Índice baseado em aprendizado de máquina para buscas por similaridade

Matheus Barbiero Bastos[1], **Daniel dos Santos Kaster**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – CEP 86057-970 – Londrina – PR – Brasil

matheus.bastos@uel.br, dskaster@uel.br

Abstract. *With complex data being stored on databases across the world more and more, efficient techniques to query them are demanded. These data are often represented in metric spaces and queried through similarity searches, such as the k-nearest neighbors query, which retrieves the k most similar objects in the entire dataset. To speed up this retrieval, indexes are used, in this case very different from the traditional ones. Graph-based methods have been great in lowering the number of calculations needed, consequently decreasing the search time, but they can be improved. More recently, learned indexes have emerged with some new concepts, like using machine learning to take advantage of the data distribution and other characteristics, showing promising results. This work aims to create an efficient learned index to speed up graph-based methods for similarity searching. It will be done through seed vertex predicting using supervised learning, and evidence shows that it may be able to compete with state-of-the-art techniques in terms of search performance.*

Resumo. *Dados complexos são cada vez mais armazenados em bancos de dados, demandando técnicas eficientes para fazer consultas sobre eles. Esses dados são geralmente representados em espaços métricos e as consultas são buscas por similaridade, como a dos k vizinhos mais próximos, que retorna os k objetos mais similares em todo o conjunto de dados. Para acelerar a busca, índices são empregados, nesse caso bem diferentes dos tradicionais. Métodos baseados em grafos têm sido boas alternativas para diminuir o número de cálculos necessários, assim reduzindo o tempo de busca, mas ainda podem ser melhorados. Mais recentemente,* learned indexes *surgiram com novos conceitos, como o uso de aprendizado de máquina para se aproveitar da distribuição de dados e de outras características, obtendo resultados promissores. Este trabalho tem o objetivo de criar um índice eficiente para acelerar métodos baseados em grafos para buscas por similaridade. A ideia é predizer os vértices iniciais de busca usando aprendizado supervisionado, e evidências indicam que o índice será competitivo com técnicas do estado da arte em termos de tempo de resposta.*

1. Introdução

Em bancos de dados, o assunto de índices não é novidade. Novas técnicas vêm sendo pesquisadas há muitas décadas. Por exemplo, a B-Tree [4], uma das estruturas de dados usadas em índices mais conhecidas, foi inventada em 1970. Apesar de já ter sido

explorado de diversas formas, a literatura recente mostra que ainda há muito a se aprimorar, ao passo que a taxa de armazenamento (e consumo) de dados no mundo cresce exponencialmente [19].

Estudos clássicos no assunto abordam principalmente dados estruturados (ou escalares), em que os objetos têm seus atributos tratados como dimensões independentes. Eles podem ser números ou pequenas cadeias de caracteres [34]. Seus registros podem ser ordenados, possibilitando consultas como busca exata, busca parcial e por intervalo. A criação de um índice em um conjunto de dados desse tipo geralmente envolve a ordenação de seus elementos, muitas vezes empregando algoritmos baseados em busca binária.

Por outro lado, mais recentemente dados complexos viram sua utilização crescer nos bancos de dados. Eles possuem muito menos estrutura definida e sua especificação é menos precisa. Exemplos incluem imagens, sons e documentos de texto. Diferentemente dos estruturados, eles não podem ser ordenados e comparações exatas não fazem tanto sentido [18]. Por exemplo, a busca exata por uma imagem não seria útil na maioria dos casos (a não ser que se estivesse verificando a existência exata de uma dada imagem), pois até a variação em um único pixel, mesmo que imperceptível ao olho humano, causaria diferenças na representação dos dados.

No caso desse tipo de dados, buscas por similaridade são mais adequadas. Duas consultas são consideradas as principais: a ***k-nearest neighbors query*** [39] (k vizinhos mais próximos, representada por *k-NNq*) e a ***range query*** [35] (consulta por abrangência, representada por Rq). A primeira, a partir de um objeto de consulta q, retorna os k objetos com menor distância para q. A segunda, a partir de um objeto q e uma distância limite ξ, retorna todos os objetos cuja distância para q é menor ou igual a ξ.

Assim como nos dados estruturados, a construção de índices acelera consultas em conjuntos de dados complexos. Porém, o desafio neste caso é um pouco diferente devido à maneira em que os dados são representados: são representados através **vetores de características**, que tipicamente são conjuntos de valores escalares [18]. Esses valores representam diferentes características do objeto, como cor e forma no caso de uma imagem. Assim, a similaridade entre dois objetos pode ser medida através da aplicação de uma função de distância.

Muitas vezes, esses vetores assumem um número muito grande de dimensões, ocasionando dificuldades como a "maldição da dimensionalidade" [12]. Também existem tipos de dados que naturalmente não têm dimensões. Portanto, em alguns casos é mais conveniente representar os dados em um **espaço métrico** [40]. Neles, a única operação possível é a computação da distância entre pares de objetos. Por esse motivo, são diferentes as técnicas de particionamento e filtragem utilizadas para reduzir a computação necessária em uma busca.

Para realizar a tarefa de busca por proximidade, várias estruturas de índice foram propostas na literatura, como o M-Index [30] e a Slim-tree [37]. Mais recentemente, algoritmos baseados em grafos surgiram como opções eficientes em realizar essa tarefa [26, 16]. Nota-se que muitos desses trabalhos focam na busca *aproximada* dos vizinhos mais próximos, que é, no geral, muito mais eficiente que a exata [25]. A troca de exatidão por performance é um *trade-off* aceitável em muitos casos de uso.

