Enumeração de Ciclos em Grafos Não-direcionados Simples

Lucas Azevedo Victor Colen
Wanderson de Souza
Pontifícia Universidade Católica de Minas Gerais

Data de Entrega: 1 de Abril de 2024

Resumo

Este trabalho aborda o desafio da enumeração de todos os ciclos presentes em um grafo não-direcionado simples, um problema com aplicações significativas em diversas áreas, como otimização de redes, sistemas de roteamento e análise de circuitos. Propomos uma solução computacional desenvolvida em C++ que emprega duas estratégias principais: a permutação dos vértices do grafo e a exploração de caminhos através de busca em profundidade para a identificação de ciclos. Além disso, implementamos um mecanismo para eliminar duplicatas nos conjuntos de ciclos encontrados, visando otimizar a eficiência e a precisão dos resultados. Os métodos foram avaliados em termos de desempenho computacional, variando o tamanho e a densidade dos grafos analisados, para determinar sua aplicabilidade e eficácia em cenários diversificados. Os resultados obtidos refletem a viabilidade das abordagens adotadas, apresentando insights valiosos sobre as diferenças de desempenho e potenciais aplicações práticas.

1 Introdução

Este relatório detalha o desenvolvimento e a implementação de um programa em C++ destinado à enumeração de ciclos em grafos não direcionados. A identificação de ciclos em grafos é uma tarefa fundamental na teoria dos grafos, com aplicações práticas em áreas como análise de redes, circuitos elétricos e sistemas de navegação. Este trabalho foca na implementação de métodos eficientes para encontrar e enumerar todos os ciclos possíveis em um grafo, considerando variações na densidade de arestas do grafo.

2 Metodologia

A enumeração de ciclos em grafos não direcionados foi realizada por meio de duas abordagens distintas, cada uma explorando estratégias diferentes para identificar todos os ciclos possíveis dentro de um grafo. Ambas as abordagens foram

implementadas na linguagem de programação C++, tirando vantagem da orientação a objetos e das bibliotecas padrão para estruturas de dados e medição de tempo.

2.1 Enumeração de Ciclos por Caminhamento

A primeira abordagem de enumeração de ciclos envolve um caminhamento no grafo, utilizando um algoritmo de busca em profundidade (DFS). Partindo de um vértice inicial, o algoritmo explora recursivamente todos os caminhos possíveis, voltando ao vértice de origem para formar um ciclo. Durante a exploração, um caminho é registrado somente se ele forma um ciclo fechado que não contenha vértices repetidos, exceto o vértice inicial e final que são o mesmo. Este método é particularmente eficiente para identificar ciclos em grafos de menor densidade, onde o número de caminhos a serem explorados é relativamente menor.

2.2 Enumeração de Ciclos por Permutação

A segunda abordagem adotada foi a enumeração de ciclos por permutação dos vértices do grafo. O algoritmo gera todas as permutações possíveis dos vértices e verifica para cada permutação se ela representa um ciclo válido. Um ciclo é considerado válido se cada par de vértices consecutivos na permutação estiver conectado por uma aresta no grafo e o ciclo terminar no vértice de partida, formando um laço. Após a geração de todas as permutações possíveis, o algoritmo elimina ciclos duplicados para garantir que cada ciclo seja contabilizado uma única vez. Essa abordagem tem um desempenho superior em grafos de alta densidade, apesar da maior complexidade computacional devido ao número crescente de permutações com o aumento do tamanho do grafo.

2.3 Otimizações e Complexidade

Ambas as abordagens foram otimizadas para evitar a contagem de ciclos duplicados. No caso da permutação, isso é realizado por meio da ordenação dos ciclos encontrados e a subsequente remoção de duplicatas. As diferenças na complexidade algorítmica entre as duas abordagens são discutidas em relação ao tamanho do grafo e à densidade das arestas, refletindo o trade-off entre o custo computacional e a completude na enumeração de ciclos.

2.4 Infraestrutura Experimental

Para a realização dos experimentos, foram utilizados grafos com uma variação controlada do número de vértices e da densidade das arestas. A densidade é ajustada pela proporção de arestas presentes em relação ao número total possível de arestas. Os tempos de execução para cada abordagem foram medidos e registrados para permitir uma análise comparativa detalhada.

2.5 Aplicabilidade em Grafos Direcionados

Embora as técnicas descritas tenham sido desenvolvidas e testadas em grafos não direcionados, com modificações adequadas, é possível adaptá-las para funcionar em grafos direcionados. A busca de ciclos por caminhamento já é intrinsecamente aplicável a grafos direcionados, pois a direção das arestas é naturalmente considerada durante o caminhamento. No entanto, a técnica de permutação requer ajustes para garantir que apenas os ciclos direcionados sejam considerados, onde cada aresta percorrida no ciclo respeite a direção do arco do grafo. Futuras extensões deste trabalho poderiam explorar essas adaptações com o objetivo de lidar com a enumeração de ciclos em grafos direcionados.

