Resolução do Problema n-Puzzle por Algoritmos de Busca com e sem Informação

Maria Larissa de Oliveira S. Sousal[1], Priscila do Espírito Santo Sousa[2]

[1]Instituto Federal do Piauí - Campus Picos Curso: Análise e Desenvolvimento de Sistemas
Disciplina: Inteligência Artificial

{capic.2024118tads0013@aluno.ifpi.edu.br[1], capic.2024118tads0031@aluno.ifpi.edu.br[2]

Abstract. *This paper presents a comparative analysis of information-free and information-based search algorithms applied to the n-Puzzle problem. The BFS, DFS, IDS, A* and Greedy Search algorithms were used, evaluating execution time, solution depth, number of expanded nodes and the effectiveness of heuristics.*

Resumo. *Este trabalho apresenta uma análise comparativa de algoritmos de busca sem informação e com informação aplicados ao problema n-Puzzle. Foram utilizados os algoritmos BFS, DFS, IDS, A* e Busca Gulosa (Greedy), avaliando tempo de execução, profundidade da solução, quantidade de nós expandidos e a eficácia de heurísticas.*

1. Introdução

O n-Puzzle é um problema clássico de Inteligência Artificial, composto por uma grade de tamanho n x n contendo peças numeradas e um espaço vazio representado pelo número 0. O objetivo é reorganizar as peças a partir de um estado inicial até atingir o estado objetivo, através de movimentos válidos.

Algoritmos Implementados: Foram implementados os seguintes algoritmos:

- **Busca sem informação:** BFS, DFS, IDS
- **Busca com informação:** A* e Greedy, com heurísticas de Manhattan e peças fora do lugar

2. Fundamentação Teórica

Os algoritmos de busca em Inteligência Artificial (IA) são métodos utilizados para explorar espaços de estados em busca de soluções para problemas complexos. Eles são fundamentais para a resolução de tarefas como planejamento, tomada de decisão e jogos, onde é necessário encontrar uma sequência de ações que leva de um estado inicial a um estado objetivo. A eficiência desses algoritmos está diretamente ligada à forma como eles percorrem o espaço de busca e utilizam (ou não) informações adicionais sobre o problema.

2.1. Diferença entre Busca Sem Informação e Busca Com Informação

A busca sem informação (também chamada de busca cega) é aquela que não possui conhecimento adicional sobre o problema além da definição dos estados e das ações possíveis. Os algoritmos exploram o espaço de forma sistemática, sem orientação sobre qual caminho pode ser mais promissor. Exemplos incluem BFS, DFS e IDS.

A busca com informação (ou busca heurística) utiliza estimativas ou funções heurísticas para guiar a busca em direção ao objetivo, reduzindo o número de estados explorados. Essas heurísticas fornecem uma estimativa do custo restante para atingir o objetivo, tornando o processo mais eficiente. Exemplos são A* e Busca Gulosa.

2.2. Descrição dos Algoritmos Utilizados

- **BFS (Busca em Largura)**: Explora os nós nível por nível, usando fila, garantindo encontrar a solução de menor profundidade (em termos de número de passos), se existir. É completa e ótima, porém consome muita memória em espaços grandes.
- **DFS (Busca em Profundidade)**: Explora os caminhos até o fundo antes de retroceder, utilizando pilha ou recursão. É mais econômica em memória, mas pode entrar em ciclos ou explorar caminhos longos desnecessariamente. Não é ótima nem completa sem limites.
- **IDS (Busca em Profundidade Iterativa)**: Combina DFS com BFS: realiza múltiplas buscas em profundidade com limites crescentes. É completa e ótima como BFS, mas com uso de memória similar ao DFS.
- **A***: Utiliza uma função de custo total f(n)=g(n)+h(n), onde g(n) é o custo até o nó atual e h(n) é a estimativa do custo restante (heurística). É eficiente e ótima se a heurística for admissível.
- **Busca Gulosa**: Considera apenas a heurística h(n) para guiar a busca, ignorando o custo acumulado. Costuma ser mais rápida que A*, mas não garante encontrar a solução ótima.

