Igor Raphael Magollo

Método Escalável Para Aproximar uma MST Utilizando um Grafo de *k* Vizinhos Mais Próximos

São Carlos, Brasil

2021

Igor Raphael Magollo

Método Escalável Para Aproximar uma MST Utilizando um Grafo de *k* Vizinhos Mais Próximos

Monografia apresentada ao curso de Ciência da Computação, como requisito para obtenção do Título de Graduado em Ciência da Computação, no Departamento de Computação da Universidade Federal de São Carlos.

Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Graduação em Ciência da Computação

Orientador: Prof. Dr. Murilo Coelho Naldi

São Carlos, Brasil

2021

Igor Raphael Magollo

Método Escalável Para Aproximar uma MST Utilizando um Grafo de *k* Vizinhos Mais Próximos

Monografia apresentada ao curso de Ciência da Computação, como requisito para obtenção do Título de Graduado em Ciência da Computação, no Departamento de Computação da Universidade Federal de São Carlos.

Trabalho aprovado. São Carlos, Brasil, 17 de junho de 2021:

Prof. Dr. Murilo Coelho Naldi
Orientador

Prof. Dr. Diego Furtado Silva
Convidado 1

Prof. Dr. Mário César San Felice
Convidado 2

São Carlos, Brasil
2021

Resumo

Encontrar a Árvore Geradora Mínima (do inglês *Minimum Spanning Tree* MST) consiste em induzir uma árvore geradora cuja soma dos custos das arestas é mínima. Dentre muitas aplicações, a *MST* é muito utilizada em algoritmos de aprendizado de máquina não supervisionados, como é o caso dos agrupamentos. Com o aumento do volume de dados utilizados para análise e tomada de decisão, plataformas como *Apache Spark* têm se tornado mais necessárias. No entanto, existe uma dificuldade para encontrar a *MST* em um ambiente distribuído quando as distâncias ponto a ponto ainda precisam ser calculadas, pois esse cálculo de dissimilaridade possui uma complexidade quadrática em função do número de pontos, tornando-se muito ineficiente e até mesmo inviável para grandes conjuntos de dados. Neste trabalho é apresentado um método eficiente para encontrar uma boa aproximação para a *MST*. O método consiste em aplicar o algoritmo de Boruvka de forma distribuída sobre uma aproximação do grafo dos k vizinhos mais próximos construído através de uma versão modificada do algoritmo *NNDescent* na plataforma *Apache Spark*. Em todos os casos abordados nos experimentos, a *MST* aproximada atingiu menos de 2% de custo superior ao custo da *MST* exata quando o parâmetro k do algoritmo *NNDescent* era apenas 1% da quantidade de pontos dos conjuntos de dados. Além disso, o método mostrou possuir complexidade computacional linearmente proporcional com a quantidade de pontos e levemente sublinear com relação ao parâmetro k.

Palavras-chave: árvore geradora mínima. *Apache Spark*. *NNDescent*.

Abstract

Finding the Minimum Spanning Tree (MST) consists of inducing a spanning tree whose sum of edge costs is minimal. Among many applications, the MST is widely used in unsupervised machine learning algorithms, such as clustering. With increasing volume, data used for analysis and decision making, platforms such as Apache Spark have become more influential. However, there is a difficulty in finding the MST in a distributed environment when the point-to-point distance needs to be calculated, since this calculation has a quadratic complexity with respect to the number of data points, making it very inefficient and even unfeasible for large data sets. In this work an efficient method to find a good approximation of the MST is presented. The method consists of applying the Boruvka algorithm in a distributed way over an approximated k nearest neighbors graph constructed through a modified version of the NNDescent algorithm on the Apache Spark platform. In all cases of experiments, the approximate MST reached less than 2% of the cost higher than the cost of the exact MST when the parameter k of the NNDescent algorithm was greater than just 1% of the number of points in the data sets. Furthermore, the method has showed computational complexity linearly proportional to the number of points and is slightly sublinear in relation to parameter k.

Keywords: *minimum spanning tree. Apache Spark. NNDescent.*

Sumário

1 Introdução

1.1 Contexto e problemática

Dado um grafo não-dirigido e ponderado G, induzir a árvore geradora mínima (do inglês *Minimum Spanning Tree – MST*) consiste em encontrar uma árvore geradora cuja soma dos custos seja mínima. A *MST*, como um grafo de proximidade, carrega consigo importantes informações estruturais acerca do grafo G por conectar todos os seus vértices com o menor custo possível e sem ciclos (SHIMOMURA et al., 2021).

Dentre as muitas aplicações da *MST* estão alguns métodos de aprendizado de máquina não supervisionados como algoritmos de agrupamento hierárquico e agrupamento por densidade (BATENI et al., 2017; SHAHZAD; COENEN, 2020). De acordo com Bateni et al. (2017), os agrupamentos baseados em ligações (do inglês *linkage-based*) são equivalentes a encontrar as *MSTs* nos conjuntos de dados. E além disso, a maior parte das novas técnicas de agrupamento baseadas na *MST* são sequenciais, o que gera dificuldade em suas aplicações em grandes conjuntos de dados, pois tanto o processamento quanto a memória necessária crescem em proporção quadrática à quantidade de pontos, pois é necessário calcular todas as arestas do grafo completo.

Com a ascensão da industria 4.0 no mundo, as áreas de tecnologia têm ganhado força ao passo que o volume de dados utilizados para análise e tomada de decisões tem crescido. Como previsto por Achsan et al. (2018), *Big Data* faz parte do grupo de segmentos que tem dominado as pesquisa nos últimos dois anos. Portanto, grandes conjuntos de dados têm se tornado comuns e o seus processamentos necessários, fazendo com que ambientes distribuídos, como a plataforma *Apache Spark*, que abstrai o particionamento dos dados e utiliza padrões de código que auxiliam no paralelismo, ganhem destaque (ZAHARIA et al., 2016).

O conceito de busca por similaridade (do inglês *Similarity Search*) consiste na busca pela (dis)similaridade entre os objetos em conjuntos de dados (SHIMOMURA et al., 2021). Esse conceito se relaciona com grafos de proximidade por resumirem as vizinhanças entre os objetos em um conjunto de dados e otimizarem a busca pelas suas distâncias quando considerado um espaço euclidiano. Um importante grafo de proximidade muito utilizado para buscas por similaridade é o grafo dos k vizinhos mais próximos (do inglês k *Nearest Neighbors Graph – k-NNG*). Esse grafo armazena as conexões com os k vizinhos mais próximos, ou seja, com os menores custos, para cada vértice.

Em Dong, Moses e Li (2011), o autor propôs um método heurístico denominado *NNDescent* para construir um *k-NNG* aproximado a partir de uma dissimilaridade ge-

nérica com complexidade empírica quase linear. Baseado nesta heurística, o algoritmo inicia com um *k-NNG* aleatório e a cada iteração procura por vizinhos de vizinhos na tentativa de encontrar vértices ainda mais próximos e assim otimizar sua vizinhança. Posteriormente, Warashina et al. (2014) propôs uma versão distribuída utilizando o modelo *MapReduce* que foi retomada em Brati et al. (2019) e obteve uma complexidade empírica $\Theta(|V|^{1,35})$. Essa versão escalável se mostrou quase tão eficiente quanto sua versão em memória compartilhada. Todas as arestas calculadas pelo *NNDescent* durante seu processo de otimização que não se incluem no *k-NNG* aproximado serão chamadas de arestas residuais. Além disso, as arestas residuais são informações de conexões cujas distâncias já foram calculadas durante o processo de otimização do algoritmo, ou seja, seu cálculo foi inevitável.

Dado o crescimento quadrático em função da quantidade de pontos no grafo, estratégias aproximadas se tornaram mais aceitáveis, fazendo com que um *k-NNG* aproximado obtido em tempo próximo a proporção linear utilizando o *NNDescent* seja uma boa abordagem para estimar as arestas presentes na *MST*. No entanto, nem sempre é possível se obter uma *MST* exata a partir de um *k-NNG*, pois mesmo que ele seja conexo, podem existir arestas que estão presentes na *MST* mas não no *k-NNG*. Todas as arestas que estão presentes na *MST* mas não são encontradas no *k-NNG* serão chamadas de arestas faltantes, representando a ideia de que estão em falta no *k-NNG* para se obter uma *MST* exata.

1.2 Objetivos

O objetivo deste trabalho é apresentar um método escalável para construir uma *MST* de maneira aproximada lidando com os dois problemas apresentados: (1) complexidade quadrática em função do número de pontos e (2) estimar boas aproximações para arestas da *MST* a partir do *k-NNG*. O método utiliza o algoritmo *NNDescent* para aproximar um *k-NNG* de forma eficiente, dado um conjunto de objetos no espaço euclidiano.

Além disso, o trabalho possui os seguintes objetivos específicos:

- Desenvolver um método distribuído e escalável baseado no algoritmo de Boruvka para construção da *MST* a partir de um grafo.
- Verificar a possibilidade de reutilizar as arestas residuais do *NNDescent* como boas estimativas de arestas faltantes, uma vez que não existe custo computacional extra em mantê-las.
- Através da experimentação, levantar medidas quantitativas da acurácia e a performance do método proposto.

