TP1 - Busca em Espaço de Estados

Felipe Cadar Chamone

September 2020

1 Introdução

Esse trabalho implementa 4 algoritmos de busca:

- *BFS* - Busca em Largura,
- *DFS* - Busca em Profundidade,
- *IDS* - Busca com Aprofundamento Iterativo,
- *A** - Algoritmo A*.

Esses algoritmos foram utilizados para ajudar o Márcio, que teve a ideia de utilizar conhecimentos da disciplina de Introdução a IA para programar um *AGV - Automated guided vehicle*. Apesar de parecer algo muito custoso para os grandes ambientes industriais, uma grande restrição do projeto é que os veículos adquiridos não podem dar mais de W passos sem passar por um ponto de localização. Esse detalhe torna o problema bem interessante pela adaptação necessária para o algoritmo *A** funcionar corretamente, e também corta muitos estados de possibilidades dos outros algoritmos, acelerando muito o processo de busca.

2 Modelagem

Os 3 primeiros algoritmos (*BFS*, *DFS* e *IDS*) foram modelados de maneira praticamente idêntica, então só especificaremos quando necessário.

2.1 Estados

Cada estado dos algoritmos *BFS*, *DFS* e *IDS* possuem 3 informações:

1. Posição: Posição atual,
2. Solução: Caminho percorrido na até esse estado,
3. Passos: Passos dados desde o último posto de localização.

Provavelmente poderíamos simplificar esses estados para remover a solução e guardar apenas os passos totais (para a saída requisitada na especificação do trabalho), ou remover a posição atual e extrair essa informação do último nó da solução, mas desta maneira temos um algoritmo mais simples de encontrar erros e fácil de visualizar (com a ajuda de bibliotecas gráficas).

O *A** precisou ser levemente adaptado. Classicamente o algoritmo possui uma lista de **abertos**, **fechados**, valores da heurística **H** e número de passos **G** desde o nó inicial. No nosso caso também precisamos salvar o número de passos dados **S** desde o último ponto de localização para atualizar nós estados que chegaram no mesmo destino com valores diferentes de **S**.

2.2 Função sucessora

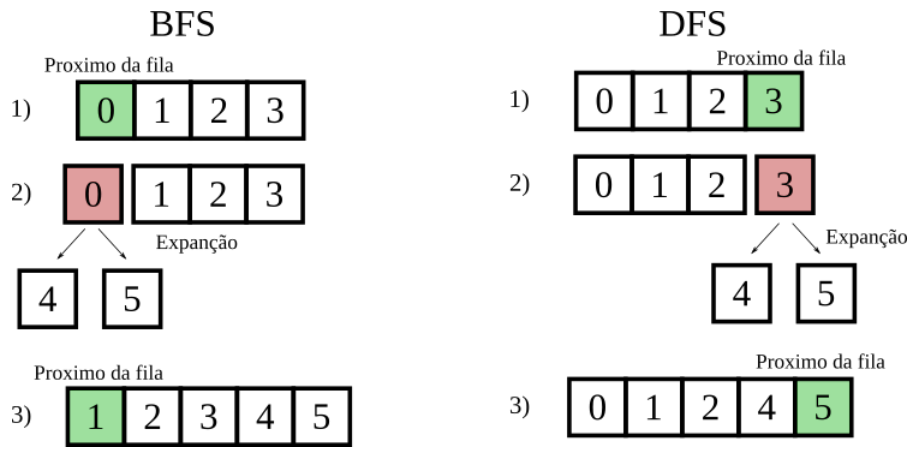


Figure 1: Princípio de iteração dos algoritmos *BFS* e *DFS*

No ***BFS***, dado que começamos com alguns estados na fila de estados, vamos expandir nossas possibilidades de soluções e as adicionar ao final da fila. Na próxima iteração selecionamos o estado que esta no incício da file, como ilustrado na esquerda da Figura 1

No ***DFS*** temos um comportamento muito semelhante, mas escolhemos o próximo nó do final da fila, como ilustrado na direita da Figura 1

No ***IDS*** seguimos o mesmo comportamento do *DFS*, com exceção que temos um limite máximo de profundidade definido, e sempre que atingimos esse limite, paramos de expandir o nó. Se necessário aumentamos esse limite em 1 e reiniciamos o algoritmo. Nesse ponto também exite muito espaço para otimização, mas não era o objetivo do trabalho ter a implementação mais eficiente possível.

No ***A**** já temos algo bem diferente. Como já conhecemos, cada nó no A* tem seu valor **H**, da heurística, e **G**, da distância percorrida, e a soma deles é o que devemos usar para ordenar a lista de **abertos** para escolher o próximo estado para explorar. Nesse caso, o algoritmo não funciona pela restrição da

quantidade de passos permitidos. Para adaptar o algoritmo devemos adicionar também o valor de **S** ao estado, que é a quantidade de passos dados desde a última estação de localização. Quando vamos expandir um estado olhando seus vizinhos, se um vizinho já foi visitado mas por esse novo caminho possui um **S** menor, atualizamos o **S** dele e o recolocamos na lista de **abertos**. Então com a lista atualizada, a ordenamos normalmente para escolher o próximo estado.

