Consulta Eficiente aos K-Vizinhos mais Próximos com Seleção Inteligente de Nó em Árvores Métricas

Enzo Seraphim[1], Luis Filipe L. J.Pedras[1], Thatyana F. P. Seraphim[1], Lucio F. D. Santos[2], Luiz Olmes Carvalho[1]

[1]Universidade Federal de Itajubá (UNIFEI) – Itajubá (MG), Brasil

[2]Instituto Federal do Norte de Minas Gerais (IFNMG) – Montes Claros (MG), Brasil

{seraphim, d2023001970, thatyana, olmes}@unifei.edu.br,
lucio.santos@ifnmg.edu.br

Abstract. *The K-Nearest Neighbors (KNN) query is a widely used method in data mining and machine learning. To enable its application to data volumes that exceed RAM capacity, metric access structures are used to reduce its computational cost. This proposal introduces the Neural KNN algorithm, which modifies the classic algorithm by using a neural network to infer the initial node and determine the pruning radius. Experiments show an average reduction of up to 60% in execution time compared to the classic KNN.*

Resumo. *A consulta K-Vizinhos Mais Próximos (KNN) é um método amplamente usado em mineração e aprendizado de máquina. Para viabilizar a sua aplicação em volumes de dados que excedem a memória RAM, utilizam-se estruturas de acesso métricas que reduzem seu custo computacional. Esta proposta introduz o algoritmo KNN Neural que altera o algoritmo clássico utilizando uma rede neural para inferir o nó inicial e determinar o valor do raio de corte. Os experimentos mostram uma redução média de até 60% no tempo de execução em relação ao KNN clássico.*

1. Introdução

A consulta K-Vizinhos Mais Próximos (KNN) é um dos métodos mais importantes para Mineração de Dados e Aprendizado de Máquina, notabilizando-se por sua abordagem não paramétrica, direta e eficaz para a classificação de padrões complexos [Wu et al., 2008]. Em sistemas preditivos, o KNN atua com aprendizado baseado em instâncias, onde a classificação de novos dados depende estritamente do cálculo de similaridade e distância geométrica em relação aos elementos armazenados.

Para processar consultas KNN com grande volume de dados que excedem a capacidade da memória RAM, foram concebidos métodos de acesso métricos como a M-Tree e a Slim-Tree [Ciaccia et al. 1997; Traina et al. 2002] que utilizam a desigualdade triangular para podar espaços de busca e reduzir drasticamente os cálculos de distância [Benrazek et al. 2026]. Ambas organizam seus dados com base em distâncias relativas possibilitando que a consulta KNN mantenha uma alta escalabilidade e baixos tempos de resposta ao processar dados de alta dimensionalidade [Silva et al. 2026].

Apesar desta concepção algorítmica clássica, pesquisas recentes demonstram que o KNN segue sendo aprimorado com novas técnicas de ponderação de distância e redução de dimensionalidade [Halder et al. 2024], consolidando sua robustez no processamento de grandes volumes de dados [Gou et al. 2012; Daulay et al., 2023].

A abordagem de aprimoramento do KNN explorada por este trabalho é a definição de um valor inicial para o raio de corte, que é minimizado durante o percurso das estruturas M-Tree e Slim-Tree através das podas da desigualdade triangular.

Dessa maneira, o objetivo deste trabalho é otimizar a consulta KNN clássica sobre a M-Tree e Slim-Tree através de uma rede neural que faz seleção inteligente do nó inicial para estimativa do valor do raio de corte. As principais contribuições deste trabalho são:

- Proposta de um algoritmo que usa redes neurais no processamento de consultas KNN na M-Tree e na Slim-Tree.
- Melhoria de desempenho em termos de tempo e acesso a disco em comparação a implementação clássica da consulta KNN na M-Tree e na Slim-Tree.
- Análise da inferência sobre a consulta KNN Neural.

O artigo está estruturado da seguinte forma: a Seção 2 aborda os trabalhos correlatos; a Seção 3 detalha a seleção inteligente de nó para a consulta KNN; a Seção 4 apresenta os experimentos e resultados obtidos; a Seção 5 conclui o artigo e lista os passos futuros.

