

# Bancos de Dados de Grafos Distribuídos

Gustavo Estrela de Matos  
*Instituto de Matemática e Estatísticas*  
*Universidade de São Paulo*  
São Paulo, Brasil  
gestrela@ime.usp.br

Hector Montenegro Terceros  
*Instituto de Matemática e Estatísticas*  
*Universidade de São Paulo*  
São Paulo, Brasil  
hector@ime.usp.br

**Resumo**—Vamos fazer este por último...

**Index Terms**—component, formatting, style, styling, insert **Isso aqui eu não entendi direito, não sei como fazer aqui.**

## I. INTRODUÇÃO

Grafos são objetos matemáticos que podem ser usados como estruturas de dados em diversas aplicações computacionais. Sistemas de Bancos de Dados baseados em grafos são considerados bancos de dados NoSQL [1] e podem ser aplicados em diversos contextos, como o de aplicativos da web; áreas industriais de transportes, telecomunicação e comércio [2]; e também em áreas de pesquisa, como em bioquímica [4], biologia molecular [5] e web semântica [6].

Um grafo pode ser definido como uma dupla  $G = (V, E)$ , em que  $V$  é um conjunto de vértices (ou nós) e  $E$  é um conjunto de arcos, que conectam dois nós. Podemos representar um arco  $e \in E$ , por uma dupla  $e = (v_i, v_j)$  com  $v_i \in V$  e  $v_j \in V$ . No contexto de bancos de dados de grafos, usualmente um nó representa uma entidade modelada, e arcos representam relacionamentos entre essas entidades. É comum que os nós recebam rótulos que estão associados ao tipo de entidade modelada, e também um conjunto de propriedades em formato chave-valor, capaz de armazenar atributos da entidade. Além disso, os arcos do modelo também podem receber rótulos que identificam o tipo de relacionamento que é modelado, e um conjunto de propriedades que representam atributos do relacionamento.

Bancos de dados baseados em grafos costumam ser aplicados em contextos em que os dados de interesse possuem relacionamentos complexos ou simplesmente quando boa parte da informação está contida nos relacionamentos. Nestes casos, um banco de dados relacional pode ser inadequado ou ineficiente. Considere, por exemplo, a relação `FABRICA (id_fabricante, id_produto)`. Em uma consulta em que se deseja saber as informações dos produtos fabricados por uma fábrica, no modelo relacional, é necessário fazer uma junção com a relação que armazena os dados dos produtos, enquanto no modelo baseado em grafos, basta percorrer os arcos do relacionamento `FABRICA` que estão ligados ao nó que representa a fábrica de interesse. Além de ser mais eficiente para algumas consultas, sistemas baseados em grafos podem ter consultas mais expressivas, capazes de representar relacionamentos complexos entre os dados [3].

Assim como outros bancos de dados NoSQL, os bancos de dados em grafos são muito utilizados em aplicações que precisam armazenar um grande volume de dados. Para atender a este requisito, alguns bancos de dados baseados em grafos permitem armazenamento distribuído. Este tipo de solução precisa implementar particionamento de dados. Além disso, um sistema distribuído deve providenciar maneiras de responder a consultas que acessam informações de vértices alocados em diferentes máquinas de uma rede.

Neste artigo, nosso principal objetivo é apresentar os conceitos fundamentais e as principais soluções para implementar bancos de dados baseados em grafos distribuídos, e também sistemas para o processamento distribuído de grafos. Ao longo deste trabalho, vamos... **informações a serem adicionadas no futuro, com o que vamos colocar de fato no artigo.**

## II. METODOLOGIA

- Como escolhemos os artigos que lemos para criar esse artigo? - Qual tipo de grafos estamos focados em tratar? (Os que a gente achou... acho que é maioria usado pra rede social??)

## III. CONCEITOS FUNDAMENTAIS

Nesta seção apresentamos os conceitos fundamentais para entender o funcionamento de bancos de dados e ferramentas de processamento de dados baseados em grafos, mais especificamente, em contextos distribuídos.

