Clusterização de grafos pelo algoritmo *K-Medoids* com utilização de múltiplas *threads*.

André Marcelino de Souza Neves

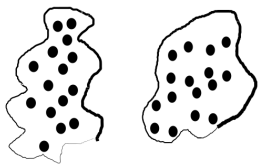
***Resumo* - Os algoritmos *K-Means* e *K-Medoids* são métodos de mineração de dados que definem *(K)* grupos para *(n)* pontos dados, inicialmente sem nenhuma rotulação. O agrupamento é baseado em alguma definição de semelhança entre as informações. Em contextos reais, os algoritmos são utilizadas grandes quantidades de dados, de forma que a otimização por paralelização do algoritmo pode trazer um significativo ganho de desempenho. Nesse trabalho, foi feita a paralelização do algoritmo *K-Medoids* implementado para clusterização de grafos. Os resultados obtidos foram satisfatórios, onde no pior caso de teste mais complexo houve *speedup* de $2,23$.**

I. INTRODUÇÃO

As técnicas de mineração de dados são formas de agregar alguma informação a partir de um conjunto de dados brutos, de forma a descobrir padrões existentes e realizar predições para eventos futuros ou gerar estatísticas interessantes [2].

Dentre muitas técnicas de mineração de dados, existem os métodos de clusterização, os quais trabalham em um conjunto inicial de dados dispersos com o objetivo de organizá-los em grupos. A divisão em grupos segue o seguinte critério: Elementos em um mesmo grupo devem ser altamente semelhantes, e o inverso para itens de grupos disjuntos, conforme a figura 1 [5].

Fig. 1: Dados agrupados em clusters



Fonte: Abraão Gonçalves Maia [1].

A. Contextualização

Com o aumento do uso das redes sociais, muitas informações podem ser obtidas a partir de dados sobre o comportamento das pessoas em tais plataformas. A análise crítica desses dados podem ser úteis para direcionamento campanhas de marketing, por exemplo. Como as informações registradas em redes sociais podem ser representadas em grafos, é possível usar tal estrutura de dados para definir a similaridade entre os objetos de acordo com as interações existentes, e a partir de algum algoritmo de clusterização, obter segmentos específicos de pessoas.

No entanto, o volume de informações produzidas diariamente pode gerar uma massa de dados grande, de forma que a clusterização dos grupos sociais torna-se uma tarefa computacional custosa e demorada. Com isso, torna-se necessária a adoção de técnicas para otimização de desempenho.

II. OBJETIVO

Desenvolver uma implementação concorrente do algoritmo *K-Medoids* para clusterização de

grafos.

III. FUNDAMENTAÇÃO TEÓRICA

A. Algoritmos de Clusterização

Dentre as possibilidades de algoritmos para clusterização, é válido citar o *K-Means* e *K-Medoids*:

- *K-Means*: Um algoritmo de clusterização que recebe como entrada dados que são dispostos em um espaço vetorial n-dimensional, e determina a similaridade entre os dados com base na distância euclidiana entre pontos [5]. Para cada *cluster* exite um centroide, que é interpretado como o ponto central da tal grupo. Nesse método, os centroides não são, necessariamente, pontos que pertencem ao conjunto de entrada.

- *K-Medoids*: Algoritmo similar ao *K-Means*, com a diferença conceitual de que os centroides devem, obrigatoriamente, pertencer ao conjunto de entrada [5]. Essa definição tem consequências diretas na implementação e nos resultados.

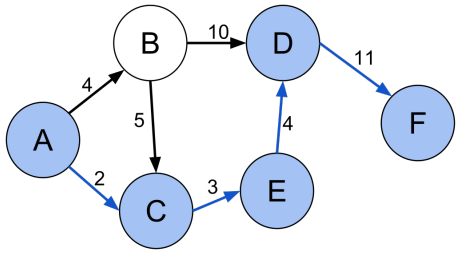
B. Algoritmo de Dijkstra para determinação de caminho mais curto

Dado um grafo com pesos positivos, esse algoritmo é utilizado quando existe a necessidade de encontrar o caminho mais curto entre dois vértices, como na figura 2 [3].

O algoritmo deve partir de um ponto inicial *(I)* qualquer. É utilizada uma tabela que indica, para cada vértice:

- Sua estimativa de distância mínima em relação ao ponto *(I)*;
- Se o vértice já foi fechado, isto é, já teve sua menor distância determinada;

Fig. 2: Caminho mínimo entre dois pontos



Fonte: Wikimedia Commons, 2013.

