Introdução à Inteligência Artificial

Trabalho Prático I - 2024/2

Igor Lacerda Faria da Silva

1. Introdução

O Trabalho Prático I de Introdução à Inteligência Artificial consistiu na implementação (em C++) de 5 algoritmos para cálculo de distância (exploração de espaço de busca) em mapas de um jogo simples. Os 5 algoritmos foram: *Breadth First Search* (BFS), *Iterative Deepening Search* (IDS), *Uniform Cost Search* (UCS), Busca Gulosa (*Greedy*) e A* (A-Star ou A-Estrela). A função heurística escolhida para os algoritmos que dependem dela (*Greedy* e A*) foi a **distância de Manhattan**. O repositório deste trabalho pode ser encontrado neste link.

A documentação está divida da seguinte maneira: esta breve introdução descreve o que foi realizado, a Seção 2 descreve a modelagem do programa, entrando nos detalhes dos algoritmos e estruturas de dados; a Seção 3, por sua vez, foca na heurística utilizada e, por fim, a Seção 4 é uma análise experimental dos diversos algoritmos implementados.

2. Algoritmos, Estruturas de Dados e Modelagem

Sem dúvidas, a principal estrutura de dados do trabalho foi uma matriz de `double`, com os pesos das posições do mapa. Outras estruturas são específicas dos algoritmos: a maioria dos algoritmos usa um *heap* (`priority_queue` do C++) para controlar qual será o próximo estado da busca, exceto pela BFS, que uma fila (`queue`) e a IDS, que usa uma pilha (`stack`). No mais, as estruturas restantes são estruturas padrões de C++, como vetores (`vector`) e pares ordenados (`pairs`).

2.1. Formulação do Problema

Os 5 componentes da formulação do problema foram:

1. **Estados**: posições na matriz, com os peso associados
2. **Estado Inicial**: posição inicial, da entrada
3. **Sequência de Ações**: subir, descer, ir para esquerda ou para direita
4. **Teste do Objetivo**: posição final atingida
5. **Custo da Solução**: soma dos custos pelo qual a busca passou por

A função sucessora recebe uma das ações e faz o movimento da posição atual para a nova posição correspondente.

2.2. Algoritmos

A seguir, uma breve descrição das principais diferenças entre os algoritmos, baseada na descrição de cada um. A BFS é usada como base e os outros algoritmos são vistos como "incrementos".

2.2.1. BFS

A BFS é um algoritmo relativamente simples. Dado um nó inicial, sua vizinhança é colocada na fila (primeiro a entrar é o primeiro a sair) e os nós são visitados na ordem em que foram inseridos, recursivamente. Obviamente, nós inválidos ou já visitados são excluídos da busca.

2.2.2. IDS

A IDS é uma DFS aplicada repetida diversas vezes, para profundidades cada vez maiores, até que o objetivo seja alcançado, ou o limite de profundidade seja atingido. A DFS é uma busca muito semelhante à BFS, a única diferença é que é usada uma pilha para se gerenciar os nós (ou seja, o primeiro a entrar é o último a sair). Além disso, como a busca da IDS tem a profundidade limitada, é necessário checar se a profundidade máxima daquela iteração foi atingida.

2.2.3. UCS

A UCS é também muito similar aos algoritmos anteriores. A diferença aqui é que usado um *heap*, que prioriza o nó de menor custo até então. Para que isso possa ser feito de maneira eficiente, é importante manter em memória uma matriz com o custo de cada posição.

2.2.4. *Greedy*

A busca gulosa é semelhante à UCS, pois também usa um *heap* (apesar de não precisar de matriz de custos). A ideia é seguir pelo "melhor" caminho, mas usando uma aproximação definida por uma heurística (mais detalhes na Seção 3). Na busca gulosa, é explorado o "melhor neste momento", em detrimento do "melhor geral".

2.2.5. A*

Por fim, o A-Estrela é uma combinação da busca gulosa com a UCS, que considera a soma dos custos gerados nestes algoritmos. A ideia é fazer uma busca que sempre é ótima, como na UCS, mas que é rápida porque é guiada pela heurística.

2.3. Outras decisões de implementação

Presume-se que sempre existe um caminho de A (início) para B (objetivo). Caso algum dos nós tenha custo infinito, o programa é encerrado. Na implementação da IDS, a profundidade máxima foi definida como $W * H$.

3. Heurísticas

A heurística escolhida foi a distancia de Manhattan. Ela é bem simples, pois é a soma soma da diferença absoluta das coordenadas. É por causa dessa simplicidade que ela é admissível: como o custo mínimo (por nó) é 1, o melhor custo é, na situação ideal, é justamente a distancia de Manhattan: deslocar-se $|x_i - x_f|$ unidades na horizontal e $|y_i - y_f|$ na vertical. É claro que esse é o cenário ideal, em que não existem impedimentos no meio do caminho e existe um caminho composto exclusivamente por nós de valor 1. De qualquer modo, a heurística nunca é mais cara do que o custo real para o objetivo e, portanto, pode-se dizer que ela é admissível.