Este trabalho concentra-se no método ***HGraph*** [36]. Ele é um método para

construção de *grafos de proximidade*, como grafos *k-NN*. A construção é feita através de uma estratégia de divisão e conquista, que cria várias partições. Elas são conectadas através de regiões de sobreposição e *long-range edges*, arestas criadas posteriormente que aumentam a qualidade das consultas. Com o grafo construído, um algoritmo baseado em *aproximação espacial* [2] pode ser utilizado para fazer consultas sobre ele.

Um problema muito grande nos algoritmos de busca baseados em aproximação espacial é quando o vértice inicial está "muito longe" do resultado ideal. Dependendo do tamanho do grafo, essa condição causa uma demora muito grande na obtenção da resposta [36]. Para contornar esse problema, diversos trabalhos têm proposto métodos para identificar vértices iniciais (*seed nodes*) de busca. A proposta desse projeto é criar um método de seleção desses vértices iniciais baseado em aprendizado de máquina, o que teoricamente reduziria o tempo médio de execução e aumentaria a probabilidade de se encontrar a resposta correta.

O fator que corrobora com essas ideias é a forte tendência da literatura recente no uso de aprendizado de máquina em índices [1, 9, 10, 28]. O trabalho de Antol et al., por exemplo, sugere substituir pivôs utilizados em estruturas hierárquicas por modelos treinados, o que possibilita "pular" várias computações custosas de distância. Os resultados obtidos impressionam e mostram que, se bem aplicados, modelos de aprendizado de máquina são fortes contribuições para as estruturas já existentes, como o próprio grafo de proximidade construído pelo método HGraph.

Este trabalho é inspirado nessa pesquisa de Antol et al., procurando compartilhar ideias a respeito de como construir modelos de classificação para prever posições aproximadas de dados em outras estruturas já construídas. Por exemplo, será feita a utilização de aprendizado supervisionado, no qual outra estrutura é mapeada em dados de treinamento para os modelos. No caso deste trabalho, essa estrutura será um grafo construído pelo método *HGraph*.

Na seção 2, a fundamentação metodológica do trabalho é apresentada, com o conteúdo necessário para a realização dele. Na seção 3, o problema é discutido juntamente com os objetivos do trabalho e uma proposta de solução é feita. Na seção 4, é mostrado o planejamento para atingir os objetivos.

2. Fundamentação Teórico-Metodológica e Estado da Arte

2.1. Buscas por similaridade

Em domínios de dados complexos, como imagens, sons e dados georreferenciados, geralmente não se utiliza buscas exatas para recuperar conteúdo. Isso ocorre porque comparações devem ser feitas através da avaliação de diferentes aspectos dos objetos, independentemente da representação utilizada para armazená-los. Para imagens, por exemplo, seria de pouca significância a comparação pixel a pixel de dois elementos [13].

No lugar de comparações diretas, uma alternativa é a utilização de buscas por similaridade. A principal maneira que elas são realizadas possui dois aspectos fundamentais. Primeiro, deve ser formulado um método para extrair características dos dados do domínio que está sendo trabalhado. Elas geram **vetores de características**, que representam numericamente um ou mais significados nos quais há interesse [22].

Então, a fim de comparar os vetores gerados (e consequentemente os objetos que eles representam), deve ser formulada uma **função de distância**. Ela indica de forma quantitativa o quão distantes são dois elementos, ou seja, o quanto são dissimilares. Um resultado próximo de zero representa grande similaridade, enquanto que um valor maior ocorre em pares mais dissimilares [40].

Na literatura, encontra-se várias funções de distância métrica, e a melhor a ser usada depende do domínio de aplicação. As mais empregadas são as da família Minkowski, representadas na equação 1, onde n é o tamanho do vetor de características, X e Y são vetores da forma $X = \{x_1, x_2, ..., x_n\}$ e $Y = \{y_1, y_2, ..., y_n\}$, e $1 \leq p < \infty$.

$$L_p(X, Y) = \sqrt[p]{\sum_{k=1}^{n} |x_k - y_k|^p} \tag{1}$$

De todas os possíveis valores para p, alguns dos mais utilizados são: $p = \infty$, correspondente à função L_∞ ou L_0, conhecida como distância de Chebyshev; $p = 1$, correspondente à L_1, chamada de distância de Manhattan; e $p = 2$, correspondente à L_2, a distância Euclidiana. Essa última é a mais próxima do nosso senso de distância, pois representa a distância entre dois pontos em uma linha reta. A figura 1 ilustra a diferença entre essas funções no espaço, exibindo os pontos equidistantes a um elemento central u em cada uma delas.

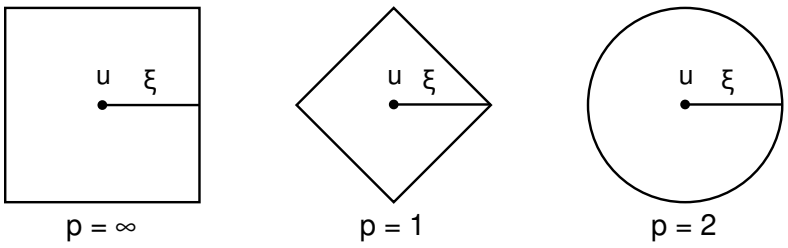


Figura 1. Distâncias L_∞, L_1 e L_2

2.1.1. Espaço métrico

A combinação do vetor de características e de uma função de (dis)similaridade é chamada de **espaço de similaridade**. Nesse caso, como a função é de distância, o espaço também pode ser considerado métrico. Um espaço métrico é um par $(\mathbb{S}, d)$, onde $\mathbb{S}$ é o domínio e d é uma função $d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}$. Também, para qualquer $x, y, z \in \mathbb{S}$, as seguintes propriedades são mantidas [40]:

- $d(x, y) \geq 0$ (não-negatividade);
- $d(x, y) = 0 \Leftrightarrow x = y$ (identidade);
- $d(x, y) = d(y, x)$ (simetria);
- $d(x, y) \leq d(x, z) + d(y, z)$ (desigualdade triangular).