### A. Graph DBMS versus Graph Analytics Systems

### B. Cortes de grafos

O conceito de corte em grafo é importante para formalização e análise do particionamento de um grafo em um contexto distribuído. Para se trabalhar com grafos de maneira distribuída, é necessário criar um particionamento do grafo, para que cada parte seja alocada em um nó da rede de computadores usada. Um particionamento precisa ser feito de maneira que o processamento do grafo seja balanceado, ou seja, as máquinas devem ter números de acessos similares; além disso, é importante fazer com que os processamentos do grafo sejam feitos com o menor número de nós de computação possível, pois o processamento que envolve mais de um nó de computação necessita de maior comunicação pela rede, aumentando a latência da operação. Para analisar estes dois aspectos com maior formalismo, definimos o conceito de corte.

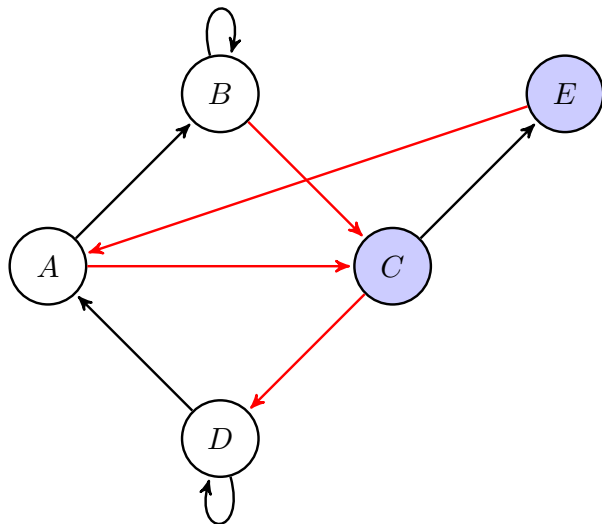


Figura 1. Grafo com exemplo de corte. O corte apresentado é definido pelos conjuntos de vértices  $E, C$ , em azul, e  $A, B, D$ , em branco. Os arcos de cor vermelha são os arcos deste corte. No total, são 4 arcos vermelhos neste corte, ou seja, o tamanho deste corte é 4.

Um corte é um particionamento em dois conjuntos disjuntos de vértices de um grafo. Seja  $G = (V, E)$  um grafo, então um conjunto  $S \subseteq V$  de vértices induz o corte  $(S, V \setminus S)$ . Um arco  $e \in E$  é chamada de arco do corte se ele atravessa as partes, ou seja, se  $e = (u, v)$  com  $u \in S$  e  $v \notin S$ , ou  $u \notin S$  e  $v \in S$ . Chamamos de tamanho do corte o número de arcos que são arcos do corte. A figura 1 apresenta um exemplo de corte em um grafo.

Note que qualquer particionamento (não apenas bipartições) do conjunto de vértices de um grafo pode ser realizado após recursivas aplicações de cortes. Portanto, é possível construir e analisar cortes olhando apenas para bipartições do grafo original. Perceba também que ao fazer um particionamento, podemos analisar o balanceamento de processamento dos nós de computação de acordo com o número de partes (e vértices) que são alocadas aos nós de computação. Além disso, é possível analisar a quantidade de comunicação entre nós de acordo com a quantidade de arcos que conectam diferentes partes, que podem estar alocadas em dois nós de computação diferentes.

Se formos falar de cortes baseados em arcos, é melhor adicionar mais um pedaço de texto explicando esse tipo de corte.

#### IV. PARTICIONAMENTO DE GRAFOS

O particionamento de um grafo é o processo que permite dividir um grafo em diferentes partes, que podem ser alocadas em diferentes máquinas, permitindo o processamento e armazenamento distribuído do grafo. Nesta seção, apresentaremos três diferentes abordagens de particionamento. A primeira, baseada no algoritmo METIS [7], é uma heurística que tenta minimizar a quantidade de arcos que conectam partes

diferentes; a segunda faz um particionamento aleatório; e a terceira particiona os vértices de acordo com o valor de alguns atributos destes vértices.

