CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS DIVINÓPOLIS
CURSO DE ENGENHARIA DE COMPUTAÇÃO
DISCIPLINA: INTELIGÊNCIA ARTIFICIAL

Análise de Algoritmos de Busca em Grafos: BFS, DFS, Busca Gulosa e A*

Autores:
Bruno Prado Dos Santos

João Francisco Teles da Silva

Professor:
Tiago Alves de Oliveira

Divinópolis – MG
Outubro de 2025

Resumo

Este trabalho apresenta uma análise comparativa de algoritmos de busca em espaço de estados aplicados a um problema de navegação em labirinto. Foram implementados e avaliados quatro algoritmos centrais da Inteligência Artificial: dois de busca não informada, Breadth-First Search (BFS) e Depth-First Search (DFS), e dois de busca informada, Greedy Best-First Search (GBFS) e A*. O desempenho foi medido com base no tempo de execução, nós expandidos, uso de memória e custo do caminho. Adicionalmente, foi investigado o impacto das heurísticas de Manhattan e Euclidiana nos algoritmos informados. Os resultados experimentais confirmaram que o A* oferece o melhor equilíbrio entre eficiência e otimalidade, garantindo o caminho de menor custo. A heurística de Manhattan demonstrou ser computacionalmente mais leve que a Euclidiana, resultando em tempos de execução ligeiramente menores.

Sumário

Resumo **1**

1 Introdução **3**

2 Fundamentação Teórica **3**
2.1 Algoritmos de Busca Não Informada 3
2.1.1 Busca em Largura (BFS) 3
2.1.2 Busca em Profundidade (DFS) 4
2.2 Algoritmos de Busca Informada 5
2.2.1 Heurísticas 5
2.2.2 Busca Gulosa (GBFS) 6
2.2.3 Busca A* 6

3 Descrição do Problema **7**

4 Implementação **7**
4.1 Buscas Não Informadas 8
4.1.1 Busca em largura (BFS) 8
4.1.2 Busca em Profundiade (DFS) 8
4.2 Buscas Informadas 9
4.2.1 Busca Gulosa 9
4.2.2 Busca A* 10

5 Resultados e Discussão **11**
5.1 Tempo de Execução 12
5.2 Nós Expandidos 13
5.3 Memória Máxima Utilizada 14
5.4 Custo do Caminho 15
5.5 Resumo Comparativo dos Algoritmos 15
5.6 Comparativo entre Heurísticas 16

6 Conclusão **18**

1 Introdução

A busca em espaço de estados é um dos conceitos fundamentais da Inteligência Artificial, sendo utilizada para resolver problemas em que é necessário encontrar uma sequência de ações que conduza de um estado inicial até um estado objetivo. Entre as abordagens mais estudadas estão as buscas não informadas, que exploram o espaço de forma sistemática, e as buscas informadas, que utilizam funções heurísticas para guiar o processo de exploração.

O presente trabalho tem como objetivo implementar, aplicar e comparar diferentes algoritmos de busca em um problema de navegação em labirinto. Foram analisados dois algoritmos de busca não informada — Breadth-First Search(BFS) e Depth-First Search(DFS) — e dois algoritmos de busca informada — Greedy Best-First Search(Gulosa) e A*.

2 Fundamentação Teórica

Os algoritmos de busca são responsáveis por explorar o espaço de estados de um problema até encontrar uma solução. Cada estratégia difere na forma como escolhe o próximo nó a ser expandido, afetando diretamente o desempenho e a qualidade da solução encontrada. Nesta seção, será descrita brevemente a teoria por trás dos algoritmos utilizados neste trabalho: Busca em Largura (BFS),Busca em Profundida(DFS),Busca Gulosa e Busca A*.

2.1 Algoritmos de Busca Não Informada

Os algoritmos de busca não informada, também conhecidos como buscas cegas, são aqueles que exploram o espaço de estados sem utilizar qualquer tipo de informação adicional sobre a localização do objetivo.

Entre os principais algoritmos dessa categoria estão a Busca em Largura (Breadth-First Search — BFS) e a Busca em Profundidade (Depth-First Search — DFS).

2.1.1 Busca em Largura (BFS)

O algoritmo Breadth-First Search (BFS) é uma estratégia de busca não informada que explora o espaço de estados de forma sistemática, expandindo primeiro todos os nós de um determinado nível antes de avançar para o próximo, garantindo que os estados mais próximos do ponto de partida sejam analisados antes dos mais distantes.

O funcionamento da BFS baseia-se em explorar todos os vizinhos de um nó antes de prosseguir para os nós do nível seguinte. O processo inicia a partir do estado inicial, que é inserido em uma fila (FIFO – First In, First Out). Em cada iteração, o algoritmo remove o primeiro nó da fila, verifica se ele corresponde ao estado objetivo e, caso contrário, insere na fila todos os seus sucessores ainda não visitados. Esse procedimento se repete até que o objetivo seja encontrado ou até que todos os estados possíveis tenham sido explorados.Além disso, um conjunto de visitados é mantido para evitar revisitar estados já explorados, prevenindo ciclos e redundâncias.

