Caminho Hamiltoniano

Marcos Adriano

January 2026

1 Introdução

O Caminho Hamiltoniano é um problema clássico da Teoria dos Grafos que consiste em determinar se existe, em um grafo, um caminho que visite todos os vértices exatamente uma vez. Diferentemente do Ciclo Hamiltoniano, não é exigido que o caminho retorne ao vértice inicial[5]. Ele pode ser definido da seguinte forma:

Entrada: Um grafo G = (V, E).
Problema de decisão: Existe um caminho simples que visita todos os vértices de G exatamente uma vez?
Saída: SIM se um tal caminho existe; NÃO caso contrário

Esse problema surge em diversas situações práticas, como redes de computadores [4][1], teoria dos jogos[2] e planejamento de rotas[3], onde se deseja percorrer todos os pontos exatamente uma vez sem repetição. Apesar de sua definição simples, o Caminho Hamiltoniano apresenta grande complexidade computacional, tornando-se um objeto central de estudo tanto teórico quanto aplicado.

2 Redução de Ciclo Hamiltoniano para Caminho Hamiltoniano

O problema do Caminho Hamiltoniano é NP-Completo, para demonstrar isso, podemos tomar outro problema conhecido que é NP-Completo e realizar a redução desse problema para o HP. O problema escolhido será o Ciclo Hamiltoniano, cuja prova de NP-completude é demonstrada em [6].

Para demonstrar que o problema do Caminho Hamiltoniano (HP) é NP-Completo, utilizamos uma redução polinomial a partir do problema do Ciclo Hamiltoniano (HC), que é conhecido como NP-Completo.

Dada uma instância de HC, representada por um grafo $G = (V, E)$, escolhe-se um vértice $v \in V$ e constrói-se um novo grafo G' a partir de G. O vértice v é substituído por dois vértices v_1 e v_2, e são adicionados dois novos vértices s e t, que forçam o início e o fim do caminho.

A construção é feita de modo que qualquer caminho hamiltoniano em G' precise iniciar em s, passar por v_1, visitar todos os demais vértices e terminar em v_2 e t. Dessa forma, existe um caminho hamiltoniano em G' se, e somente se, existir um ciclo hamiltoniano no grafo original G.

A transformação descrita adiciona apenas um número constante de vértices e arestas, sendo portanto realizada em tempo polinomial. Assim, concluímos que $HC \leq_p HP$, o que implica que o problema do Caminho Hamiltoniano é NP-Completo.

3 Soluções Possíveis

Devido à sua natureza NP-Completa, não se conhece atualmente nenhum algoritmo que resolva o problema do Caminho Hamiltoniano em tempo polinomial para todos os grafos, a menos que P = NP. Ainda assim, diversas abordagens têm sido desenvolvidas, cada uma adequada a diferentes contextos e tamanhos de instância.

3.1 Busca Exaustiva e Backtracking

A abordagem mais direta consiste em testar todas as possíveis permutações de vértices e verificar se alguma delas forma um caminho válido. Essa estratégia possui complexidade exponencial, da ordem de $O(n!)$, tornando-se inviável para grafos moderadamente grandes. Técnicas de *backtracking* permitem podar parte do espaço de busca ao interromper a exploração de caminhos que já violam as restrições do problema. Essa abordagem é mais adequada para grafos pequenos ou para fins didáticos.

3.2 Programação Dinâmica

Uma solução clássica baseada em programação dinâmica utiliza a técnica de *bitmasking*, armazenando estados que representam subconjuntos de vértices visitados e o último vértice do caminho. Essa abordagem resolve o problema em tempo $O(n^2 \cdot 2^n)$, sendo significativamente mais eficiente que a busca exaustiva. Apesar disso, ainda possui custo exponencial e é indicada apenas para grafos de pequeno porte, tipicamente com até algumas dezenas de vértices.

3.3 Reduções para Outros Problemas

O problema do Caminho Hamiltoniano pode ser reduzido a outros problemas bem estudados, como o problema de satisfatibilidade booleana (SAT). Nessa abordagem, o grafo é codificado como uma fórmula lógica cujas soluções correspondem a caminhos hamiltonianos. A resolução fica a cargo de *SAT solvers* modernos, que utilizam heurísticas avançadas, aprendizado de conflitos e poda eficiente. Essa estratégia é particularmente eficaz em instâncias estruturadas, onde os solvers conseguem explorar regularidades do problema.