3 Implementação

O programa foi desenvolvido em C++, aproveitando as funcionalidades orientadas a objetos da linguagem para organizar o código de maneira clara e modular. A estrutura do grafo é mantida por uma classe que encapsula as listas de adjacências e oferece métodos para adicionar arestas e realizar a busca e enumeração de ciclos. Além disso, foram utilizadas bibliotecas padrão para manipulação de vetores e listas, bem como para medição de tempo, permitindo uma análise precisa do desempenho do algoritmo.

3.1 Representação do Grafo

Optamos pela representação do grafo através de uma lista de adjacências, considerando a facilidade de adicionar ou remover arestas e a eficiência na busca de vértices adjacentes. Cada vértice do grafo é associado a uma lista que contém seus vértices adjacentes, permitindo uma navegação eficiente durante a busca de ciclos.

4 Experimentos

Os experimentos foram conduzidos utilizando grafos de diferentes tamanhos e densidades, com o objetivo de avaliar o desempenho das duas abordagens implementadas. Medimos o tempo de execução e o número de ciclos encontrados para cada método, proporcionando uma base comparativa entre as abordagens de permutação dos vértices e de caminhamento no grafo.

5 Resultados

Os experimentos foram conduzidos para avaliar o desempenho das duas abordagens implementadas – busca de ciclos e enumeração de ciclos – sob diferentes configurações de grafos, variando o número de vértices e a densidade das arestas. A Tabela 1 apresenta os tempos de execução para cada abordagem, medidos em

microssegundos (µs), permitindo uma comparação direta do impacto das variações na densidade sobre o desempenho dos algoritmos.

Vértices	Densidade	Caminhamento (µs)	Permutação (µs)
4	0.50	1	10
4	1.00	1	4
5	0.50	16	12
5	1.00	28	15
6	0.50	1	65
6	1.00	12	672
7	0.50	2	1,242
7	1.00	315	6,743
8	0.50	3	11,721
8	1.00	711	30,354
9	0.50	38	43,773
9	1.00	2,240	138,334
10	0.50	0	204,291
10	1.00	98,797	3,579,921
11	0.50	308	8,167,740
11	1.00	106,213	34,390,892
12	0.50	5,768	107,138,568
12	1.00	6,329,256	261,237,977

Tabela 1: Comparação dos Tempos de Execução para Diferentes Quantidades de Vértices e Densidades em diferentes métodos de Enumeração de ciclos.

Os resultados indicam um aumento significativo no tempo de execução para ambos os métodos à medida que o número de vértices e a densidade das arestas aumentam. Notavelmente, a abordagem de enumeração de ciclos apresenta um crescimento mais acentuado no tempo de execução, refletindo a complexidade computacional mais elevada desse método em comparação com a simples busca de ciclos, especialmente em grafos densos.

6 Análise de Desempenho

A avaliação do desempenho dos algoritmos implementados revela considerações cruciais sobre sua aplicabilidade e eficiência. Discutiremos cada algoritmo separadamente e depois faremos uma comparação direta entre eles.

6.1 Complexidade do Algoritmo

findCycles (Caminhamento):A complexidade do algoritmo DFS é determinada em termos de tempo. Para um grafo com V vértices e E arestas, assume uma complexidade de $O(V + E)$.A modificação proposta para encontrar todos os ciclos apresenta uma complexidade aproximada de $O(2^{V \cdot (V+E)})$.

enumerateCycles (Permutação): Este método apresenta uma complexidade substancialmente maior. A permutação dos vértices possui uma complexidade de $O(V!)$, sendo V o número de vértices do grafo. Além disso, para cada permutação, o algoritmo verifica a presença de um ciclo, adicionando mais um fator de V à complexidade, resultando em uma complexidade aproximada de $O(V! \cdot V)$.

6.2 Comparação de Desempenho

Em comparação direta, o método **findCycles** é inerentemente mais eficiente devido à sua complexidade que não é fatorial. Isso é particularmente evidente em grafos com um grande número de vértices, onde o método **enumerateCycles** pode se tornar inviável devido ao rápido crescimento do número de permutações necessárias.

6.3 Melhorias Potenciais

Identificamos que o método **enumerateCycles** pode não ser prático para grafos de tamanho substancial devido à sua natureza exponencial. Uma melhoria significativa seria a busca por abordagens mais eficientes para a enumeração de ciclos, possivelmente por meio de algoritmos de backtracking especializados ou heurísticas que reduzam o espaço de busca.

7 Participação dos integrantes

1. Wanderson ficou por conta do desenvolvimento do código;
2. Lucas fez a análise de desempenho;
3. Victor criou o relatório;

8 Conclusão

A análise comparativa indica que, enquanto o **findCycles** oferece uma abordagem eficiente e prática para a maioria dos grafos, o **enumerateCycles** enfrenta desafios significativos em termos de escalabilidade e pode se beneficiar de otimizações adicionais para se tornar viável em contextos de grafos mais densos e extensos.