2.3. Heurísticas Utilizadas

- **Número de Peças Fora do Lugar (Hamming):** Conta quantas peças estão em posições incorretas, desconsiderando a distância real. É simples, rápida, mas menos precisa.
- **Distância de Manhattan:** Soma as distâncias horizontais e verticais que cada peça está de sua posição correta. É mais precisa que Hamming, pois considera a movimentação real no tabuleiro, sendo bastante utilizada em A* e busca gulosa.

3. Descrição do Problema

- **Estado Inicial**: Configuração inserida pelo usuário.
- **Estado Objetivo**: Peças ordenadas de forma crescente, com 0 no final.
- **Ações**: Mover a peça vazia para cima, baixo, esquerda ou direita.
- **Custo**: Uniforme, cada ação tem custo 1.

4. Metodologia

Os testes foram realizados em instâncias de tamanhos 3x3 (8-puzzle), 4x4 (15-puzzle) nos algoritmos de busca sem informação e 3x3 (8-puzzle), 4x4 (15-puzzle), 5x5 (24-puzzle) nos de busca com informação. Cada algoritmo foi executado sobre instâncias com

dificuldade crescente, coletando dados de tempo de execução, profundidade da solução e quantidade de nós expandidos.

4.1. Descrição da Implementação

Linguagem e Ferramentas Utilizadas:

O programa foi desenvolvido em Python 3 utilizando apenas bibliotecas padrão da linguagem. A implementação é modular, com cada algoritmo implementado em arquivos separados (bfs.py, dfs.py, ids.py, astar.py, greedy.py), o que facilita a manutenção e a análise individual. A execução foi realizada em um notebook Acer Aspire 5, equipado com 512 GB SSD e 8 GB de RAM, sem uso de paralelismo ou aceleração por GPU.

4.2. Estrutura do Tabuleiro

O tabuleiro é representado como uma tupla imutável de inteiros, em uma única linha (linearizado). Por exemplo, para um tabuleiro 3x3, o estado (1, 2, 3, 4, 5, 6, 7, 8, 0) representa a configuração com as peças organizadas e o 0 indicando a posição vazia.

4.3. Estado Inicial e Objetivo

Estado inicial: definido pelo usuário como uma permutação válida dos números de 0 a n² - 1. **Os estados iniciais usados nos experimentos foram:**

- 3x3: `(1, 2, 3, 5, 0, 6, 4, 7, 8)`
- 4x4: `(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 13, 14, 15)`
- 5x5: `(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24)`

Estado objetivo: definido como a sequência ordenada crescente dos números de 1 até n² - 1, com o 0 na última posição, ou seja:

- 3x3: (1, 2, 3, 4, 5, 6, 7, 8, 0)
- 4x4: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)
- 5x5: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 0)

4.4. Operadores

As ações disponíveis em cada estado são: Cima (Up), Baixo (Down), Esquerda (Left), Direita (Right). Cada ação move a peça vizinha em direção à posição vazia, gerando um novo estado válido.

4.5. Custo Uniforme

O custo de cada ação é uniforme: todos os movimentos têm custo 1. Isso permite que algoritmos como A* e Busca em Largura operem corretamente com base na profundidade dos caminhos.

4.6. Heurísticas Aplicadas

Para os algoritmos com informação (A* e Busca Gulosa), duas heurísticas foram implementadas:

Distância de Manhattan: soma das distâncias verticais e horizontais de cada peça até sua posição correta.

Número de peças fora do lugar (Hamming): conta quantas peças estão posicionadas incorretamente em relação ao objetivo.

4.7. Entrada e Saída do Programa

Entrada: O usuário fornece o tamanho do puzzle (ex.: 3 para 3x3) e o estado inicial como uma sequência de números separados por espaço.

Saída: Após a execução do algoritmo escolhido, o programa imprime: o caminho percorrido (movimentos), a profundidade da solução, o número de nós expandidos, o tempo de execução, e a sequência de estados gerada até alcançar o objetivo.

5. Experimentos e Resultados

5.1. Análise Comparativa dos Algoritmos Sem Informação

BFS (Busca em Largura) 3x3: Visitou 33 nós, encontrou a solução em profundidade 4 e tempo de execução de 0.0002 segundos. Já o 4x4: Teve um desempenho razoável com apenas 17 nós visitados e profundidade 3, porém o tempo subiu para 0.003 segundos. A vantagem é que garante solução ótima e profundidade mínima a desvantagem é o consumo de memória cresce rapidamente com o aumento do tamanho do tabuleiro.