3.4 Heurísticas e Metaheurísticas

Para instâncias grandes, onde métodos exatos se tornam impraticáveis, são utilizadas heurísticas e metaheurísticas, como algoritmos genéticos, busca local, simulated annealing e colônia de formigas. Essas técnicas não garantem a obtenção de uma solução ótima ou mesmo a existência de um caminho hamiltoniano, mas frequentemente produzem boas soluções em tempo razoável. Elas são amplamente aplicadas em problemas reais de planejamento e otimização de rotas.

3.5 Casos Especiais de Grafos

Em determinadas classes de grafos, o problema pode ser resolvido de forma eficiente. Por exemplo, em grafos completos é possível obter a solução em O(1) simplesmente verificando o número de vértices e arestas, além disso, grafos com certas condições de grau mínimo ou grafos de intervalos admitem algoritmos polinomiais para a existência de caminhos hamiltonianos. Nesses casos, propriedades estruturais específicas do grafo são exploradas para reduzir a complexidade do problema.

Em resumo, a escolha da abordagem depende diretamente do tamanho do grafo, de suas propriedades estruturais e do contexto da aplicação. Métodos exatos são preferíveis para instâncias pequenas ou quando a garantia de correção é essencial, enquanto heurísticas são mais adequadas para instâncias grandes e aplicações práticas.

4 Implementação

Para a implementação do problema do Caminho Hamiltoniano foram escolhidas duas abordagens clássicas: *Backtracking* e *Programação Dinâmica*. Ambas as soluções foram implementadas utilizando grafos representados por listas de adjacência e retornam uma resposta booleana indicando a existência ou não de um Caminho Hamiltoniano. O backtracking explora recursivamente as possíveis sequências de vértices, aplicando podas sempre que uma restrição do problema é violada, enquanto a programação dinâmica utiliza a técnica de *bitmasking* para armazenar estados intermediários e evitar recomputações. A escolha dessas abordagens permite comparar estratégias distintas para a resolução de um problema NP-Completo, cujos resultados experimentais são apresentados na Seção 5.

4.1 Backtracking

Para implementação do backtracking foram implementadas 2 funções [1], uma função principal chamada hasHamiltonian (Algoritmo 1) que percorre todos os vértices do gráfico e vê se existe um caminho a partir de cada um deles chamando

[1] A implementação real, e arquivos de instâncias se encontram em `https://github.com/drickchote/hamiltonian_path`

checkPath. A função checkPath (Algoritmo 2) tenta construir um caminho com o mesmo tamanho do número de vértices, logo, um caminho hamiltoniano.

Algorithm 1 Verificação de Caminho Hamiltoniano

```
1:  function HasHamiltonian(G, n)
2:      if n = 1 then return true
3:      end if
4:      if n ≠ |V(G)| then return false
5:      end if
6:      for all v ∈ V(G) do
7:          if CheckPath(G, v, 1, ∅) then return true
8:          end if
9:      end for
         return false
10: end function
```

Algorithm 2 Busca Recursiva para Caminho Hamiltoniano

```
1:  function CheckPath(G, v, visitedCount, visited)
2:      visited ← visited ∪ {v}
3:      if visitedCount = |V(G)| then return true
4:      end if
5:      for all u ∈ Adj(v) do
6:          if u ∉ visited then
7:              if CheckPath(G, u, visitedCount + 1, visited) then return
    true
8:              end if
9:          end if
10:     end for
11:     visited ← visited \ {v} return false
12: end function
```

A poda do backtracking ocorre no Algoritmo 2 linha 6. Nela, não seguimos pelos caminhos que já foram visitados anteriormente. Além disso, o vértice é removido do caminho principal, e outro caminho é tentado.

4.2 Programação Dinâmica

A abordagem de programação dinâmica apresentada no Algoritmo 3 utiliza a técnica de *bitmasking* para representar subconjuntos de vértices visitados. Inicialmente, o conjunto de vértices e o número total de vértices do grafo são definidos (linhas 1 e 2). Casos triviais, nos quais o grafo possui zero ou um vértice, são tratados diretamente (linhas 3 e 4).