1.3 Hipótese

O objetivo deste trabalho segue da hipótese de que a heurística do algoritmo *NN-Descent* faz com que suas arestas residuais sejam melhores candidatas a arestas faltantes do que arestas obtidas por amostragem aleatória. E com isso, é possível reutilizar as arestas residuais sem custo computacional extra para obter uma melhor aproximação da *MST*.

1.4 Percurso metodológico

Foi utilizado o método de pesquisa descritiva, com a finalidade de analisar o aproveitamento das arestas residuais para melhorar a aproximação da *MST* obtida a partir do algoritmo de Boruvka executado sobre o *k-NNG* construído pelo *NNDescent*.

Para avaliar o método proposto será utilizado um procedimento experimental com uma abordagem quantitativa. Os experimentos serão realizados em um computador com um processador AMD Ryzen™ 5 1600 com 12 *threads*, 16 GB de memória RAM rodando o sistema operacional Ubuntu 20.04.2 LTS e a plataforma Apache Spark 3.0.0.

Os conjuntos de dados que serão utilizados ao longo dos experimentos foram gerados a partir de bolhas gaussianas isotrópicas. No total, cinco conjuntos de dados foram construídos com três bolhas variando a dimensão e a quantidade de pontos entre 2 e 32, e entre 2000 e 8000, respectivamente. A tabela 1 descreve e atribui um identificador para cada base.

Tabela 1 – Descrição dos conjuntos de dados artificiais gerados para teste.

ID	Tipo de dado	Dimensão	Nº de pontos	Nº de bolhas
D2N2	Bolhas Gaussianas Isotrópicas	2	2000	3
D2N4	Bolhas Gaussianas Isotrópicas	2	4000	3
D2N8	Bolhas Gaussianas Isotrópicas	2	8000	3
D8N2	Bolhas Gaussianas Isotrópicas	8	2000	3
D32N2	Bolhas Gaussianas Isotrópicas	32	2000	3

Fonte: Author

Os testes foram elaborados visando obter medidas quantitativas do desempenho e da qualidade da árvore geradora aproximada com relação a *MST* exata. Além disso, cada teste será executado cinco vezes para que uma média seja obtida devido aos fatores aleatórios do algoritmo *NNDescent*.

Para medir o desempenho do algoritmo em função da quantidade de pontos, os conjuntos D2N2, D2N4 e D2N8 serão utilizados com $k = 25$. Da mesma forma, será variado k com os valores 10, 20 e 50 no conjunto D2N2 para medir o desempenho em função de k.

Posteriormente, será feita uma comparação entre os custos das aproximações e o custo da *MST* exata. Para essa comparação, as aproximações obtidas pelos métodos MSTN e MSTR com os mesmos parâmetros utilizarão um único *k-NNG* e um único conjunto de arestas amostradas para eliminar o fator aleatório. Além da comparação de custos, será realizada uma comparação de arestas para verificar a quantidade de arestas diferentes entre a *MST* exata e as aproximações.

Além disso, nas etapas de comparação de custo e quantidade de arestas diferentes, será executado um método para encontrar k vizinhos aleatórios que será considerado um caso ruim. Então a *MST* será construída a partir do grafo aleatório para que o seu custo relativo e sua quantidade de arestas diferentes da *MST* exata sejam comparados com o método proposto.

1.5 Síntese dos resultados obtidos

Os resultados obtidos foram promissores e validaram a hipótese deste trabalho em partes. O método proposto utilizando as arestas residuais de fato mostrou que as aproximações de *MSTs* obtidas foram levemente melhores do que as aproximações encontradas pelo método ingênuo. No entanto, o método ingênuo, que não utiliza as arestas residuais, se mostrou mais rápido devido a menor quantidade de arestas no grafo utilizado para construir a *MST*, embora o *NNDescent* não tenha consumido mais tempo computacional ao mante-las.

1.6 Organização do texto

Para melhor situar o leitor, apresenta-se aqui uma breve explicação da estrutura deste trabalho. Ele e dividido em 5 capítulos abrangendo a fundamentação teórica, o método proposto, os resultados experimentais e a conclusão. Mais especificamente:

- No Capítulo 2, apresenta-se a fundamentação teórica, onde é detalhado os conceitos de similaridade entre objetos, alguns tipos de grafos importantes para este trabalho, o algoritmo de Boruvka e o *NNDescent*, bem como a plataforma *Apache Spark*. Nesse capítulo também pode ser encontrado uma breve discussão sobre trabalhos relacionados.

- No Capítulo 3, o método proposto é detalhado através de explicações e descrições algorítmicas.

- No Capítulo 4 todos os resultados obtidos pelos experimentos são mostrados e interpretados.

- No Capítulo 5 discute-se a conclusão obtida a partir dos resultados e os trabalhos futuros.

2 Fundamentação teórica

Neste capítulo o leitor encontrará os principais conceitos teóricos nos quais este trabalho se fundamenta. Aqui é dado ênfase nos grafos sendo utilizados como formas de representar relações de proximidade e algoritmos para extrair grafos com essa característica, bem como a plataforma utilizada para escalar o método proposto.

2.1 Similaridade entre objetos

O conceito de similaridade entre objetos vem da ideia de se encontrar elementos que são semelhantes a outros elementos tomados como referencia de acordo com suas características (SHIMOMURA et al., 2021). De acordo com Shimomura et al. (2021), um espaço de similaridade é composto por um conjunto de vetores de características, onde cada vetor descreve de maneira intrínseca um dado complexo[1], e uma função de (dis)similaridade que descreve o quão similar (ou diferente no caso da dissimilaridade) dois vetores de características são.

No âmbito de consultas de similaridade, um dos tipos mais básicos é a busca pelos k vizinhos mais próximos (k-NN). Esse tipo de busca consiste em encontrar os k vizinhos mais próximos de acordo com uma função de (dis)similaridade dada (SHIMOMURA et al., 2021).

É comum encontrar o conceito de similaridade sendo representado por uma função de distância δ. Dessa forma, a distância entre dois vetores de características passa a representar a dissimilaridade entre os mesmos (ZEZULA et al., 2006). Logo, um espaço de similaridade é comumente modelado como um espaço métrico. Um espaço métrico é um conjunto $\mathbb{S}$ que possuí uma função $\delta : \mathbb{S} \times \mathbb{S} \longrightarrow \mathbb{R}^+$ de relação, denominada métrica, satisfazendo as quatro propriedades seguintes (ZEZULA et al., 2006):

Dados $x, y, z \in \mathbb{S}$,

1. Não negatividade – $\delta(x, y) \geq 0$;
2. Simetria – $\delta(x, y) = \delta(y, x)$;
3. Indentidade – $\delta(x, y) = 0 \iff x = y$;
4. Desigualdade triangular – $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$.

[1] Muitas vezes o vetor de características representa um objeto através de suas características.

Dentre várias funções de distância, segundo Shimomura et al. (2021), as mais utilizadas são as chamadas normas L_p (ou normas Minkowski) para $1 \leq p \leq \infty$. Ainda dentre essas, destacam-se a L_1 (distância Manhattan), L_2 (distância Euclidiana) e L_∞ (distância Chebyshev). A norma Minkowski é dada pela equação 2.1 onde n representa o comprimento do vetor de características.

$$\delta(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{2.1}$$

Algumas vezes é necessário usar uma função de (dis)similaridade que não garante todas as propriedades de uma métrica. Quando esse é o caso, o espaço é denominado não métrico (SHIMOMURA et al., 2021). Um exemplo de dissimilaridade não métrica é a distância Cosseno, que consiste em calcular o cosseno do ângulo formado pelos dois vetores de características comparados (WEINSHALL; JACOBS; GDALYAHU, 1998).

2.2 Grafos

Nesta seção é descrito de maneira sucinta a definição de um grafo e algumas propriedades importantes para o compreendimento das subseções 2.2.2 e 2.2.3.

Os grafos são vértices com conexões entre vértices, dando a ideia de vizinhança. Formalmente um grafo é definido como um par $G = (V, E)$, onde V é o conjunto finito de vértices e $E \subseteq \{\{a, b\} \in V^2\}$ representa arestas que conectam pares não ordenados de vértices em v. Um grafo denominado dirigido, é um grafo onde suas arestas possuem direção, ou seja, partem de um vértice e chegam em outro mas sem o caminho contrário. Neste trabalho será considerado apenas grafos não-dirigidos (SZWARCFITER, 1988).

As arestas de um grafo podem ser ponderadas, nesse caso elas possuem uma propriedade extra que determina seu peso. É chamado de passeio uma sequência de vértices em V tais que cada um desses vértices se conecta através de um aresta com o seu sucessor na sequência. Quando o passeio não possui arcos repetidos, é dito que a sequência é um caminho. A definição de caminhos nos trás a ideia de alcance, onde dados dois vértices $a, b \in V$, a alcança b se existe um caminho que os conecta, e vice-versa. Dito isso, dois vértices são conectados quando existe um caminho que os conecta (BOAVENTURA NETTO, 2006). Quando todos os vértices são conectados entre si, é dito que o grafo é conexo.