Essa última propriedade, a da desiguldade triangular, é fundamental para técnicas que visam otimizar o número de computações de distância feitas em uma consulta. Por

exemplo, ela pode ser usada para determinar a impossibilidade de determinado elemento no conjunto de dados fazer parte da resposta, tornando desnecessários cálculos de distância a ele.

2.1.2. Tipos de consultas por similaridade

Em conjuntos de dados complexos, são feitas consultas por similaridade para buscar dados de interesse. Elas consistem em selecionar elementos do conjunto que satisfaçam determinado critério de similaridade. Existem dois tipos de consulta por similaridade fundamentais que são tratados na literatura: as consultas aos *k-vizinhos mais próximos* ($kNNq$) e as consultas por abrangência ou *range queries* (Rq) [6].

Seja $\mathbb{S}$ um domínio de dados, $S \subseteq \mathbb{S}$ o conjunto dos dados, $q \in \mathbb{S}$ um elemento de consulta e d uma função de distância definida sobre $\mathbb{S}$:

- a consulta $kNNq(q, k)$ encontra os k elementos de menor distância a q que estão presentes no conjunto S. Formalmente, o resultado da consulta K é definido por $K = \{s \in S \mid \forall t \in S \setminus K, |K| = k, d(q, s) \leq d(q, t)\}$.
- a consulta $Rq(q, r)$, com $r \in \mathbb{R}^+$, encontra todos os elementos no conjunto S cuja distância para q é menor ou igual a r. Formalmente, o resultado da consulta é dado por $\{s \in S \mid d(q, s) \leq r\}$.

2.1.3. Métodos de acesso aos dados

Uma maneira trivial para ser realizar uma busca por similaridade, como a $kNNq$ apresentada na seção anterior, é comparar cada objeto do conjunto de dados com o elemento de busca, verificando um a um quais satisfazem as condições desejadas. Essa estratégia resolve o problema, porém não de maneira eficiente: funções de distância geralmente são custosas [32], e executá-las para todos os elementos é inviável, principalmente em espaços com alta dimensionalidade e muitos dados. Com isso em mente, várias técnicas de busca surgiram para evitar ao máximo computações de distância. Podemos englobá-las em algumas categorias [38], sendo as principais apresentadas a seguir.

- Baseadas em *hashing*: a função *hash* tem o papel de agrupar elementos similares, tendo a probabilidade de colisão muito maior nesses casos [8]. Um método conhecido é o *locality-sensitive hashing* [20].
- Baseadas em árvores: as estruturas dos índices são árvores, utilizando diferentes técnicas de particionamento hierárquico. Exemplos incluem a *M-Tree* [7] e a *Slim tree* [37].
- Baseadas em grafos: O espaço de similaridade é modelado como um grafo. Geralmente, os elementos do conjunto de dados são representados como vértices. Alguns exemplos são o NSW [26] e o k-NNG [16].

Na literatura, os métodos mais comuns são os baseados em árvore e os baseados em grafos. Eles são capazes de particionar o conjunto de dados de maneira similar, mas há uma limitação inerente com as árvores [15]: a busca começa pela raiz, e quando um caminho errado é escolhido, não há como corrigir o erro em camadas mais inferiores. Portanto, é necessário realizar *backtracks*, forçando com que a árvore inteira ou partes

dela sejam percorridas novamente. Isso geralmente ocorre várias vezes em uma única busca, percorrendo um caminho levemente diferente em cada uma [17].

Esse problema tem relação com as estruturas hierárquicas globais presentes nas árvores [17], as quais impossibilitam a resposta de ser encontrada (sem *backtracks*) caso a subárvore incorreta esteja sendo percorrida, mesmo tendo elementos próximos. Os métodos baseados em grafos evitam essa característica, permitindo que buscas iniciadas em vértices mais próximos do elemento de consulta, no geral, exijam poucos cálculos de distância [31]. Esse fator é chave para este trabalho, e é devido a ele que o foco será dado em métodos baseados em grafos.

2.2. Grafos de proximidade

Um grafo é definido por $G(V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas que conectam pares de vértices de V. Grafos de proximidade são grafos especializados para a realização de consultas por similaridade [31]. Neles, uma propriedade P, chamada de *critério de vizinhança*, é definida, e cada par de pontos $u, v \in V$ é conectado por uma aresta $e \in E$ se, e somente se, eles satisfizerem a propriedade P. As arestas podem ou não ter pesos, e caso tenham ele geralmente é a distância entre os dois elementos [32]. Esses grafos refletem a proximidade entre seus vértices conectando objetos espacialmente próximos.