Em seguida, os rótulos originais dos vértices são mapeados para índices inteiros no intervalo $0 \dots n-1$ (linhas 5 a 7), o que permite a representação

eficiente dos subconjuntos por meio de máscaras de bits. A lista de adjacência indexada é então construída a partir desse mapeamento (linhas 8 a 14).

A estrutura de programação dinâmica dp é inicializada na linha 15, onde cada estado $dp[mask]$ representa os possíveis vértices finais de caminhos que visitam exatamente os vértices indicados por $mask$. Os estados iniciais, correspondentes a caminhos formados por um único vértice, são definidos nas linhas 16 a 18.

As transições da programação dinâmica ocorrem nas linhas 19 a 30. Para cada subconjunto de vértices representado por $mask$, o algoritmo tenta estender os caminhos existentes adicionando vértices adjacentes ainda não visitados, gerando novos estados válidos.

Por fim, o algoritmo verifica se existe algum estado que corresponda a um subconjunto contendo todos os vértices do grafo (linhas 31 a 34). Caso exista, conclui-se que o grafo possui um Caminho Hamiltoniano; caso contrário, a resposta é negativa.

Algorithm 3 Caminho Hamiltoniano usando Programação Dinâmica com Bitmask

Require: Grafo $G = (V, E)$ representado por lista de adjacência
Ensure: **true** se existe um Caminho Hamiltoniano em G, caso contrário **false**

$V \leftarrow$ lista de vértices de G
$n \leftarrow |V|$
if $n = 0$ **ou** $n = 1$ **then return true**
end if
▷ Mapeia rótulos dos vértices para índices $0 \dots n-1$
for $i \leftarrow 0$ até $n-1$ **do**
    $idx[V[i]] \leftarrow i$
end for
▷ Constrói lista de adjacência indexada
for $v \in V$ **do**
    for $u \in Adj(v)$ **do**
        if $u \in V$ **then**
            $adj[idx[v]] \leftarrow adj[idx[v]] \cup \{idx[u]\}$
        end if
    end for
end for
▷ $dp[mask]$ é um bitset indicando os vértices finais possíveis
Inicialize $dp[mask] \leftarrow 0$ para todo $mask \in [0, 2^n - 1]$
▷ Estados iniciais: caminhos de tamanho 1
for $v \leftarrow 0$ até $n-1$ **do**
    $dp[2^v] \leftarrow dp[2^v] \cup \{v\}$
end for
▷ Transições da DP
for $mask \leftarrow 0$ até $2^n - 1$ **do**
    if $dp[mask] = \emptyset$ **then**
    end if
    for cada vértice $v \in dp[mask]$ **do**
        for cada $u \in adj[v]$ **do**
            if $u \notin mask$ **then**
                $nextMask \leftarrow mask \cup \{u\}$
                $dp[nextMask] \leftarrow dp[nextMask] \cup \{u\}$
            end if
        end for
    end for
end for
$fullMask \leftarrow 2^n - 1$
if $dp[fullMask] \neq \emptyset$ **then return true**
elsereturn false
end if

5 Experimentos

Os testes foram conduzidos considerando diferentes classes de grafos, de modo a evidenciar o comportamento dos algoritmos em cenários estruturalmente distintos.

5.1 Metodologia Experimental

Os experimentos foram executados em Python e o tempo de execução foi medido em **milissegundos (ms)**. Para cada instância, cada algoritmo foi executado uma vez, e o objetivo foi observar o crescimento do tempo de execução conforme o tamanho do grafo e sua estrutura.

Foram consideradas três famílias de instâncias:

- **Grafos completos** K_n: grafos densos com grande quantidade de caminhos hamiltonianos;
- **Grafos densos sem caminho hamiltoniano**: obtidos a partir de K_n com a adição de 2 vértices e 2 arestas, de modo a garantir que não exista caminho hamiltoniano;
- **Grafos esparsos (caminho)**: grafos que consistem em um único caminho simples com n vértices e $n - 1$ arestas.

Os resultados são apresentados por meio de gráficos de barras e tabelas comparativas.

