Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

DCC642 IIA — TP 1

8-puzzle solver

Alexander Thomas Mol Holmquist

31 de maio de 2022

1 Introdução

Este trabalho apresenta uma solução para uma instância do problema N-puzzle: o 8-puzzle, e provê diversas análises do desempenho de diferentes algoritmos para resolvê-lo. Apesar do problema geral do N-puzzle ser NP-difícil, o 8-puzzle é tratável. Nossa solução inclui os seguintes algoritmos:

- Breadth-first search (BFS)
- Iterative-deepening search (IDS)
- Uniform-cost search
- A* search
- Greedy best-first search
- Hill Climbing

O *BFS* procura o estado-objetivo um nível por vês. O *IDS* procura o estado-objetivo realizando uma Depth-first Search com limite fixo, e incrementando este limite para alcançar estados cada vez mais profundos. O *Uniform-cost search* foi implementado com o conhecido algoritmo de *Dijkstra*, que expande sempre o estado de menor custo até o estado inicial – neste caso, o custo é a profundidade. *Greedy best-first search* é parecido com Dijkstra, mas sempre expande o estado de menor custo de acordo com uma heurística. *A* search* é uma combinação do Dijkstra e Greedy best-first: sempre se expande o estado de menor custo somado *profundidade + heurística*. *Hill Climbing* é um algoritmo de busca local, e portanto esquece do seu histórico de busca: só é guardado o último estado, que é expandido na direção que minimiza uma heurística.

A Tabela 1 indica, para cada algoritmo, se ele é *ótimo* (sempre encontra o melhor caminho até a solução), e se ele é *completo*.

Algoritmo	**Completo**	**Ótimo**
BFS	T	F
IDS	T	F
Dijkstra	T	T
Greedy	F	F
A*	T	T
Hill Climbing	F	F

Tabela 1: Comparação de completude e otimalidade entre os algoritmos. *T* significa sim, e *F* significa não.

2 Implementação

A seguir, apresentamos as principais estruturas de dados da implementação, assim como as heurísticas utilizadas para os algoritmos de busca com informação e busca local. O objetivo desta seção é facilitar o entendimento do código para o leitor.

2.1 Estruturas de dados

2.1.1 Explorer

Explorer, cuja implementação pode ser encontrada no arquivo `solver/_internal/explorer/explorer.py`, é a principal classe do programa. Esta classe é utilizada por todos os algoritmos. Ela mantém uma tabela hash com todos os estados já visitados. O hash utilizado na tabela é simplesmente a concatenação dos dígitos de um dado estado. Por exemplo, para o estado $[1, 2, 3, 4, 5, 6, 7, 8, 0]$, o hash é 1234567890. O lugar de um estado na árvore de busca é determinado por duas informações: o hash de seu pai, e o movimento (transição de estado) que leva do pai a ele.

Explorer contém uma função, `branch`, que usuários podem utilizar para expandir um certo estado. O que faz o *Explorer* versátil é o fato de que a ordem de expansão dos estados é deixada por conta do usuário da classe. Essa versatilidade permitiu que todos os algoritmos fizessem uso de *Explorer*, e o código ficou bastante enxuto.

2.1.2 Outras estruturas de dados

Cada algoritmo precisa de uma ordem específica de expansão dos estados. É conhecido, por exemplo, que se utiliza uma fila simples para implementar BFS, enquanto Dijkstra precisa de uma fila de prioridade. Estas e outras estruturas de dados foram implementadas minimalisticamente. O código pode ser encontrado na pasta `solver/_internal/datastruct`.

2.1.3 Observações

Para o algoritmo IDS, utilizamos um limite de aprofundamento de 3. Isto quer dizer que o algoritmo é permitido avançar 3 níveis de profundidade a cada iteração.

2.2 Heurísticas utilizadas

Utilizamos duas heurísticas: *Manhattan Distance* e *Displaced Pieces*. Ambas são admissíveis, como é bem conhecido. *Manhattan Distance* calcula o número de movimentos verticais e horizontais mínimo entre o estado atual e a solução, considerando que se pode movimentar entre células que não contém o espaço vazio do puzzle. Por exemplo, para o estado abaixo, o valor desta heurística seria 16.

$$\begin{matrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{matrix}$$

Displaced Pieces conta o número de peças que não está no devido lugar. Para o estado acima, esta heurística teria o valor 9. É fácil mostrar que *Manhattan Distance* domina *Displaced Pieces*.

