Introdução à Inteligência Artificial
Solucionador de Sudoku

Jorge Augusto de Lima e Silva - 2021032005 - UFMG - DCC642

Maio de 2024

1 Introdução

A proposta do trabalho consistia na implementação de cinco algoritmos distintos para a solução automática de um jogo de Sudoku. Os algoritmos em questão foram Breadth-First Search, Iterative Deepening Search, Uniform-Cost Search, A^* Search e Greedy Best-First Search.

2 Implementação

O programa foi implementado em C++ e a estrutura de arquivos do está dividida nos seguintes diretórios principais: obj, include e src, tal que os arquivos .o ficam em obj, o cabeçalho das principais funções que serão utilizadas ficam em include/sudoku.hpp e os arquivos fonte com as implementações dos métodos em questão ficam no diretório src. Além disso, no diretório raiz do projeto existe um arquivo Makefile, que pode ser usado para a compilação do programa.

Dentro do diretório src temos as implementação dos 5 algoritmos principais, da função main que vai coordenar o programa e algumas funções auxiliares que são utilizadas em alguns dos solucionadores desenvolvidos.

2.1 Breadth-First Search

O algoritmo de Busca em Largura (BFS) é um método de busca que expande todos os nós vizinhos do nó atual antes de passar para os nós subsequentes. No contexto da resolução do Sudoku, o BFS funciona expandindo o tabuleiro atualmente considerado para incluir todas as possíveis configurações do tabuleiro que surgem a partir da adição de um novo número em uma célula vazia.

Para implementar o algoritmo BFS, utilizamos uma fila, onde inicialmente inserimos o tabuleiro inicial. Em cada iteração, retiramos o tabuleiro da frente da fila e examinamos o próximo espaço vazio. Se não houver espaços vazios (ou seja, se o tabuleiro estiver completo), retornamos o tabuleiro. Caso contrário, para cada número possível que possa ser inserido na célula vazia, geramos um novo tabuleiro com esse número inserido e o colocamos na fila.

O algoritmo continua esse processo até que um tabuleiro completo seja encontrado, representando uma solução válida para o Sudoku.

2.2 Iterative Deepening Search

A Busca de Profundidade Iterativa (IDS) é uma estratégia de busca que combina as características da busca em profundidade com a busca em largura. A ideia básica é executar repetidamente uma busca em profundidade limitada, aumentando progressivamente o limite de profundidade, até encontrar a solução desejada.

Para aplicar o IDS à resolução do Sudoku, implementamos uma função de busca em profundidade limitada (LDFS) que recebe um limite de profundidade como parâmetro. Esta função executa uma busca em profundidade no tabuleiro atual, limitando a profundidade da busca ao valor especificado. Se a solução não for encontrada dentro desse limite de profundidade, aumentamos o limite e tentamos novamente.

O IDS continua esse processo até encontrar uma solução ou até atingir um limite de profundidade máximo, que neste caso é o tamanho do tabuleiro de Sudoku (81 células).

2.3 Uniform-Cost Search

O algoritmo de Busca de Custo Uniforme é uma variante da busca em largura que considera o custo associado a cada transição entre estados. Neste caso, utilizamos uma fila de prioridade em que o critério de prioridade é o custo acumulado para alcançar o estado atual. O algoritmo continua expandindo os estados até encontrar um estado objetivo, ou seja, um tabuleiro de Sudoku completo.

A cada iteração, o algoritmo seleciona o estado com o menor custo acumulado na fila de prioridade e expande suas possíveis transições, calculando o custo de cada transição e adicionando os estados resultantes na fila. Esse processo continua até que um estado objetivo seja encontrado.

2.4 Greedy Best-First Search

Este é um algoritmo heurístico que seleciona o próximo estado a ser expandido com base em uma heurística que estima a proximidade do estado ao objetivo. Neste caso, utilizamos uma fila de prioridade em que o critério de prioridade é a estimativa de proximidade de cada estado com base na quantidade de células vazias restantes no tabuleiro.

Em cada iteração, o algoritmo seleciona o estado com a menor estimativa de proximidade na fila de prioridade e expande suas possíveis transições. Esse processo continua até que um estado objetivo seja encontrado.