##### A. Particionamento com cortes mínimos

O algoritmo METIS tem como objetivo produzir um particionamento com  $k$  partes que minimiza a quantidade de arcos atravessando partes, ou seja, produz cortes mínimos no grafo. O problema de encontrar tal partição é chamado  $k$ -particionamento de um grafo, e vamos defini-lo assim: dado um grafo  $G = (V, E)$ , com  $|V| = n$ , particione  $V$  em  $k$  conjuntos  $V_1, V_2, \dots, V_k$  de maneira que  $V_i \cap V_j = \emptyset$  para  $i \neq j$ ,  $\bigcup_{i=1}^k V_i = V$ , e  $|V_i| = n/k$  para  $i \in \{1, \dots, k\}$  (sem perda de generalidade, se  $n$  não é divisível por  $k$ , então adicione nós "fantasmas" até que  $k|n$ ). O  $k$ -particionamento também pode ser generalizado ao considerar pesos para arcos, adicionando uma informação de importância ou relevância para as conexões, mas por facilidade vamos considerar apenas o caso em que todos os pesos dos arcos são iguais, o que é consistente com a definição que apresentamos. Lembre também que um corte em um grafo também é uma bipartição do grafo; a escolha por um dos nomes ao longo desta seção será feita de maneira conveniente à explicação dos conceitos.

O  $k$ -particionamento possui importância em aplicações de computação paralela e distribuída, e foi provado ser NP-completo [8], ou seja, o problema se torna intratável rapidamente com o aumento da instância. Por isso, o algoritmo METIS é uma heurística, ou seja, a solução produzida não é ótima em geral.

A heurística METIS resolve o problema do  $k$ -particionamento realizando aplicações recursivas de um procedimento de biparticionamento. O número total de chamadas recursivas feitas é no máximo  $\lceil \log x \rceil$ . O algoritmo de bipartição é também uma heurística que tenta achar o corte de menor custo no grafo. Como achar esta partição no grafo pode tomar muito tempo, uma etapa de encolhimento é feita antes, e depois disso o grafo bipartido é expandido e refinado até se obter uma bipartição do grafo original. O algoritmo de biparticionamento é composto por três etapas:

- encolhimento: pares de vértices são escolhidos dois a dois, e são acoplados em um vértice novo que reúne todos os arcos que estavam nos vértices antigos. Esta etapa é repetida até o grafo ser considerado pequeno;
- biparticionamento: uma heurística (novamente) determina um corte com tamanho pequeno;
- expansão: os vértices acoplados são desfeitos ao mesmo tempo que a bipartição é atualizada de acordo com os novos nós.

Durante o encolhimento, os vértices devem ser escolhidos dois a dois sem repetições, o que é equivalente a escolher um emparelhamento do grafo. Um emparelhamento é um conjunto de arcos de um grafo que não tem vértices em comum. O algoritmo METIS implementa uma heurística de emparelhamento máximo (com maior número de arcos), para escolher os pares de vértices para serem acoplados. A vantagem de fazer um emparelhamento máximo durante o encolhimento é diminuir

o número de iterações até que o grafo fique pequeno. Segundo Karypis et al., um grafo de tamanho 100 é pequeno o suficiente para seguir para etapa de bipartição.

Com o grafo encolhido, uma bipartição deve ser mais facilmente encontrada. Porém, como este problema ainda é NP-difícil [8], uma outra heurística é usada. Esta heurística constrói um corte do grafo, e é um algoritmo guloso que começa com um corte que contém apenas um vértice, escolhido aleatoriamente, e a cada iteração aumenta o conjunto de vértices do corte ao adicionar um novo vértice, até que o corte tenha metade dos vértices do grafo. Em cada etapa do algoritmo, um vértice de fora do corte que está conectado a um arco do corte é escolhido. Esta escolha é feita de acordo com o ganho de arcos de corte que o novo vértice pode adicionar (ou remover) no corte atual. Seja  $S_i$  o conjunto de vértices do corte da  $i$ -ésima iteração, então, definimos o ganho de adicionar um vértice como:

$$g_v = |\{u \mid (v, u) \in E, u \notin S_i\}| - |\{u \mid (v, u) \in E, u \in S_i\}| \quad (1)$$

A medida  $g_v$  também é útil para a última etapa do biparticionamento, que ao mesmo tempo que expande o grafo colapsado, faz um refinamento do biparticionamento produzido. A cada iteração desta etapa, um nó é desacoplado e então todos os vértices da borda do corte do biparticionamento são analisados para uma possível troca de partes. Na prática, esta etapa é aplicada com otimizações que tornam desnecessário analisar todos os vértices da borda por várias iterações, tornando essa heurística mais eficiente.