- Vértice anterior, que faz parte do menor caminho a partir do ponto inicial.

O fluxo principal do algoritmo é um processo iterativo que contém os seguinte passos:

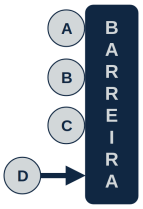
1) Escolher algum vértice *(V)* que não esteja fechado, e que tenha a menor estimativa de distância para *(I)*. O vértice escolhido é marcado como fechado, ou seja, tem a distância mínima confirmada;
2) Para cada vizinho *(N)* de *(V)*, fazer a verificação: Distância entre *(N)* e *(I)*, em um caminho que tenha *(V)* como intermediário, é menor que a estimativa atual de distância mínima? Em caso positivo, a estimativa de *(N)* é atualizada, e *(V)* é definido como vértice anterior a *(N)* no caminho mínimo.

C. Sincronização de threads com utilização de barreiras

Um problema recorrente em programação paralela é a necessidade de distribuição de partes de uma tarefa entre várias *threads*, mas com a consideração de que as *threads* deverrão ter sua execução pausada em um ponto do programa.

O recurso usado nesses casos é chamado **barreira**, que considera um ponto do programa onde nenhuma *thread* pode transpôr, até que todas o tenham atingido [4]. As *threads* que atingem o ponto antes da hora são suspensas temporariamente, conforme mostra a figura 3.

Fig. 3: Sincronização por barreiras



Fonte: Autor.

IV. DESENVOLVIMENTO

A. Implementação do algoritmo de clusterização de grafos

1) Algoritmo K-Medoids: O algoritmo possui funcionamento iterativo, onde uma configuração inicial é definida, e a partir dela, obtém-se melhorias em cada iteração realizada. Possui os seguintes passos:

1) Escolher *(K)* pontos aleatórios para serem centroides dos *clusters*;
2) Para cada ponto *(P)*, calcular a similaridade entre *(P)* e os *(K)* centroides;
3) Atribuir cada ponto *(P)* ao *cluster* cujo centroide seja o mais similar;
4) Para cada *cluster*, definir o novo centroide de forma que seja o ponto tal que a similaridade total para outros pontos seja a máxima.
5) Se após o passo 4, algum *cluster* teve seu centroide alterado, repetir o passo 2.

À cada iteração, no passo 2, o algoritmo sempre posiciona os pontos de forma mais próxima dos centroides do *cluster* o qual pertencem. Isso implica em que o algoritmo tenha convergência para um ótimo local, estado em que a distância entre os centroides e os demais pontos de um *cluster* seja a menor possível.

2) Adaptação do algoritmo para grafos: Fundamentalmente, os algoritmos *K-Means* e *K-Medoids* trabalham com pontos dispostos em um espaço vetorial n-dimensional. Nessa situação, a determinação da similaridade entre dois pontos é feita por meio do cálculo da distância euclidiana.

Para determinação da similaridade entre dois vértices, foi utilizado o algoritmo de *Dijkstra*.

O algoritmo *K-Medoids* adaptado para grafo recebe como entrada um grafo ponderado não direcionado, e um número *(K)* que representa a quantidade de *clusters* que deverão ser criados.

As seguintes considerações são feitas em relação ao procedimento padrão do *K-Medoids*:

- O passo 2) consiste na execução do algoritmo de *dijkstra* a partir de cada centroide;
- O passo 4) consiste na execução do algoritmo de *dijkstra* a partir de cada vértice *(V)*, de forma a obter a distância entre *(V)* e os demais pontos do *cluster* o qual *(V)* está inserido. O novo centroide do *cluster* será aquele que o somatório das distâncias para os demais vértices seja a menor possível.

B. Investigação do uso de paralelismo em nível de threads

Na busca para uma estratégia de paralelização desse algoritmo, foi analisada cada uma das etapas de sua execução, de forma a encontrar sub rotinas que poderiam ser paralelizadas sem condições de corrida, ou com o mínimo possível. As seguintes etapas foram consideradas inadequadas para paralelização:

- Etapa 1: A execução é rápida, pois envolve apenas escolher pontos aleatórios sem repetição, com complexidade $O(n)$, onde n é a quantidade de vértices;
- Etapa 3: A execução dessa etapa tem complexidade $O(K \cdot n)$, onde *(K)* é a quantidade de *clusters*. Com isso, pode-se assumir que a etapa não é custosa o suficiente para utilizar concorrência, pois no geral, há um número relativamente menor de *clusters* em relação a vértices. Além disso, a execução concorrente dessa etapa gera condições de corrida, pois duas *threads* podem ter uma situação onde

ambas tentam atribuir um vértice *(V)* à *clusters* diferentes;

- Etapa 5: Essa etapa tem complexidade $O(K)$, sendo de forma que não possui custo suficiente para paralelização.