4. Análise Quantitativa e Discussão

Os 2 mapas (`floresta.map` e `cidade.map`) de exemplo disponibilizados via *Moodle* foram usados para se realizar um *benchmarking* dos algoritmos. De antemão, vale ressaltar que a ordem dos operadores é um fator muito importante no tempo de execução, especialmente da IDS. Com isso em mente, as comparações vão incluir os casos simétricos – ou seja, invertendo o nó de início e o nó objetivo. Além disso, na execução dos experimentos, a `-O3`, de otimização do compilador, foi habilitada.

4.1. Tempo de Execução

Para avaliar o tempo de execução, foram propostas 5 pares de coordenadas em cada mapa. Como são dois mapas e a simetria é levada em conta, foram realizados 20 execuções por mapa.

4.1.1. Cidade

Como este mapa possui menos estados inválidos, o início do mapa foi fixado em (0, 0) e, os destinos, foram da forma $(2^i - 1, 2^i - 1)$, com i variando de 4 até o limite do mapa, que gera a coordenada (255, 255). Para estes casos, os algoritmos *Greedy* e BFS são extremamente rápidos, sendo que, boa parte do tempo de execução está associada a algum *overhead* e não ao tempo de processamento dos estados propriamente ditos. A segunda "classe" de algoritmos é composta pelos algoritmos UCS e A*: na teoria, o A* deveria ser mais rápido, mas para os casos considerados nesta seção, não houve diferença significativa: ambos ficaram na faixa de dezenas de milissegundos para a maior instância, não ultrapassando 60ms.

O único algoritmo que apresentou um comportamento interessante foi a IDS. Na Figura 1, o caso base é até bem comportado, mas vale destacar que ele demorou muito mais do que os outros algoritmos: para a maior instância, seu tempo de execução ficou próximo de meio segundo. O caso em que a execução termina em (0, 0) é ainda mais intrigante, pois, além de demorar (ainda) mais, a entrada "intermediária" teve um tempo de execução muito semelhante ao caso mais distante, ambos beirando os 2.5s. O porquê disso será melhor discutido na Seção 4.2.

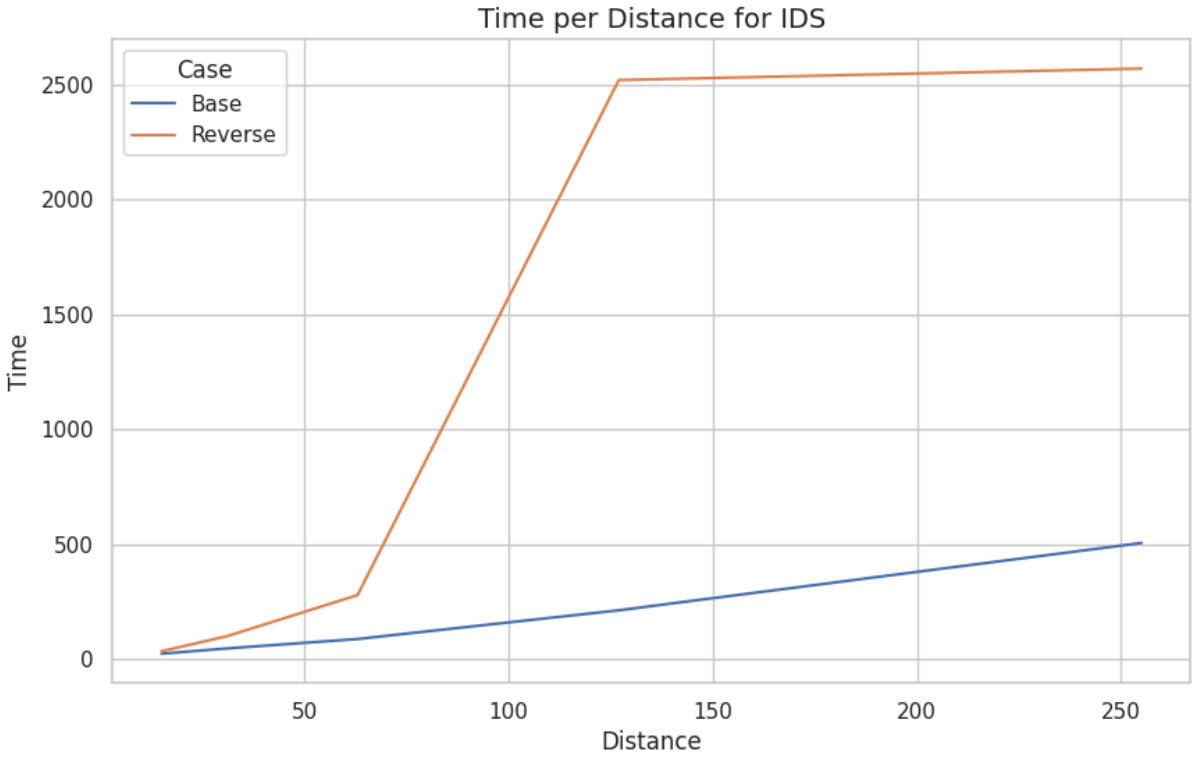

Figura 1: Tempo de execução da IDS no mapa Cidade.