2.5 A^* Search

O algoritmo A^* é uma extensão da busca de custo uniforme que combina a função de custo acumulado com uma heurística que estima o custo restante para alcançar o estado objetivo. Neste caso, utilizamos uma fila de prioridade em que o critério de prioridade é a soma do custo acumulado e da estimativa do custo restante para cada estado.

A cada iteração, o algoritmo seleciona o estado com o menor valor da função de custo na fila de prioridade e expande suas possíveis transições, calculando o custo acumulado e a estimativa do custo restante para cada transição e adicionando os estados resultantes na fila. Esse processo continua até que um estado objetivo seja encontrado.

3 Resultados

Para cada dificuldade, foram realizados testes com 10 jogos diferentes, e os resultados obtidos podem ser observados nas figuras 1 e 2. Nestas figuras, é possível observar um comportamento geral dos algoritmos, com os algoritmos de busca informada sendo consistentemente melhores do que os de busca não informada, tanto em número de iterações quanto em tempo gasto.

Uma observação interessante é o comportamento do algoritmo Iterative Deepening Search quando comparado com os outros métodos de busca não informada. Devido à sua premissa de repetir a busca em profundidade limitada inteira cada vez que a profundidade é aumentada, este algoritmo visita um mesmo estado um número considerável de vezes, contribuindo para uma alta quantidade de iterações.

Além disso, é importante considerar que a profundidade da solução é fixa e já conhecida, tornando o uso deste algoritmo pouco eficiente em quase todos os contextos, comportando-se de forma semelhante à busca em largura, mas com uma repetição excessiva de estados.

Por outro lado, o Iterative Deepening Search consome menos memória, uma vez que não precisa armazenar estados a serem explorados em uma estrutura de dados auxiliar, e a pilha de execução contém no máximo 81 estados empilhados devido à recursão, um número muito menor em comparação com os outros métodos.

Outro ponto relevante é a escolha da heurística para os algoritmos A^* e Greedy Best-First Search. Ao selecionar a heurística de preencher a célula com o menor número de candidatos possíveis, o branching-factor do problema é significativamente reduzido, o que facilita a chegada a uma opção correta para a célula, diminuindo também a quantidade de candidatos das outras células relacionadas. Além disso, se uma célula vazia sem candidatos associados for encontrada durante a busca pela célula com o menor número de candidatos, este estado pode ser descartado em favor de outros ramos do espaço de busca.

As figuras 3, 4, 5, 6, 7 e 8, baseadas nos testes disponibilizados na proposta do trabalho, destacam os comportamentos discutidos ao longo desta seção, especialmente a diferença significativa em iterações e tempo entre os algoritmos de busca não informada e informada.

4 Conclusão

A partir deste trabalho, foi possível obter uma compreensão abrangente dos diferentes tipos de algoritmos de busca em espaços de estados de um problema, destacando a importância do uso de informações adicionais para melhorar o desempenho. Os resultados obtidos evidenciaram que os algoritmos de busca informada superaram consistentemente os algoritmos de busca não informada, tanto em termos de tempo de execução quanto de número de iterações.

Uma observação relevante foi o comportamento peculiar do algoritmo Iterative Deepening Search. Apesar de ter demonstrado um desempenho inferior em comparação com os outros métodos, sua utilidade se destaca em situações em que se busca soluções com profundidades menores, tornando-o uma opção viável em contextos específicos.

Além disso, foi possível identificar o tipo de cenário em que cada algoritmo é mais adequado para uso prático. Por exemplo, o A^* e o Greedy Best-First Search demonstraram ser eficazes em situações em que se pode aplicar uma heurística precisa para estimar o custo restante até a solução. Por outro lado, a Busca em Largura e o Uniform-Cost Search são mais adequados quando não há informações adicionais disponíveis sobre o espaço de busca.