DFS (Busca em Profundidade) 3x3: Expandiu um número muito elevado de nós (13.689) e chegou a uma profundidade de 34, com tempo de 0.0485 segundos. Já 4x4: Os resultados não foram obtidos, pois quando ele vai revolver o problema ele pega o estado e expande todos os nós fazendo com que fique num loop infinito e estoure a memoria, fazendo com que o computador trave e não consiga resolver o problema do puzzle indicando que o algoritmo pode ter ficado preso em ciclos ou executado por tempo excessivo. Com isso, não garante solução ótima; propenso a caminhos longos e ineficientes.

IDS (Busca em Profundidade Iterativa) 3x3: Solução eficiente com apenas 16 nós visitados, profundidade 4 e tempo de 0.0001s. O 4x4: Também eficiente com 7 nós visitados, profundidade 3 e tempo idêntico. Possui a vantagem de combinar a completude e otimalidade do BFS com o uso de memória do DFS. E sua principal desvantagem é a possibilidade de repetir muitas buscas em profundidades inferiores, mas o impacto é pequeno para puzzles pequenos.

Table 1. Resultados dos algoritmos de busca sem informação.

Algoritmo	Nós visitados	Profundidade	Tempo de Execução
BFS 3x3	33	4	0.0002 segundos
BFS 4x4	17	3	0.003 segundos
DFS 3x3	13689	34	0.0485 segundos
DFS 4x4	----	----	----
IDS 3x3	16	4	0.0001s
IDS 4x4	7	3	0.0001s

Logo, O IDS demonstrou ser o algoritmo mais eficiente para as instâncias testadas, conseguindo resultados ótimos com o menor número de nós visitados e tempo extremamente baixo. O BFS também teve bom desempenho, mas com maior custo computacional e memória. E o DFS se mostrou pouco eficiente, com alto custo de tempo e número de nós, sendo inviável para o 4x4.

5.2. Análise Comparativa dos Algoritmos Com Informação

Table 2. Resultados dos algoritmos de busca com informação.

Algoritmo	Nós visitados	Profundidade	Tempo de Execução
A* (Manhattan) 3x3	5	4	14.3565 segundos
A* (Manhattan) 4x4	4	3	2.1039 segundos
A* (Manhattan) 5x5	213506	38	14.5759 segundos
A* (Fora do lugar) 3x3	5	4	1.3887 segundos
A* (Fora do lugar) 4x4	4	3	2.0747 segundos
A* (Fora do lugar) 5x5	----	----	----
Gulosa (Manhattan) 3x3	5	4	2.9545 segundos
Gulosa (Manhattan) 4x4	4	3	1.5032 segundos
Gulosa (Manhattan) 5x5	5814	298	1.5831 segundos
Gulosa (Fora do lugar) 3x3	5	4	1.6268 segundos
Gulosa (Fora do lugar) 4x4	4	3	1.9260 segundos
Gulosa (Fora do lugar) 5x5	29063	676	1.8062 segundos

5.3. Heurística: Distância de Manhattan

A* obteve excelentes resultados em 3x3 e 4x4, com apenas 5 e 4 nós visitados respectivamente, e profundidades de 4 e 3. O tempo, no entanto, foi alto em 3x3 (14.3565s), evidenciando um custo computacional elevado, apesar do ótimo desempenho heurístico. No 5x5, a quantidade de nós visitados foi extremamente alta (213.506), embora a profundidade da solução tenha sido relativamente moderada (38). Isso mostra que, apesar da eficácia da heurística, o espaço de busca cresce rapidamente.

A Busca Gulosa, com a mesma heurística: Teve desempenho parecido em 3x3 e 4x4 (nós = 5 e 4), mas tempo inferior ao A* (2.9545s e 1.5032s). No 5x5, visitou muito menos nós (5814) que o A*, porém encontrou uma solução menos profunda (298), o que mostra que não garante a melhor solução (não ótima), mas é mais rápida Logo: A* é mais precisa (soluções ótimas), mas mais lenta. E a Gulosa é mais rápida, porém menos confiável para soluções mínimas.