De forma geral, não se pode dizer que o tempo escala proporcionalmente à distância de Manhattan. Esse efeito foi observado, em menor escala, com os algoritmos rápidos também.

4.1.2. Floresta

A Floresta é um mapa mais denso, então não é possível selecionar os pontos usando-se a "estratégia de base 2" da seção anterior. No entanto, ela foi usada de base para se definir o ponto de início (17, 0), que é o canto superior mais à esquerda. Os pontos de destino escolhidos foram: (17, 16), (35, 31), (102, 63), (132, 127) e (208, 255). A escolha se deu a partir do deslocamento dos pontos da estratégia anterior para posições próximas que não estivesse bloqueadas.

No geral, o comportamento foi bem semelhante ao outro mapa, mas a velocidade dos algoritmos foi ainda mais acentuada devido ao espaço de busca reduzido. Foi possível separar os algoritmos em dois grupos: os ultrarrápidos e a IDS. De fato, o tempo dos ultrarrápidos não excedeu 10ms nas instâncias escolhidas. A IDS, por outro lado, apresentou o comportamento que pode ser visto na Figura 2.

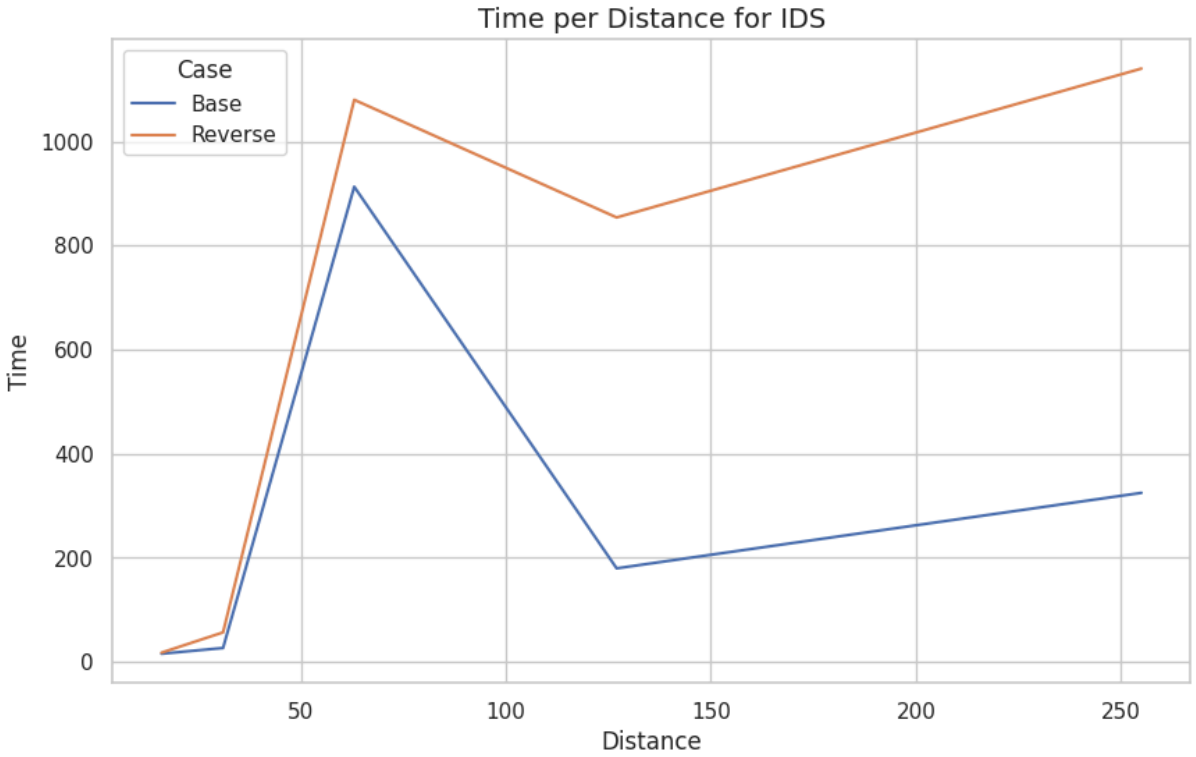

Figura 2: Tempo de execução da IDS no mapa Floresta.

Como as coordenadas x e y dos pontos são diferentes, foi adotado o y como base para a distância (para aproximar consistência com o gráfico anterior, da Figura 1). Devido à alta quantidade de caminhos bloqueados neste mapa, a execução de destaque foi a de distância 63 (comportamento consistente em ambas direções), com tempo de execução elevado para sua distância. De fato, neste mapa também não parece existir relação clara entre a distância e o tempo de execução.

Há, ainda, a questão do *overhead*, que pode ter influenciado na análise de tempo como um todo. Como o programa é rápido, a execução em ambos os mapas está sujeita a grandes variações considerando fatores externos, como o *load* do sistema operacional. Uma análise mais aprofundada exigiria mapas maiores.