Em resumo, este trabalho proporcionou uma visão geral das diferentes estratégias de busca em inteligência artificial, destacando suas vantagens e limitações em contextos específicos. Essas

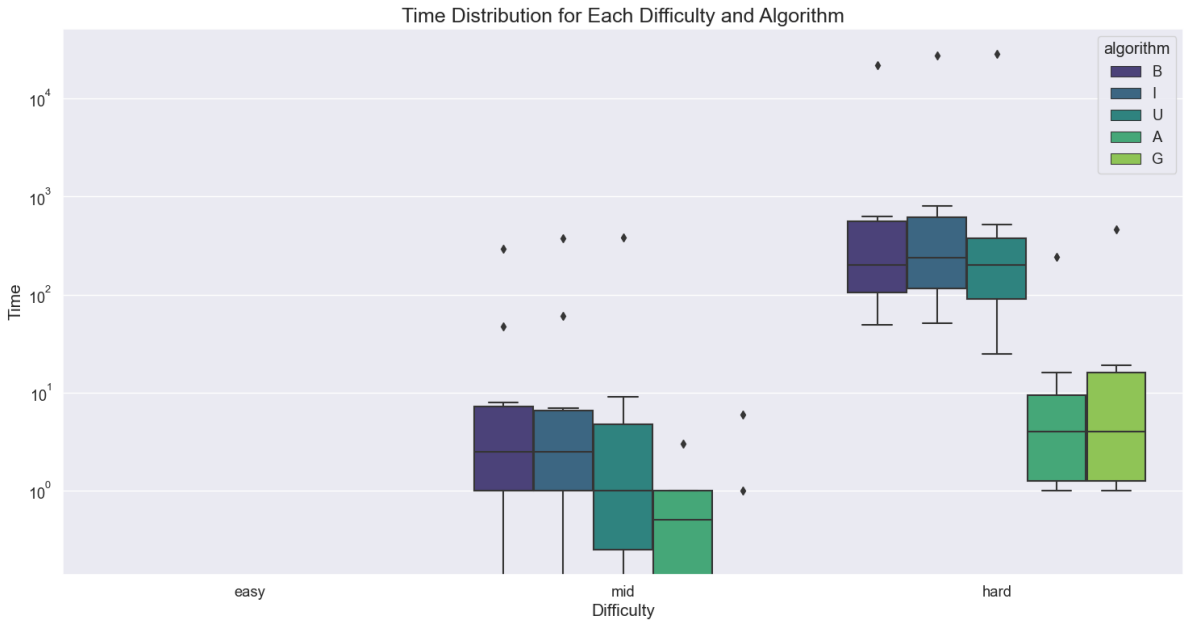

Figura 1: Comparação do tempo de execução dos algoritmos para diferentes dificuldades de Sudoku.

informações são fundamentais para a seleção adequada de algoritmos em problemas do mundo real, levando em consideração as características individuais de cada situação.