Como dito anteriormente nessa seção, as etapas 2) e 4) envolvem várias execuções do algoritmo de *dijkstra*, sendo que cada uma utiliza um ponto inicial diferente, a partir de um conjunto de pontos. Para a paralelização, esse conjunto foi particionado e teve cada partição atribuída a uma *thread* diferente, estratégia denominada *SPMD (Single program, multiple data - Único programa para diversos dados)* [4].

As *threads* são criadas no início da execução do algoritmo, e são mantidas até a completa finalização. Ao serem criadas, as *threads* aguardam por conjunto de dados para serem processados.

Ao atribuir *threads* filhas para execução do *dijkstra* nas etapas 2) e 4), a *thread* principal precisa aguardar pela finalização do trabalho de todas as *threads*. Isso é feito por meio de um método de uma barreira de sincronização. O método é chamado dentro do objeto de uma *thread* específica *(T)*, e bloqueia a execução da *thread* que o chamou, nesse caso, a principal, até que *(T)* tenha concluído o processamento.

Esse método é invocado pela *thread* principal para todas as *threads* filhas, de forma a aguardar a finalização de todas.

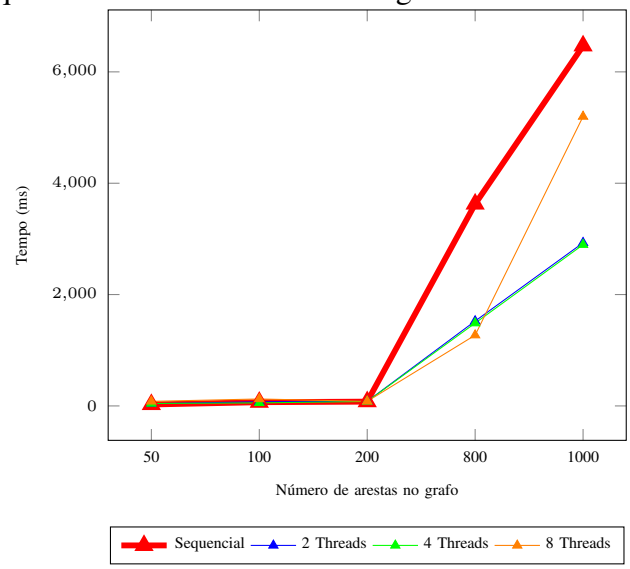
Quando uma *thread* filha finaliza seu trabalho, ela aguarda pela chegada de um novo conjunto de dados a serem processados.

V. RESULTADOS OBTIDOS

Para testes de eficiência da implementação, foram utilizados grafos de 20, 50, 100, 600 e 800 vértices, cada um com respectivamente 50, 100, 200, 800, 1000 arestas, e respectivamente 6, 16, 33, 200 e 266 clusters.

Foram medidos os tempos de execução para versão sequencial, com 2, 4 e 8 *threads*. Os valores de tempo total estão mostrados no gráfico da figura 4.

Fig. 4: Tempo gasto para execução em função da quantidade de arestas de cada grafo



Fonte: Autor.

Conforme o gráfico da figura 4, é possível perceber o ganho proporcional de desempenho quando houve a utilização de duas e quatro *therads*, em comparação com a execução sequencial. A execução no pior caso, a qual utilizou um grafo de 800 vértices, 1000 arestas e 266 *clusters*, teve *speedup* de $2,23$.

REFERÊNCIAS

[1] Abraão Gonçalves Maia, Lucas Pantuza Amorim, and Douglas de Oliveira Nunes. Algoritmos concorrentes para reconhecimento de padrões de recebimento e armazenamento de sucata metálica em uma usina siderúrgica. Technical report, 2016.

[2] Adelaja Oluwaseun Adebayo and Mani Shanker Chaubey. Data mining classification techniques on the analysis of student's performance. *Department of System Programming*, 2019.

[3] Muhammad Adeel Javaid. Understanding Dijkstra's Algorithm. 2013.

[4] P. Pacheco. *An Introduction to Parallel Programming*. Elsevier Science, 2011.

[5] Tijn Witsenburg and Hendrik Blockeel. K-Means Based Approaches to Clustering Nodes in Annotated Graphs. *International Symposium on Methodologies for Intelligent Systems*, 2011.