2. Referencial Teórico

A relevância do KNN entre os principais algoritmos de mineração de dados [Wu et al., 2008] tem motivado estudos que mapeiam o seu aprimoramento contínuo para manter a robustez no processamento de grandes volumes de dados. O trabalho de revisão de literatura conduzido em [Daulay et al. 2023] compila as principais abordagens e adaptações propostas para otimizar o funcionamento e a aplicação do algoritmo KNN. De maneira complementar, a revisão abrangente elaborada por [Halder et al. 2024] analisa o desempenho de modificações específicas do KNN, destacando o emprego de novas técnicas de redução de dimensionalidade e de ponderação de distância.

Um trabalho pioneiro que usa regressão polinomial para prever a posição de registros de consulta KNN na M-Tree foi o LIMS (*Learned Index for Metric Spaces*) [Tian et al. 2022]. Essa regressão infere a posição (rank) de um objeto em uma página de disco, dado seu valor LIMS transformado por pivôs.

[Wang et al. 2026] apresenta o LM-Tree (*Learned M-Tree*) um índice de aprendizado híbrido que combina pivôs e modelos de aprendizado com a M-Tree para aprimorar o algoritmo de KNN. Seu modelo neural otimiza o processo de corte de ramos reduzindo substancialmente o número de cálculos.

Em [Procházka et al. 2025] foi apresentado o LMI (*Learned Metric Index*) que demonstrou ganho na escalabilidade usando uma arquitetura de modelos de rede neurais para responder consultas aproximadas sobre bases de dados de alta dimensionalidade.

Finalmente, o trabalho [Hjaltason; Samet 2003] comprova que o valor do raio de corte do algoritmo KNN atua como um limiar adaptativo de exclusão em que à medida que objetos melhores são descobertos, o raio sofre contração. Essa contração interage diretamente com a desigualdade triangular das estruturas métricas para podar espaços de busca. Consequentemente, explorar ramos promissores mais cedo faz com que o raio diminua drasticamente o custo de execução. Assim, um raio menor significa um poder de poda muito mais agressivo para o resto da execução do algoritmo, reduzindo o número total de cálculos de distância dispendiosos.

3. Algoritmo K-Vizinhos Mais Próximos Neural

Consultas KNN podem ser executadas em estruturas como a M-Tree e Slim-Tree, organizadas em páginas internas e folhas. Cada entrada na página interna contém: objeto de roteamento, raio de cobertura, distância ao representativo e referências para subárvores. As entradas das folhas contém: objetos e distância ao representativo.

Um dos aprimoramentos feitos no KNN pela literatura é a estimativa do raio de corte no processamento das estruturas métricas que evita o usar o valor infinito para o raio de corte inicial. O problema da sugestão de raio de corte é não ter *k* objetos suficientes dentro da região delimitada. Para resolver esse problema, este trabalho propõe o processamento dos objetos de uma página folha para o cálculo do raio de corte.

A página ideal a ser processada é a que contém o objeto de resposta mais próximo na lista de resultados. Para encontrar tal página, optou-se por usar uma rede neural treinada com características do objeto, que devolve o identificador página folha. Desta maneira, o Algoritmo 1 propõe a inserção das cinco primeiras linhas no KNN clássico para inicializar o raio de corte. A rede faz a inferência da folha, que é processada usando uma *heap* temporária de resposta com os primeiros *k* objetos mais próximos ao objeto de consulta. Havendo mais de *k* objetos na folha, será atualizado o raio de corte.

A implementação clássica KNN usa duas *heaps*: a *heap* de respostas (*heapResp*) que está organizada com distâncias decrescentes para o objeto de consulta; e a *heap* de páginas a visitar (*heapVisitar*) que está organizada com as distâncias crescentes para o objeto de consulta para priorizar as páginas que estão mais próximos.

Algoritmo 1: neuralKnn(objCons,k) : heapResp

```
numPag ← redeNeural(objCons) //classifica objeto em página folha
raioCorte ← ∞
heapTemp ← novo Heap //decrescente topo tem maior distância
folhaTemp ← novo Pagina(numPag)
procFolha(folhaTemp, objCons, raioCorte, ∞, heapTemp,k)
heapResp ← novo Heap //decrescente topo tem maior distância
heapVisitar ← novo Heap //crescente topo tem menor distância
heapVisitar.adiciona(∞, raiz, raioCorte)
enquanto heapVisitar.vazio()==false
  topo ← heapVisitar.removeTopo()
  se topo.distRep - topo.cobertura ≤ raioCorte
    pagina ← novo Página(topo.numSubPag)
    se pagina é folha
      procFolha(pagina,objCons,raioCorte,topo.distRep,heapResp,k)
    senão
      procIndice(pagina,objCons,raioCorte,topo.distRep,heapVisitar)
```

O processamento das páginas índices da M-Tree ou Slim-Tree é ilustrado no Algoritmo 2, que prioriza, usando a *heapVisitar,* o processamento das páginas com menores distâncias entre o objeto de consulta e o representativo e que tiveram sobreposição de suas coberturas com o raio de corte do objeto de consulta.