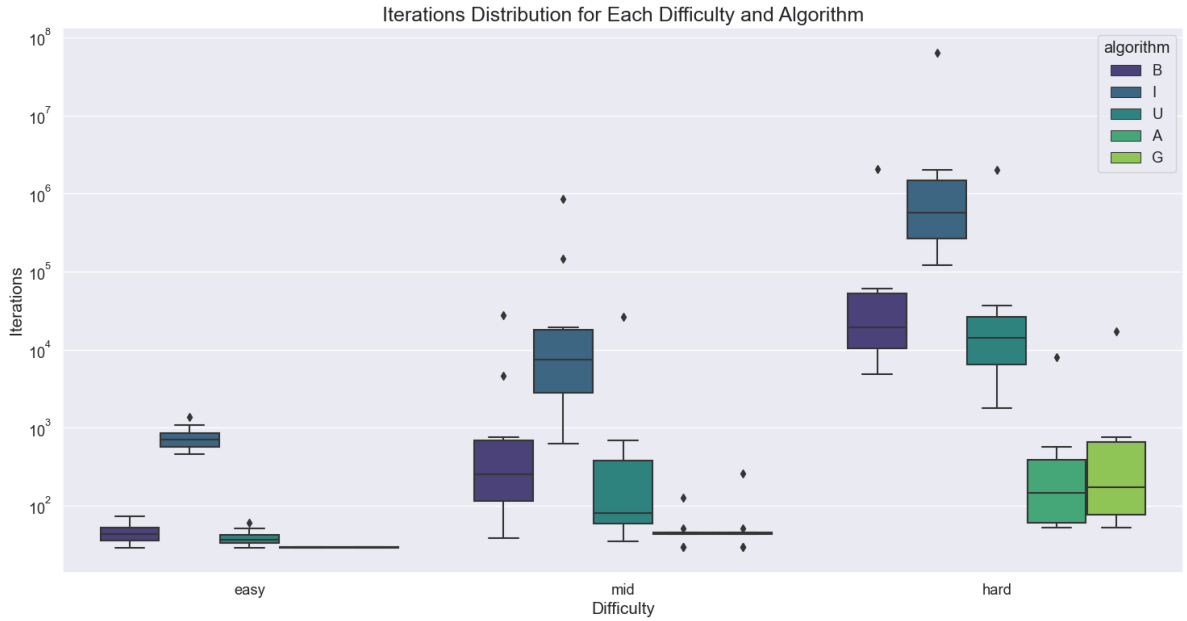

Figura 2: Comparação do número de iterações dos algoritmos para diferentes dificuldades de Sudoku.

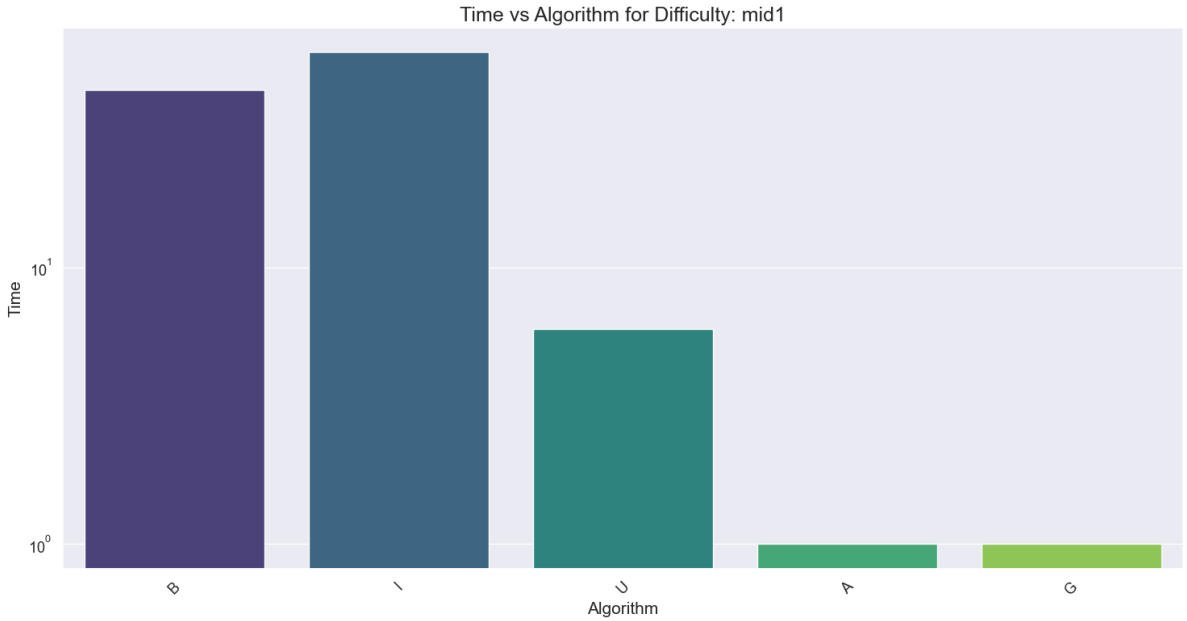

Figura 3: Tempo de execução para o primeiro jogo de dificuldade média.