3 Resultados

3.1 Desempenho

A seguir, apresentamos em forma de gráfico alguns resultados que nos permite comparar o desempenho dos vários algoritmos. Utilizamos os primeiros 25 exemplos de quebra-cabeças

dados na especificação do trabalho. O eixo X sempre é "Puzzle difficulty", que indica o número mínimo de passos para resolver o puzzle. O algoritmo de Hill Climbing não encontrou solução para 20 das 25 instâncias. *B* indica BFS, *I* indica IDS, *U* indica Dijkstra, *A* indica A*, *G* indica Greedy best-first, *H* indica Hill Climbing.

A Figura 1 mostra o tempo que cada algoritmo levou para executar, **em segundos**. O algoritmo G parece ser o mais eficiente neste quesito, mas como se pode ver na Figura 3, muitas de suas soluções são de qualidade ruim. O algoritmo A tem a segunda melhor performance, em termos de tempo de execução.

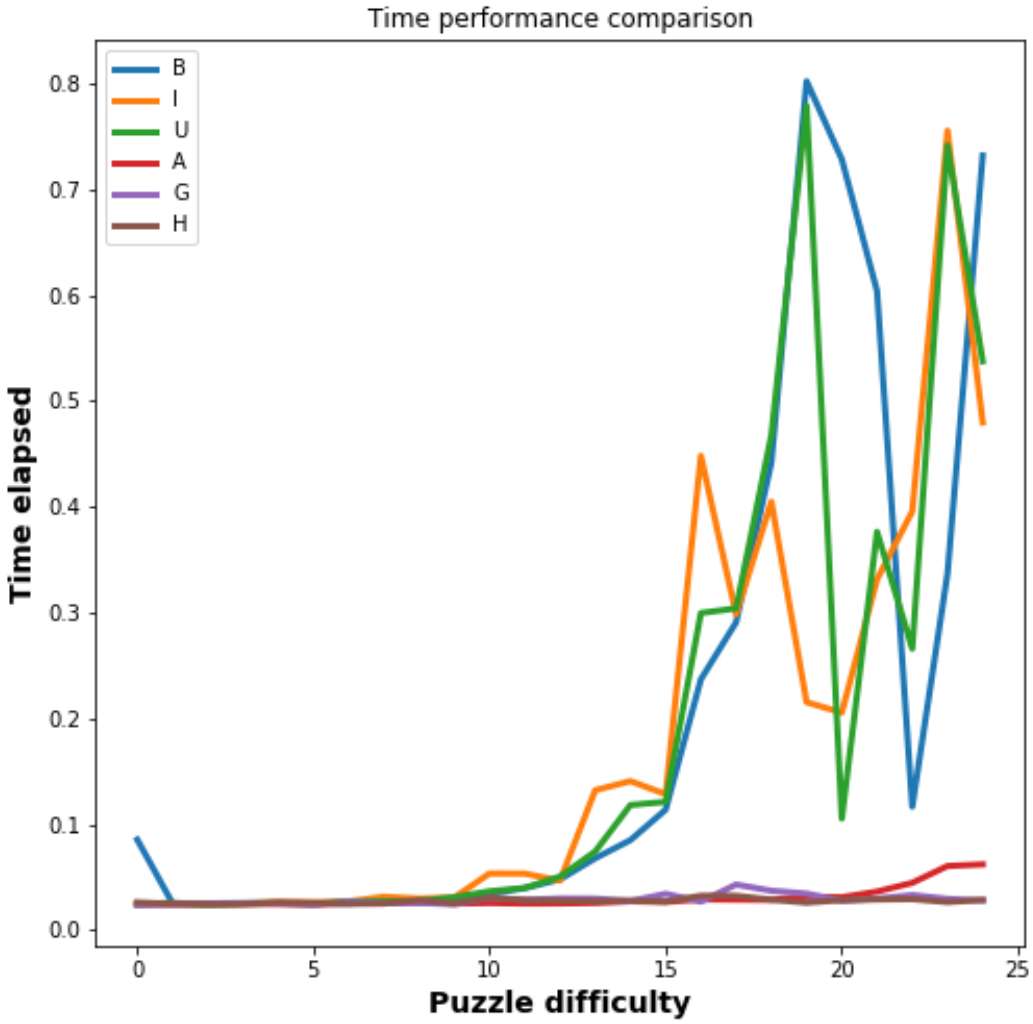

Figura 1: Tempo de execução para cada algoritmo.