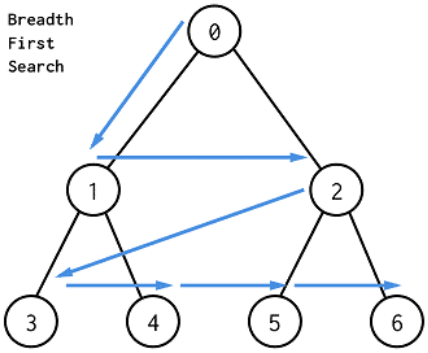

Figura 1: Funcionamento da busca em largura (BFS)

Como mostrado na Figura 1, a BFS expande primeiro todos os nós do nível mais próximo do estado inicial antes de avançar para os níveis seguintes. Esse comportamento garante que, caso o objetivo esteja presente em um dos níveis anteriores, ele será encontrado antes que o algoritmo explore estados mais distantes.

2.1.2 Busca em Profundidade (DFS)

O algoritmo Depth-First Search (DFS) é uma estratégia de busca não informada que explora o espaço de estados priorizando a profundidade da árvore de busca. Diferentemente da BFS, que expande todos os nós de um mesmo nível antes de avançar, a DFS segue um caminho até o final possível antes de retroceder para explorar caminhos alternativos.

O funcionamento da DFS baseia-se em uma estrutura de pilha (LIFO – Last In, First Out). O algoritmo inicia no estado inicial, expandindo o primeiro nó e inserindo seus sucessores na pilha. A cada iteração, o nó mais recentemente adicionado é removido e expandido. Caso o nó corresponda ao estado objetivo, a busca é encerrada. Se atingir um caminho sem saída, o algoritmo retrocede para explorar outros ramos da árvore.

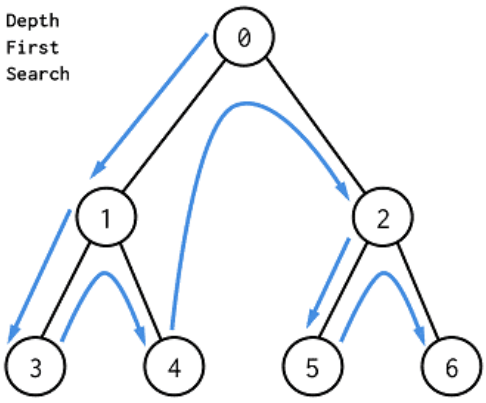

Figura 2: Funcionamento da busca em profundidade (DFS)

Como ilustrado na Figura 2, a DFS percorre os nós sempre indo o mais fundo possível antes de retornar para níveis anteriores, explorando os caminhos em profundidade.

2.2 Algoritmos de Busca Informada

As buscas informadas, também chamadas de buscas heurísticas, utilizam informações adicionais sobre o problema para guiar a exploração do espaço de estados. Ao contrário das buscas cegas, que exploram de forma sistemática, os algoritmos informados utilizam funções heurísticas para estimar a proximidade de cada estado em relação ao objetivo, priorizando a expansão dos nós mais promissores e, assim, aumentando a eficiência da busca.

2.2.1 Heurísticas

Distância de Manhattan A heurística de Manhattan calcula a soma das diferenças absolutas entre as coordenadas do nó atual (x, y) e do objetivo (x_{goal}, y_{goal}). Essa abordagem considera apenas movimentos em linha reta nas direções horizontal e vertical, sendo adequada para labirintos ou grades onde os movimentos diagonais não são permitidos.

$$h_{\text{Manhattan}}(n) = |x - x_{goal}| + |y - y_{goal}|$$

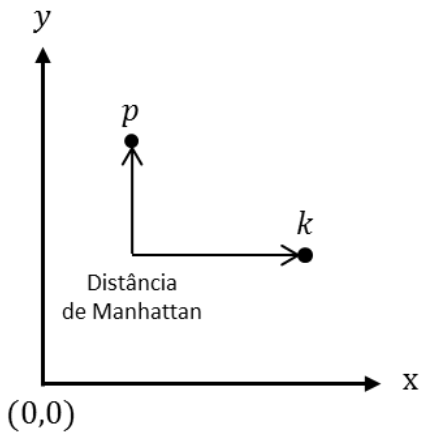

Figura 3: Heurística de Manhattan: soma das diferenças horizontais e verticais

Distância Euclidiana A heurística Euclidiana calcula a distância em linha reta entre o nó atual e o objetivo, fornecendo uma estimativa direta do caminho restante. É útil em ambientes que permitem movimentos diagonais, mas envolve operações de raiz quadrada, resultando em maior custo computacional.

$$h_{\text{Euclidiana}}(n) = \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2}$$

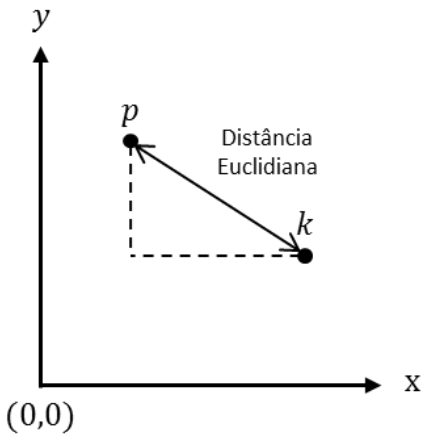

Figura 4: Heurística Euclidiana: distância em linha reta até o objetivo

2.2.2 Busca Gulosa (GBFS)

A Busca Gulosa é um algoritmo de busca informada, que utiliza uma função heurística $h(n)$ para estimar o custo de um nó até o objetivo. Diferentemente das buscas não informadas, como BFS e DFS, o algoritmo Guloso orienta sua exploração de acordo com o valor da heurística, sempre escolhendo expandir o nó que aparenta estar mais próximo do objetivo.