Um grafo $H = (U, F)$ é dito subgrafo de um grafo $G = (V, E)$ quando $U \subseteq V$ e $F \subseteq E$. Particularmente, quando $U = V$ o grafo H é denominado grafo gerador. Quando um subgrafo $C \subseteq G$ é conexo e não é possível aumentar seu tamanho adicionando mais vértices e arestas contidas em G de forma que ele continue conexo, então C é chamado de componente conectado.

2.2.1 Grafos de Proximidade

Em um grafo as arestas podem abstrair critérios de conexão. Um dos desses critérios pode ser o critério de vizinhança, onde uma conexão entre dois vértices representa que eles são vizinhos. Nesse caso, é comum que as arestas sejam ponderadas e seus pesos representem a distância entre os vértices (SHIMOMURA et al., 2021).

Os grafos de proximidade (GP), são grafos onde o par $u, v \in V$ é conectado por uma aresta $e = (u, v), e \in E$, se e somente se u e v satisfazem um critério de vizinhança definido P (OCSA; BEDREGAL; CUADROS-VARGAS, 2007).

Um dos GPs mais importantes na literatura é o grafo de Delaunay (DG). Este grafo é equivalente ao diagrama de Voronoi e consiste em um grafo planar onde os pontos situados em regiões de Voronoi adjacentes são conectados por uma aresta (FORTUNE, 1992; AURENHAMMER, 1991). Os subgrafos mais representativos do DG e da sua propriedade de proximidade P, considerando o espaço euclidiano são o grafo de Gabriel (GG), o grafo de vizinhança relativa (RNG), a árvore geradora mínima (MST) e o grafo do vizinho mais próximos (NNG) (OCSA; BEDREGAL; CUADROS-VARGAS, 2007). A equação 2.2 mostra a relação entre esses grafos (SHIMOMURA et al., 2021).

$$NNG \subseteq MST \subseteq RNG \subseteq GG \subseteq DG \tag{2.2}$$

2.2.2 *K* vizinhos mais próximos

O grafo dos k vizinhos mais próximos (k-NNG) é definido como $G = (V, E)$, onde $E = \{(u, v, \delta(u, v) \mid v \in NN_k(u)_\delta\}$ tal que $NN_k(u)_\delta$ é o conjunto contendo os k vizinhos mais próximos de u no conjunto V com respeito a função de similaridade δ (SHIMOMURA et al., 2021). As arestas do k-NNG podem ser dirigidas ou não-dirigidas, e dependendo do valor k ele pode ser desconexo. Um exemplo de k-NNG com $k = 2$ pode ser visto na Figura 1.

2.2.3 Árvore geradora mínima

Para compreender o que é uma árvore geradora mínima (MST), é necessário primeiro compreender o que é uma árvore e o que é o custo do grafo. Uma árvore nada mais é do que um grafo conexo e acíclico. Já o custo de um grafo corresponde a soma dos pesos de suas arestas.

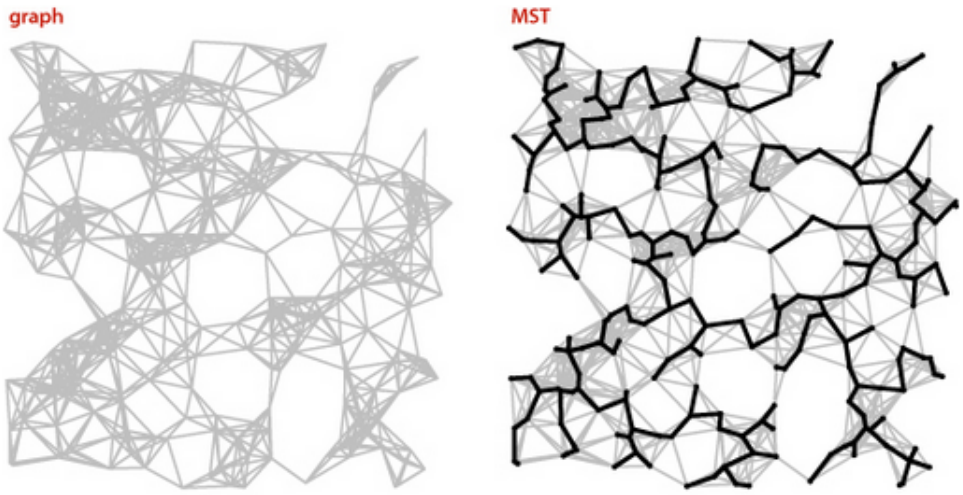
Sabendo disso, dado um grafo G, uma MST para esse grafo é o subgrafo gerador $T \subseteq G$ conexo cujo custo das arestas é mínimo. A Figura 2 mostra um MST construída a partir de um grafo.

Figura 1 – 2-NNG formado a partir de um conjunto de dados que assemelha-se a uma ferradura.

(a) Conjunto de pontos quaisquer. (b) k-NNG com $k = 2$.

Fonte: Autor

Figura 2 – Exemplo de MST (direita) extraída de um grafo (esquerda).



Fonte: (SEDGEWICK; WAYNE, 2011)

2.2.4 A relação entre o *k-NNG* e a *MST*

Como visto na Equação 2.2, é conhecido que o $NNG \subseteq MST$, mas quando consideramos um k-NNG com $k > 1$, não se pode estabelecer esse tipo de relação de maneira genérica com a MST. No entanto o k-NNG trás consigo noções de proximidade que podem ser utilizadas para estimar arestas que podem estar presentes na MST, chamadas de arestas candidatas.

Definimos aqui o conceito de aresta faltante se referindo a toda aresta $v \in E_{MST} - E_{k-NNG}$, ou seja, toda aresta que está na MST mas não está no k-NNG. É fácil notar que mesmo quando o k-NNG é conexo, ele pode não conter a MST. Tomando novamente o exemplo dado pela Figura 1, a Figura 3 mostra a MST construída a partir do conjunto de dados considerando a distância euclidiana e a MST construída a partir do grafo 2-NNG.

Figura 3 – Exemplo da ferradura – Comparação entre a *MST* extraída de um grafo completo e a *MST* extraída de um *2-NNG*.

(a) *MST*. (b) *MST* extraída a partir do *2-NNG*.

Fonte: Autor

Esse exemplo será chamado por exemplo da ferradura.

2.3 Boruvka

Existem vários algoritmos que resolvem o problema da *MST*, como os clássicos Prim e Kruskal. No entanto, neste trabalho destaca-se o algoritmo de Boruvka por ser conhecido como naturalmente paralelizável (BADER; CONG, 2006). De acordo com Bader e Cong (2006), algoritmo de Boruvka é composto de iterações com três etapas mandatórias:

1. **Encontra arestas mínimas:** para cada vértice, encontra a aresta de menor peso e a marca como presente na *MST*.

2. **Conecta componentes:** identifica os componentes conexos no grafo induzido pelas arestas marcadas na etapa 1.

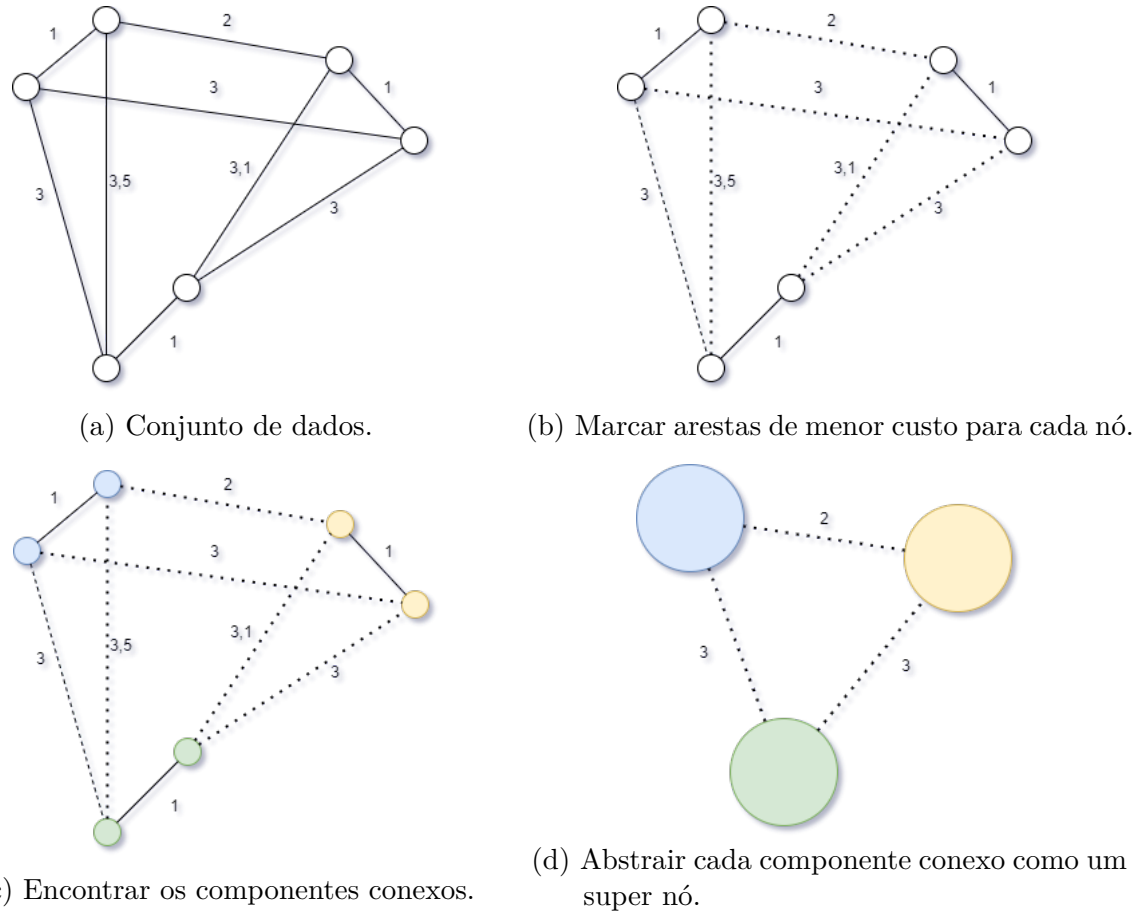
3. **Compacta o grafo nos componentes conexos:** compacta cada componente conexo em um super vértice, remove as arestas para si mesmos e renomeia os vértices para manter a consistência.