A Figura 2 mostra o número de estados expandidos antes de terminar a execução, para cada algoritmo. É interessante notar como o algoritmo IDS é mais instável que BFS e Dijkstra. O comportamento das curvas parece ser exponencial sobre a dificuldade do puzzle. O algoritmo G é o que termina expandindo menos estados (novamente, como veremos na Figura 3, G não fornece boas soluções). O algoritmo A termina expandindo muito menos estados que B, I, e U.

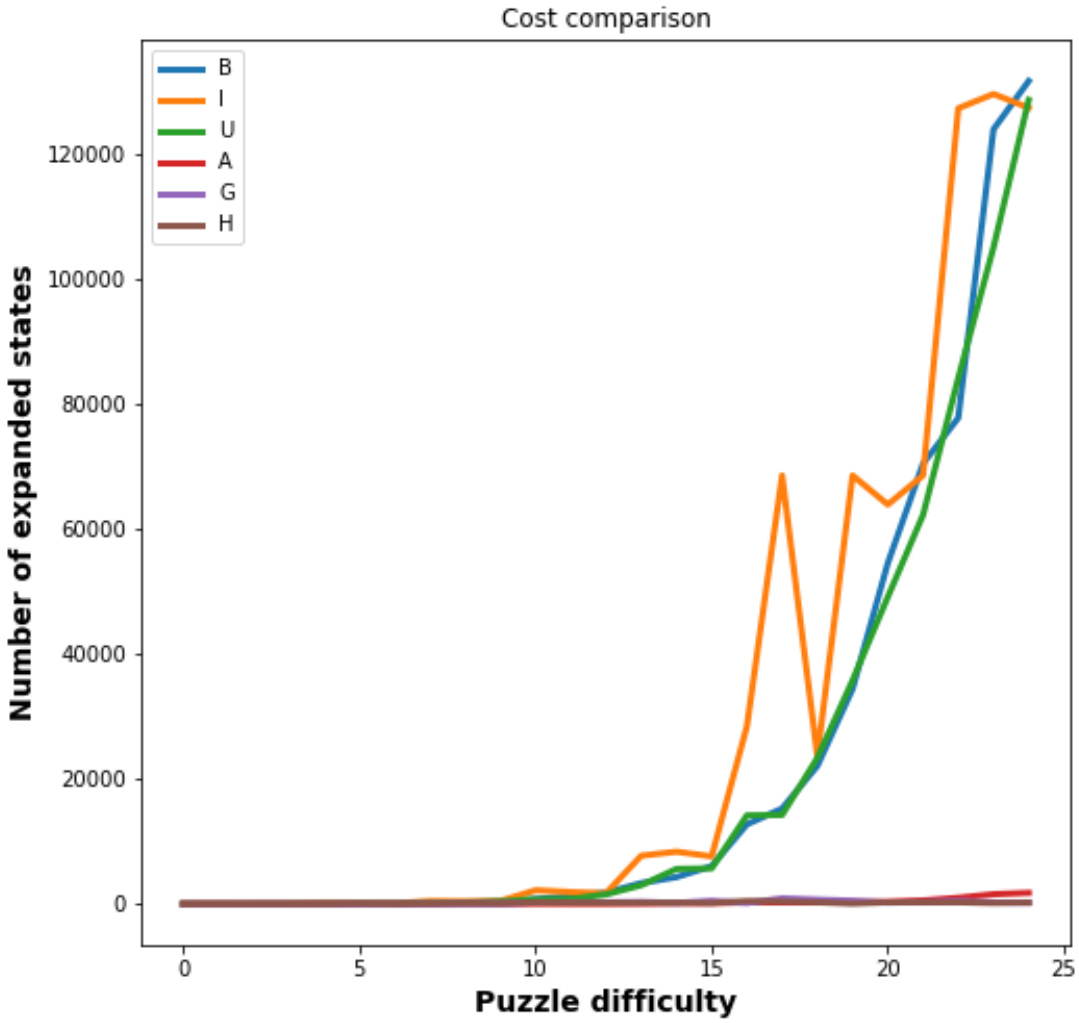

Figura 2: Número de estados expandidos para cada algoritmo.