Uma das maneiras mais reconhecidas de se responder consultas de similaridade com um grafo de proximidade é através da *aproximação espacial*, definida por Navarro [29]. Começando de um elemento qualquer $a \in V$, é escolhido um de seus vizinhos $b \in N(a)$, o qual deve ser mais próximo da consulta do que a e todos os seus outros vizinhos, de maneira a chegar cada vez mais perto do resultado desejado. Isso é repetido até que não exista tal vizinho, o que significa que o nó atual é o mais próximo da consulta.

O grafo mais simples onde esse algoritmo funciona é o grafo completo, onde todos os vértices são adjacentes a todos os outros vértices. Porém, tal estrutura é extremamente ineficiente, pois cada iteração requer $|V|$ cálculos de distância. O objetivo, portanto, é obter o grafo com o menor número de arestas possível, sem deixar de responder as consultas corretamente. Tal grafo[1] $G(V, \{(a, b), a \in V, b \in N(a)\})$, representando um espaço métrico $(\mathbb{S}, d)$, onde $V \in \mathbb{S}$, deve respeitar a seguinte propriedade [29]:

$$\forall a \in V, \forall q \in \mathbb{S}, \text{ se } \forall b \in N(a), d(q, a) \leq d(q, b), \text{ então } \forall b \in \mathbb{S}, d(q, a) \leq d(q, b) \quad (2)$$

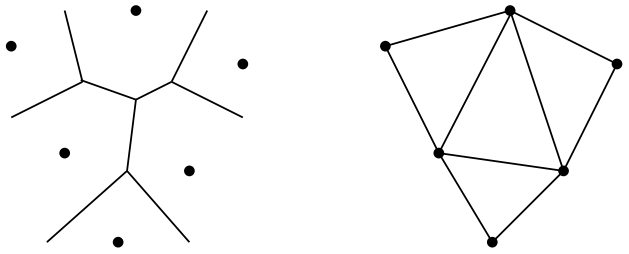
Isso significa que, para determinado elemento q, se não for possível chegar mais perto de q a partir de a indo a algum de seus vizinhos $b \in N(a)$, a é o elemento mais próximo de q em todo o conjunto V.

2.2.1. Exemplos de grafos de proximidade

Um exemplo de grafo que satisfaz a propriedade de aproximação espacial é o da triangulação de Delaunay. Nele, os vértices que são vizinhos no diagrama de Voronoi

[1] e qualquer outro onde se deseja realizar aproximação espacial

são conectados. O diagrama de Voronoi divide o espaço em subrregiões, onde cada uma é correspondente a um ponto p do conjunto de dados e, nessa subrregião, qualquer ponto tem como elemento mais próximo o ponto p [29]. Uma visualização do mesmo conjunto de pontos para o diagrama de Voronoi e o grafo de Delaunay se encontra na figura 2.



(a) Diagrama de Voronoi (b) Grafo de Delaunay

Figura 2. Diagrama de Voronoi e grafo de Delaunay com os mesmos pontos.

Esse grafo, se generalizado para espaços arbitrários, seria o ideal para o método de aproximação espacial, contendo o número mínimo de arestas enquanto satisfaz a propriedade necessária. No entanto, ele é adequado apenas para espaços euclidianos, ao passo que Navarro mostra que o grafo de Delaunay deveria ser o grafo completo em espaços métricos [29]. Ainda, em dados com alta dimensionalidade, o Delaunay e extensões propostas a ele sofrem com a *maldição da dimensionalidade*, rapidamente se tornando um grafo completo conforme o número de dimensões aumenta [36].

Como uma alternativa que engloba espaços métricos, Navarro propõe a *Spatial Approximation Tree* (SAT), que é um caso particular do grafo de Delaunay [29]. Para construí-la, primeiramente um elemento aleatório $a \in V$ é selecionado para se tornar a raiz da árvore. Então, os nós seguintes são inseridos de acordo com a propriedade da aproximação espacial. O restante dos elementos é ordenado de acordo com a distância para a, e então eles começam a ser inseridos como filhos de a, mas com uma regra: o elemento deve ter sua distância para a menor do que sua distância para qualquer outro elemento $b \in N(a)$ já inserido anteriormente. Esse processo é repetido recursivamente até que todos os elementos estejam na árvore.

Por ser uma árvore, a escolha da raiz é crítica na SAT, e uma raiz ruim pode levar à exploração de uma parte muito significativa da árvore, conforme já foi discutido anteriormente. Portanto, Ocsa et al. [31] sugerem o uso de *Relative Neighborhood Graphs* [21] (RNG). Por serem subgrafos dos grafos de Delaunay, a propriedade de aproximação espacial é garantida neles. Ao mesmo tempo, podem ser construídos com apenas as distâncias entre os elementos do conjunto de dados disponíveis, sendo adequados para espaços métricos. Apesar disso, sua construção é muito custosa, com sua complexidade de tempo sendo $O(n^3)$ [21].

Outro grafo de proximidade a ser levado em consideração é o *k-Nearest Neighbor Graph* (*k-NNG*) [3]. Nele, cada vértice é conectado com os k elementos mais próximos a ele no conjunto de dados, de acordo com determinada função de distância. A complexidade temporal para construção desse grafo, utilizando um algoritmo trivial de força bruta, é de $O(n^2)$. Isso não é aceitável para grandes bases de dados, motivando várias pesquisas

focadas em algoritmos mais eficientes para construção de *k-NNGs* [11, 33].