O funcionamento baseia-se em uma fila de prioridade, onde os nós são ordenados pelo valor de $h(n)$. A cada iteração, o nó com menor valor heurístico é removido e expandido. Seus sucessores são avaliados pela função h e inseridos novamente na fila, repetindo o processo até que o estado objetivo seja alcançado.

Embora nem sempre encontre o caminho ótimo, a Busca Gulosa tende a ser eficiente em termos de tempo, pois direciona a exploração para regiões promissoras do espaço de busca.

2.2.3 Busca A*

O algoritmo A* combina as vantagens da busca de custo uniforme e da busca gulosa, sendo também uma busca informada. Ele utiliza uma função de avaliação definida como:

$$f(n) = g(n) + h(n)$$

onde:

- $g(n)$ representa o custo real do caminho percorrido até o nó n;
- $h(n)$ é a estimativa heurística do custo restante até o objetivo.

O A* expande o nó com o menor valor de $f(n)$, equilibrando entre explorar caminhos promissoress (via $h(n)$) e caminhos baratos até o momento (via $g(n)$).

Se a heurística $h(n)$ for admissível (isto é, nunca superestima o custo real até o objetivo), o algoritmo A* é completo e ótimo, garantindo o caminho de menor custo possível.

3 Descrição do Problema

O problema proposto consiste em encontrar o caminho entre um ponto inicial e um ponto objetivo dentro de um labirinto bidimensional representado por uma matriz. O ambiente é composto por células que podem conter diferentes tipos de elementos, conforme a convenção:

- `S`: posição inicial (*start*);
- `G`: posição objetivo (*goal*);
- `#`: parede ou obstáculo;
- `.`: célula livre para movimentação.

O ambiente de teste específico utilizado para todas as execuções e comparações de algoritmos foi o `labirinto3.txt`, cuja estrutura é apresentada na Figura 5.

```
S.#########
..........#
.#.######.#
.#....#...#
.####.#.#.#
....G.#.#.#
#####.###.#
#.....#...#
###########
```

Figura 5: Representação do labirinto utilizado nos testes (`labirinto3.txt`).

As ações possíveis são os movimentos nas quatro direções principais — Norte, Sul, Leste e Oeste — desde que o movimento seja válido, isto é, não ultrapasse os limites do labirinto nem atravesse paredes. Cada ação possui custo unitário.

A representação interna do problema foi implementada como uma matriz (ou grade), na qual cada célula corresponde a uma posição do agente.

O objetivo é encontrar um caminho viável entre o ponto inicial e o ponto final, respeitando as restrições do ambiente e minimizando o custo total do percurso. Além da obtenção do caminho, os algoritmos foram comparados quanto à eficiência e qualidade das soluções obtidas.

4 Implementação

Os algoritmos apresentados neste trabalho foram implementados em Python, tendo como base os pseudo-algoritmos descritos no livro *Artificial Intelligence: A Modern Approach* de Russell e Norvig, adaptados para lidar com labirintos bidimensionais e estruturas de dados da linguagem. A implementação do código priorizou a clareza, modularidade e a possibilidade de visualização do caminho durante a execução.

4.1 Buscas Não Informadas

4.1.1 Busca em largura (BFS)

O BFS utiliza uma fila (`deque`) para gerenciar a fronteira, explorando os nós em ordem de inserção. Um conjunto `visited` garante que células já exploradas não sejam revisitadas. O caminho é construído progressivamente ao adicionar novas posições válidas, e o algoritmo registra o número de nós expandidos e a memória máxima utilizada.

Algorithm 1: Breadth-First-Search(*problem*)

Entrada: *problem*
Saída: a solution or failure

```
1  node ← nó with STATE = problem.INITIAL-STATE, PATH-COST = 0;
2  if GOAL-TEST(node.STATE) then
3      return SOLUTION(node);
4  frontier ←a FIFO queue with node as the only element;
5  explored ← an empty set;
6  while true do
7      if EMPTY(frontier) then
8          return failure;
9      node ← POP(frontier) ;   // chooses the shallowest node in frontier
10     add node.STATE to explored;
11     foreach action in ACTIONS(node.STATE) do
12         child ← CHILD-NODE(problem, node, action);
13         if child.STATE is not in explored or frontier then
14             if GOAL-TEST(child.STATE) then
15                 return SOLUTION(child);
16             frontier ← INSERT(child, frontier);
```

4.1.2 Busca em Profundiade (DFS)

O DFS implementa a fronteira como uma pilha (`list`), explorando sempre o último nó inserido (LIFO). As células visitadas são rastreadas pelo conjunto `visited`, e o caminho é atualizado conforme a exploração avança. Também mede o número de nós expandidos e o uso máximo de memória durante a execução.

Algorithm 2: Depth-First-Search(*problem*)

Entrada: *problem*
Saída: a solution or failure

```
1  node ← nó with STATE = problem.INITIAL-STATE, PATH-COST = 0;
2  frontier ← a LIFO stack with node as the only element;
3  explored ← an empty set;
4  while true do
5      if EMPTY(frontier) then
6          return failure;
7      node ← POP(frontier) ;        // chooses the deepest node in frontier
8      if GOAL-TEST(node.STATE) then
9          return SOLUTION(node);
10     add node.STATE to explored;
11     foreach action in ACTIONS(node.STATE) do
12         child ← CHILD-NODE(problem, node, action);
13         if child.STATE is not in explored or frontier then
14             frontier ← INSERT(child, frontier);
```

4.2 Buscas Informadas

4.2.1 Busca Gulosa

A Busca Gulosa utiliza uma `PriorityQueue` para ordenar os nós de acordo com a heurística. Cada nó processado é registrado em `visited` para evitar revisitas, e o caminho é atualizado dinamicamente ao expandir nós válidos. O algoritmo acompanha os nós expandidos e a memória máxima usada, mas considera apenas o custo heurístico na prioridade.