A Figura 3 mostra o número de passos até o objetivo na solução retornada por cada algoritmo. Note que somente A e U são algoritmos ótimos, mas B (BFS) se mostrou eficaz em encontrar as melhores soluções para todas as instâncias. O algoritmo I não encontrou solução ótima para o quebra-cabeça cuja solução ótima tem tamanho 17. Talvez o mais interessante desta figura é o comportamento do algoritmo G (Greedy best-first). Por ser um algoritmo que se baseia tão somente na heurística fornecida, ele tem capacidade de encontrar soluções muito rapidamente, como foi mostrado nas figuras anteriores. Contudo, esta figura prova um contraponto: suas soluções são piores que o ótimo para todas as instâncias a partir da dificuldade *10*.

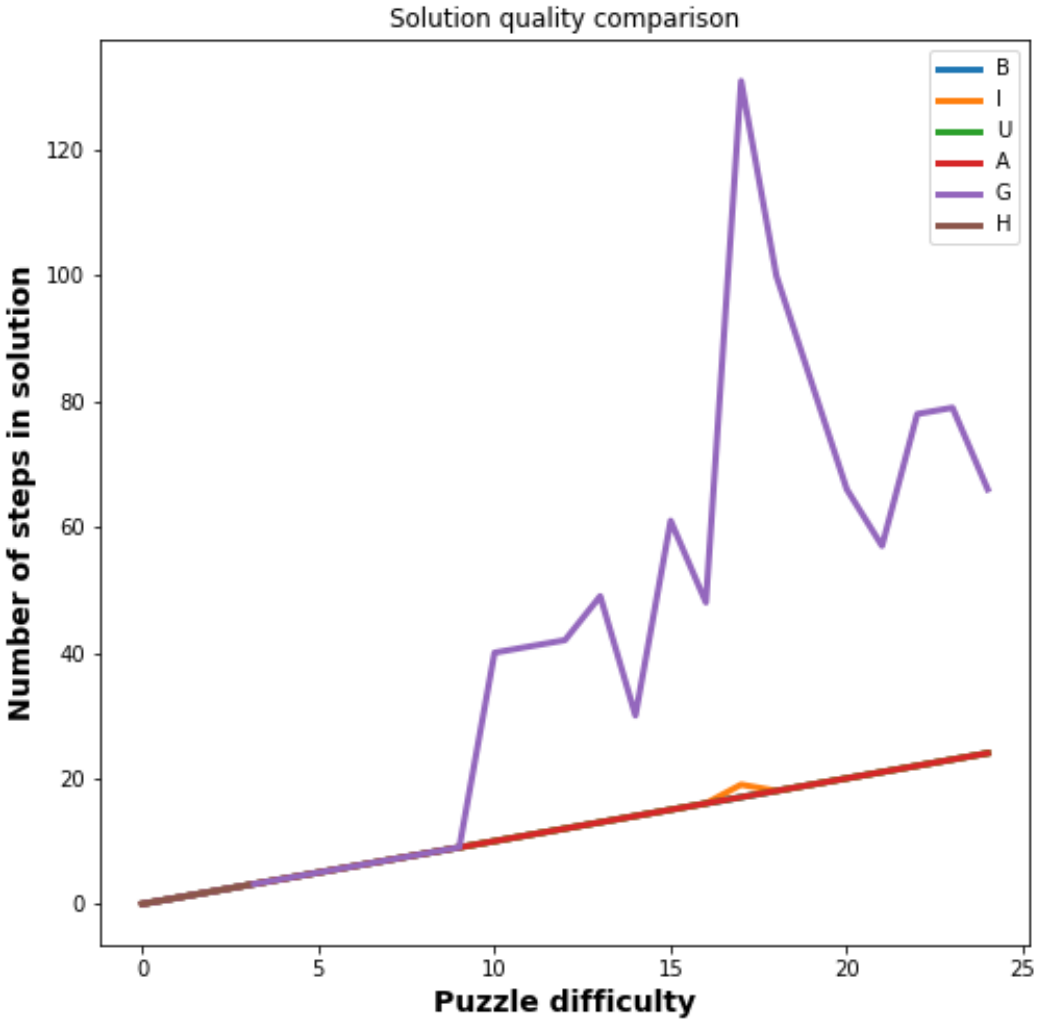

Figura 3: Número de passos até objetivo, para cada algoritmo.