5.4. Heurística: Número de Peças Fora do Lugar (Hamming)

No 3x3 e 4x4, tanto A* quanto Gulosa tiveram desempenho praticamente idêntico aos resultados com Manhattan, com tempo significativamente menor no A* em 3x3 (1.3887s). No 5x5, os dados da A* não foram apresentados porque excedeu a memoria RAM, mas a Busca Gulosa visitou 29.063 nós e retornou uma solução profunda (676), evidenciando que essa heurística é menos informativa do que Manhattan, resultando em caminhos mais longos. Com isso, em puzzles pequenos, a heurística é rápida e eficiente, e em puzzles grandes, é pouco precisa, aumentando a profundidade e a carga computacional.

Dessa forma, A* com Manhattan é a abordagem mais completa (ótima e informada), porém seu custo de tempo aumenta significativamente com o tamanho do puzzle.

Em contrapartida, a Gulosa com Manhattan é mais rápida e visita menos nós, mas não garante a solução ótima.

Hamming (peças fora do lugar) é útil apenas em puzzles pequenos; em instâncias maiores (5x5), sua performance decai drasticamente.

Para puzzles maiores, o uso de Manhattan é mais eficaz, mesmo com tempo de execução mais elevado, pois entrega soluções melhores.

5.5. Gráficos Comparativos

Para facilitar a análise do desempenho dos algoritmos de busca aplicados ao problema n-Puzzle, foram elaborados gráficos comparativos com base nos dados obtidos durante os experimentos. Os gráficos a seguir comparam os algoritmos quanto ao tempo de execução, número de nós visitados e profundidade da solução encontrada.

Essas representações visuais permitem observar com maior clareza as vantagens e limitações de cada abordagem, destacando o impacto do uso de heurísticas informadas e o comportamento dos algoritmos em instâncias de diferentes tamanhos.

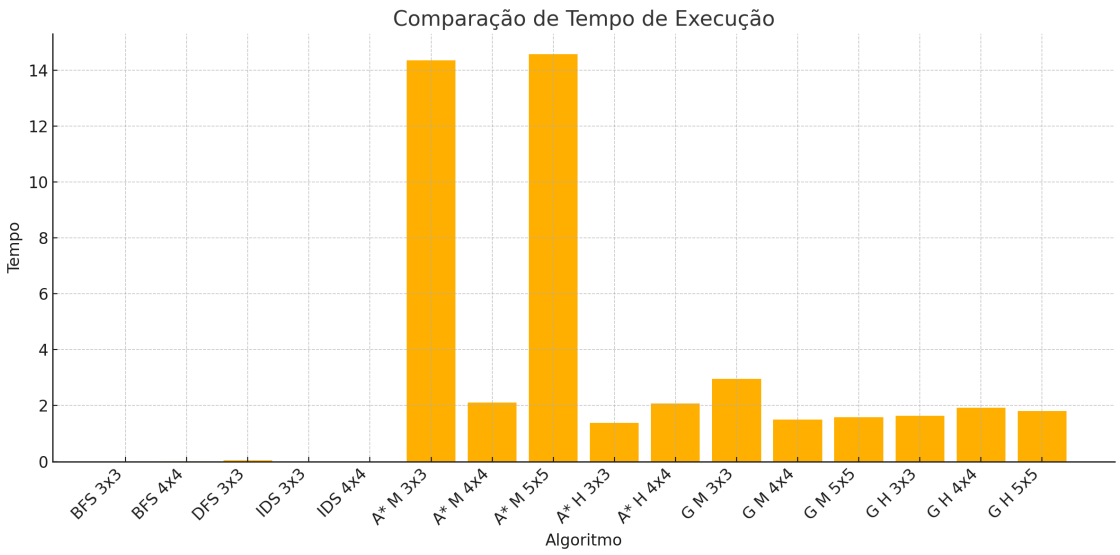

Figure 1. Comparação do tempo necessário para que cada algoritmo encontrasse a solução do n-Puzzle. Algoritmos como A* com Manhattan apresentam tempos elevados em instâncias pequenas, enquanto heurísticas menos precisas mantêm tempos menores, porém com soluções menos profundas.

Figure 2. Número total de estados (nós) explorados durante a execução de cada algoritmo. Algoritmos informados, especialmente A* com heurísticas eficazes, tendem a visitar menos nós que algoritmos sem informação, como DFS. No entanto, o custo cresce exponencialmente em instâncias maiores (como no 5x5 com A*).