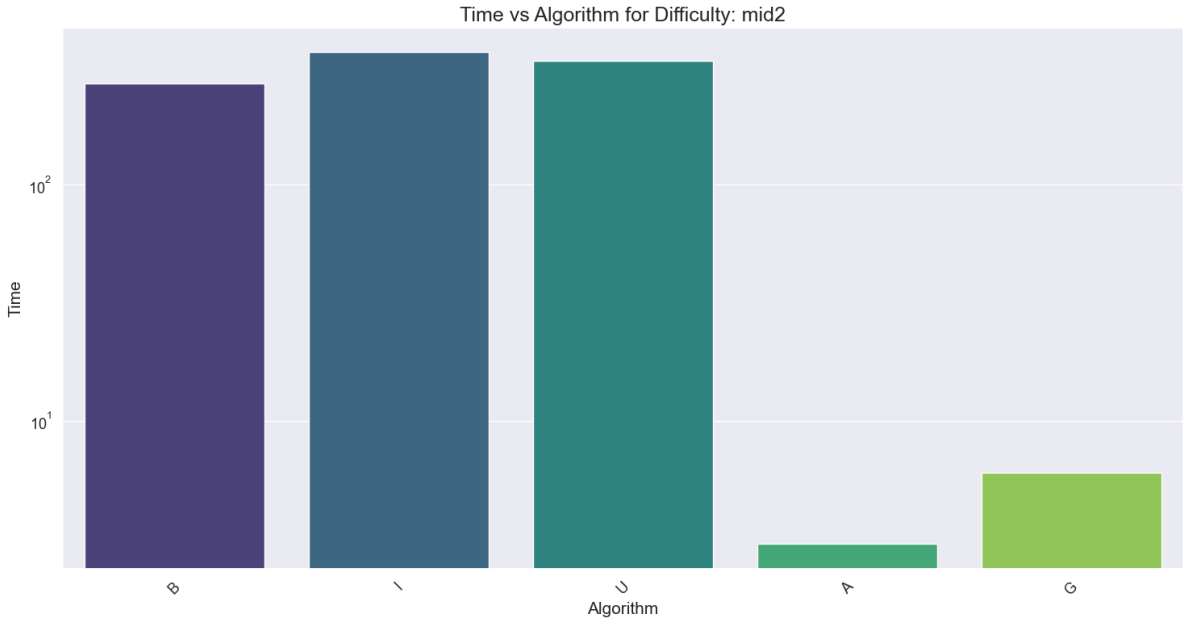

Figura 4: Tempo de execução para o segundo jogo de dificuldade média.

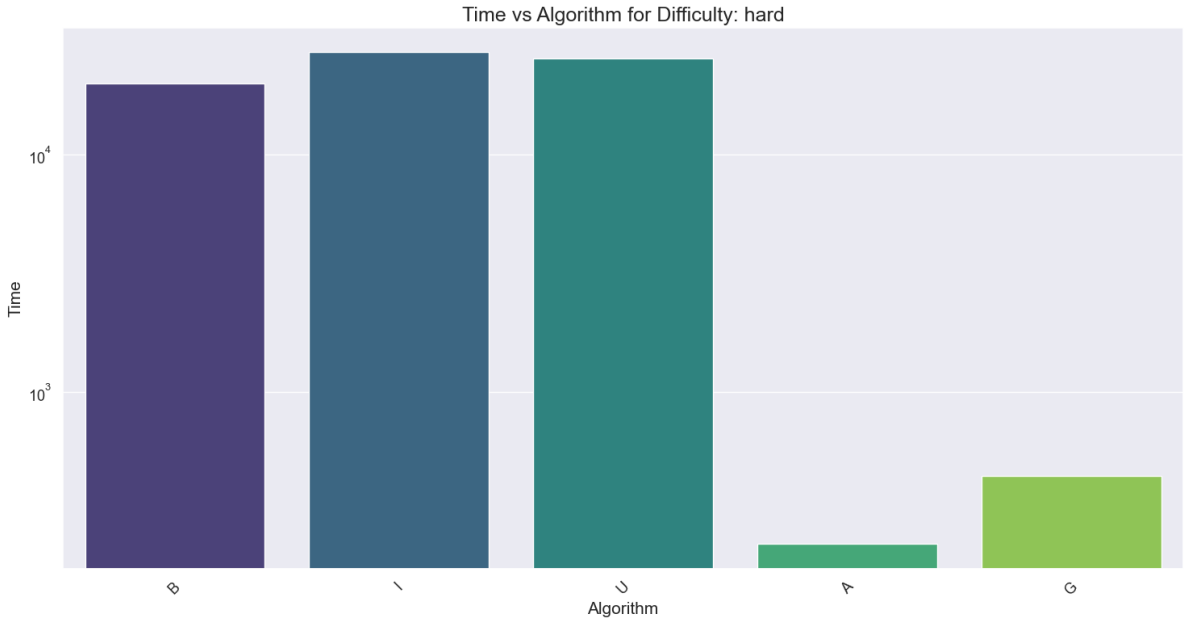

Figura 5: Tempo de execução para o jogo de dificuldade alta.