O algoritmo METIS é até hoje um dos principais algoritmos de particionamento de grafos, e por isso é usado como base de comparação para outras propostas [9].

### B. Particionamentos aleatórios e por intervalos

Um particionamento aleatório atribui partes aos vértices de maneira aleatória e uniforme. Esta atribuição é feita de acordo com alguma função de hash, que pode ser aplicada, por exemplo, no atributo de identificação do vértice. Este tipo de particionamento é facilmente implementado visto que o único requisito é que uma função de hash distribua os vértices de maneira equilibrada entre os nós.

Um particionamento por intervalo é similar, e também precisa de uma função de hash, mas ao invés de atribuir partes a valores de um atributo, essa função atribui partes a intervalos dos valores de atributos. Esta abordagem é útil quando há um atributo nos vértices que determina a chance de relacionamento entre vértices de acordo com seus valores; por exemplo, se os vértices são pessoas com um atributo de latitude e longitude, então é razoável admitir que os relacionamentos (arcos) ocorrerão mais entre pessoas que tem valores próximos para esses atributos.

Além de ser facilmente implementadas, essas estratégias precisam de pouco processamento e pouco espaço de memória para identificar a qual parte um vértice foi alocado. Outra vantagem desses métodos é que um novo vértice do grafo pode

Tabela I  
CARACTERÍSTICAS DE ABORDAGENS DE PARTICIONAMENTO

|  | Particionamento de corte mínimo | Particionamento aleatório e por intervalo |
|--|---------------------------------|---|
| Usa estrutura do grafo                       | Sim                             | Não                                       |
| Dificuldade de cálculo do particionamento    | Difícil                         | Fácil                                     |
| Necessita de armazenar partes explicitamente | Sim                             | Não                                       |
| Atribuição de parte para um novo vértice     | Depende de criação de política  | Automática                                |

ser atribuído para uma parte de maneira rápida, diferente da abordagem de cortes mínimos.

### C. Comparação entre abordagens de particionamento

Ambas abordagens apresentadas anteriormete podem ser aplicadas vantajosamente. A estratégia baseada em cortes se aproveita da estrutura da estrutura do problema de interesse e pode produzir um particionamento que possui poucos arcos conectando duas partes, ou seja, um particionamento que agrupa as entidades que se relacionam constantemente e, portanto, faz as consultas precisarem de menos comunicações entre nós de computação. No caso dos particionamentos aleatórios ou por intervalos, o processo de particionamento é simples de se implementar e necessita de pouco espaço de armazenamento. Além disso, particionamentos aleatórios podem atribuir partes a novos vértices assim que eles são criados.

Por outro lado, ambas abordagens também possuem desvantagens. A principal desvantagem do particionamento por cortes é a sua complexidade, de implementação e de recuperação das partes. Além disso, no particionamento baseado em cortes, uma política precisa ser definida para quando um novo vértice é criado: este vértice precisa ser eventualmente colocado em alguma parte, mas deve ser considerado que refazer todo processo de particionamento sempre que um novo nó é criado é inviável. A desvantagem do método particionamento aleatório e não usar a estrutura dos dados do problema, o que pode fazer com que operações comuns causem um grande tráfego na rede, pois os nós fortemente relacionados não estarão disponíveis nos mesmos locais.

O trabalho de Khayyat et al. mostra que, de fato, não um consenso de qual é a melhor alternativa de particionamento [10]. Portanto, a escolha de uma estratégia de particionamento depende da aplicação. A tabela I apresenta os principais aspectos estratégias de particionamento que apresentamos nesta seção.

## V. SISTEMAS GERENCIADORES DE BANCOS DE DADOS EM GRAFOS

### VI. GRAPH ANALYTICS SYSTEMS

Com o crescimento das aplicações na internet e em outras áreas como transportes e comércio, a demanda por processamento eficiente e distribuído de grafos cresceu. Outros

arcabouços de processamento distribuído com propósitos mais genéricos, como MapReduce, podem não ser adequados para alguns problemas de grafos. Por isso, tornou-se necessário a criação de arcabouços de processamento distribuído especializados em grafos, que permitem escrever programas que fazem cálculos sobre grafos de maneira mais intuitiva e eficiente. Chamamos estes arcabouços de *Graph Analytics Systems* (GASs).