A Figura 4 ilustra os passos descritos acima.

2.4 *NNDescent*

O algoritmo *NNDescent* possui uma abordagem gananciosa para construir uma aproximação do *k-NNG* com um baixo custo computacional, empiricamente estimado como $\Theta(|V|^{1,14})$ (DONG; MOSES; LI, 2011). Essa abordagem utiliza uma função de

Figura 4 – Exemplo de iteração do algoritmo de Boruvka



(a) Conjunto de dados.

(b) Marcar arestas de menor custo para cada nó.

(c) Encontrar os componentes conexos.

(d) Abstrair cada componente conexo como um super nó.

Fonte: Autor

similaridade arbitrária e baseia-se em uma heurística que diz que os vizinhos de um vizinho provavelmente também são vizinhos.

Dado um conjunto de vértices V, uma similaridade arbitrária σ, a quantidade de vizinhos k, uma taxa de amostragem ρ e um limitante inferior de mudanças δ para término antecipado, o algoritmo inicia construindo uma vizinhança de k vértices aleatórios para cada vértice $v \in V$. Após isso, esse método melhora a aproximação inicial de forma iterativa, comparando cada vértice v com ρ amostras de vizinhos de seus vizinhos e também com sua vizinhança reversa $\{u \mid v \in k\text{-}NN(u)\}$, finalizando quando a quantidade de vizinhos mais próximos atualizada for menor do que o limitante δ (DONG; MOSES; LI, 2011).

Apesar desse método ser eficiente com relação a quantidade de objetos no conjunto de dados, seu desempenho diminui rápido conforme aumenta-se a quantidade de vizinhos desejada. Além disso, esse método tem sua qualidade prejudicada conforme a dimensionalidade do conjunto de dados aumenta. Isto se deve ao *hubness phenomenon*, que é um fenômeno derivado da maldição da dimensionalidade (LOW et al., 2013; BRATI et al.,

Algoritmo 1: NNDescent

Entrada: conjunto de dados V, função de similaridade σ, K, ρ, δ
Resultado: k-NNG
início
  $B[v] \longleftarrow Amostragem(V, K) \times \{\langle \infty, 'verdadeiro' \rangle\}\ \forall v \in V$
  repita
    para $v \in V$ **faça**
      $antigo[v] \longleftarrow$ todos os items em $B[v]$ com um indicador 'falso'
      $novo[v] \longleftarrow \rho K$ items em $B[v]$ com um indicador 'verdadeiro'
      Marca todos os items amostrados em $B[v]$ como 'falso'
    fim
    $antigo' \longleftarrow Inverso(antigo)$
    $novo' \longleftarrow Inverso(novo)$
    para $v \in V$ **faça**
      $antigo[v] \longleftarrow antigo[v] \bigcup Amostragem(antigo'[v], \rho K)$
      $novo[v] \longleftarrow novo[v] \bigcup Amostragem(novo'[v], \rho K)$
      para $u1, u2 \in novo[v], u1 < u2$ *ou* $u1 \in novo[v], u2 \in antigo[v]$ **faça**
        $l \longleftarrow \sigma(u1, u2)$
        $c \longleftarrow c + AtualizaNN(B[u1], \langle u2, l, 'verdadeiro' \rangle)$
        $c \longleftarrow c + AtualizaNN(B[u2], \langle u1, l, 'verdadeiro' \rangle)$
      fim
    fim
  até $c < \delta NK$;
  retorna B
fim

função $Amostragem(S, n)$ **início**
  retorna n *amostras do conjunto* S
fim

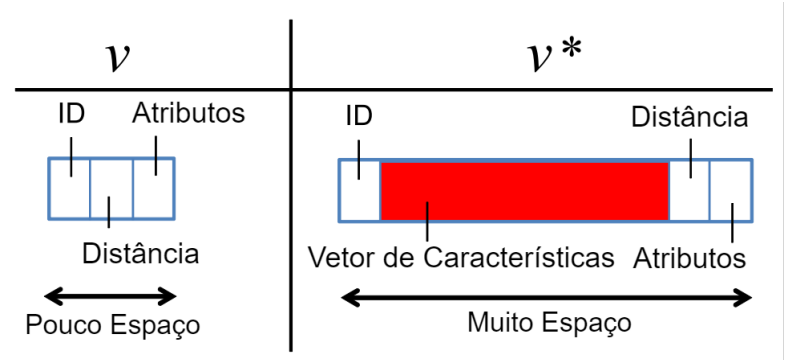
função $Inverso(B)$ **início**
  $R[v] \longleftarrow \{u \mid \langle v, ... \rangle \in B[v]\} \forall v \in V$
  retorna R
fim

função $AtualizaNN(H, \langle u, l, ... \rangle)$ **início**
  Atualiza a vizinhança H
  retorna *1 se mudou, ou 0 caso não tenha mudado.*
fim

2019). Esse fenômeno diz que conforme a dimensão do conjunto de dados aumenta, alguns poucos pontos se tornam "*hubs*", concentrando de forma muito desproporcional a maior parte das conexões em um grafo de k vizinhos mais próximos, ou seja, possuindo um grau muito alto.

De acordo com Brati et al. (2019), entretanto, é possível melhorar a qualidade do grafo estimado aumentando a quantidade de vizinhos mais próximos (parâmetro k), reduzindo os efeitos do *hubness phenomenon*. Desta forma é possível obter uma boa apro-

Figura 5 – Vetor representando um objeto de um conjunto de dados.



Fonte: Warashina et al. (2014)

ximação com k grande o suficiente.

2.4.1 *NNDescent* utilizando *Hadoop MapReduce*

Posteriormente, Warashina et al. (2014) desenvolveu um método eficiente para construir o *k-NNG* em grandes volumes de dados utilizando o arcabouço Hadoop MapReduce (LEE et al., 2012). Nesse método, o autor compôs duas estruturas para armazenar cada objeto do conjunto de dados sem replicar suas características. A primeira não armazena as características, e é denotada sem a utilização do caractere "*", como pode ser observado na figura 5. Já a segunda possui o vetor de características e é denotada com o caractere "*".

Cada iteração dessa versão atualiza as vizinhanças utilizando quatro rotinas: *CreateRevVertices*, *CreateKVertices*, *CreateAdjVertices* e *UpdateKVertices*. Além disso, cada uma destas etapas é composta por uma operação *Map*, uma operação de agregação por chave descrita como *Shuffle* e uma operação *Reduce* (WARASHINA et al., 2014; LEE et al., 2012).

Operações de *Map* aplicadas em um vetor mapeiam cada uma dos valores no vetor de acordo com uma função de mapeamento. Essa função de mapeamento não possui dependências externas e pode mapear cada valor do vetor de forma independente e paralela. Uma operação de *Reduce* reduz um vetor para um único valor através de uma função que se aplica a cada dois elementos os reduzindo para um, como por exemplo soma ou multiplicação. No caso da soma, o vetor é reduzido para a soma de todos os seus valores, enquanto a multiplicação reduz o vetor para a multiplicação de todos os seus valores.

A operação descrita como *Shuffle* no entanto, utiliza uma chave presente em cada elemento do vetor para agrupar os elementos com a mesma chave em um mesmo vetor, criando vetores com elementos de mesma chave.