Algoritmo 2: procIndice(índice,objCons,raioCorte,distConsRep,heapVisitar)

```
para cada entrada(e) ∈ índice faça
  se distConsRep - e.distRep ≤ e.cobertura + raioCorte //desigualdade
    distRep ← distância(e.objeto, objConsulta)
    se distRep ≤ e.cobertura + raioCorte //interseção
      heapVisitar.adiciona(<distRep,e.numSubPag, e.cobertura>)
```

O Algoritmo 3 ilustra o processamento das folhas da M-Tree ou Slim-Tree que priorizam a remoção dos objetos mais distantes da resposta em uma *heap*, e que tiveram interseção com a cobertura do raio de corte do objeto de consulta. Os objetos excedentes ao *k* mais distantes são removidos e atualiza-se o raio de corte. As últimas linhas do Algoritmo 3 tratam os casos de empate na última posição do *heap* de resposta.

Algoritmo 3: procFolha(folha,objCons,raioCorte,distConsRep,heapResp,k)

```
para cada entrada(e) ∈ folha faça
  se distConsRep - e.distRep ≤ e.cobertura + raioCorte //desigualdade
    distRep ← distância(e.objeto, objCons)
    se distRep ≤ e.cobertura + raioCorte //interseção
      heapResp.adiciona(<distRep,e.objeto>)
      se heapResp.size > k
        enquanto heapResp.valorTopo() == raioCorte
          listaRepetido.add(heapResp.removeTopo())
        se heapResp.size < k
          heapResp.adicionaTodos(listaRepetido)
        raioCorte ← heapResp.valorTopo() //atualiza raio corte
```

4. Resultados

Esta seção avalia a proposta em relação ao tempo de execução e quantidade de acessos a disco. Os testes foram executados sobre o Windows 11, AMD Ryzen 7 5800H, 32 GB RAM, 931GB NVME, e os algoritmos implementados em Java no *framework* Ob-Inject [Carvalho et al. 2013]. Os experimentos foram divididos em 4 fases: (1) criação dos índices M-Tree e Slim-Tree; (2) criação da normalização e da rede neural; (3) execução do KNN e KNN neural; e (4) análise da inferência sobre a consulta KNN Neural.

Os experimentos empregam o conjunto de dados ALOI [Schubert; Zimek 2010], composto por 110.240 objetos com 13 dimensões, correspondentes às características de Haralick. Esse conjunto foi dividido aleatoriamente em: (i) 88.084 objetos (80%) para a inserção M-Tree e Slim-Tree, e treinamento da rede neural; e (ii) 22.156 objetos (20%) para verificar o desempenho da consulta KNN.

Na Fase 1, os 88.084 objetos (i) foram inseridos em 4 estruturas em disco, todas com 3 níveis, parametrizada com métrica Euclidiana: M-Tree e Slim-Tree com páginas de 8K e folhas com até 103 objetos, e com páginas de 16K e folhas com até 206 objetos.

Na Fase 2, cada objeto na estrutura foi submetido à consulta KNN com $k = 1$, recuperando o mesmo objeto e a informação do número da página folha de armazenamento. Esse procedimento foi repetido para cada uma das 4 estruturas para gerar um arquivo de treinamento com as 13 características do objeto (entrada da rede neural) e seu respectivo número da página folha (resposta esperada). Ainda na Fase 2, as características dos objetos foram normalizadas (*z-score*) e usadas no treinamento de 4 redes neurais (Tabela 1) usando a biblioteca DeepLearning4J [Eclipse 2026]. Todas as 4 redes neurais contém 6 camadas (1 de entrada, 4 ocultas e 1 de saída) e configuradas com: algoritmo Adam para adaptar a taxa de aprendizado individualmente para cada parâmetro; inicialização dos pesos via Xavier; critério de parada antecipada; função de ativação *Exponential Linear Unit*. Na camada de saída das 4 redes neurais usou-se a função Softmax para transformar os valores em uma distribuição de probabilidade entre a quantidade de classes (Tabela 1) e a função de perda é a *Negative Log Likelihood*. A expansão das camadas ocultas (Tabela 1) das redes ocorre porque as páginas de 8K comportam uma quantidade menor de objetos (até 103) em comparação às de 16K (até

206). Consequentemente, a fragmentação dos dados em folhas menores faz com que os modelos de 8K precisem inferir um número quase duas vezes maior de páginas.