3 Experimentos

Só foram disponibilizados 3 mapas para testes, e em um deles o **W**, número de passos entre estações de localização, é pequeno demais. Para fazer os experimentos repetimos esses mesmo 3 mapas com um **W** maior, para mostrar que a quantidade de passos muda muito na performance dos algoritmos.

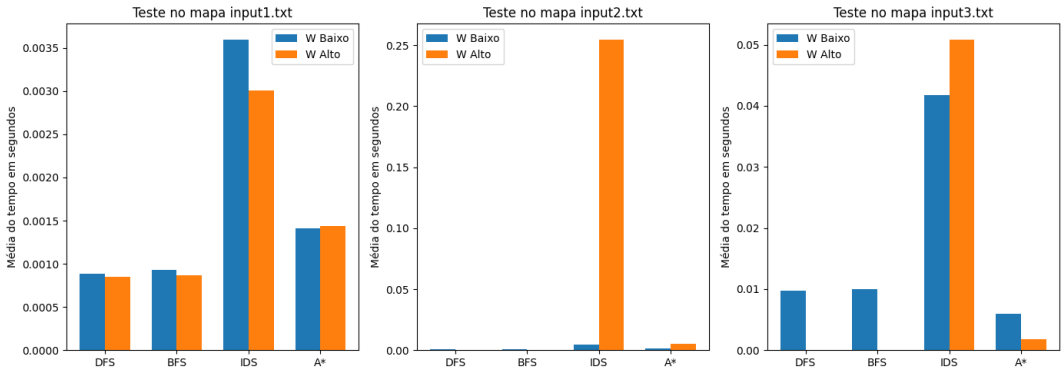


Figure 2: Comparação de tempos entre os algoritmos usando valores de W diferentes

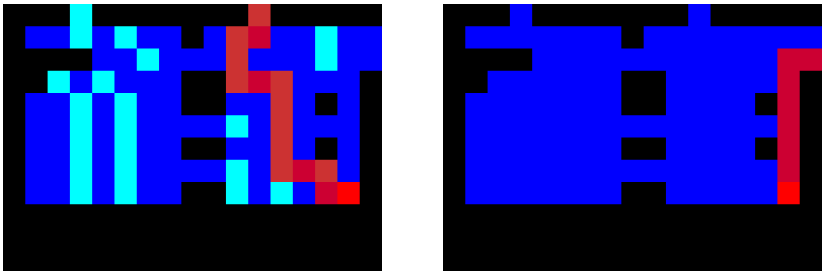
Fizemos 30 execuções para cada combinação de mapa e algoritmo e tomamos a média dos tempos de busca. O gráfico da figura 2 mostra a comparação dos algoritmos em cada mapa usando diferentes valores de **W**. Nele, quando dizemos "W baixo" nos referimos ao W original do mapa, e "W Alto" a um valor de **W** que não limite nenhuma solução.

Podemos observar que em mapas muito pequenos, como o *input1.txt* no gráfico mais a esquerda, os valores para diferentes **W**s são quase idênticos. Isso acontece pois não há como gerar soluções grandes o suficiente mesmo que o número de passos seja ilimitado, então os algoritmos praticamente não variam. Nesse gráfico também podemos ver que o processo de ordenação do ***A**** e as iterações repetidas do ***IDS*** acabam os deixando mais lentos, no geral, do que os competidores menos sofisticados.

No gráfico do meio o valor original de **W** não era suficiente para produzir uma solução válida, então os algoritmos acabam explorando ainda menos soluções e o *overhead* do ***A**** e do ***IDS*** ficam aparentes. Quando aumentamos o valor

de **W** começamos a observar um comportamento interessante, agora o ***DFS*** e o ***BFS*** demoraram tanto que não conseguimos colocar no gráfico, o ***IDS*** acabaou achando uma solução em tempo hábil por não precisar explorar todas as profundidades máximas de soluções , e o ***A**** foi o melhor com muita facilidade pela sua heurística de direção.

No gráfico mais a direita já temos um mapa um pouco maior. Novamente, com W baixo, os algoritmos ***DFS*** e o ***BFS*** vão bem e chegam bem próximos do tempo do ***A****, mesmo assim o mapa é grande suficiente para que o ***A**** saia na vantagem. Quando usamos o W alto, os algoritmos ***DFS*** e o ***BFS*** novamente não conseguem terminar em tempo aceitável e o ***A**** fica mais rápido ainda por conseguir traça um caminho mais curto e direto sem as restrições de passos entre estações de localização, como podemos observar na figura 3.



(a) Melhor solução **com** restrição de passos

(b) Melhor solução **sem** restrição de passos

Figure 3: Comparação entre soluções com e sem restrição de passos. Blocos azul claros representação estações de localização, vermelhos representam o caminho da solução, azul escuro representa blocos caminháveis e preto blocos não caminháveis