2.2.2. O *HGraph*

O *HGraph* é um método que visa acelerar o tempo de construção de qualquer tipo de grafo de proximidade proposto na literatura, incluindo o *k-NNG*, ao mesmo tempo que melhora a qualidade dos resultados através da adição de *arestas de longa distância* [36]. Essas arestas não seguem o critério de proximidade estabelecido no grafo, e conectam os pivôs utilizados no processo de particionamento do *HGraph*, consequentemente conectando as regiões do grafo gerado.

O método está no contexto de buscas *aproximadas* por similaridade, que são consultas que retornam resultados aproximados. Elas são motivadas pelo alto custo de se produzir respostas exatas para buscas por similaridade, tendo o objetivo de reduzir o tempo gasto com o mínimo de erro possível. Para aumentar a velocidade, os algoritmos têm a permissão de diminuir a qualidade da resposta, podendo retornar elementos que não fazem parte dos resultado exato. Isso geralmente não é um problema em espaços métricos, considerando que muitas vezes a imprecisão é inerente aos dados [1].

A fundamentação do *HGraph* se dá em duas partes: (1) uma estratégia de divisão e conquista para construir qualquer tipo de grafo, dividindo o conjunto de dados recursivamente até que uma certa cardinalidade de um subconjunto seja atingida; (2) a adição de arestas de longa distância a vértices selecionados do grafo, que são os pivôs em cada partição recursiva. Isso é feito seguindo o critério de vizinhança do grafo cada vez que um conjunto é dividido.

O resultado do processo de divisão são vários grafos menores, cujos pivôs são conectados pelas arestas de longa distância. Para que elementos próximos, mas que foram delegados para subconjuntos diferentes, não fiquem desconexos, é definida uma região de sobreposição na qual elementos são duplicados, isto é, ficarão presentes em mais de um subconjunto. Essa região fica próxima às bordas dos subconjuntos, sendo mais uma forma em que os grafos ficam conectados.

Os experimentos realizados mostram que o *HGraph* foi capaz de melhorar a qualidade da consulta, ao mesmo tempo que obteve um tempo de construção reduzido, principalmente quando usado em conjunto com o *k-NNG*. Também demonstrou melhorias, apesar de com alguns *trade-offs*, no *Navigable Small World Graph*. Mais detalhes podem ser vistos no artigo [36]. Sem dúvidas, há potencial para melhorias no tempo de consulta dos grafos construídos pelo método. Uma proposta com essa finalidade é discutida na seção 3.

2.3. *Learned Indexes*

Como muitas estruturas de dados, os índices tradicionais são de propósito geral, ou seja, não assumem características fortes a respeito dos dados sendo armazenados [24]. Sendo assim, é comum que não sejam capazes de tirar vantagem da distribuição dos dados ou de aproveitar padrões existentes neles. Ao mesmo tempo, dados do mundo real muitas vezes não seguem padrões previamente conhecidos e que podem ser generalizados. Desenvolver soluções especializadas para cada caso de uso envolveria um esforço muito grande, que no final provavelmente não valeria a pena.

Trabalhos recentes têm tratado dessa falta de proveito através da criação de ***learned indexes*** [1, 9, 10, 24]. Eles fazem o uso de aprendizado de máquina, o que pode os tornar capazes de aprender modelos que refletem padrões intrínsecos aos dados em questão. Um dos precursores, o artigo de Tim Kraska et al. considera índices tradicionais como modelos, os quais podem ser aprimorados, ou até substituídos, por outros tipos de modelos, incluindo os que utilizam redes neurais [24]. Os autores também argumentam que aparentes obstáculos, como o pensamento tradicional que modelos de *Machine Learning* são muito custosos, não são tão problemáticos como podem parecer.

2.3.1. *Learned indexes* em dados ordenáveis

Seguindo a linha do artigo supracitado, índices para buscas por intervalo, como a B-Tree, já são modelos: dada uma chave, "predizem" a posição de um valor no conjunto indexado [24]. Por questão de eficiência, é comum que não sejam todas as chaves indexadas na B-Tree, mas sim, por exemplo, apenas a primeira chave de cada página. Assim, a posição retornada pelo índice não será exata: terá um erro mínimo e máximo, sendo o mínimo 0 e o máximo o tamanho da página. Vendo através desse ângulo, é possível considerar a B-Tree como uma árvore de regressão, ou seja, um modelo de *Machine Learning*.

Ainda mais, para buscas por intervalo, onde os dados devem estar ordenados de acordo com a chave de busca (se for desejado que elas sejam eficientes), um modelo que prediz a posição de uma chave no conjunto ordenado dos dados aproxima a **função de distribuição acumulada** [27] (CDF, sigla em inglês). Com uma CDF, é intuitivo inferir a posição aproximada de uma dada chave através da equação 3:

$$p = F(Chave) * N \tag{3}$$

onde p é a posição estimada, $F(Chave)$ é o resultado da CDF que estima a probabilidade de existir uma chave menor ou igual à chave de busca $P(X \leq Chave)$ e N é o número total de chaves. Esse fato significa que, para que a indexação seja feita, basta aprender a distribuição dos dados para construir a CDF. É nesse ponto que são baseados alguns *learned indexes*, como o RMI [24], o PGM-index [14] e o RadixSpline [23].

O ***Recursive Model Index***, ou RMI [24], é uma estrutura motivada pela dificuldade de se atingir uma precisão satisfatória com apenas um modelo quando há muitos elementos. Por exemplo, reduzir o erro de predição para a ordem das centenas quando existem 100M de elementos dessa maneira não é simples. Por outro lado, a redução de 100M para 10 mil é muito mais fácil até com modelos simples, e de 10 mil para 100 é um problema similar. Então, a ideia do RMI é construir uma hierarquia de modelos com vários estágios, onde em cada estágio é escolhido um modelo com base na chave, até que no estágio final a posição é predita. No primeiro estágio, há apenas um modelo.