4.2. Número de Estados Expandidos

O número de estados expandidos não pode ser obtido diretamente da saída esperada do programa. É possível *plotar* o caminho, mas isso limita significativamente o poder de análise. Por exemplo, usando apenas a comparação do caminho, os algoritmos UCS e A* são indistinguíveis, pois ambos sempre são ótimos. Tendo isso em vista, essa seção possui uma breve análise preliminar, que foca em algumas execuções específicas, comparando os caminhos.

Depois, é introduzida uma variação do programa, que rastreia quais estados foram expandidos, permitindo extrair sua totalidade. Como essa operação introduz um certo *overhead* e é incompatível com a saída especificada para o programa, essa variação foi considerada exclusiva desta análise.

4.2.1. Caminhos

Para analisar os caminhos, foram selecionadas duas execuções da Seção 4.1: $(0, 0) \rightarrow (127, 127)$ e o caminho reverso, no mapa Cidade.

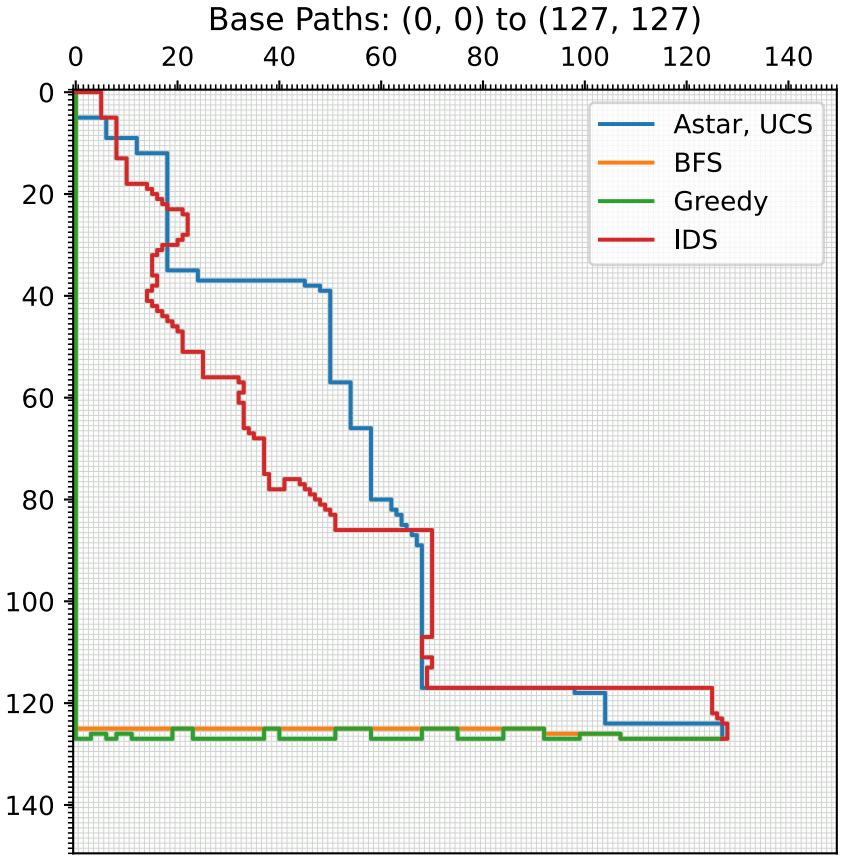

Figura 3: Caminhos, iniciando em (0, 0) e terminando em (127, 127).