5.2 Grafos completos K_n

Nos grafos completos K_n existe um número muito elevado de caminhos hamiltonianos possíveis. Nesse caso, o algoritmo de backtracking tende a encontrar uma solução rapidamente, pois há muitas alternativas válidas para estender o caminho e a busca encerra assim que encontra a primeira solução.

Os gráficos mostram que, para valores de n entre 10 e 20, o tempo do *Backtracking* permanece muito baixo. Em contrapartida, a *Programação Dinâmica com bitmask* apresenta crescimento exponencial acentuado, pois explora sistematicamente estados correspondentes a subconjuntos de vértices, independentemente da facilidade da instância.

5.3 Grafos densos sem caminho hamiltoniano

Para avaliar o comportamento em instâncias densas **sem solução**, foram construídos grafos densos derivados de K_n e, em seguida, foram adicionados dois vértices e duas arestas de forma a assegurar que o grafo resultante não possuísse caminho hamiltoniano. Esse tipo de instância tende a ser desfavorável para o backtracking, pois o algoritmo precisa explorar um grande espaço de busca antes de concluir que a resposta é **NÃO**.

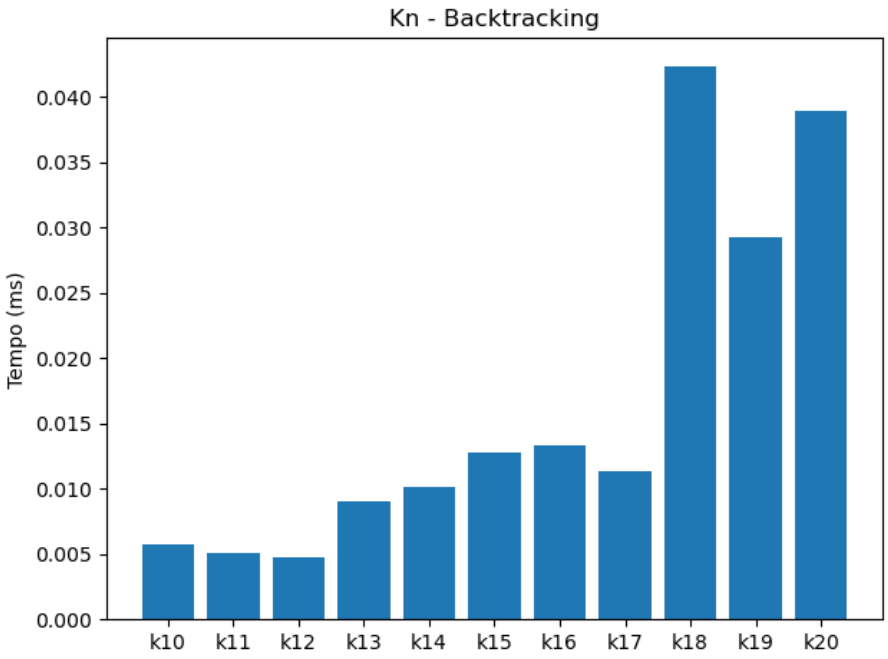

Figure 1: Execuções com Grafos Completos com Backtracking

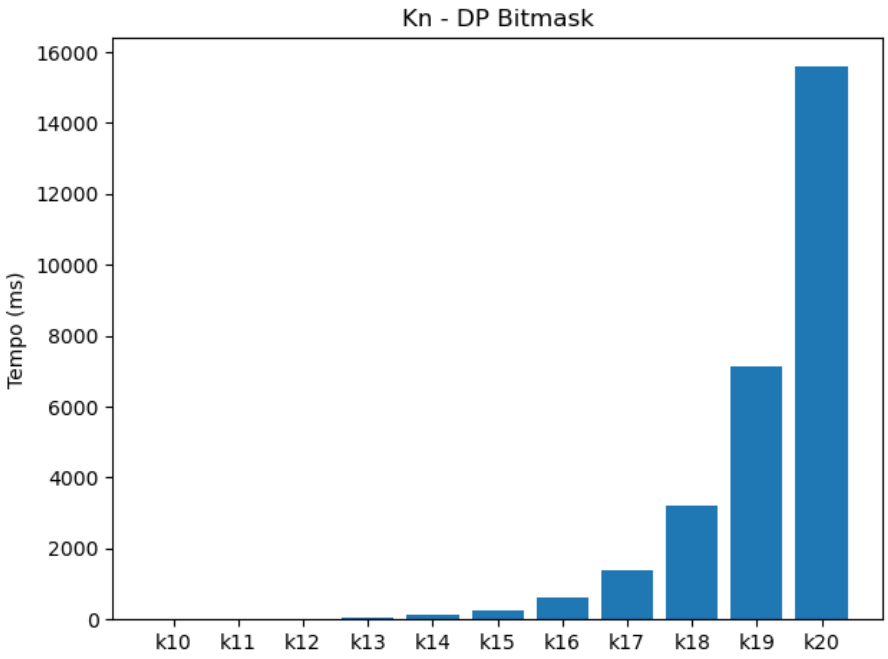

Figure 2: Execuções com Grafos Completos com Programação Dinâmica

A Tabela 1 apresenta os tempos obtidos para essa classe.

Table 1: Tempo de execução para grafos densos sem caminho hamiltoniano

Vértices	**Arestas**	**Backtracking (ms)**	**PD (ms)**
7	12	0,122	0,078
8	17	0,697	0,157
9	23	5,448	1,096
10	30	41,497	1,304
11	38	426,324	2,772
12	47	3889,595	7,101

Observa-se que o tempo do backtracking cresce rapidamente com o número de vértices, evidenciando seu comportamento exponencial quando é necessário provar a inexistência de uma solução. A programação dinâmica também cresce exponencialmente, porém com valores menores nessa faixa de tamanho, em linha com sua complexidade $O(n^2 \cdot 2^n)$.

5.4 Grafos esparsos (caminho simples)

Por fim, foram avaliados grafos esparsos formados por um único caminho simples, isto é, grafos com n vértices e $n - 1$ arestas. Nessa classe de instâncias existe um caminho hamiltoniano (o próprio caminho do grafo), e a estrutura do grafo reduz significativamente o número de escolhas possíveis em cada passo.