Algorithm 3: Greedy-Best-First-Search(*problem*)

Entrada: *problem*
Saída: a solution or failure

```
1  node ← nó with STATE = problem.INITIAL-STATE, PATH-COST = 0;
2  frontier ← a priority queue ordered by h(n) with node as the only element;
3  explored ← an empty set;
4  while true do
5      if EMPTY(frontier) then
6          return failure;
7      node ← POP(frontier) ;        // chooses the node with lowest h(n) in
        frontier
8      if GOAL-TEST(node.STATE) then
9          return SOLUTION(node);
10     add node.STATE to explored;
11     foreach action in ACTIONS(node.STATE) do
12         child ← CHILD-NODE(problem, node, action);
13         if child.STATE is not in explored or frontier then
14             frontier ← INSERT(child, frontier);
15         else if child.STATE is in frontier with higher h(n) then
16             replace that frontier node with child;
```

4.2.2 Busca A*

O A* combina custo do caminho (g) e heurística (h) em uma `PriorityQueue` com prioridade $f = g + h$. O conjunto `visited` evita nós repetidos, e o caminho é construído incrementalmente à medida que novos nós são expandidos. O algoritmo retorna o caminho encontrado, o total de nós expandidos e a memória máxima utilizada.

Algorithm 4: A*-Search(*problem*)

Entrada: *problem*
Saída: a solution or failure

```
node ← nó with STATE = problem.INITIAL-STATE, PATH-COST = 0;
frontier ← a priority queue ordered by f(n) = g(n) + h(n) with node as the only element;
explored ← an empty set;
while true do
    if EMPTY(frontier) then
        return failure;
    node ← POP(frontier) ;        // chooses the node with lowest f(n) in frontier
    if GOAL-TEST(node.STATE) then
        return SOLUTION(node);
    add node.STATE to explored;
    foreach action in ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action);
        if child.STATE is not in explored or frontier then
            frontier ← INSERT(child, frontier);
        else if child.STATE is in frontier with higher f(n) then
            replace that frontier node with child;
```

5 Resultados e Discussão

Esta seção detalha os resultados experimentais obtidos pela aplicação dos algoritmos de busca — BFS, DFS, Greedy Best-First Search (GBFS) e A* — no ambiente de labirinto descrito. A análise comparativa foca nas seguintes métricas de desempenho: tempo de execução, quantidade de nós expandidos, memória máxima utilizada e custo do caminho encontrado. O estudo inclui, ainda, uma avaliação específica sobre a influência das heurísticas de Manhattan e Euclidiana no desempenho dos algoritmos informados (GBFS e A*).

5.1 Tempo de Execução

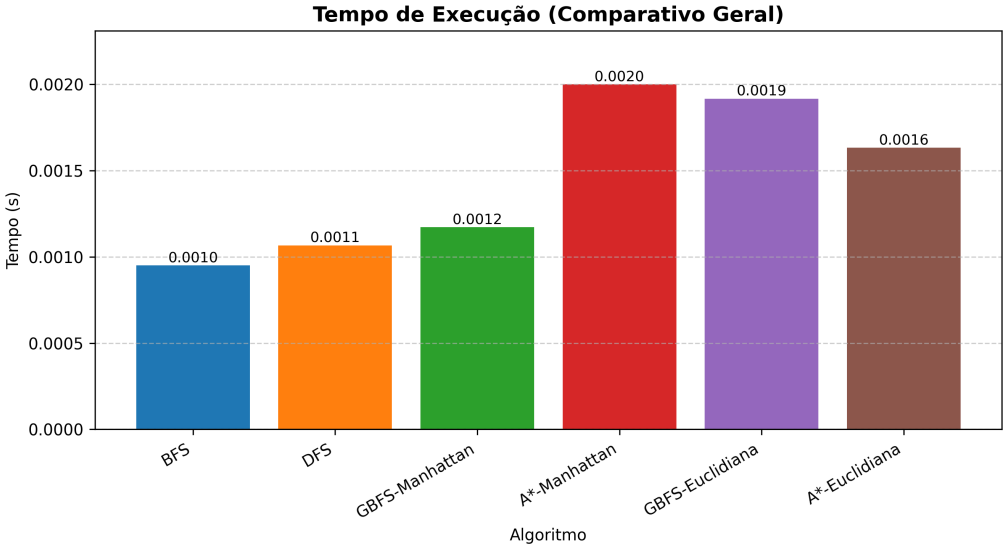

Figura 6: Tempo de execução — Comparativo geral

O gráfico da Figura 6 mostra que o tempo de execução dos algoritmos foi bastante próximo entre si, variando entre 0.0008 s e 0.0014 s. A Busca em Largura (BFS) apresentou um tempo ligeiramente menor que os algoritmos informados, devido à sua simplicidade e ausência de cálculos heurísticos. A Busca em Profundidade (DFS) teve desempenho similar, reforçando que ambos são eficientes em tempo para problemas de pequeno porte como o labirinto em questão analisado.

Já os algoritmos GBFS e A* apresentaram tempos um pouco maiores, reflexo do custo adicional de cálculo da função heurística. A versão com heurística Euclidiana foi a mais lenta, o que é consistente com a teoria, já que a distância Euclidiana envolve operações de raiz quadrada, enquanto a Manhattan utiliza apenas somas e diferenças absolutas. Esse comportamento confirma a relação direta entre a complexidade da heurística e o tempo computacional total.