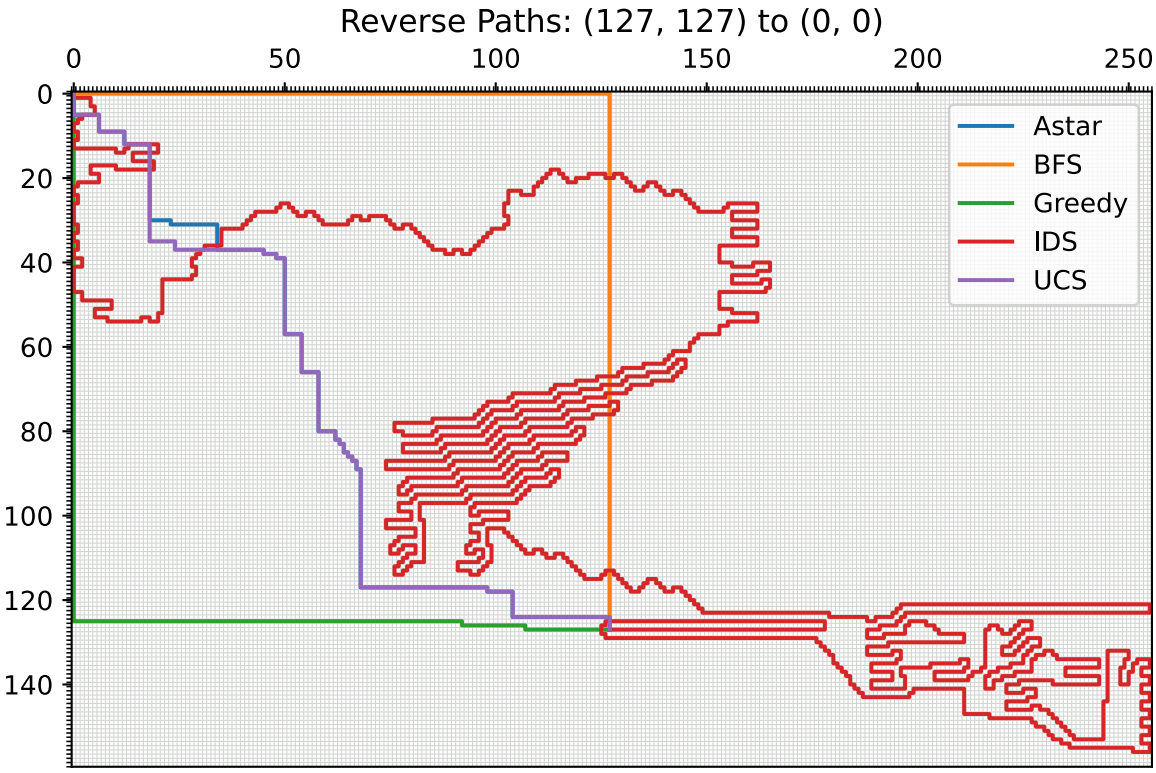

Figura 4: Caminhos, iniciando em (127, 127) e terminando em (0, 0).

A grande diferença das figuras é que, a IDS, no caminhamento reverso, explora muito mais estados devido à ordem das ações. Isso acontece porque a configuração padrão favorece posições que começam no canto superior esquerdo, pois a região explorada inicialmente é "para baixo e para a direita". É por isso que o tempo de execução da IDS foi tão elevado na seção Seção 4.1.

A ordem das ações acaba impactando em alguns dos outros algoritmos também, mas em menor grau. Por exemplo, a BFS fez um caminho quase perfeitamente simétrico. Outro ponto interessante é que os algoritmos ótimos não fizeram exatamente o mesmo caminho na "volta", apesar de terem feito na "ida". Isso acontece porque existe mais de um caminho ótimo e, devido à ordem das operações, houve um trecho (próximo a $P = (40, 40)$) em que o A* e a UCS divergiram.

4.2.2. Estados

Para finalizar, a análise de estados explorados. Como mencionado no início da seção, foi feita uma modificação no programa, para que ele pudesse produzir dados mais relevantes para esta análise. Em suma, a modificação troca o custo da solução pela quantidade de estados explorados e o caminho gerado por uma lista com todos os nós visitados. Novamente, a IDS introduz certa dificuldade: o mesmo nó é visitado diversas vezes, o que contribui consideravelmente para seu tempo de execução. No entanto, a abordagem de somente contar os nós visitados não é capaz de detectar isso.

Além disso, em outros algoritmos, como a UCS e o A*, também podem acontecer repetições. No entanto, nesta seção preferiu-se apenas comparar os nós visitados, porque algumas conclusões importantes já foram deduzidas nas outras análises: independentemente das repetições, a IDS visita muito mais nós (ver Figura 4). De fato, as questões ainda não respondidas podem ser exploradas justamente a partir da exploração da quantidade "líquida" de estados visitados.

Por exemplo: o quão melhor é o A* em relação à UCS? A partir da discussão na Seção 4.2.1, não era possível extrair uma resposta. Para responder a essa pergunta, o experimento da Seção 4.2.1 foi repetido com a modificação proposta aqui. O resultado para os caminhos "base" é exibido na Figura 5. Em termos absolutos, o A* visitou 8078 estados, enquanto a UCS visitou um total de 23367 estados. Ou seja, nesse caso específico, a busca do A* visitou aproximadamente $\frac{1}{3}$ dos estados da busca da UCS.