A etapa *CreateRevVertices* recebe uma coleção de pares contendo os objetos e suas

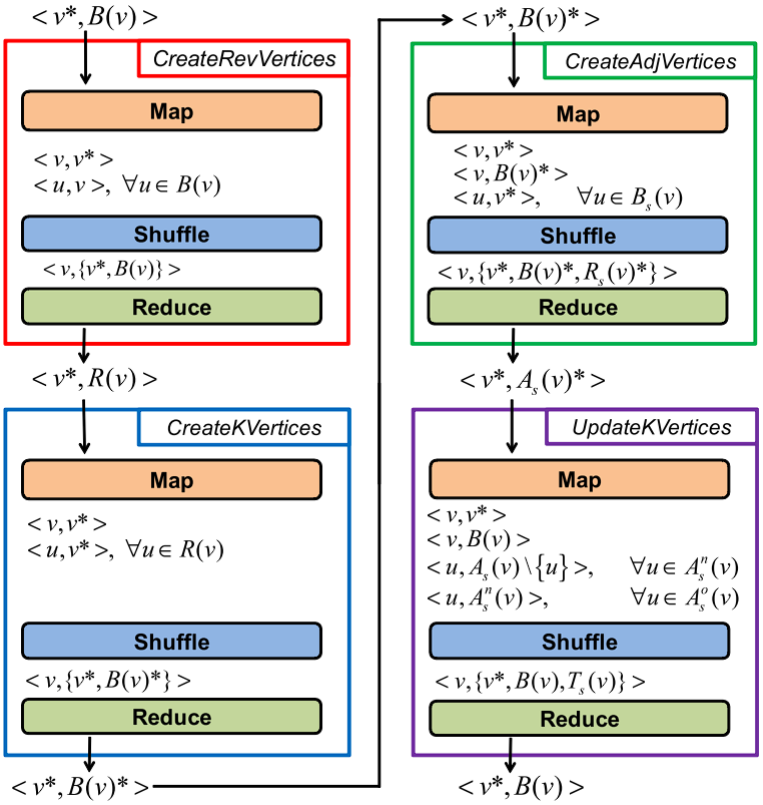
vizinhanças $\langle v^*, B(v)\rangle$. É então responsável por computar a vizinhança reversa de cada objeto e retornar em par com o objeto correspondente $\langle v^*, R(v)\rangle$.

A partir da coleção de $\langle v^*, R(v)\rangle$, a rotina *CreateKVertices* constrói novamente a aproximação da vizinhança dos k mais próximos $B(v)^*$ utilizando uma amostra aleatória da vizinhança reversa $R(v)$.

Na etapa *CreateAdjVertices*, é computado o conjunto de vértices adjacentes $A(v) = B(v) \bigcup R_s(v)$ para todos os objetos v, onde $R_s(v)$ represente os objetos amostrados da vizinhança reversa.

Por fim, a rotina *UpdateKVertices* calcula a distância entre os objetos novos e os antigos e atualiza a lista de vizinhos $B(v)$ para todos os objetos, emitindo novamente o par $\langle v^*, B(v)\rangle$.

Figura 6 – Iteração do algoritmo *NNDescent* utilizando o arcabouço *MapReduce*.



Fonte: Warashina et al. (2014)

De acordo com Warashina et al. (2014), diferente do método original sequencial, a versão distribuída demonstrou possuir uma complexidade empírica de $\Theta(|V|^{1,35})$ com relação ao aumento de vértices e uma complexidade levemente sublinear com o aumento da quantidade de vizinhos k. Os testes realizados com cinco vezes mais unidades de processamento apresentaram um *speedup* entre 4 e 4,5.

2.5 *Apache Spark*

O projeto *Apache Spark* foi iniciado em 2009 com o intuito de criar uma plataforma unificada para processamento de dados de maneira distribuída (ZAHARIA et al., 2016). Essa plataforma estendeu o *MapReduce* criando um modelo de abstração de compartilhamento de dados chamado *Resilient Distributed Dataset* (RDD). Esse modelo permite que a plataforma *Spark* consiga processar uma vasta quantidade de diferentes tipos de dados que anteriormente necessitava de outras plataformas, como SQL, *streaming*, aprendizado de máquina e processamento de grafos.

2.6 Trabalhos relacionados

Em Arefin et al. (2012), o autor propõe o uma abordagem escalável para aplicar o método de agrupamento *kNN-MST-Agglomerative* que executa em GPU. A abordagem baseia-se na construção de um *k-NNG* com $k = \lfloor \ln(n) \rfloor + c$, onde n é a quantidade de pontos do conjunto de dados e c é uma constante pequena, seguido de uma aplicação do algoritmo de Boruvka para extrair a *MST*. Em seguida, é realizado um agrupamento aglomerativo. O autor concluiu que a abordagem obteve um agrupamento relevante ao comparar com outros algoritmos estado da arte, além de ser um método eficiente com relação a memória graças a utilização do *k-NNG* para reduzir a quantidade de arestas armazenadas. Apesar desse trabalho ter sido desenvolvido para GPU e não ser distribuído, ele demonstra que existe uma necessidade por processar maiores quantidades de dados e que utilizar um *k-NNG* para diminuir a quantidade de arestas e se obter uma *MST* aproximada é uma boa estratégia.

Um método denominado *affinity clustering* para agrupamento hierárquico em escala baseado no modelo *MapReduce* é descrito no trabalho Bateni et al. (2017). O método busca adaptar métodos de agrupamento equivalentes ao problema da *MST* para uma forma paralela e distribuída. Assim como em nosso trabalho, o autor aborda o problema de encontrar a *MST* em um ambiente distribuído incluindo o algoritmo de Boruvka dentre os algoritmos estudados. No entanto duas soluções baseadas no particionamento do espaço foram apresentadas. A primeira assume que o grafo é completo e encontra uma *MST* exata, a segunda utiliza uma estrutura denominada *distributed hash table* que particiona os objetos fazendo com que pontos próximos no espaço possuam alta probabilidade de colidirem através de uma função de *hash*.

No trabalho desenvolvido por Jothi, Mohanty e Ojha (2018), foi proposto um método de agrupamento baseado em uma *MST* aproximada utilizando um particionamento baseado no *NNG* para reduzir a quantidade de arestas calculadas e o tempo computacional. O autor demonstrou que o novo método calcula todas as arestas necessárias para aproximar a *MST* em $O(n^{3/2})$ em contraste com a forma exata de calcular a distância

ponto a ponto em $O(n^2)$. Além disso, verificou a quantidade de arestas erradas e a diferença de peso entre a *MST* aproximada e a *MST* exata e concluiu que essa diferença não influenciou nos grupos obtidos através do agrupamento proposto. Apesar desse trabalho possuir um princípio semelhante ao nosso, legitimando nossa motivação, ele foi desenvolvido de forma sequencial.

Em Shahzad e Coenen (2020) é proposto um método de agrupamento distribuído baseado em *MST*. O método é composto por um algoritmo de Kruskal distribuído proposto pelo autor. O método pressupõe que as arestas já foram calculadas e as distribui igualmente entre as unidades de processamento. Ele utiliza uma estrutura conhecida como *disjoint sets* que foi desenvolvida em cima de duas tabelas de *hash*. Assim como no nosso trabalho, o autor utiliza a plataforma *Spark* e busca encontrar uma *MST* de forma distribuída. No entanto, ele pressupõe que as arestas entre os pontos já foram calculadas e implementa apenas o método da *MST* sobre as arestas conhecidas para utilizar no agrupamento, diferente do método proposto em nosso trabalho, onde conhecemos apenas os pontos no espaço, e nada sobre suas distâncias.

3 Método proposto

Dado um conjunto de vetores de características, duas abordagens serão apresentadas para construir uma *MST* sobre esse conjunto em três principais etapas:

1. **Construção de um *k-NNG* aproximado:** a partir do conjunto de vetores de características, constrói-se o *k-NNG* aproximado utilizando o algoritmo *NNDescent*.
2. **Construção da *MST/MSF* sobre o *k-NNG*:** utilizando uma implementação distribuída do algoritmo de Boruvka, construir uma *MST* sobre o resultado da etapa anterior. Caso o grafo obtido anteriormente não seja conexo, constrói-se uma floresta geradora mínima (do inglês *Minimim Spanning Forest – MSF*) aproximada.
3. **Amostragem para estimar AFs e completar a *MST*:** A partir do resultado da etapa anterior, estima-se as AFs que conectam as árvores da *MSF* para aproximar a *MST*.

3.1 Abordagens estudadas

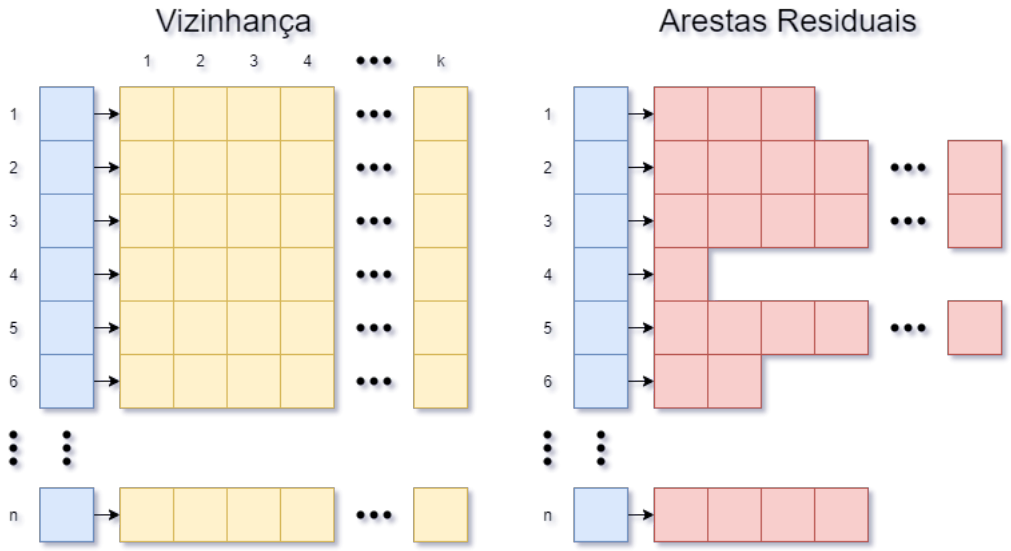
A primeira abordagem consiste em aplicar a primeira etapa da maneira mais ingênua, aplicando a versão do *NNDescent* descrita na Seção 2.4.1. Essa abordagem será denominada daqui em diante como Árvore Geradora Mínima - Ingênua (*MSTN*, do inglês *Minimum Spanning Tree - Naive*).