3.2 Comparação entre heurísticas

Também produzimos alguns gráficos interessantes comparando as duas heurísticas que utilizamos: *Manhattan Distance* e *Displaced Pieces*. Veja Seção 2.2 para uma explicação de cada uma. Nos gráficos a seguir, o caracter de prefixo denota o algoritmo, e o caracter de sufixo denota a heurística. Por exemplo, *AM* é A* + *Manhattan Distance*, *GD* é Greedy best-first + *Displaced Pieces*.

Existem pelo menos três pontos dignos de nota nestes gráficos. Primeiro, o desempenho com *Manhattan Distance* parece ser consideravelmente melhor para ambos os algoritmos. Segundo, o desempenho de AD é péssimo. Finalmente, a Figura 6 mostra, mais uma vez, que o algoritmo Greedy best-first encontra soluções ruins, em geral.

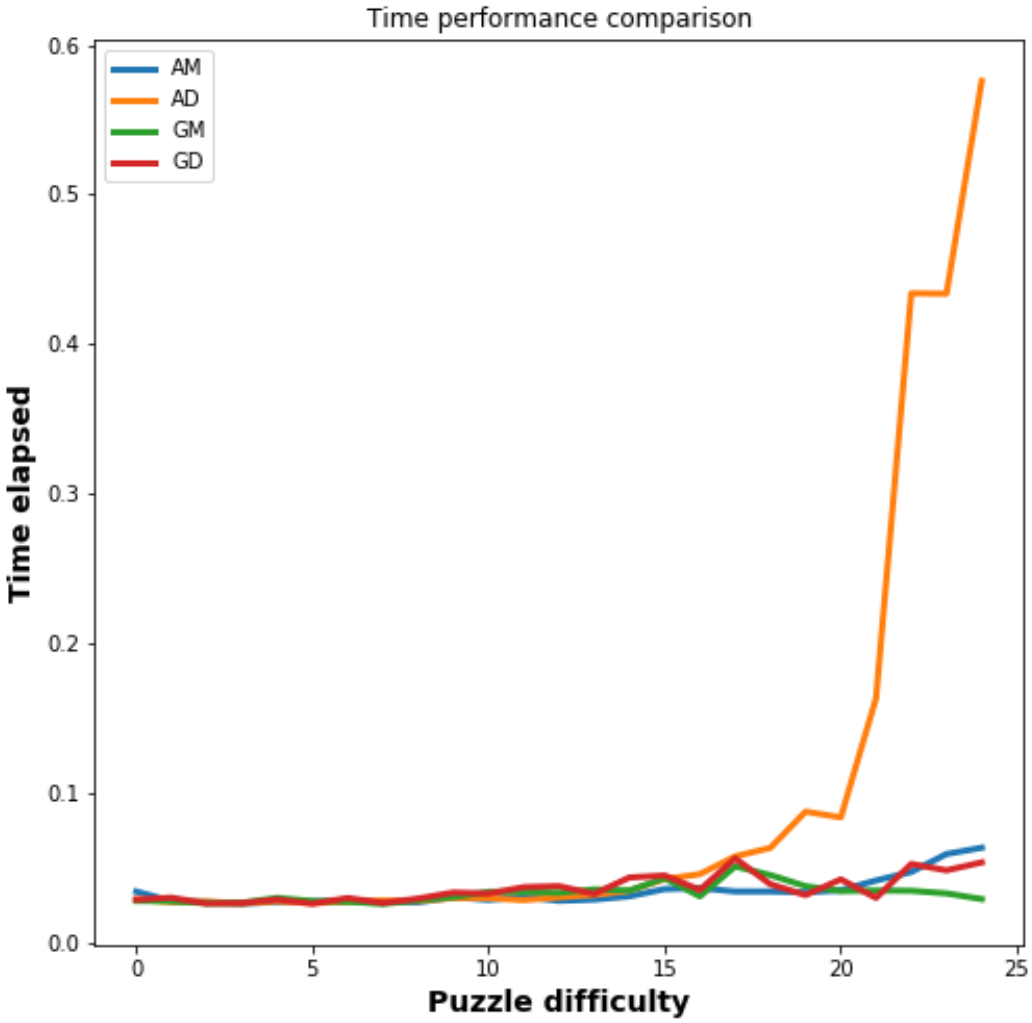

Figura 4: Tempo de execução para cada heurística utilizada.