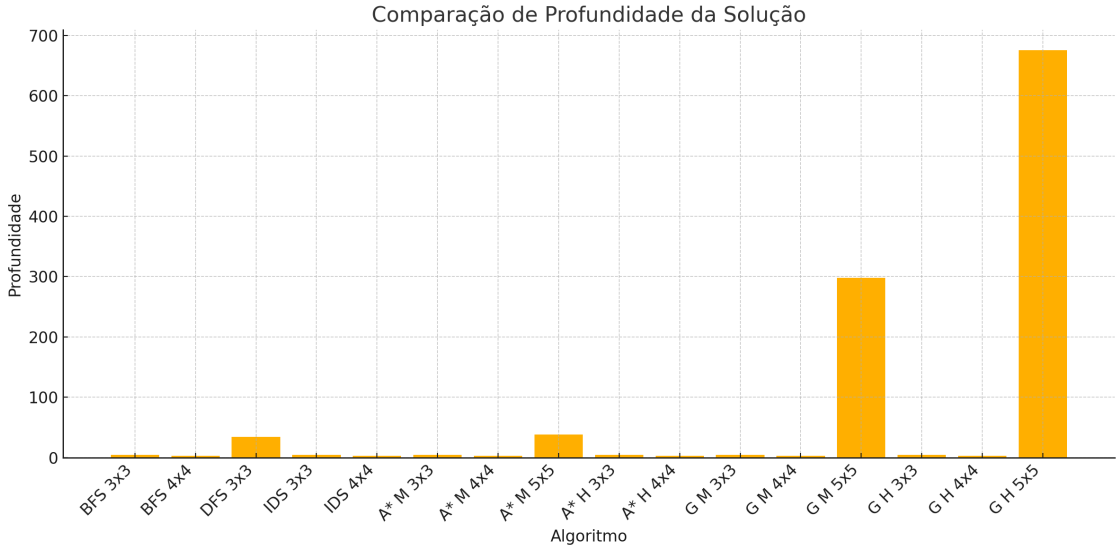

Figure 3. Profundidade (número de movimentos) da solução encontrada por cada algoritmo. A* garante soluções ótimas com profundidades mínimas. Algoritmos como a busca gulosa, apesar de rápidos, podem gerar caminhos significativamente mais longos em problemas maiores.

6. Exemplos de Caminho da Solução com Árvore Gerada

Nesta seção, são apresentados exemplos visuais da estrutura em árvore construída a partir do estado inicial até o estado objetivo para duas instâncias do problema n-Puzzle: 8-puzzle (3x3) e 15-puzzle (4x4).

Cada árvore representa a sequência de estados visitados pelo algoritmo até alcançar a solução, destacando o caminho correto por meio das ações executadas em cada nível da árvore. O objetivo é ilustrar a forma como os algoritmos de busca constroem o espaço de estados e conduzem a resolução do problema a partir da raiz até o objetivo.

As figuras a seguir evidenciam esse processo de maneira hierárquica e organizada, facilitando a compreensão da lógica de busca aplicada.

6.1. Exemplo de Caminho da Solução para 8-Puzzle (Profundidade: 4):

A Figura apresenta a árvore de busca gerada para uma instância do problema 8-Puzzle, resolvida pelo algoritmo BFS (Busca em Largura). O estado inicial está representado na raiz da árvore, com profundidade 0, e cada nível subsequente mostra a aplicação de uma ação que transforma o estado atual em um novo estado filho.

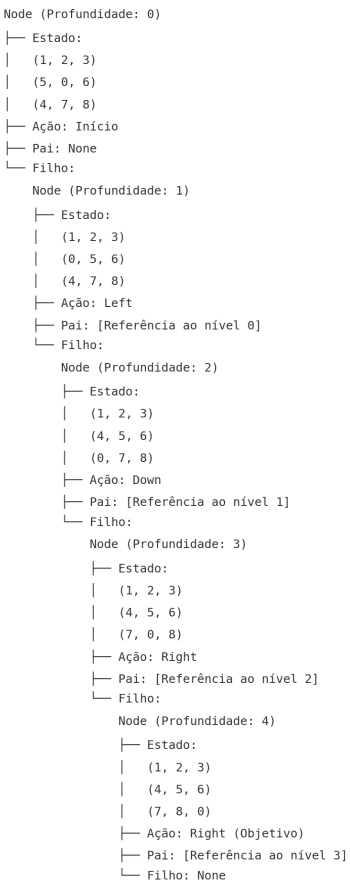

Figure 4. Estrutura de dados em árvore representando o caminho da solução no 8-puzzle.