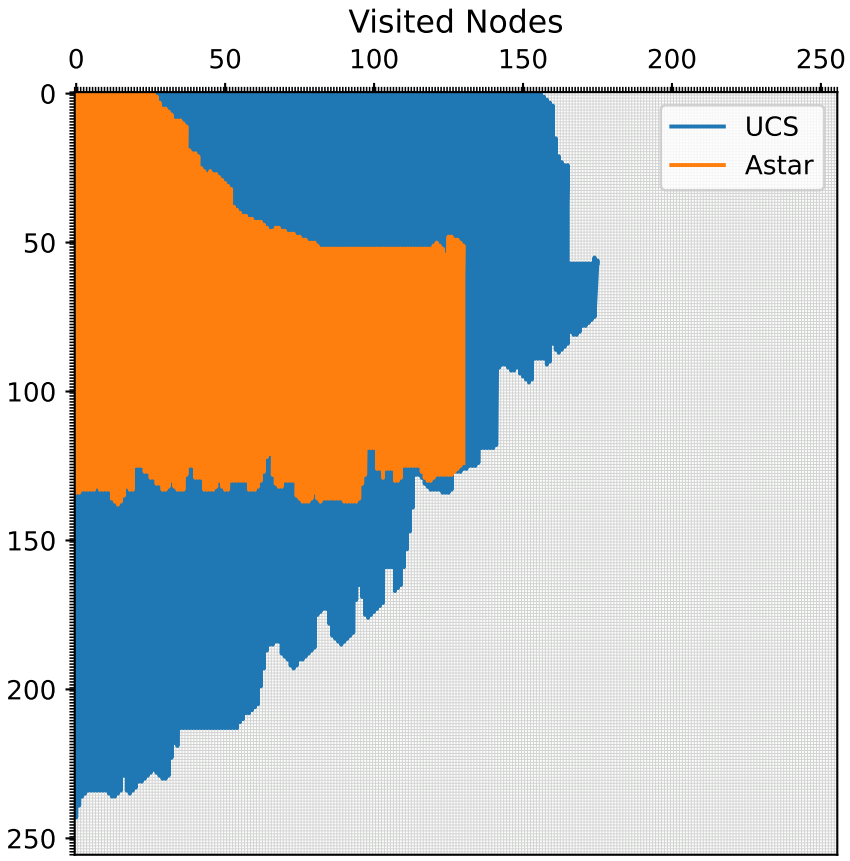

Figura 5: Nós visitados, iniciando em (0, 0) e terminando em (127, 127).

De forma mais geral, os experimentos da seção de tempo (Seção 4.1) foram repetidos e os gráficos, com a contagem de estados visitados, podem ser encontrados nas figuras a seguir.

Primeiramente, no mapa Cidade, fica evidente a eficiência do *Greedy*, que visita pouquíssimos estados. Na maioria dos casos, o A* deixa a busca bem mais rápida que o UCS, como avaliado na Figura 5. No entanto, para as instâncias maiores, o ganho é pequeno. Não foi encontrada uma explicação para o fato de a BFS e a UCS possuírem curvas tão semelhantes. Outro caso interessante foi a IDS ter uma redução tão drástica na maior busca (para o caminho de "ida").

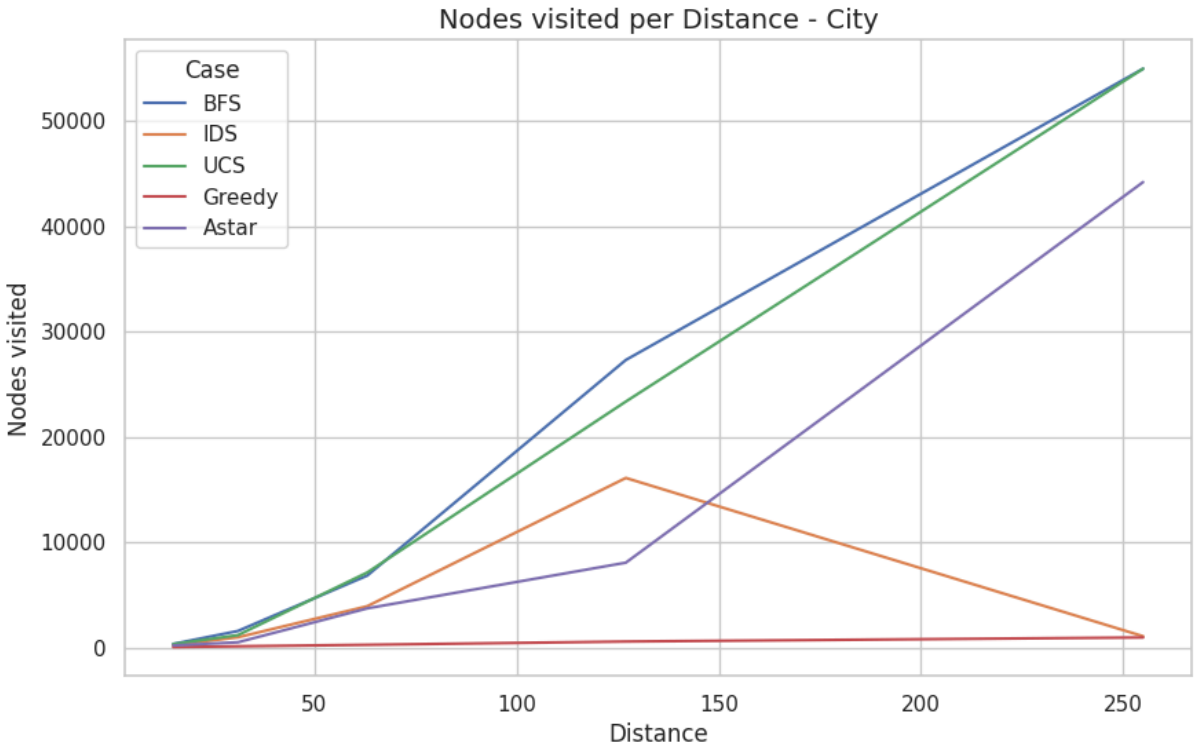

Figura 6: Nós visitados, no mapa Cidade.

