Trabalho apresentado no XLI CNMAC, Unicamp - Campinas - SP, 2022.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics
Preprint

Uma nova abordagem para detecção de ciclo incremental

João Victor P. de Almeida[1]
Instituto de Ciência e Tecnologia, Universidade Federal Fluminense, Rio das Ostras, RJ
Danilo Artigas[2]
Instituto de Ciência e Tecnologia, Universidade Federal Fluminense, Rio das Ostras, RJ

O problema de **detecção de ciclo incremental** consiste em, dado um grafo direcionado (inicialmente vazio) tal que as arestas são adicionadas uma a uma, detectar quando um ciclo é formado. Recentemente, dois artigos foram apresentados sobre o assunto [1, 2]. O primeiro apresenta dois algoritmos, um para grafos esparsos e um para grafos densos, enquanto o segundo artigo apresenta um algoritmo randomizado tanto para grafos densos quanto para grafos esparsos. Neste trabalho apresentamos um novo algoritmo para o problema, comparando-o computacionalmente com aquele apresentado em [2] e com dois algoritmos clássicos : Busca em Profundidade [6] e Busca em Largura [5]. O algoritmo proposto apresentou um desempenho computacional superior aos algoritmos comparados em todos os casos.

Consideraremos um grafo direcionado $G = (V, E)$ com um conjunto V de n vértices e um conjunto de arestas E, inicialmente vazio. Ao longo da execução, m arestas serão adicionadas aleatoriamente uma a uma. Um **ciclo** em um grafo é uma sequência de vértices $v_0, v_1, \ldots, v_k$ tal que existe uma aresta (v_i, v_{i+1}) para $0 \leq i < k$ e $v_0 = v_k$. O problema da detecção de ciclo incremental consiste em identificar quando a adição de uma nova aresta formará um ciclo e interromper, a partir desse ponto, novas inserções de arestas.

Uma abordagem natural para o problema seria por meio dos bem conhecidos algoritmos Busca em Profundidade [6] e Busca em Largura [5]. Ambos utilizam uma mesma ideia: a cada nova aresta (v, w) inserida ao grafo, exploramos o grafo a partir de w e verificamos se é possível alcançar v o que formaria um ciclo. Ambas as buscas podem ser executadas em tempo $O(m)$ e, após m inserções, teríamos um tempo computacional $O(m^2)$.

Em [2], Bernstein et al. apresentaram um algoritmo randomizado, que garante o resultado correto, porém o tempo de execução varia de acordo com a entrada. Por isso, a complexidade desse algoritmo é $\tilde{O}(m\sqrt{n})$, em que $\tilde{O}$ denota o tempo esperado de execução. A ideia do algoritmo consiste em manter, para cada vértice v, dois conjuntos: o conjunto $A(v)$ dos ancestrais de v (vértices u tais que existe um caminho de u a v em G); e o conjunto $D(v)$ dos descendentes de v (vértices w tais que existe um caminho de v a w em G). Se dois vértices x e y estão em um ciclo, então $A(x) = A(y)$ e $D(x) = D(y)$. Como pode ser muito custoso verificar todos os vértices dos conjuntos de ancestrais e descendentes de um dado par de vértices x, y, o algoritmo utiliza um subconjunto S de vértices, sorteado aleatoriamente e verifica se $A(x) \cap S = A(y) \cap S$ e $D(x) \cap S = D(y) \cap S$.

O algoritmo que propomos armazena o conjunto de ancestrais de cada vértice e, para cada nova aresta (x, y) inserida, se y é ancestral de x, então identificamos um ciclo. Senão, atualizamos o conjunto de ancestrais de todos os vértices alcançáveis a partir de uma Busca em Largura iniciada em y. Para toda aresta (v, w) alcançada na referida busca, adicionamos a $A(w)$ o conjunto $A(v)$. No entanto, a representação do conjunto de ancestrais de um vértice v será realizada por uma sequência de n bits onde o i-ésimo bit é igual a 1 caso o vértice i seja antecessor de v e 0 caso

[1] joaovictoralmeida@id.uff.br
[2] daniloartigas@id.uff.br

contrário. Armazenaremos essa sequência de bits como um número inteiro de n bits em que a representação binária deste número é igual a sequência de bits.