Já a segunda abordagem utiliza da hipótese deste trabalho. Durante o processo de otimização das vizinhanças de cada ponto, o *NNDescent* calcula muitas arestas que no final do processo não estão entre os k vizinhos. No entanto, essas arestas, denominadas arestas residuais, são estimadas de acordo com a heurística de conectar possíveis vizinhos. Logo espera-se que elas sejam boas candidatas a arestas que fazem parte da *MST* mas não estão entre os k vizinhos mais próximos. Além disso, seu cálculo faz parte do processo de otimização, e armazená-las não exige custo computacional adicional, embora custe mais memória. A partir dessa informação, é proposta uma verão modificada do *NNDescent* para armazenar as arestas residuais. Essa versão compõe a abordagem que será denominada Árvore Geradora Mínima - Residual (*MSTR*, do inglês *Minimum Spanning Tree - Residual*).

O *NNDescent* modificado é construído de maneira semelhante a sua versão distribuída descrita na Seção 2.4.1. A única diferença é a adição de uma estrutura de lista para cada vértice para armazenar as arestas residuais obtidas no processo de otimização. A Figura 7 exemplifica o armazenamento em formato de lista de adjacências. Cada bloco

azul corresponde a um ponto do conjunto de dados, e os blocos amarelos e vermelhos possuem o identificador do ponto vizinho e sua distância para o ponto no respectivo bloco azul. Os elementos das listas são armazenados ordenados de acordo com a distância até o respectivo ponto.

Figura 7 – Exemplo de armazenamento da vizinhança e armazenamento das arestas residuais em estruturas de lista de adjacências.



Fonte: Autor

Para que uma aresta seja considerada residual e assim adicionada na lista de arestas residuais, um de dois casos deve acontecer:

1. A aresta pertencia a vizinhança dos k mais próximos mas foi substituída por arestas menores que foram encontradas no processo de otimização.

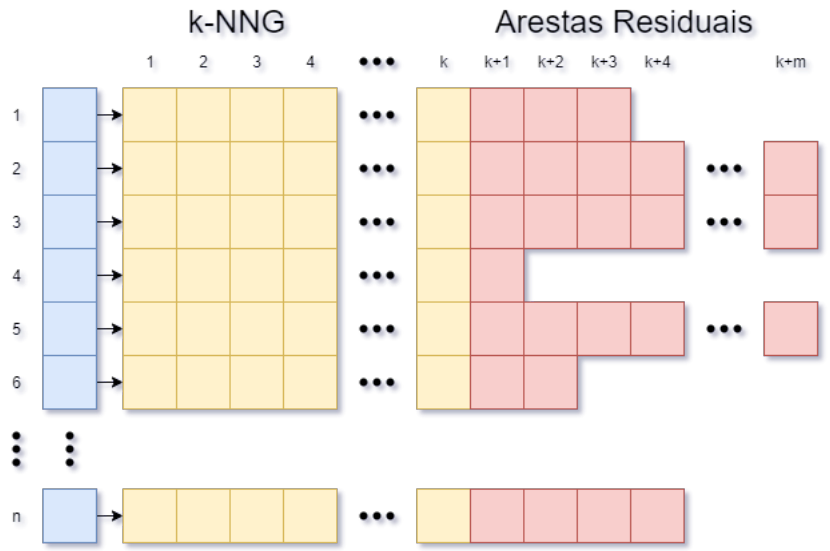
2. A aresta era candidata a pertencer a vizinhança dos k mais próximos, mas ao calcular sua distância foi verificado que seu custo era maior do que o custo das k arestas mais próximas naquela iteração.

Quando os critérios de parada do *NNDescent* modificado são alcançados, as arestas residuais são adicionadas ao grafo para sua utilização na construção da *MST*. Veja a Figura 8.

3.2 Boruvka distribuído

O algoritmo foi desenvolvido para trabalhar com um grafo disposto na forma de lista de adjacência. Ele possui quatro rotinas principais, elas são: **pegarCandidatos**,

Figura 8 – *k-NNG* mais arestas residuais ao fim do *NNDescent* modificado.



Fonte: Autor

Algoritmo 2: Boruvka-distribuído

Entrada: *k-NNG*
Resultado: *MST* ou *MSF*
início
  $G \longleftarrow k - NNG$
  $E \longleftarrow$ // conjunto de arestas da MST
  $C \longleftarrow \{\langle v, v\rangle \mid v \in V\}$ // par vértice com o componente a qual pertence
  $candidatos \longleftarrow pegarCandidatos(G)$
  repita
    $candidatosSeguros \longleftarrow pegarCandidatosSeguros(candidatos, C)$
    $E \longleftarrow E \bigcup candidatosSeguros$
    $C \longleftarrow encontrarComponentesConectados(E, C)$
    $G \longleftarrow removerNãoCandidatos(G, C)$
    $candidatos \longleftarrow pegarCandidatos(G)$
  até $conta(candidatos) = 0$;
  retorna E
fim

removerNãoCandidatos, **pegarCandidatosSeguros** e **encontrarComponentesConectados**.

A rotina **pegarCandidatos** consiste em, para cada nó do grafo, pegar o vizinho mais próximo que ainda está presente no grafo. Essa rotina corresponde diretamente à primeira etapa do método de Boruvka descrito na Seção 2.3. Veja o Algoritmo 3.

Na rotina **removerNãoCandidatos**, os rótulos dos componentes conexos são distribuídos entre todos os nós e então todos os vizinhos que pertencem ao mesmo componente conexo do nó ao qual a lista de adjacência se refere são removidos da vizinhança.

Algoritmo 3: pegarCandidatos

Entrada: Grafo como lista de adjacência
Resultado: O elemento mais próximo para cada vértice $v \in V$

função Map($\langle v$,componente$_v$, vizinhos$\rangle$) **início**
  $\langle u, w_u, componente_u \rangle \longleftarrow vizinhos[0]$
  retorna $\langle (componente_v,\ componente_u), (v, u, w_u) \rangle$
fim

Veja o Algoritmo 4.

A rotina **pegarCandidatosSeguros** corresponde à terceira etapa descrita na Seção 2.3. O objetivo dessa rotina é abstrair os componentes conexos como se fossem nós e encontrar a aresta de menor custo que sai de cada um deles. Para isso, as arestas obtidas na etapa **pegarCandidatos** são emitidas duas vezes, cada uma com um rótulo de um dos componentes a qual ela conecta como chave. Na sequência, as arestas são agrupadas por chave formando uma grafo representado como uma lista de adjacência onde cada nó chave é um componente conexo. Ao final, é retornada a aresta de menor custo para cada componente conexo. Veja o Algoritmo 5

Por fim, a etapa **encontrarComponentesConectados** apenas encontra os componentes conexos no grafo que esta sendo induzido. Pode-se utilizar qualquer método distribuído para encontrar os componentes conexos e neste trabalho foi utilizado um método disponibilizado pela biblioteca *GraphX*, que é disponibilizada pela própria plataforma *Spark*.

Ao final do algoritmo de Boruvka, caso uma *MSF* tenha sido obtida, é necessário conectar os as árvores geradas. Para isso, são estimadas arestas amostrando cinco pontos em cada componente conectado e calculando a distância ponto a ponto desde os dois pontos não pertençam ao mesmo componente conexo. Em seguida são selecionadas as arestas de menor custo por componente e adicionadas na *MSF* para formar uma *MST*. Essa estratégia de amostragem é utilizada nas duas abordagens sempre que o resultado do algoritmo de Boruvka for uma *MSF*.