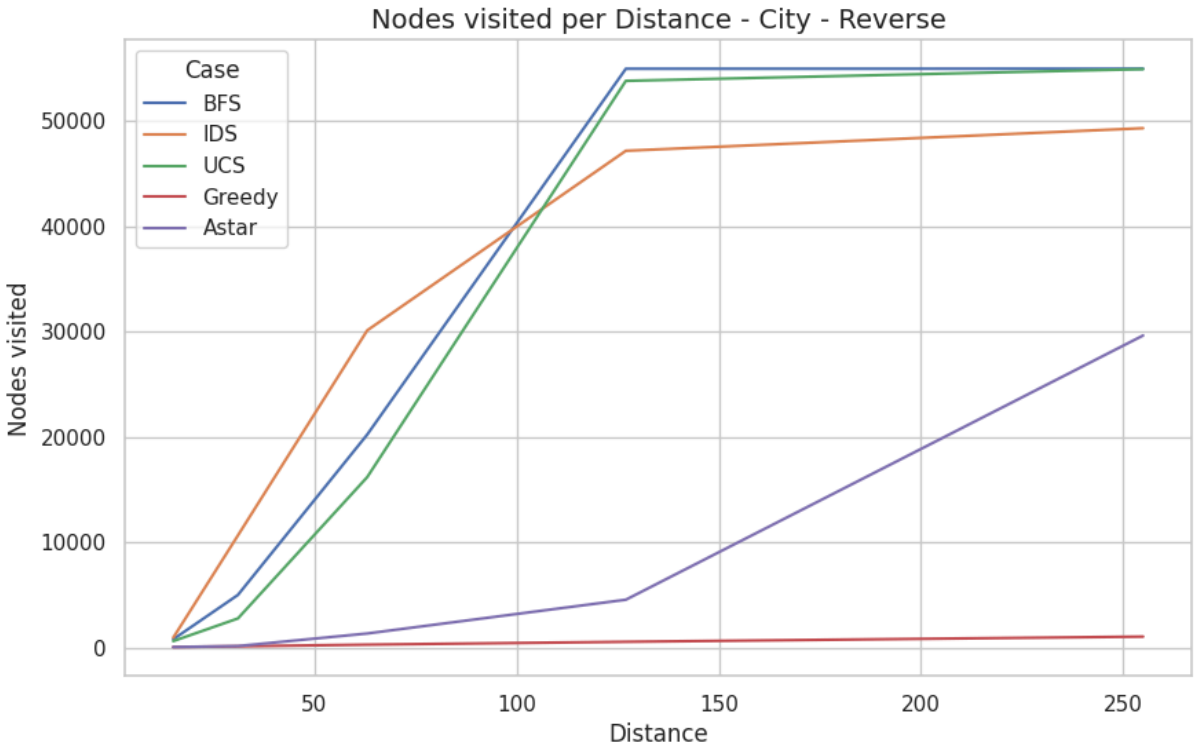

Figura 7: Nós visitados, no mapa Cidade, caminho reverso.

Comparando-se ambos os gráficos, é possível observar que quase todos os algoritmos "pioram" (visitam mais estados) na busca reversa, exceto quando ela é colossal (distância 255), momento em que já não há mais tanta diferença. O mapa Floresta é discutido a seguir.