Uma das primeiras soluções GAS foi o sistema Pregel, criado por engenheiros do Google [11]. Pregel possui uma abordagem de processamento centrada nos vértices do grafo. Os programas deste arcabouço funcionam com uma sequência de rodadas chamadas de *supersteps*; a cada *superstep* um vértice pode receber uma mensagem, fazer uma computação, enviar uma mensagem, ou decidir parar. Um programa acaba quando todos os vértices decidem parar. Este método de computação também é a base para outros GAS, como o Giraph [12], Surfer [13], GoldenOrb [14] e Mizan [10]. Estas variações do sistema Pregel normalmente se diferenciam pelo sistema de armazenamento utilizado e também pelo tipo de particionamento de grafo utilizado.

O sistema Pregel e suas variantes podem ainda não ser adequados para algumas aplicações, principalmente aquelas em que são construídos grafos densos. Em um grafo denso, a quantidade de arcos pode ser muito grande, criando um alto tráfego de informação localmente e também entre os nós de computação. Como solução para este problema o sistema Blogel foi desenvolvido. Este sistema utiliza o mesmo método de processamento que o Pregel, porém faz cálculos centrados em blocos de vértices, evitando comunicação excessiva entre vértices.

Vamos, nesta seção, explicar em maiores detalhes os funcionamentos dos arcabouços Pregel e Blogel.

#### A. Pregel

Pregel é um arcabouço criado por engenheiros do Google, para processamento de grafos. Este arcabouço surgiu por conta da inadequação de outros arcabouços de processamento, como MapReduce, no processamento de grafos. Apesar de surgir como uma alternativa ao MapReduce, Pregel utiliza uma ideia de sucesso do primeiro arcabouço: processamento particionado e local. Um programa Pregel é composto por uma sequência de iterações, chamadas de *supersteps* em que cada vértice do grafo faz processamentos independentes que podem compor a resposta do programa.

Em cada *superstep* um vértice pode receber uma mensagem, fazer um processamento, ou enviar uma mensagem (ou qualquer combinação dos três). Um vértice é capaz de receber uma mensagem que lhe foi enviada na iteração anterior, e esta mensagem normalmente é utilizada como argumento para algum processamento que o vértice pode ter na iteração. O processamento de cada vértice é determinado pelo usuário, que escreveu o programa no arcabouço. Além de receber uma mensagem e fazer um processamento, um vértice também pode enviar uma mensagem, que será recebida pelo destinatário na próxima iteração. Um vértice normalmente envia mensagens

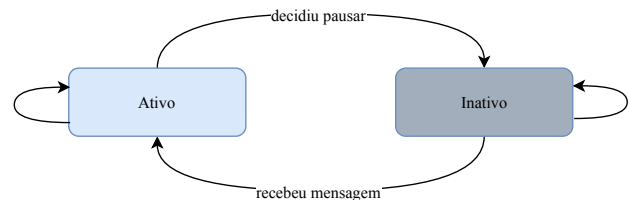


Figura 2. Diagrama de estado de um vértice em um programa Pregel.