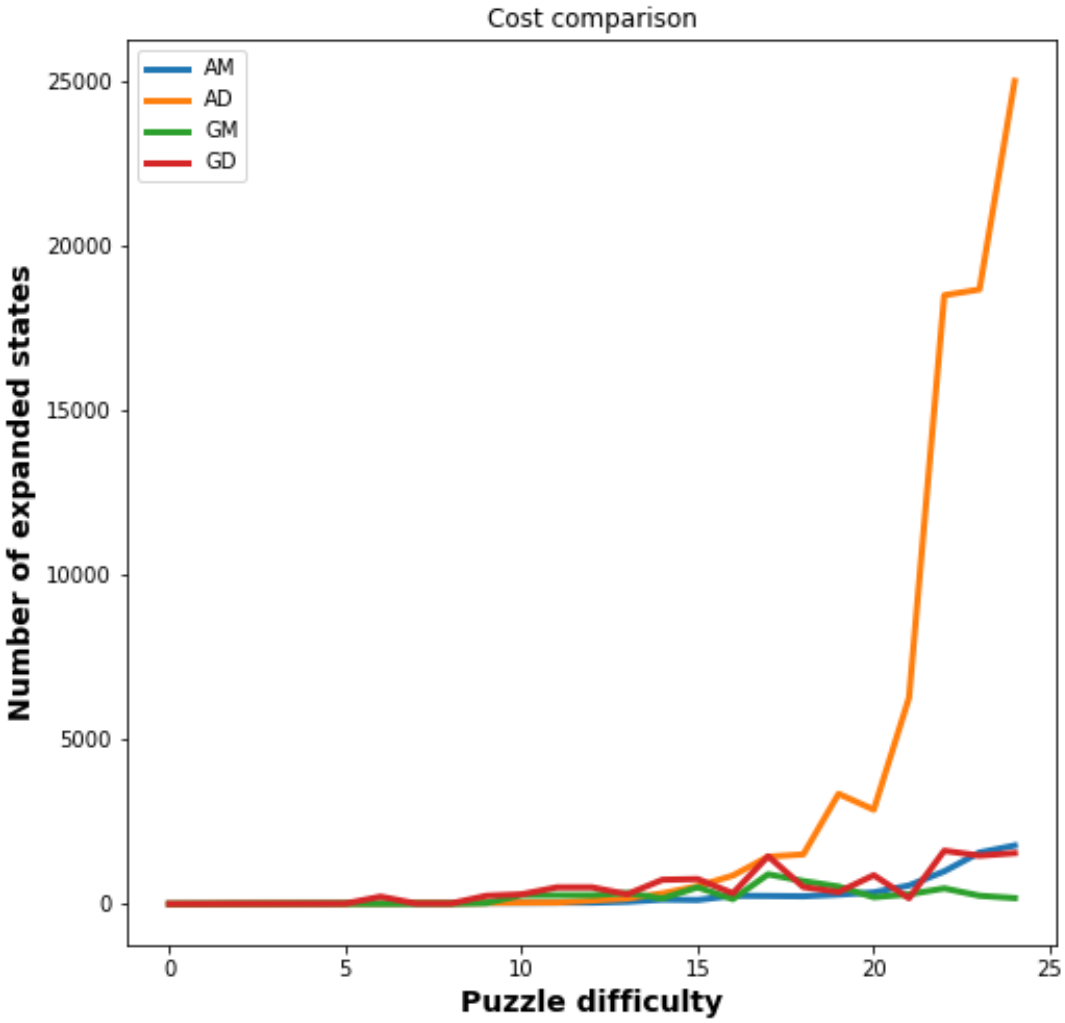

Figura 5: Número de estados expandidos para cada heurística utilizada.