Uma vantagem importante é que os modelos podem ser de tipos diferentes. Por exemplo, um RMI de dois estágios pode usar um modelo linear no estágio 0, o qual seleciona um modelo cúbico do estágio 1 para efetivamente predizer a posição da chave. Esse exemplo é ilustrado na figura 3. Os modelos no RMI são treinados de maneira *top-down*, ou seja, primeiramente é treinado o modelo no estágio 0 (o primeiro), com ele são treinados os modelos do estágio 1, os quais são posteriormente usados para treinar

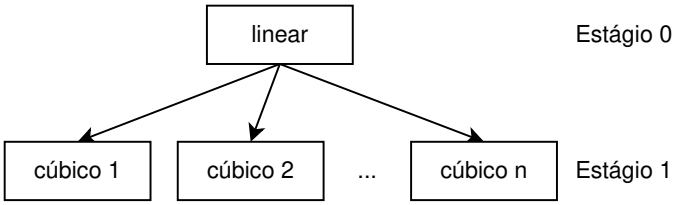


Figura 3. Um RMI de dois estágios. Fonte: [27]

.

os modelos do estágio 2, e assim sucessivamente. Formalmente, com x sendo a chave, $y \in [0, N)$ a posição correspondente e M_ℓ o número de modelos no estágio ℓ, cada modelo k no estágio ℓ, denotado por $f_\ell^{(k)}$, é treinado visando minimizar os seguintes erros:

$$L_\ell = \sum_{(x,y)} (f_\ell^{(\lfloor M_\ell f_{\ell-1}(x)/N \rfloor)}(x) - y)^2 \qquad L_0 = \sum_{(x,y)} (f_0(x) - y)^2 \tag{4}$$

O RMI se mostrou competitivo com os índices tradicionais, reduzindo pela metade, em relação a uma B-Tree otimizada, o tempo de execução de consultas em vários conjuntos de dados. Mais detalhes podem ser encontrados no artigo [24].

2.3.2. *Learned indexes* em dados complexos

As estruturas apresentadas na seção 2.3.1 são capazes de indexar dados ordenáveis, sendo habilitadas a substituir índices tradicionais como a B-Tree. Elas obtêm resultados satisfatórios, muitas vezes reduzindo o tempo de consulta e o tamanho do índice [27]. Porém, o intuito original delas não trata dados complexos, desestruturados, que se encaixam no modelo de um espaço métrico. Em contrapartida, um trabalho recente aplica o conceito de *learned indexes* a esse tipo de dados, introduzindo o ***Learned Metric Index***, ou LMI [1]. Antol et al. acreditam que sua pesquisa é original, apesar de reconhecerem alguns experimentos anteriores utilizando aprendizado de máquina para realizar buscas por similaridade em espaços métricos. No mínimo, é a primeira do tipo a ter sucesso.

O LMI pode constituir um problema de aprendizado supervisionado, onde uma estrutura já existente é utilizada, considerando como *label* de um elemento a sua posição no índice original. Também pode ser um problema de aprendizado não supervisionado, onde o índice é construído por inteiro, determinando por si só as divisões dos dados. Os autores optaram pela primeira alternativa por fins de experimentação, apesar de considerarem o LMI não supervisionado como uma das futuras direções de pesquisa.

Nos experimentos, as estruturas-base utilizadas para o treinamento supervisionado foram a M-Tree [7] e o M-Index [30]. O objetivo não é simular a estrutura do índice original através de modelos, mas usá-la de ponto de partida a fim de criar uma potencialmente melhor. Como *label* de um objeto, é considerada sua posição na estrutura original. Por exemplo, com o M-Index de base, se determinado objeto estiver contido no *cluster* $C_{2,17,1,3}$, a *label* para o modelo do primeiro nível seria 17; para modelos do segundo, 1; e do terceiro, 3.

O processo de treinamento se constitui em substituir cada nó interno da árvore

original do índice por um modelo. Se inicia no nó raiz, treinando um modelo com todo o conjunto dos dados. Seus descendentes são treinados em subconjuntos cada vez menores, conforme a profundidade na árvore. O treinamento é sequencial, ou seja, um nó é treinado somente após seu nó pai ser. Cada modelo tem um problema de classificação a ser resolvido, que é basicamente determinar as probabilidades de cada um dos filhos ter a(s) resposta(s) para alguma consulta. É importante notar que não são todos os nós preservados da estrutura original, bem como podem surgir novos nós. Mais detalhes podem ser encontrados no artigo.

Uma busca é iniciada no nó raiz e, em cada passo, um modelo é executado. A saída dele será uma distribuição de probabilidades, gerando um valor para cada filho do nó em questão. As probabilidades são inseridas em uma fila de prioridade, que é ordenada do nó mais provável ao menos provável. Então, o próximo nó escolhido será o primeiro da fila, e com ele o processo é repetido. Ao se escolher um nó-folha, que contém uma pequena partição do *dataset*, é feita uma busca linear que resultará na resposta para a consulta, ou em parte dela. Caso parte da resposta estiver faltando[2], é escolhido mais um nó na fila de prioridade, se repetindo o processo até que a resposta esteja enfim completa.