5.2 Nós Expandidos

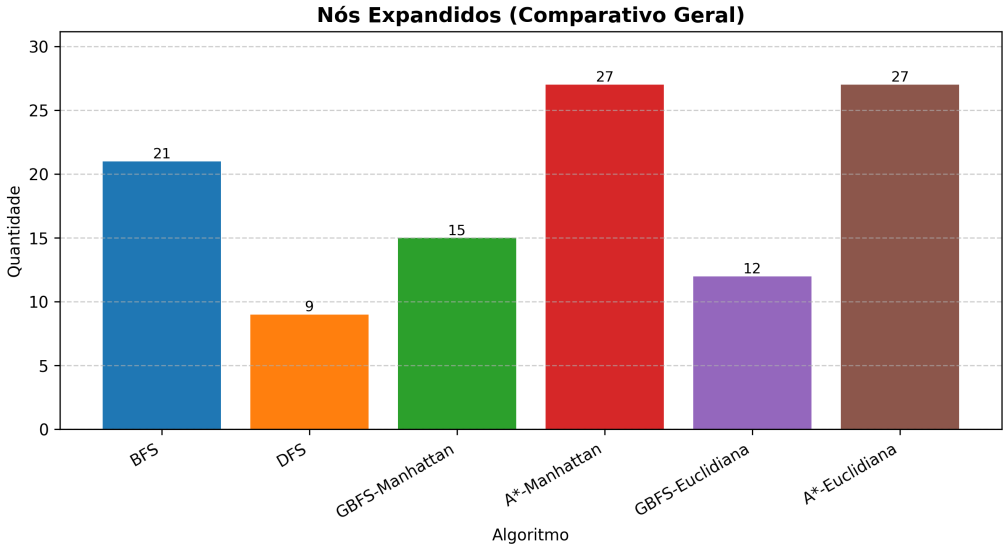

Figura 7: Nós expandidos — Comparativo geral

A Figura 7 mostra o número total de nós expandidos por cada algoritmo. Observa-se que o DFS foi o que expandiu menos nós, seguido pelos algoritmos GBFS, enquanto o BFS e o A* expandiram uma quantidade significativamente maior. Esse padrão está alinhado com a literatura: o DFS tende a explorar poucos nós porque se aprofunda rapidamente em um único caminho, ainda que isso possa levá-lo a becos sem saída. O GBFS, por se guiar pela heurística, também reduz o número de expansões em comparação com buscas cegas, porém à custa da perda de otimalidade.

O A*, por outro lado, realiza mais expansões que o GBFS, uma vez que precisa considerar tanto o custo acumulado $g(n)$ quanto a estimativa heurística $h(n)$, buscando um equilíbrio entre exploração e custo total. Esse comportamento explica a maior quantidade de nós processados, mas garante a obtenção do caminho ótimo. O BFS, por sua vez, expande todos os nós de cada nível antes de prosseguir, o que naturalmente resulta em uma das maiores quantidades de expansões observadas.

Em síntese, os resultados obtidos são coerentes com o comportamento teórico de cada algoritmo: o DFS é o mais econômico em nós e memória, porém não confiável quanto à otimalidade; o GBFS é eficiente, mas heurístico e incompleto; o BFS e o A* são completos e ótimos, ainda que à custa de maior esforço computacional.

5.3 Memória Máxima Utilizada

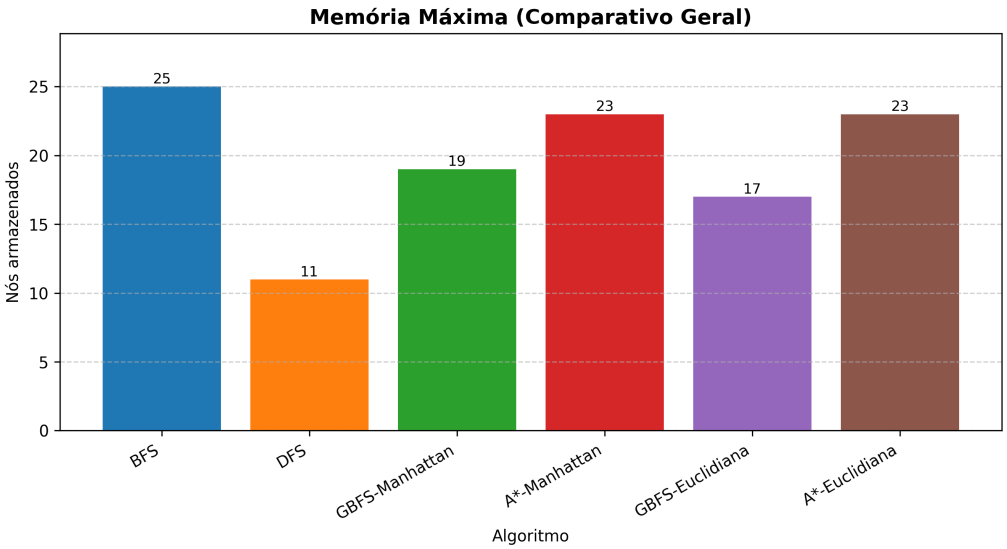

Figura 8: Memória máxima utilizada — Comparativo geral

A Figura 8 apresenta a comparação entre os algoritmos em termos de memória máxima utilizada, medida pela quantidade de nós simultaneamente armazenados na estrutura de dados durante a execução. Observa-se que a Busca em Largura (BFS) demandou a maior quantidade de memória, atingindo aproximadamente 25 nós armazenados. Esse resultado está em conformidade com a teoria, uma vez que o BFS mantém todos os nós do nível atual antes de expandir o próximo, o que implica alto consumo de memória, especialmente em ambientes com múltiplos caminhos possíveis.