Ao seguir esse caminho na árvore, é possível visualizar claramente como o algoritmo percorreu os estados até alcançar a solução ótima. Essa estrutura facilita a análise de desempenho, profundidade da solução e o custo computacional envolvido.

6.2. Exemplo de Caminho da Solução para 15-Puzzle (Profundidade: 3)

A Figura abaixo representa a estrutura da árvore de busca gerada durante a resolução do 15-Puzzle utilizando o algoritmo BFS (Busca em Largura). O estado inicial está posi-

cionado na raiz da árvore, no nível de profundidade 0. A partir dele, são aplicadas ações válidas (neste caso, movimentos do espaço vazio para a direita), gerando novos estados em níveis subsequentes.

```
Node (Profundidade: 0)
├── Estado:
│   (1, 2, 3, 4)
│   (5, 6, 7, 8)
│   (9, 10, 11, 12)
│   (0, 13, 14, 15)
├── Ação: Início
├── Pai: None
└── Filho:
    Node (Profundidade: 1)
    ├── Estado:
    │   (1, 2, 3, 4)
    │   (5, 6, 7, 8)
    │   (9, 10, 11, 12)
    │   (13, 0, 14, 15)
    ├── Ação: Right
    ├── Pai: [Referência ao nível 0]
    └── Filho:
        Node (Profundidade: 2)
        ├── Estado:
        │   (1, 2, 3, 4)
        │   (5, 6, 7, 8)
        │   (9, 10, 11, 12)
        │   (13, 14, 0, 15)
        ├── Ação: Right
        ├── Pai: [Referência ao nível 1]
        └── Filho:
            Node (Profundidade: 3)
            ├── Estado:
            │   (1, 2, 3, 4)
            │   (5, 6, 7, 8)
            │   (9, 10, 11, 12)
            │   (13, 14, 15, 0)
            ├── Ação: Right (Objetivo)
            ├── Pai: [Referência ao nível 2]
            └── Filho: None
```

Figure 5. Estrutura de dados em árvore representando o caminho da solução no 15-puzzle.

Essa representação evidencia como o algoritmo BFS explora o espaço de busca de forma sistemática, nível por nível, até encontrar a solução. Neste caso, a solução foi encontrada com profundidade 3, demonstrando uma sequência curta de ações até alcançar o objetivo.

7. Análise e Discussão

Os resultados mostram que algoritmos com heurísticas como A* são mais eficazes em termos de profundidade e número de nós expandidos, especialmente para tabuleiros maiores. Por outro lado, DFS não é viável para instâncias como 4x4 ou superiores. A BFS é eficiente para 3x3, mas se torna custosa em termos de memória em puzzles maiores. IDS é o mais econômico em memória, mas pode ser mais lento conforme a profundidade cresce.

8. Conclusão

O trabalho demonstrou a eficácia de diferentes estratégias de busca no problema n-Puzzle. A busca A* com heurísticas apropriadas se destacou em profundidade e desempenho, oferecendo soluções ótimas de forma eficiente em instâncias pequenas e médias. Além

disso, observou-se que algoritmos como IDS também apresentaram bons resultados em termos de economia de memória e tempo de execução.

A comparação evidenciou a importância do uso de heurísticas adequadas para alcançar bons desempenhos em ambientes com espaço de busca elevado, como nos puzzles maiores (15 e 24 peças). A busca Gulosa se mostrou vantajosa em termos de rapidez, embora não garanta soluções ótimas.

Trabalhos futuros podem incluir otimizações mais profundas nas heurísticas, aplicação de algoritmos paralelos para lidar com instâncias maiores, uso de aprendizado de máquina para ajuste de heurísticas e análise de novos algoritmos informados e não informados para o problema.

9. Referências

RUSSELL, Stuart J.; NORVIG, Peter. Inteligência Artificial. 3. ed. São Paulo: Pearson, 2013.

10. Links

Link do vídeo:

Link do código-fonte: https://github.com/priscilasanto/Problema-n-Puzzle.git