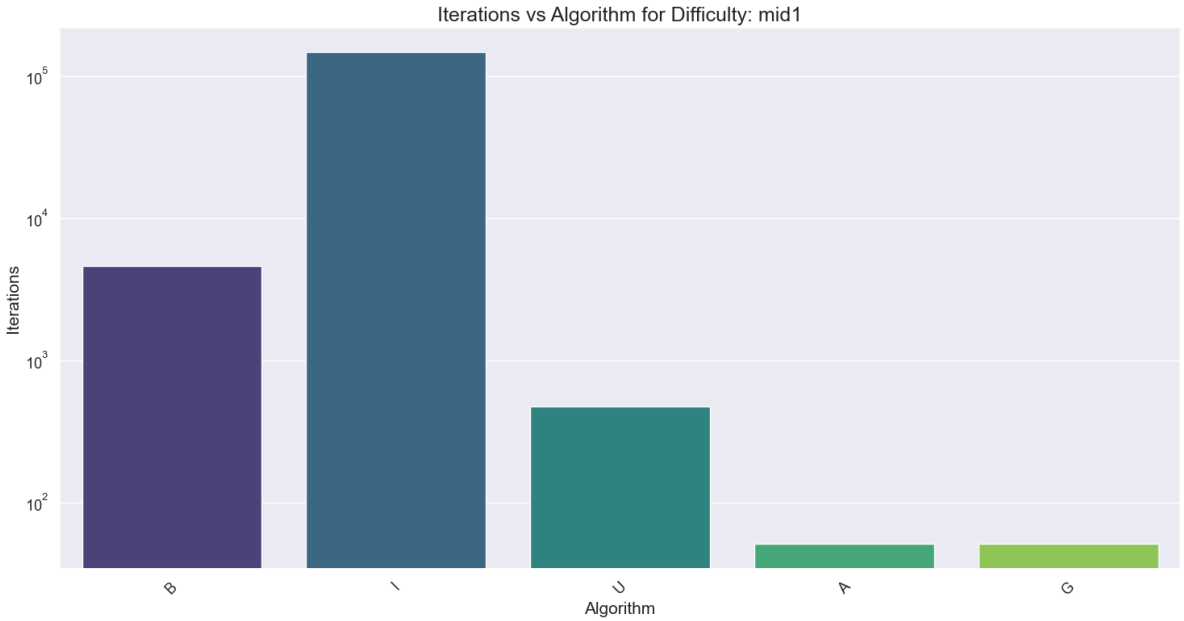

Figura 6: Número de iterações para o primeiro jogo de dificuldade média.