Por outro lado, a Busca em Profundidade (DFS) foi a mais econômica, armazenando em torno de 11 nós, pois mantém apenas o caminho atual até o nó sendo explorado. Os algoritmos informados (GBFS e A*) apresentaram consumo intermediário. O GBFS, por utilizar apenas a heurística $h(n)$ e não considerar o custo acumulado $g(n)$, armazenou uma quantidade levemente menor de nós que o A*, que precisa manter estruturas mais completas para garantir a otimalidade. Dessa forma, a ordem observada — BFS $>$ A* $>$ GBFS $>$ DFS — reflete exatamente o comportamento teórico esperado quanto ao uso de memória.

5.4 Custo do Caminho

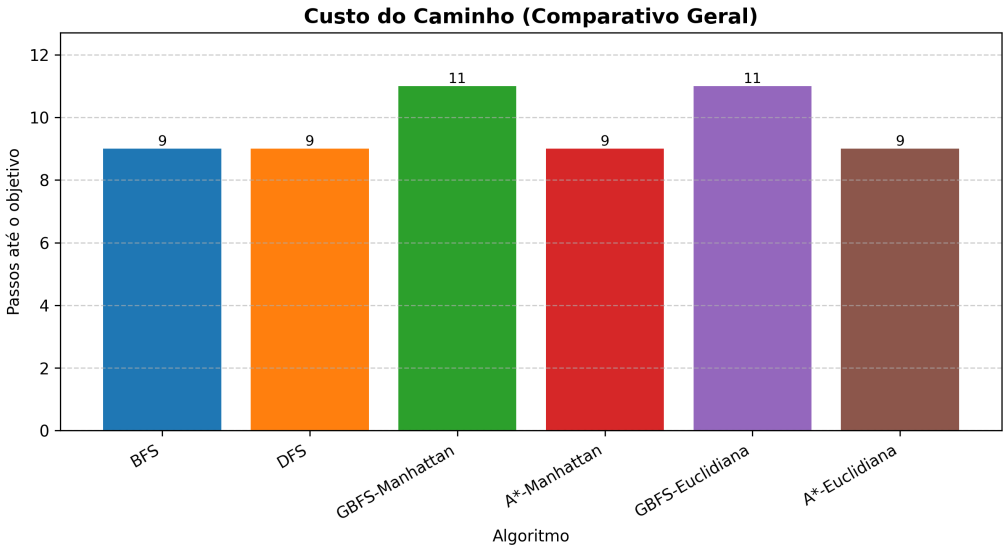

Figura 9: Custo do caminho — Comparativo geral

Na Figura 9 é possível observar que os algoritmos BFS, DFS e A* (tanto com heurística Manhattan quanto Euclidiana) encontraram caminhos com o mesmo custo total, aproximadamente 9 passos até o objetivo. Esse resultado é teoricamente consistente, pois o BFS e o A* garantem a optimalidade do caminho em grafos não ponderados quando a heurística é admissível e consistente, o que é o caso das distâncias Manhattan e Euclidiana neste contexto.

A igualdade do custo obtido pelo DFS é apenas circunstancial: embora ele tenha alcançado o mesmo resultado neste labirinto simples, esse algoritmo não assegura caminhos mínimos, podendo facilmente retornar soluções subótimas em outros cenários. Já os algoritmos GBFS apresentaram um custo de 11 passos, maior que os demais, o que está de acordo com a teoria — o GBFS é uma busca gulosa que se orienta apenas pela estimativa heurística, podendo escolher caminhos aparentemente promissores, mas que não são os mais curtos.

Assim, os resultados confirmam o comportamento esperado: o BFS e o A* encontraram o caminho ótimo, o DFS obteve um bom resultado apenas por coincidência, e o GBFS produziu uma solução mais longa, típica de sua natureza heurística não informada pelo custo real acumulado.

5.5 Resumo Comparativo dos Algoritmos

Para consolidar os resultados discutidos, a Tabela 1 resume as principais medições de desempenho obtidas experimentalmente. Ela permite observar, de forma quantitativa, a relação entre eficiência, custo e consumo de recursos de cada método.

Tabela 1: Comparativo geral dos algoritmos de busca

Algoritmo	Heurística	Tempo (s)	Nós Exp.	Memória	Custo
BFS	—	0.0008	21	25	9
DFS	—	0.0009	9	11	9
GBFS	Manhattan	0.0011	15	19	11
A*	Manhattan	0.0011	27	23	9
GBFS	Euclidiana	0.0014	12	17	11
A*	Euclidiana	0.0014	27	23	9

A partir dos dados da Tabela 1, observa-se claramente o equilíbrio do **A***, que combina a optimalidade da BFS com uma eficiência próxima da GBFS. O **DFS** destaca-se pelo baixo uso de memória e nós expandidos, enquanto a **BFS** se sobressai na completude e na garantia de menor custo, ainda que com maior gasto espacial.

5.6 Comparativo entre Heurísticas

A seguir, as Figuras 10 a 13 apresentam o comportamento específico das heurísticas Manhattan e Euclidiana nos algoritmos informados.