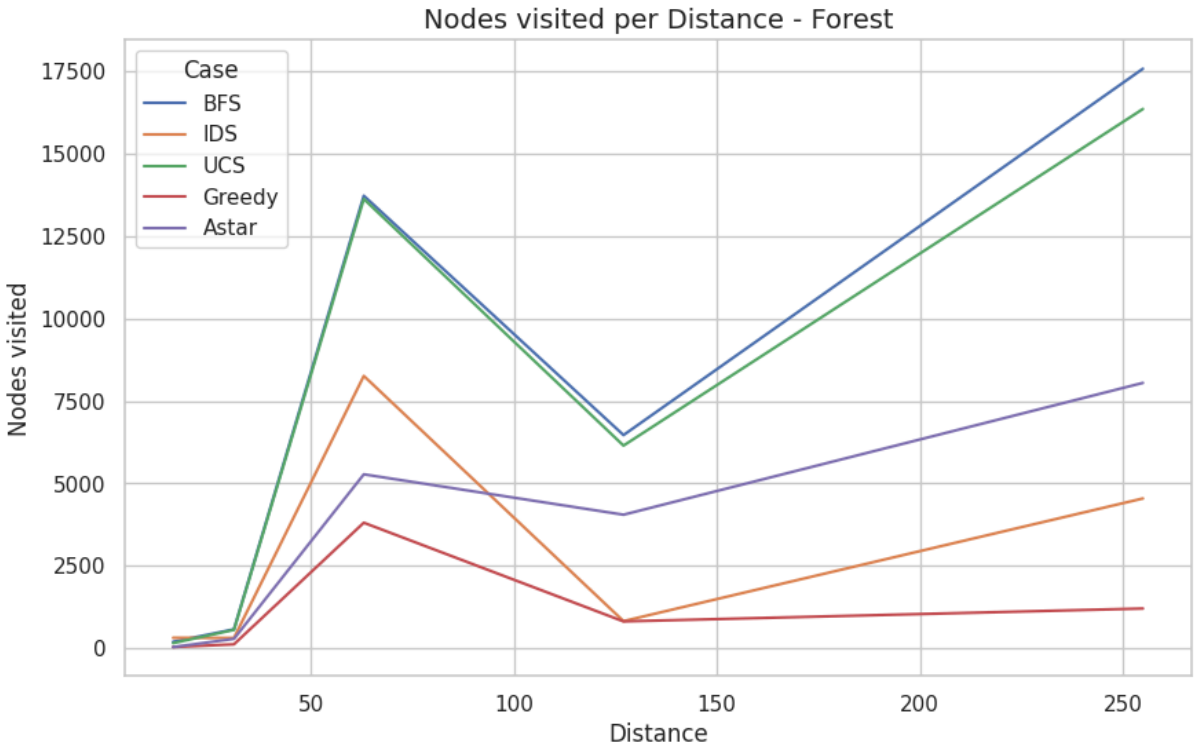

Figura 8: Nós visitados, no mapa Floresta.

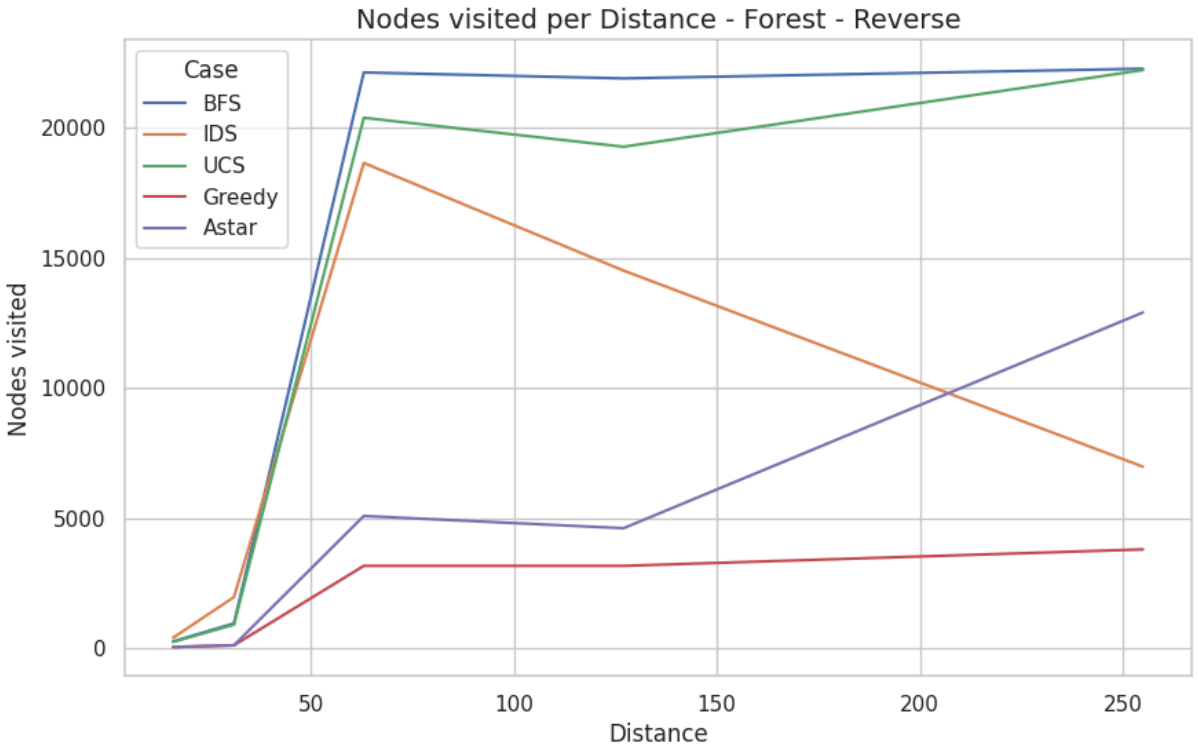

Figura 9: Nós visitados, no mapa Floresta, caminho reverso.

Fica clara a relação entre o tempo de execução e o número de estados expandidos: na Figura 2, observou-se que para a execução de distância 63, o tempo de execução foi particularmente elevado. Nas figuras acima, é possível notar que neste caso, a maioria dos algoritmos explora "desproporcionalmente" o espaço de busca (especialmente na "volta").

Tal qual no outro mapa, UCS e BFS são muito próximas em termos de estados explorados. Vale notar que os algoritmos tem um comportamento consistente no caso base e que a análise de mais pontos poderia ser interessante suavizar o formato "ondular" do gráfico. Para o caminho reverso é mais complicado de se extrair informações: tanto a UCS como a BFS atingem um pico logo no início, e ficam por volta dele, mas cada um dos outros algoritmos é diferente. Para o *Greedy*, por exemplo, o maior "contorno" que o algoritmo teve que fazer, provavelmente estava na primeira metade da busca. Já o A* tem um comportamento semelhante ao outro mapa, com rendimentos decrescentes.