Tabela 1 - Treinamento das 4 Redes Neurais

Nome Rede	Camada Oculta	Classe	Taxa	Ép.	Tempo (s)	Acur.	Prec.	Revoc.
MTree 8k	26, 52, 26, 52	1444	0,0003	82	1273	0,3030	0,2938	0,2173
Slim 8k	26, 52, 26, 52	1418	0,0003	68	1036	0,2532	0,2820	0,1910
MTree 16k	13, 26, 13, 26	724	0,0007	163	1198	0,4395	0,4793	0,3649
Slim 16k	13, 26, 13, 26	695	0,0007	167	1211	0,5164	0,5843	0,4347

Na Tabela 1, os modelos de 16K atingiram melhores indicadores em todas as métricas de avaliação comparadas às arquiteturas de 8K. No entanto, os modelos de 8K com acurácias na faixa de 25% a 30% não comprometem a exatidão do algoritmo que, no pior caso, fará uma navegação com um raio de corte superior. Uma análise sobre a exatidão da inferência da consulta KNN Neural é feita na Fase 4.

Na Fase 3, os 22.156 objetos (ii) foram submetidos a uma consulta KNN e KNN Neural com variações de *k*. A Figura 1 apresenta o resultado do processamento nas 4 estruturas.

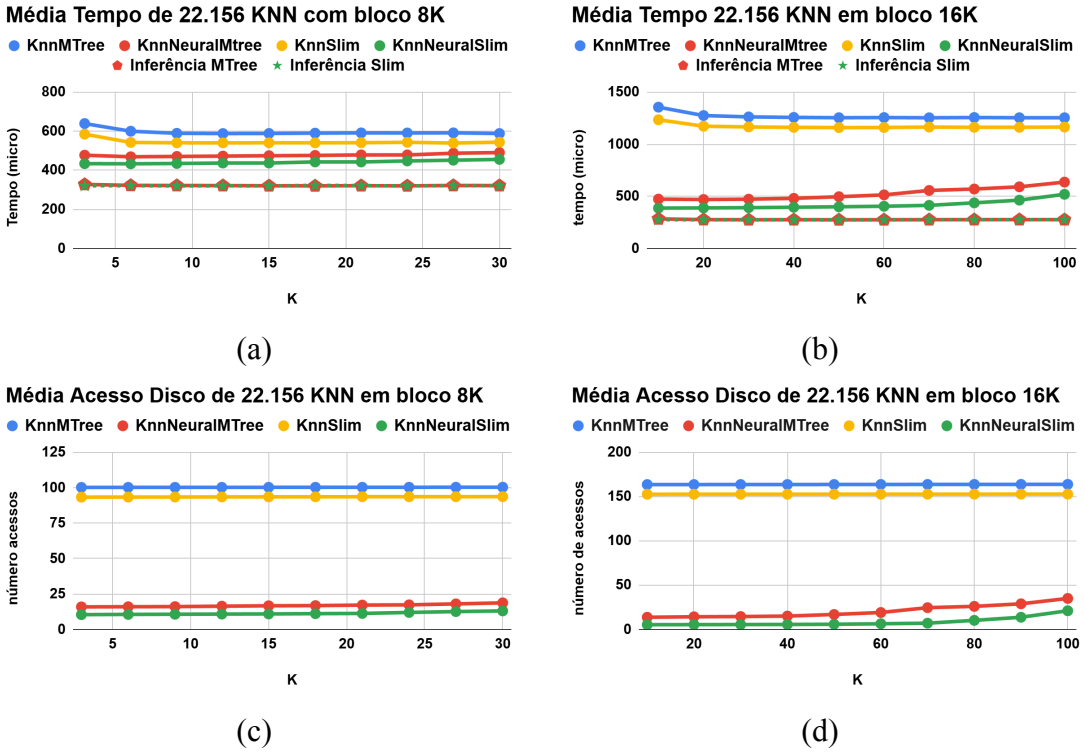

Figura 1. Gráficos da Média dos Tempos e dos Acessos a Disco do KNN.