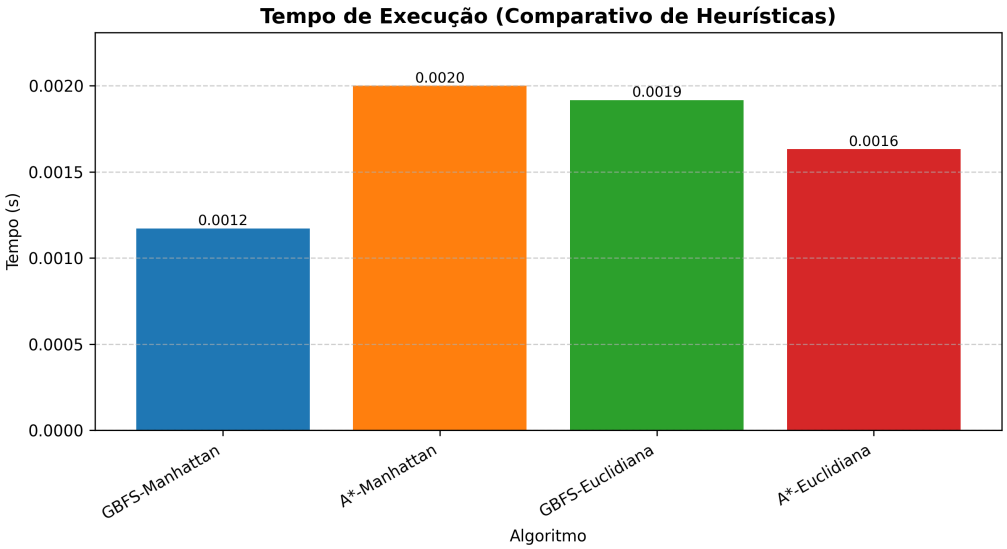

Figura 10: Tempo de execução — Comparativo de heurísticas

O tempo de execução foi ligeiramente maior para a heurística Euclidiana, tanto em GBFS quanto em A*. Isso ocorre porque a distância Euclidiana requer operações de raiz quadrada, enquanto a Manhattan é baseada apenas em somas, resultando em menor custo computacional.

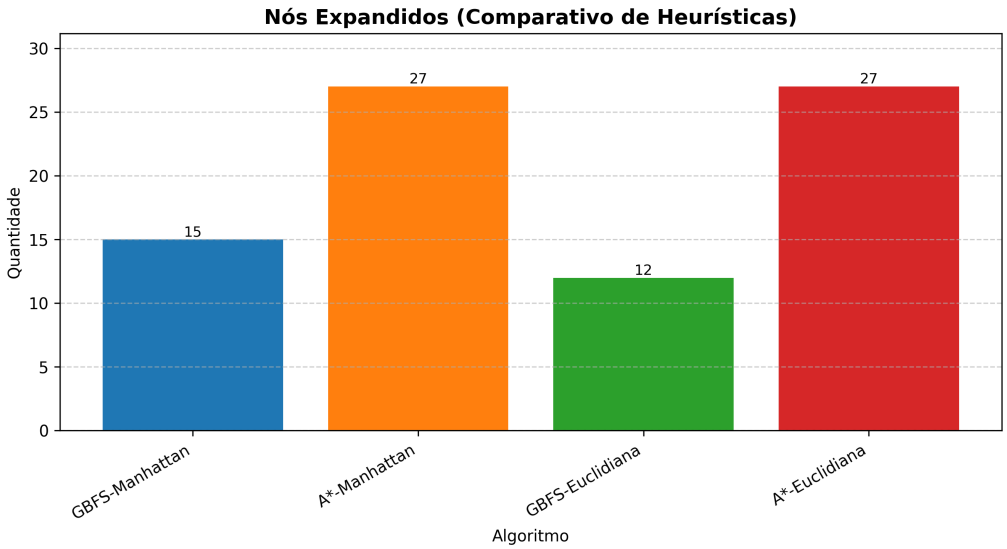

Figura 11: Nós expandidos — Comparativo de heurísticas

Na Figura 11, observa-se que a heurística Euclidiana tende a reduzir o número de nós expandidos na GBFS, já que fornece uma estimativa mais precisa da distância ao objetivo. Em contrapartida, no A*, as duas heurísticas produziram resultados idênticos em número de expansões e custo final, mostrando que ambas são **admissíveis** e, portanto, garantem a optimalidade.