Algoritmo 4: removerNãoCandidatos

Entrada: Grafo como lista de adjacência
Resultado: Grafo sem vizinhos dentro do mesmo componente conectado

função LeftJoin($\langle v$, componenteAntigo$_v$, vizinhosComComponentesAntigos$\rangle$, $\langle v$, componente$_v\rangle$) **início**
  retorna $\langle v$, *componenteAntigo$_v$, vizinhosComComponentesAntigos, componente$_v\rangle$*
fim

função FlatMap($\langle v$, componenteAntigo$_v$, vizinhosComComponentesAntigos, componente$_v\rangle$) **início**
  para $\langle u, w_u$, *componenteAntigo$_u\rangle \in$ vizinhosComComponentesAntigos* **faça**
    Emitir $\langle u$, (v, w_u, componente$_v)\rangle$
  fim
fim

função LeftJoin($\langle u$, (v, w_u, componente$_v)\rangle$, $\langle u$, componente$_u\rangle$) **início**
  retorna $\langle u, (v, w_u,$ *componente$_v$*$)$, *componente$_u\rangle$*
fim

função Map($\langle u, (v, w_u,$ componente$_v)$, componente$_u\rangle$) **início**
  retorna $\langle ($*componente$_v$*$), (u, w_u,$ *componente$_u)\rangle$*
fim

função GroupByKey($\langle\langle (v,$ componente$_v), (u, w_u,$ componente$_u)\rangle\rangle$) **início**
  retorna $\langle ($*componente$_v$*$)$, *vizinhosComComponentes* $\rangle$
fim

função Map($\langle (v,$ componente$_v)$, vizinhosComComponentes$\rangle$) **início**
  vizinhos $\longleftarrow \{\langle u, w_u,$ *componente$_u\rangle \in$ vizinhosComComponentes* | *componente$_v \neq$ componente$_u\}$*
  retorna $\langle v$, *componente$_v$*, *vizinhos*$\rangle$
fim

Algoritmo 5: pegarCandidatosSeguros

Entrada: Arestas candidatas
Resultado: A menor aresta candidata por componente conectado

função FlatMap($\langle$(componente$_v$, componente$_u$), $(v, u, w)\rangle$) **início**
  Emite $\langle$componente$_v$, ((componente$_v$, componente$_u$), (v, u, w))$\rangle$
  Emite $\langle$componente$_u$, ((componente$_v$, componente$_u$), (v, u, w))$\rangle$
fim

função ReduceByKey($\langle$(componente$_v$, componente$_u$), $(v, u, w_1)\rangle$, $\langle$(componente$_j$, componente$_k$), $(j, k, w_2)\rangle$) **início**
  se $w_1 \leq w_2$ **então**
    retorna $\langle v, u, w_1\rangle$
  fim
  senão
    retorna $\langle j, k, w_2\rangle$
  fim
fim

4 Experimentos

Neste capítulo são detalhados os resultados obtidos de acordo com a metodologia adotada. Começando pelos experimentos de desempenho computacional variando a quantidade de pontos, depois variando k. Na sequência, são mostrados os resultados obtidos pela comparação dos custos das árvores geradoras aproximadas e da diferença de arestas.

Na Tabela 2 são retomadas informações acerca dos conjuntos de dados definidos anteriormente e utilizados nos experimentos.

Tabela 2 – Descrição dos conjuntos de dados artificiais gerados para teste.

ID	Tipo de dado	Dimensão	Nº de pontos	Nº de bolhas
D2N2	Bolhas Gaussianas Isotrópicas	2	2000	3
D2N4	Bolhas Gaussianas Isotrópicas	2	4000	3
D2N8	Bolhas Gaussianas Isotrópicas	2	8000	3
D8N2	Bolhas Gaussianas Isotrópicas	8	2000	3
D32N2	Bolhas Gaussianas Isotrópicas	32	2000	3

Fonte: Author

Ao realizar os testes variando a quantidade de pontos, notou-se que o tempo computacional do *NNDescent* se manteve semelhante nos dois métodos. Em contrapartida o tempo da *MST* cresceu consideravelmente. Veja na Tabela 3 e na Figura 9a.

Tabela 3 – Tempo de execução variando o número de vértices, com $k = 25$.

Método	Base	Tempo *NNDescent* (s)	Tempo *MST* (s)	Total (s)
MSTR	D2N2	118,69	38,18	156,87
	D2N4	239,66	87,57	327,23
	D2N8	448,79	197,31	646,10
MSTN	D2N2	126,46	5,61	132,06
	D2N4	221,12	8,46	229,57
	D2N8	401,85	9,51	411,36

Fonte: Autor

Semelhante aos testes variando a quantidade de pontos, ao variar somente o parâmetro k observou-se um tempo computacional mais alto no cálculo da *MST*. Veja na Tabela 4 e na Figura 9b.

De maneira geral, o método ingênuo MSTN se mostrou mais rápido na etapa de indução da *MST* devido a menor quantidade de arestas envolvidas no processo, uma vez que ele não utiliza os cálculos residuais do *NNDescent*. A Figura 9 evidencia a diferença de tempo de execução entre os dois métodos utilizando uma terceira reta como referência.

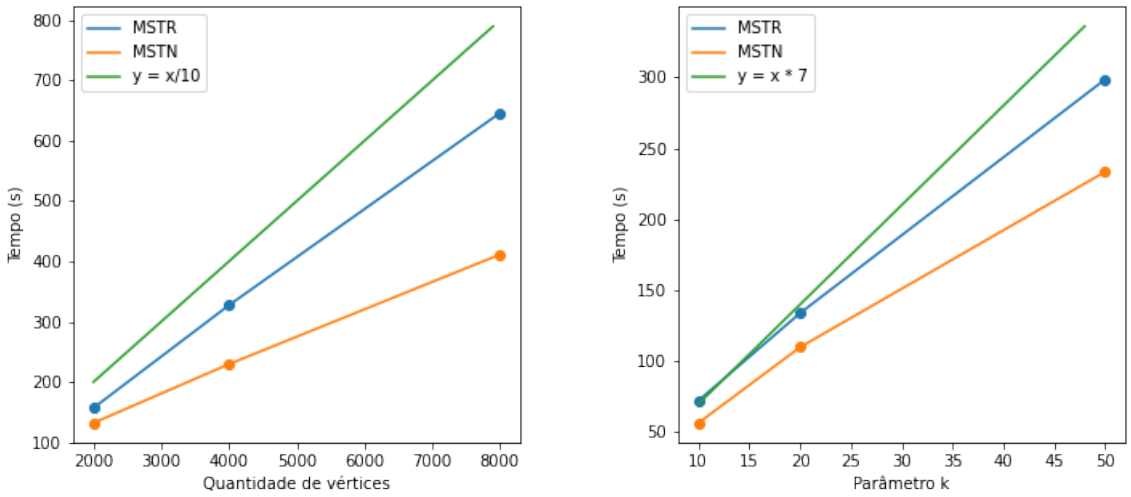
Tabela 4 – Tempo de execução variando o k na base D2N2.

Método	k	Tempo *NNDescent* (s)	Tempo *MST* (s)	Total (s)
MSTR	10	53,78	18,54	72,32
	20	101,94	31,96	133,90
	50	232,45	66,01	298,46
MSTN	10	50,93	5,66	56,60
	20	104,30	5,62	109,92
	50	226,76	6,65	233,41

Fonte: Autor

Além disso é possível visualizar que os métodos analisados possuem um tempo computacional linear com relação a quantidade de pontos e levemente sublinear com relação ao parâmetro k.

Figura 9 – Comparação de performance entre os métodos.



(a) Número de vértices x Tempo de execução (b) k x Tempo de execução

Fonte: Autor

A Tabela 5 mostra o custo relativo (Custo aproximação / Custo *MST* exata). Quanto mais próximo de 1, mais próximo do custo da *MST* exata. Ao calcular os custos relativos das árvores geradoras induzidas pelos métodos notou-se por pouco que o método MSTR obteve as melhores aproximações da *MST* exata. Em contrapartida, as árvores geradoras construídas a partir dos grafos de k vizinhos aleatórios obtiveram os maiores custos, como esperado.

Com relação às arestas obtidas nas aproximações, a Tabela 6 mostra que na maior parte dos testes, os métodos MSTR e MSTN obtiveram a mesma quantidade de arestas diferentes da *MST* exata. Isso indica que essas arestas não estavam presentes no *k-NNG*

Tabela 5 – Custo relativo (*MST* aproximada / *MST* exata) para cada método.

Base	k	Custo rel. MSTR	Custo rel. MSTN	Custo rel. Aleatório
D2N2	10	1,1307	1,1476	8,7156
	20	1,0189	1,0398	5,9857
	50	1,0048	1,0291	3,8181
D8N2	10	1,0328	1,0358	2,0495
	20	1,0036	1,0057	1,7687
	50	1,0003	1,0029	1,5354
D32N2	10	1,0177	1,0183	1,3277
	20	1,0030	1,0039	1,2457
	50	1,0001	1,0007	1,1801

Fonte: Autor

aproximado, e de acordo com a Tabela 5 o método MSTR obteve as alternativas de menor custo. A diferença de arestas diminuí consideravelmente conforme k aumenta, pois a quantidade de arestas erradas na aproximação do *k-NNG* obtidas pelo *NNDescent* é menor.

É possível observar que tanto o método MSTR quanto o método construído a partir dos k vizinhos aleatórios não precisaram amostrar arestas para conectar componentes. Isso indica que a primeira amostragem aleatória de vizinhos executada pelo algoritmo *NNDescent* tem alta probabilidade de encontrar um grafo conectado, e o método MSTR consegue carregar essas arestas através das informações residuais.