Vários tipos de modelos podem ser utilizados no LMI. Nos experimentos realizados, foram utilizados quatro, cada um com suas particularidades, desde o tempo de construção até os ajustes possíveis em hiperparâmetros. Entre eles, estão: a **Regressão Logística**, que é mais simples e consumiu menos recursos computacionais em sua construção, além de obter bons resultados em consultas; as **Redes Neurais** de 2 e 3 camadas ocultas, que foram competitivas com a regressão logística, mas tiveram sua performance reduzida em alguns casos; e a ***Random Forest***, que obteve alta precisão em consultas *1-NN* (que retorna o elemento mais próximo), mas não teve um resultado parecido em consultas mais genéricas *k-NN*, além de consumir muitos recursos computacionais.

3. Proposta de predição de vértice inicial em grafos de proximidade

Ao realizar uma busca em um grafo construído com o método HGraph, são empregados algoritmos de busca gulosa, como o *GS* e o *GNNS*. Nesses algoritmos, é selecionado um vértice aleatório do grafo para ser o inicial, ou seja, não há um critério definido, baseado em algum fator relevante, para que essa seleção seja feita.

3.1. O problema

Como a seleção do vértice inicial é aleatória, se o escolhido estiver muito "longe" do resultado, o tempo de execução da consulta será maior, pois terão que ser percorridos muito mais nós, consequentemente realizando muitas computações de distância (ou, no caso de dados contidos não apenas na memória, muitos acessos a disco). Isso, em conjuntos de dados muito volumosos, se torna um real problema, atrasando consideravelmente a obtenção do resultado. Esse não é um problema só do HGraph, mas de algoritmos de busca baseados em aproximação espacial no geral.

3.2. Objetivos

O objetivo principal do trabalho a ser desenvolvido é construir um *learned index* para grafos de proximidade eficiente.

[2]Por exemplo, se a consulta for para obter os 30-NN de um elemento de consulta q qualquer, e até o momento foram encontrados apenas 15.

Outro objetivo é propor um método de mapeamento das posições dos vértices no grafo para que elas possam ser usadas como *labels* no treinamento.

Também será necessário levantar os modelos de classificação ideais para o caso de uso, estudando os pontos fortes e fracos de alguns como ***Random Forest*** e ***Neural Network***.

3.3. Proposta de solução

Se baseando em tendências recentes da literatura, principalmente no *Learned Metric Index* [1], a proposta deste trabalho é construir um ***learned index*** sobre grafos de proximidade, capaz de determinar um vértice ideal para se iniciar uma dada busca. Assim como no LMI, o aprendizado será supervisionado, utilizando da própria estrutura do grafo como treinamento. Isso resolverá o problema mencionado, nunca escolhendo vértices distantes do resultado como iniciais, e potencialmente melhorará de maneira considerável o desempenho de busca, pois o resultado sempre estará perto do início, necessitando de poucas computações de distância para alcançá-lo.

Há razões para acreditar que o resultado seja competitivo com o estado da arte: grafos já são considerados estruturas eficientes para modelar espaços de similaridade e realizar buscas neles, e *learned indexes* têm mostrado sua força, sendo capazes, mesmo com implementações passíveis de muitas melhorias, de desbancar índices consolidados na literatura e que vêm sendo otimizados há décadas.

4. Procedimentos metodológicos/Métodos e técnicas

Inicialmente, deverá ser estudado a fundo o *Learned Metric Index* [1], visando entender as dificuldades enfrentadas e como elas foram superadas. Idealmente, os métodos seriam implementados da maneira que foram propostos a fim de fortalecer os conhecimentos necessários para realização deste trabalho. Seguindo os passos dos autores, tal implementação utilizaria bibliotecas conhecidas de *machine learning* como *scikit-learn* e *keras* para o treinamento e execução dos modelos. Outros trabalhos relacionados também deverão ser estudados, compondo a revisão bibliográfica.

Então, o estudo de como mapear os grafos de proximidade construídos pelo *HGraph* para dados de treinamento será iniciado. No percurso, será necessário um certo estudo no código-fonte da implementação do método, além de experimentos realizados com ele, que foi feito em cima da *Non-Metric Space Library* (NMSLIB) [5].

Com o mapeamento minimamente definido, os experimentos com diferentes modelos de classificação e técnicas para utilizá-los devem começar. Para tal, vários modelos existentes serão levados em consideração, desde árvores de classificação a redes neurais profundas, tentando entender os benefícios de cada um e como eles podem afetar cada resultado. A ideia inicial é fazer treinamentos supervisionados com os modelos a partir das posições mapeadas do grafo. A técnica de mapeamento pode sofrer alterações ao longo dos experimentos, sendo este um processo contínuo de melhoria.

Tendo nova técnica já desenvolvida, os resultados obtidos serão comparados com os de outros trabalhos, como o *Learned Metric Index* e o próprio *HGraph*. Para que as comparações tenham sentido, as mesmas bases de dados devem ser utilizadas, po-

dendo ser algumas das apresentadas nesses trabalhos, como: *CoPhIR*[3], que consiste de 106 milhões de imagens representadas por vetores de características de 282 dimensões; a *Profiset*[4], composta por 20 milhões de elementos obtidos pela extração de imagens através de redes neurais convolucionais; e a MNIST[5], uma coletânea de dígitos escritos a mão, representados por imagens 28x28.

5. Cronograma de Execução

As atividades mencionadas na seção 4 são enumeradas a seguir, com o cronograma de realização delas presente na tabela 1.