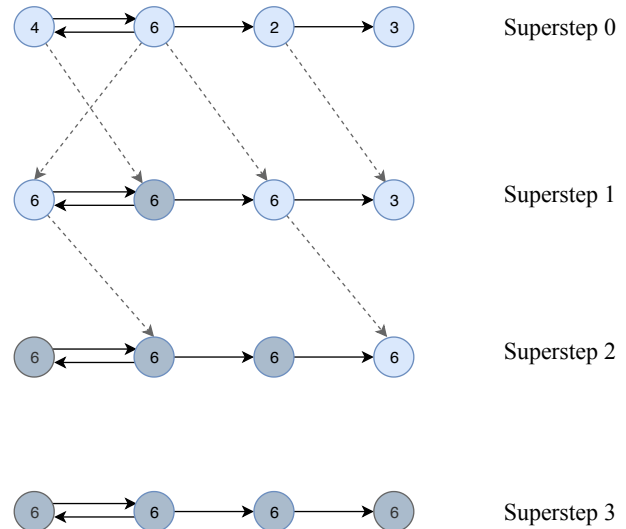


Figura 3. Exemplo de execução de um programa Pregel, que encontra o valor máximo dos vértices de um grafo. Os vértices mais escuros estão inativos, enquanto os vértices ativos possuem coloração mais clara. Os arcos tracejados representam comunicação entre *supersteps*, enquanto os arcos sólidos representam os arcos do grafo original. Em cada iteração um vértice ativo recebe o valor de um vértice adjacente que estava ativo na última iteração. Se o valor recebido por um vértice é maior do que o seu valor, então este vértice propaga o maior valor para todos os seus vértices adjacentes; se o valor recebido não for maior que o seu, então o vértice decide se tornar inativo.

para vértices adjacentes, mas uma mensagem também pode ser enviada para qualquer outro vértice.

Além fazer trocas de mensagens e processamento, um vértice também pode pausar. Quando isto acontece, o vértice fica em estado inativo e não faz nenhum tipo de processamento ou envio de mensagens; por outro lado, um vértice inativo pode voltar a ser ativo se ele receber uma mensagem. A figura 2 mostra o diagrama de estados de vértices em um programa Pregel. Quando todos os vértices estão em estado inativo, então não existe mais nenhum processamento ou comunicação a ser feito, logo o programa termina. A figura 3 mostra um exemplo de programa em Pregel.

Após a introdução do sistema Pregel, outros sistemas similares foram criados. Alguns desses sistemas mudam pontualmente alguns aspectos do sistema Pregel, incluindo o algoritmo de particionamento utilizado. O sistema Pregel usa particionamento aleatório, com *hashing*, por padrão [11], mas outros sistemas similares como Giraph e Surfer permitem o particionamento por intervalos de valores e cortes mínimos,

respectivamente.

## B. Blogel

### REFERÊNCIAS

- [1] “NoSQL Databases”, <http://nosql-database.org/>.
- [2] “Neo4j Customers”, <https://neo4j.com/customers/>.
- [3] M. Hunger, R. Boyd. “RDBMS & Graphs: SQL vs. Cypher Query Languages” Neo4j Blog. Mar. 2016. <https://neo4j.com/blog/sql-vs-cypher-query-languages/>
- [4] N. Swainston et al., “biochem4j: Integrated and extensible biochemical knowledge through graph databases”, PLOS ONE, vol. 12, no. 7, p. e0179130, Jul. 2017.
- [5] F. Olken, “Graph Data Management for Molecular Biology”, OMICS: A Journal of Integrative Biology, vol. 7, no. 1, pp. 75–78, Jan. 2003.
- [6] B. McBride, “Jena: a semantic Web toolkit,” IEEE Internet Computing, vol. 6, no. 6, pp. 55–59, Nov. 2002.
- [7] Karypis, G., & Kumar, V. (1998). “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. SIAM Journal on Scientific Computing, 20(1), 359–392. <https://doi.org/10.1137/s1064827595287997>
- [8] Andreev, K., & Racke, H. (2006). “Balanced Graph Partitioning”. Theory of Computing Systems, 39(6), 929–939. <https://doi.org/10.1007/s00224-006-1350-7>
- [9] D. Avdiukhin, S. Pupyrev, e G. Yaroslavtsev, “Multi-dimensional balanced graph partitioning via projected gradient descent”, Proceedings of the VLDB Endowment, vol. 12, n° 8, p. 906–919, abr. 2019.
- [10] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, e P. Kalnis, “Mizan”, in Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys ’13, 2013.
- [11] G. Malewicz et al., “Pregel”, in Proceedings of the 2010 international conference on Management of data - SIGMOD ’10, 2010.
- [12] Avery, Ching. “Giraph: Large-scale graph processing infrastructure on hadoop.” Proceedings of the Hadoop Summit. Santa Clara 11.3 (2011): 5-9.
- [13] Chen, Rishan, et al. “Improving large graph processing on partitioned graphs in the cloud.” Proceedings of the Third ACM Symposium on Cloud Computing. ACM, 2012.
- [14] GoldenOrb. A Cloud-based Open Source Project for MassiveScale Graph Analysis. <http://goldenorbos.org/>, 2012