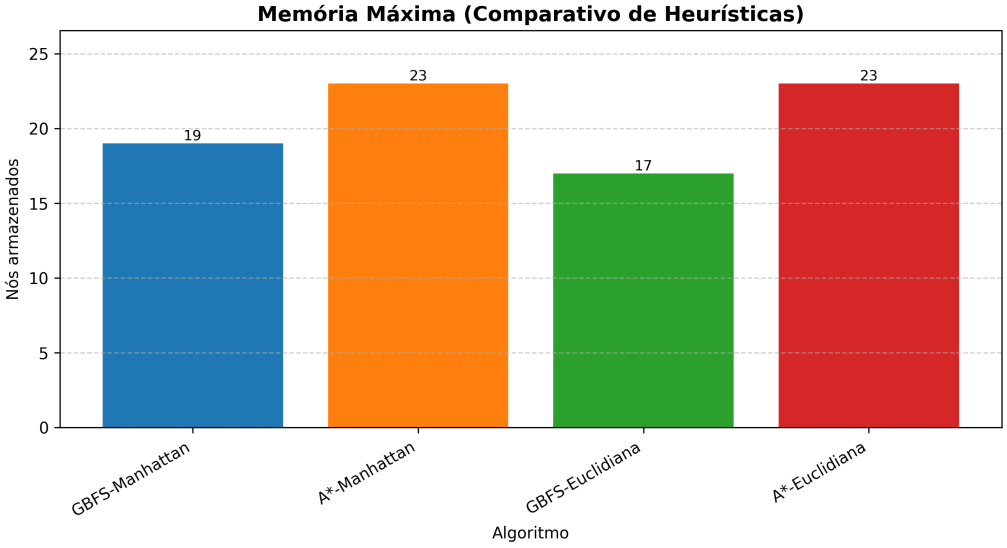

Figura 12: Memória máxima — Comparativo de heurísticas

Os resultados de memória máxima também foram semelhantes entre as heurísticas, com pequenas variações relacionadas à ordem de expansão dos nós. Essa diferença sutil é esperada e não altera o comportamento global dos algoritmos.

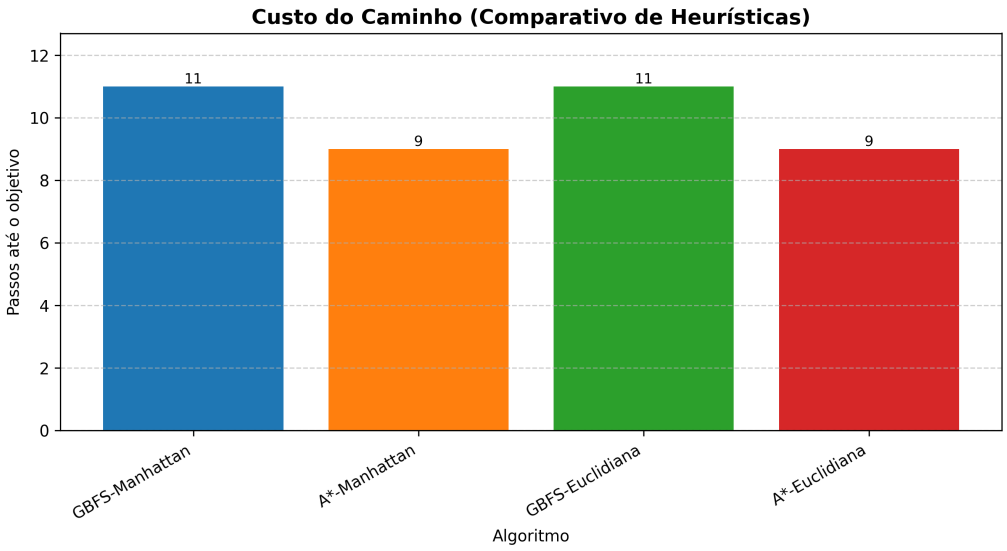

Figura 13: Custo do caminho — Comparativo de heurísticas

Por fim, observa-se que tanto na GBFS quanto no A* o custo final do caminho foi o mesmo ao utilizar Manhattan ou Euclidiana, o que reforça que, para o tipo de ambiente testado (grade ortogonal), ambas as heurísticas são adequadas. A diferença está apenas na precisão e no custo de cálculo, não afetando a qualidade da solução.

6 Conclusão

A análise comparativa realizada neste trabalho permitiu validar experimentalmente as características teóricas dos algoritmos de busca. Foi constatado que a Busca em Largura (BFS), embora garanta a otimalidade do caminho (resultando no custo 9), o faz ao preço de um consumo de memória significativamente elevado (25 nós), decorrente de sua exploração exaustiva nível a nível. Em contrapartida, a Busca em Profundidade (DFS) demonstrou ser a mais econômica em recursos de memória (11 nós), contudo, a obtenção do caminho ótimo neste cenário foi circunstancial, visto que o algoritmo não oferece garantias intrínsecas de otimalidade.

No âmbito das buscas informadas, o algoritmo A* destacou-se por apresentar o equilíbrio mais eficaz entre eficiência e garantia de solução. Utilizando tanto a heurística de Manhattan quanto a Euclidiana, o A* foi capaz de encontrar o caminho de menor custo (custo 9), equiparando-se à BFS, mas com uma exploração de estados mais direcionada e, em geral, mais eficiente. A Busca Gulosa (GBFS), por sua vez, embora tenha processado menos nós em algumas configurações, falhou em encontrar a solução ótima (resultando no custo 11), confirmando sua natureza que prioriza a estimativa heurística imediata em detrimento do custo real acumulado.

Por fim, a comparação direta entre as heurísticas revelou que a Distância de Manhattan proporcionou um desempenho superior em termos de tempo de execução. Isso reflete diretamente sua menor complexidade computacional (baseada em somas) quando comparada à Distância Euclidiana (que exige cálculos de raiz quadrada). Embora ambas tenham se mostrado admissíveis, a escolha da heurística demonstrou ser um fator relevante para a otimização do tempo de processamento. Deste modo, o trabalho conclui que o A*, especialmente quando combinado com a heurística de Manhattan, representa a solução

mais balanceada para o problema proposto.

Referências

- RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial: Uma Abordagem Moderna**. 3. ed. Rio de Janeiro: Elsevier, 2013.
- SANTOS, Bruno Prado dos; SILVA, João Francisco Teles da Silva. *Algoritmos de Busca em Labirintos: Implementação e Comparativo Experimental*. Disponível em: `https://github.com/joaofranciscoteles/Trabalhos-Busca-e-8-Rainhas-.git`. Acesso em: 22 out. 2025.