1. Revisão bibliográfica;
2. Estudo de técnicas para mapeamento das posições no grafo em dados de treinamento;
3. Desenvolvimento da técnica através da experimentação com diferentes modelos de classificação;
4. Comparação dos resultados obtidos com os trabalhos relacionados e o estado da arte;
5. Escrita do Trabalho de Conclusão de Curso.

Tabela 1. Cronograma de Execução

	ago	set	out	nov	dez	jan	fev	mar	abr	mai
Atividade 1	X	X	X							
Atividade 2			X	X	X	X	X	X		
Atividade 3					X	X	X	X		
Atividade 4							X	X	X	
Atividade 5					X	X	X	X	X	X

6. Contribuições e/ou Resultados esperados

Com o trabalho desenvolvido, é esperado que o conteúdo apresentado aborde o estado da arte na área de buscas por similaridade em dados complexos. Espera-se também aplicar o conceito de *learned indexes* a grafos de proximidade, apresentando técnicas pioneiras. Comparações de tempo de construção, de busca e de treinamento (quando aplicável) serão feitas entre as novas técnicas e as relacionadas, nas quais o trabalho foi baseado.

[3]http://cophir.isti.cnr.it/

[4]http://disa.fi.muni.cz/research-directions/software/profiset/

[5]http://yann.lecun.com/exdb/mnist/

7. Espaço para assinaturas

Londrina, *13 de setembro de 2021*.

Matheus Barbiero Bastos

Daniel dos Santos Kaster

Referências

[1] Matej Antol, Jaroslav Ol'ha, Terézia Slanináková, and Vlastislav Dohnal. Learned metric index—proposition of learned indexing for unstructured data. *Information Systems*, 100:101774, 2021.

[2] Kazuo Aoyama, Kazumi Saito, Takeshi Yamada, and Naonori Ueda. Fast similarity search in small-world networks. In *Complex Networks*, pages 185–196. Springer, 2009.

[3] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[4] R Bayer and E McCreight. ªorganization and maintenance of large ordered indices. In º *Proc. 1970 ACM-SIGFIDENT Workshop Data Description and Access*, pages 107–141, 1970.

[5] Leonid Boytsov and Bilegsaikhan Naidan. Engineering efficient and effective non-metric space library. In Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*, volume 8199 of *Lecture Notes in Computer Science*, pages 280–293. Springer, 2013.

[6] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM computing surveys (CSUR)*, 33(3):273–321, 2001.

[7] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Vldb*, volume 97, pages 426–435. Citeseer, 1997.

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[9] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. Alex: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 969–984, 2020.

[10] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *arXiv preprint arXiv:2006.13282*, 2020.

[11] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011.

[12] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. High-dimensional similarity search for scalable data science. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2369–2372. IEEE, 2021.

[13] Christos Faloutsos. *Searching multimedia databases by content*, volume 3. Springer Science & Business Media, 2012.

[14] Paolo Ferragina and Giorgio Vinciguerra. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment*, 13(8):1162–1175, 2020.

[15] Omar U Florez and SeungJin Lim. Hrg: A graph structure for fast similarity search in metric spaces. In *International Conference on Database and Expert Systems Applications*, pages 57–64. Springer, 2008.

[16] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[17] Ben Harwood and Tom Drummond. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5713–5722, 2016.

[18] Gisli R Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems (TODS)*, 28(4):517–580, 2003.

[19] A Holst. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2024. *Statista. Dec*, 3, 2020.

[20] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[21] Jerzy W Jaromczyk and Godfried T Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992.

[22] Daniel dos Santos Kaster. *Tratamento de condições especiais para busca por similaridade em bancos de dados complexos*. PhD thesis, Universidade de São Paulo, 2012.

[23] Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. Radixspline: a single-pass learned index. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–5, 2020.

[24] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[25] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2019.

[26] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[27] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. Benchmarking learned indexes. *arXiv preprint arXiv:2006.12804*, 2020.

[28] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 985–1000, 2020.

[29] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, 2002.

[30] David Novak, Michal Batko, and Pavel Zezula. Metric index: An efficient and scalable solution for precise and approximate similarity search. *Information Systems*, 36(4):721–733, 2011.

[31] Alexander Ocsa, Carlos Bedregal, and Ernesto Cuadros-Vargas. A new approach for similarity queries using neighborhood graphs. In *SBBD*, pages 131–142, 2007.

[32] Rodrigo Paredes and Edgar Chávez. Using the k-nearest neighbor graph for proximity searching in metric spaces. In *International Symposium on String Processing and Information Retrieval*, pages 127–138. Springer, 2005.

[33] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Practical construction of k-nearest neighbor graphs in metric spaces. In *International Workshop on Experimental and Efficient Algorithms*, pages 85–97. Springer, 2006.

[34] Lúcio Fernandes Dutra Santos. *Similaridade em big data*. PhD thesis, Universidade de São Paulo.

[35] Lúcio Fernandes Dutra Santos. *Explorando variedade em consultas por similaridade*. PhD thesis, Universidade de São Paulo, 2012.

[36] Larissa Capobianco Shimomura and Daniel S Kaster. Hgraph: a connected-partition approach to proximity graphs for similarity search. In *International Conference on Database and Expert Systems Applications*, pages 106–121. Springer, 2019.

[37] Caetano Traina, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *International Conference on Extending Database Technology*, pages 51–65. Springer, 2000.

[38] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631*, 2021.

[39] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[40] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.