Uma característica importante do algoritmo proposto é que a performance das operações efetuadas. Como mencionado anteriormente, o conjunto de ancestrais é codificado por meio de um número inteiro positivo com n bits. Para atualizar o conjunto de ancestrais de um vértice, o que demanda o processamento de uma operação de união, devemos efetuar uma operação **OR** lógica entre a representação binária dos inteiros que representam os conjuntos de ancestrais que devem se unir. Esta é a operação que mais se repete no algoritmo e, tipicamente, esta operação é bem eficiente computacionalmente. Não apenas esta operação, mas todas as demais escolhas de estruturas e operações para o algoritmo justificam-se pela performance computacional das mesmas. Nosso objetivo principal é desenvolver um algoritmo que possua um bom desempenho computacional. Obtivemos êxito nesta tarefa pois, conforme o parágrafo seguinte, o algoritmo apresentou melhor performance que os outros três algoritmos descritos anteriormente.

Para avaliar o nosso algoritmo, implementamos os algoritmos de Bernstein et al. [2] e as Buscas em Largura e Profundidade, e comparamos com o algoritmo proposto. Nos experimentos foram utilizados 19 grafos do projeto KONECT [4] de diferentes tamanhos e esparsidades. Nos 11 grafos com menos arestas foram testados os quatro algoritmos avaliados. Em todos os 11 casos analisados, nosso algoritmo foi consideravelmente mais rápido do que os demais. Comparado ao algoritmo de Bernstein et al., tivemos um desempenho pelo menos 6 vezes melhor, e o algoritmo de Bernstein et al foi melhor que os algoritmos clássicos em 9 dos 11 casos. Nos 8 grafos maiores foram testados apenas o algoritmo proposto e o de Bernstein et al. e, novamente, o algoritmo proposto foi superior em todos os casos com um desempenho de 10 a 1000 vezes superior.

O código está disponível em [https://github.com/Victor-Almeida/Cycle-Detection] junto com os grafos do projeto KONECT [4] utilizados para os experimentos. O algoritmo foi programado em Python com a biblioteca Networkx [3] e os testes foram executados em um AMD Ryzen 5 2400G 3.6 GHz com memória RAM DDR4 2666 MHz.

Referências

[1] Michael A. Bender et al. "A New Approach to Incremental Cycle Detection and Related Problems". Em: **ACM Trans. Algorithms** 12.2 (dez. de 2015), 14:1–14:22. DOI: 10.1145/2756553.

[2] Aaron Bernstein e Shiri Chechik. "Incremental Topological Sort and Cycle Detection in $\tilde{O}(m\sqrt{n})$ Expected Total Time". Em: **Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)**. 2018, pp. 21–34. DOI: 10.1137/1.9781611975031.2.

[3] Aric Hagberg, Daniel A. Schult e Pieter Jacob Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". Em: **Proceedings of the 7th Python in Science Conference**. Ed. por Gaël Varoquaux, Travis Vaught e Jarrod Millman. Pasadena, CA USA, 2008, pp. 11 –15.

[4] Jérôme Kunegis. "KONECT: The Koblenz Network Collection". Em: **Proceedings of the 22nd International Conference on World Wide Web**. WWW '13 Companion. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, 1343–1350. ISBN: 9781450320382. DOI: 10.1145/2487788.2488173.

[5] E.F. Moore. "The shortest path through a maze". Em: **International Symposium on the Theory of Switching**. Harvard University Press, 1959, pp. 285–292.

[6] Robert E. Tarjan. "Depth-First Search and Linear Graph Algorithms". Em: **SIAM Journal on Computing** 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010.