A Tabela 2 apresenta os tempos obtidos.

Table 2: Tempo de execução para grafos esparsos (caminho simples)

Vértices	**Arestas**	**Backtracking (ms)**	**PD (ms)**
10	9	0,008	0,058
20	19	0,021	22,040
30	29	0,133	30966,694

Nesse cenário, o backtracking apresenta desempenho muito eficiente, pois a busca é fortemente guiada pela própria estrutura do caminho e encontra a solução rapidamente. Em contrapartida, a programação dinâmica torna-se rapidamente inviável, pois seu custo está ligado ao número de subconjuntos de vértices e cresce exponencialmente com n.

5.5 Discussão

Os experimentos indicam que o desempenho prático dos algoritmos depende fortemente da estrutura do grafo:

- Em grafos com **muitas soluções** (como K_n), o *Backtracking* tende a ser muito rápido, pois encontra uma solução cedo e encerra.

- Em grafos sem solução, o *Backtracking* pode se tornar muito lento, pois precisa explorar grande parte do espaço de busca antes de concluir **NÃO**.
- A *Programação Dinâmica com bitmask* apresenta comportamento mais previsível, porém possui custo exponencial e se torna impraticável conforme n cresce, especialmente em grafos esparsos maiores.

Assim, embora ambos os métodos sejam exponenciais no pior caso, eles apresentam comportamentos bastante distintos dependendo da classe de instâncias considerada.

Referências

[1] Yongeun Bae, Chunkyun Youn, and Ilyong Chung. Application of the hamiltonian circuit latin square to the parallel routing algorithm on 2-circulant networks. In Jun Zhang, Ji-Huan He, and Yuxi Fu, editors, *Computational and Information Science*, pages 219–224, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[2] Claude Berge. *Path-Partitions in Directed Graphs and Posets*, pages 447–464. Springer Netherlands, Dordrecht, 1985.

[3] Julien Bramel and David Simchi-Levi. *The Logic of Logistics: Theory, Algorithms, and Applications.* Springer, New York, 1997.

[4] Shao Dong Chen, Hong Shen, and Rodney Topor. An efficient algorithm for constructing hamiltonian paths in meshes. *Parallel Computing*, 28(9):1293–1305, 2002.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[6] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.