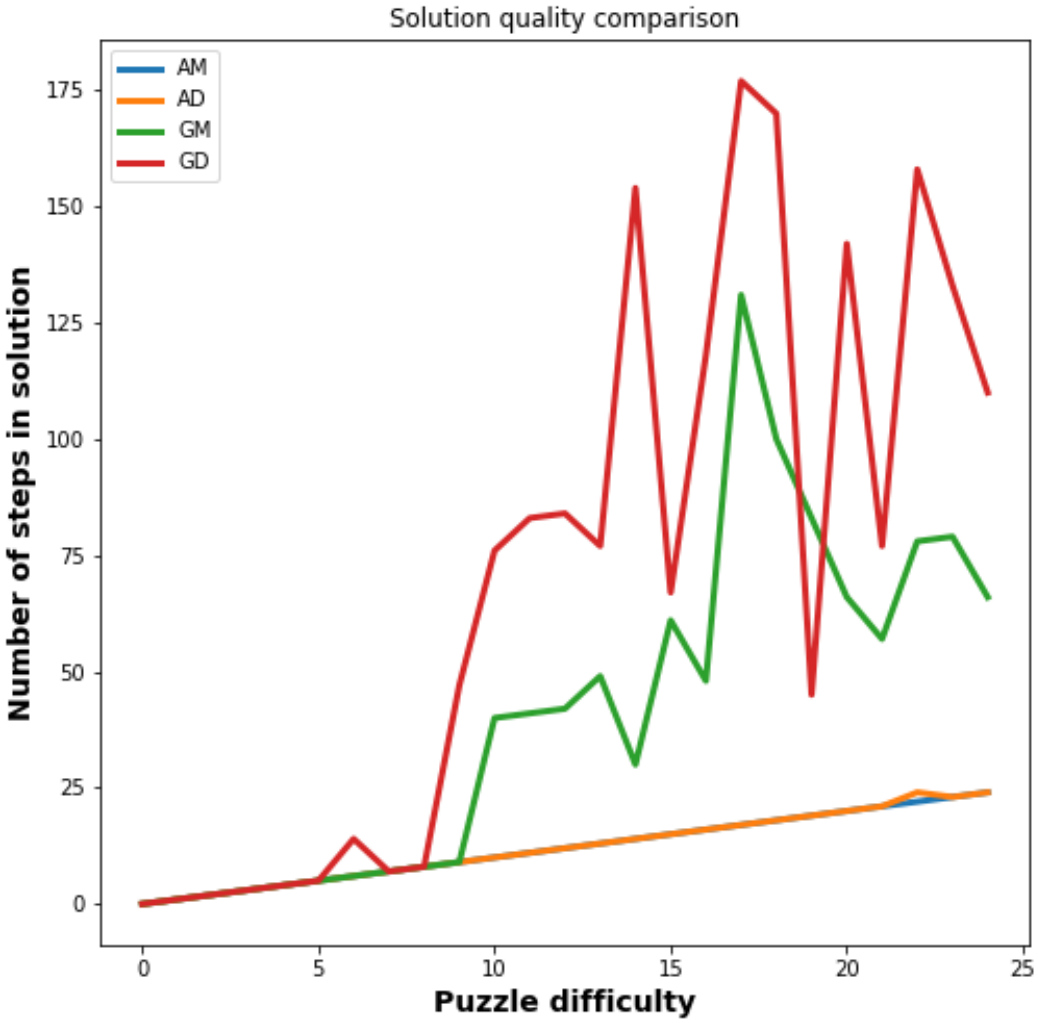

Figura 6: Número de passos até objetivo, para cada heurística utilizada. Note que há um comportamento estranho para o algoritmo AD, quando executado sobre a instância de dificuldade 22. Isso pode ser por conta de um *corner case* na implementação da heurística *Displaced Pieces* que a tornou não-admissível.

A Instruções de uso

A interface de linha de comando do programa segue a especificação. Isto é, a *usage* básica é:

```
./TP1 <algorithm> <puzzle_entries> [PRINT]
```

Utilize a bandeira `-log-level` para determinar o nível de log desejado na execução do programa. Similar ao `PRINT`, existe a opção `PRINT_STATISTICS`, que imprime estatísticas da execução ao final.

B Ambiente de testes

- **Sistema operacional:** Ubuntu 20.04.3 LTS.
- **CPU:** Intel(R) Core(TM) i7-1065G7 @ 1.30GHz-3.90GHz. 1 processador físico, 4 núcleos, 8 threads virtuais.
- **RAM:** 16GB tecnologia DDR4 a 3200MHz.