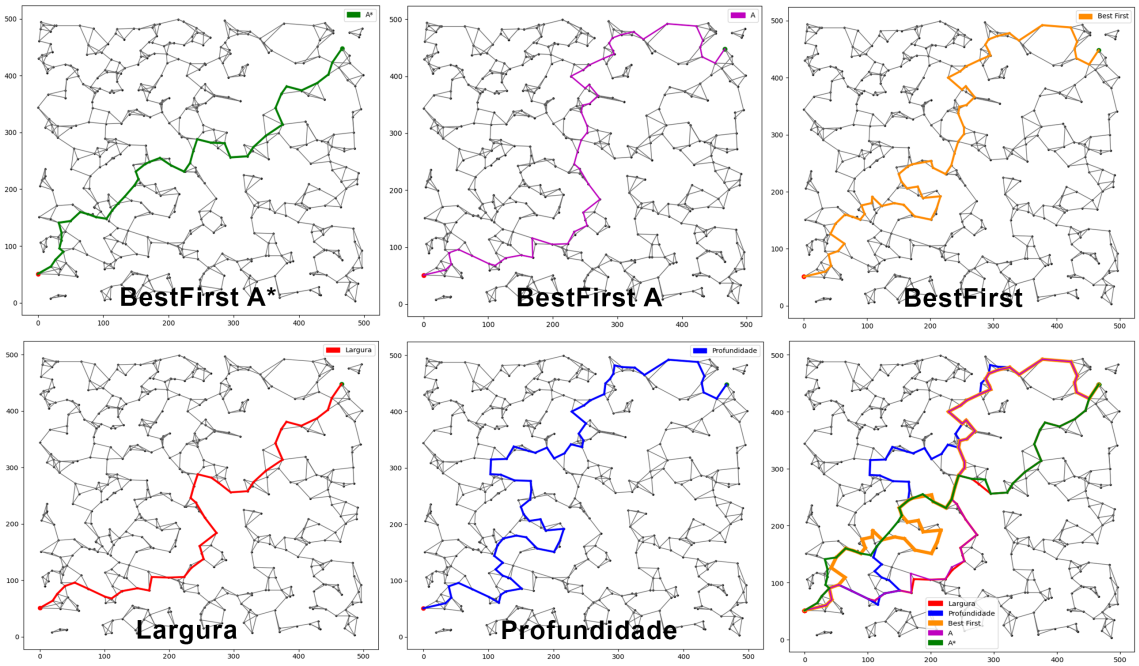


Figura 8. Comparação em grade das soluções encontradas por cada método. Fonte: Autor.