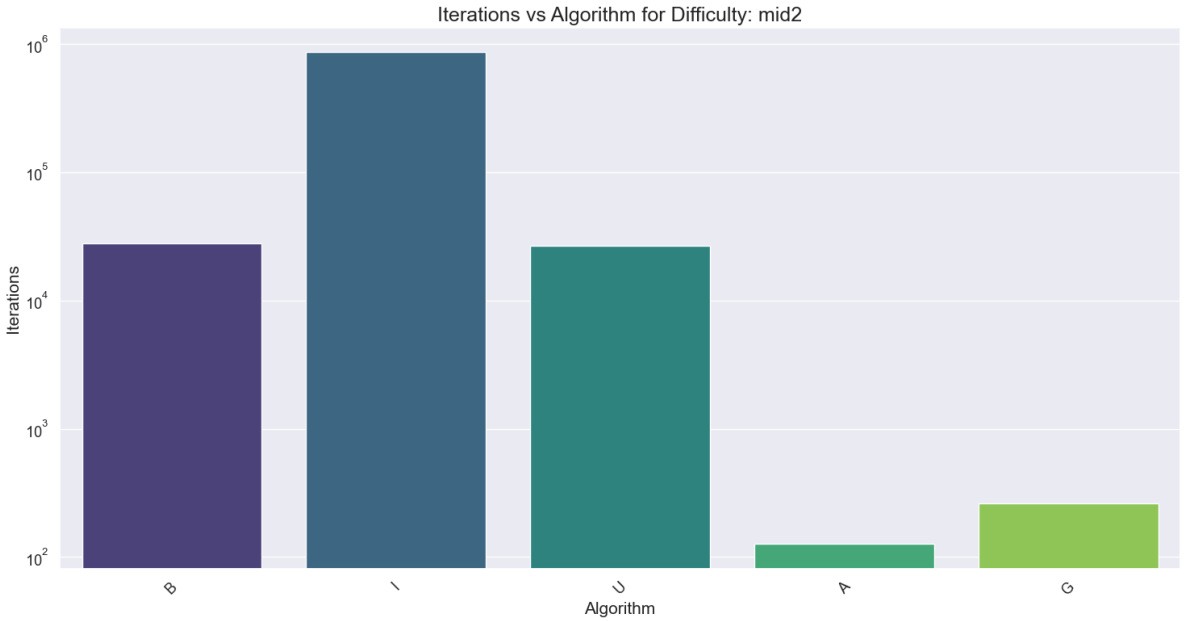

Figura 7: Número de iterações para o segundo jogo de dificuldade média.

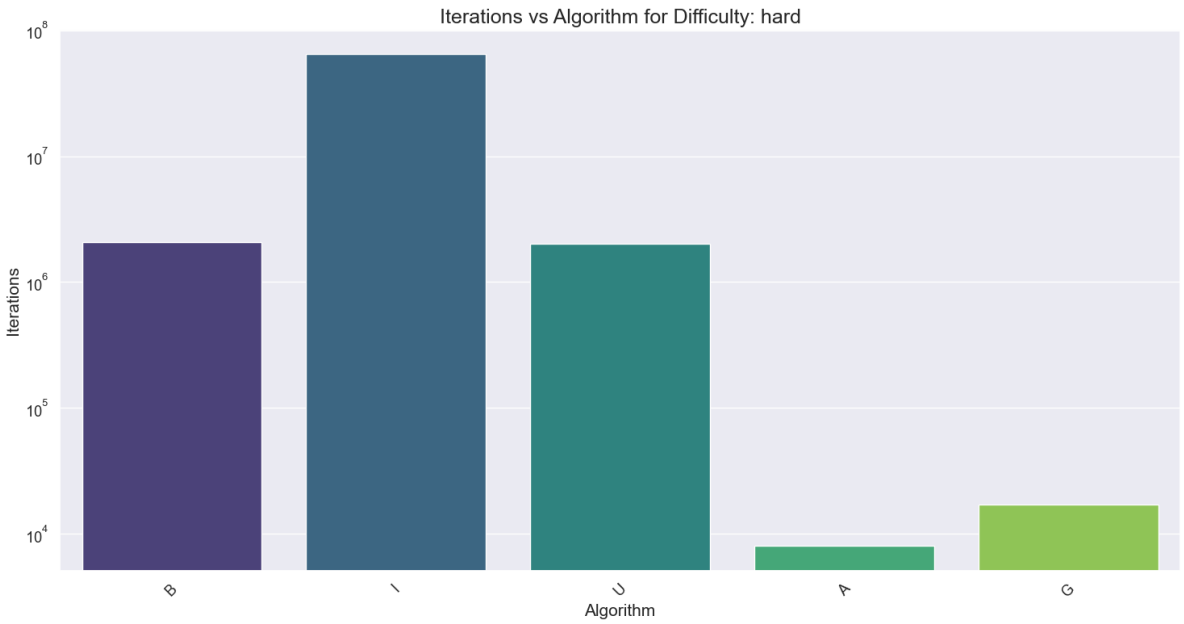

Figura 8: Número de iterações o jogo de dificuldade alta.