Além disso, nos casos de maior dimensão do conjunto de dados, a quantidade de arestas diferentes da *MST* exata é consideravelmente maior quando se utiliza os mesmos parâmetros. Isso acontece pela diminuição da precisão do *NNDescent* devido ao *hubness phenomenon* (BRATI et al., 2019).

Tabela 6 – Quantidade de arestas obtidas por amostragem e quantidade de arestas diferentes da MST exata.

Base	k	Arestas obtidas por amostragem			$\mid MST_{aprox.}(V) - MST_{exata}(V) \mid$		
		MSTR	MSTN	Aleatório	MSTR	MSTN	Aleatório
D2N2	10	0	2	0	74	74	1976
	20	0	2	0	7	7	1955
	50	0	2	0	3	3	1896
D8N2	10	0	2	0	281	281	1988
	20	0	2	0	33	33	1951
	50	0	2	0	2	3	1910
D32N2	10	0	2	0	569	569	1973
	20	0	2	0	115	115	1958
	50	0	2	0	3	4	1895

Fonte: Autor

5 Conclusão

Nesse trabalho foi discutida a relevância da *MST* para algoritmos de agrupamento, bem como o aumento da quantidade de dados e a necessidade de se utilizar algoritmos desenvolvidos para plataformas distribuídas. Além disso, foi abordado o problema da obtenção da *MST* de forma paralela quando a memoria não é compartilhada e as dissimilaridades não são conhecidas.

Foi demonstrado então a possibilidade da utilização de um método eficiente para calcular dissimilaridades importantes através da construção de um *k-NNG* aproximado com o algoritmo *NNDescent*. Foi visto que o *NNDescent* possui uma versão distribuída e que ela pode ser utilizada para calcular arestas que podem estar na *MST*, além de gerar muitos cálculos de arestas residuais durante seu processo de otimização.

Com isso, definiu-se a hipótese de que as arestas residuais calculadas pelo *NNDescent*, além de fazerem parte do processo de otimização de forma que o seus cálculos sejam inevitáveis, são melhores aproximações para arestas faltantes, ou seja, que estão na *MST* mas não no *k-NNG*, do que arestas obtidas por amostragem aleatória, e uma vez as arestas residuais já foram calculadas, bastaria utiliza-las.

Em seguida, foi desenvolvido o algoritmo de Boruvka de forma escalável para a plataforma *Spark* utilizando o modelo *MapReduce*, e utilizando ele foi desenvolvido dois métodos para colocar em prova a hipótese adotada. O primeiro utilizando as arestas residuais, chamado de MSTR, e o segundo método sem utilizar as arestas residuais, denominado MSTN.

Os experimentos demonstraram que o MSTR obteve as melhores aproximações, validando parcialmente a hipótese desse trabalho. Além disso, em todos os casos abordados no experimento, a *MST* aproximada pelo método MSTR possuía menos de 2% de custo superior ao custo da *MST* exata quando o k era apenas 1% da quantidade de pontos dos conjuntos de dados. No entanto, o MSTN mostrou os melhores tempos com a qualidade levemente inferior.

Nos conjuntos de dados com objetos de maior dimensão, ambos os métodos demonstraram possuir quantias significantes de arestas diferentes da *MST* exata.

Com relação a complexidade computacional, ambos os métodos demonstraram uma complexidade linear com a variação da quantidade de objetos e levemente sublinear com a variação do parâmetro k.

Conclui-se então que o método proposto é satisfatoriamente eficiente e promissor para ser aplicado em ambientes distribuídos com conjuntos de dados reais e volumosos.

No entanto, é preciso estudar seu comportamento em um verdadeiro ambiente distribuído para que outras medidas importantes sejam obtidas, como o seu *speedup*.

Como trabalho futuro, sugere-se o estudo de métodos de otimização das arestas obtidas por amostragem aleatória. Além disso, é necessário verificar o quão relevante são, com relação à estrutura da MST, as arestas que diferem das obtidas na MST exata quando a dimensão dos conjuntos de dados aumenta, devido ao *hubness phenomenon*.

Referências

ACHSAN, H. T. Y. et al. The importance of computer science in industry 4.0. In: *2018 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. [S.l.: s.n.], 2018. p. 29–37. Citado na página 9.

AREFIN, A. S. et al. knn-mst-agglomerative: A fast and scalable graph-based data clustering approach on gpu. In: *2012 7th International Conference on Computer Science Education (ICCSE)*. [S.l.: s.n.], 2012. p. 585–590. Citado na página 24.

AURENHAMMER, F. Voronoi diagramsa survey of a fundamental geometric data structure. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 23, n. 3, p. 345405, set. 1991. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/116873.116880>. Citado na página 17.

BADER, D. A.; CONG, G. Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. *Journal of Parallel and Distributed Computing*, v. 66, n. 11, p. 1366–1378, 2006. ISSN 0743-7315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731506001262>. Citado na página 19.

BATENI, M. H. et al. Affinity clustering: Hierarchical clustering at scale. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 68676877. ISBN 9781510860964. Citado 2 vezes nas páginas 9 e 24.

BOAVENTURA NETTO, P. O. *Grafos: teoria, modelos, algoritmos*. 4. ed. [S.l.]: São Paulo: Edgard Blucher, 2006. Citado na página 16.

BRATI, B. et al. The influence of hubness on nn-descent. *International Journal on Artificial Intelligence Tools*, v. 28, n. 06, p. 1960002, 2019. Disponível em: <https://doi.org/10.1142/S0218213019600029>. Citado 4 vezes nas páginas 10, 20, 21 e 35.

DONG, W.; MOSES, C.; LI, K. Efficient k-nearest neighbor graph construction for generic similarity measures. In: *Proceedings of the 20th International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2011. (WWW '11), p. 577586. ISBN 9781450306324. Disponível em: <https://doi.org/10.1145/1963405.1963487>. Citado 3 vezes nas páginas 9, 19 e 20.

FORTUNE, S. Voronoi diagrams and delaunay triangulations. In: _____. *Computing in Euclidean Geometry*. [s.n.], 1992. p. 225–265. Disponível em: <https://www.worldscientific.com/doi/abs/10.1142/9789812831699_0007>. Citado na página 17.

JOTHI, R.; MOHANTY, S. K.; OJHA, A. Fast approximate minimum spanning tree based clustering algorithm. *Neurocomputing*, v. 272, p. 542–557, 2018. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092523121731295X>. Citado na página 24.

LEE, K.-H. et al. Parallel data processing with mapreduce: A survey. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 40, n. 4, p. 1120, jan. 2012. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/2094114.2094118>. Citado na página 22.

LOW, T. et al. The hubness phenomenon: Fact or artifact? In: _____. *Towards Advanced Data Analysis by Combining Soft Computing and Statistics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 267–278. ISBN 978-3-642-30278-7. Disponível em: <https://doi.org/10.1007/978-3-642-30278-7_21>. Citado 2 vezes nas páginas 20 e 21.

OCSA, A.; BEDREGAL, C.; CUADROS-VARGAS, E. A new approach for similarity queries using neighborhood graphs. In: SILVA, A. S. da (Ed.). *XXII Simpósio Brasileiro de Banco de Dados, 15-19 de Outubro, João Pessoa, Paraíba, Brasil, Anais*. SBC, 2007. p. 131–142. Disponível em: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbbd/2007/SBBD09.pdf>. Citado na página 17.

SEDGEWICK, R.; WAYNE, K. *Algorithms, 4th Edition*. [S.l.]: Addison-Wesley, 2011. I-XII, 1-955 p. ISBN 978-0-321-57351-3. Citado na página 18.

SHAHZAD, A.; COENEN, F. Efficient distributed mst based clustering for recommender systems. In: *2020 International Conference on Data Mining Workshops (ICDMW)*. [S.l.: s.n.], 2020. p. 206–210. Citado 2 vezes nas páginas 9 e 25.

SHIMOMURA, L. C. et al. A survey on graph-based methods for similarity searches in metric spaces. *Information Systems*, v. 95, p. 101507, 2021. ISSN 0306-4379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306437920300181>. Citado 4 vezes nas páginas 9, 15, 16 e 17.

SZWARCFITER, J. L. *Grafos e algoritmos computacionais*. 2. ed. [S.l.]: Rio de Janeiro: Campus, 1988. Citado na página 16.

WARASHINA, T. et al. Efficient k-nearest neighbor graph construction using mapreduce for large-scale data sets. *IEICE Transactions on Information and Systems*, E97.D, n. 12, p. 3142–3154, 2014. Citado 3 vezes nas páginas 10, 22 e 23.

WEINSHALL, D.; JACOBS, D. W.; GDALYAHU, Y. Classification in non-metric spaces. In: *Proceedings of the 11th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1998. (NIPS'98), p. 838844. Citado na página 16.

ZAHARIA, M. et al. Apache spark: A unified engine for big data processing. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 59, n. 11, p. 5665, out. 2016. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/2934664>. Citado 2 vezes nas páginas 9 e 24.

ZEZULA, P. et al. *Similarity Search - The Metric Space Approach*. [S.l.: s.n.], 2006. v. 32. ISBN 978-0-387-29146-8. Citado na página 15.