O tempo de execução (Figuras 1(a) e 1(b)) reflete diretamente o custo dos acessos a disco (Figuras 1(c) e 1(d)). Nas páginas de 16K, as consultas KNN clássicas operam na faixa de 1100 a 1300 microssegundos, enquanto que as KNN Neurais executam na faixa de 400 a 600 microssegundos, com uma redução média de tempo de 60%. O custo médio da inferência na execução em bloco 16K foi de 47% na M-Tree e 34% na

Slim-Tree. Nos blocos de 8K as consultas KNN Neurais mantêm melhor desempenho operando na faixa de 500 microssegundos com redução média de tempo de 17%. O custo médio da inferência na execução em bloco 8K foi de 33% na M-Tree e 27% na Slim-Tree. As folhas maiores (16K) concentram mais objetos que favorecem a classificação pela rede neural.

As Figuras 1(c) e 1(d) mostram que, nos KNN clássicos, o número de acessos a disco é alto e constante, sendo em média 100 acessos na M-Tree 8K, 93 acessos na Slim-Tree 8K, 163 acessos na M-Tree 16K e 152 acessos na Slim-Tree 16K, independente do crescimento de k. Nos KNN Neurais, o número de acessos a disco é baixo e com leve crescimento conforme o valor de k aumenta, sendo em média 17 acessos na M-Tree 8K, 12 acessos na Slim-Tree 8K, 21 acessos na M-Tree 16K e 9 acessos na Slim-Tree 16K.

Na Fase 4 foi realizada a análise da distância entre a inferência da rede neural e o número de página correto do KNN Neural. Os 22.156 objetos (ii) foram submetidos a um novo KNN que respondeu uma lista das páginas mais próximas de forma crescente. A Figura 2 mostra a diferença acumulada entre a inferência e a posição da lista.

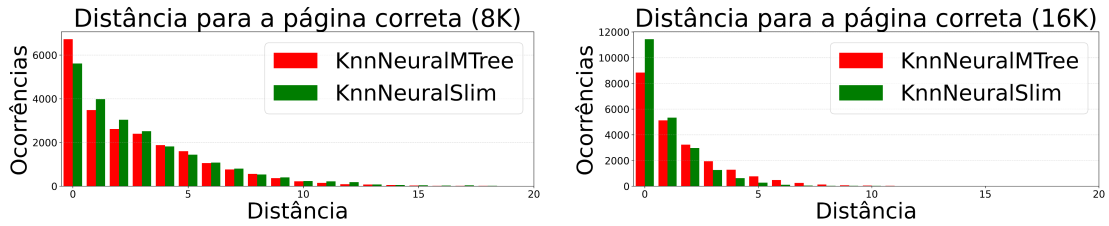

Figura 2. Histograma do posicionamento entre inferência e número da página correto.

Na Figura 2, observa-se que entre os 10 números de páginas mais próximos, estão 96,80% KNN inferidos para a M-Tree 8K, 95,72% para a Slim-Tree 8k, 99,62% para a M-Tree 16k, e 99,88% para a Slim-Tree 16k. A proximidade entre as 10 primeiras páginas mais próximas da inferência justifica a eficiência para o KNN Neural.

5. Conclusão

Este trabalho propôs uma otimização estrutural da consulta K-Vizinhos Mais Próximos em estruturas de acesso métricas M-Tree e Slim-Tree através da integração de redes neurais para a seleção inteligente do nó inicial.

O algoritmo substitui a abordagem do raio de corte infinito por uma estimativa finita, otimizando a poda do espaço de busca através da desigualdade triangular. Os experimentos com o conjunto de dados validaram a eficiência da abordagem KNN Neural em comparação com a implementação clássica em 60% na média da relação de tempo em blocos de 16K. Os principais ganhos de desempenho incluem redução no tempo de execução e no número de acessos a disco. A rede neural provou ser altamente eficaz, posicionando a inferência entre as 10 primeiras páginas mais próximas em mais de 95% dos casos.

Como trabalho futuro, pretende-se investigar índices métricos de aprendizado, como a LM-Tree e LIMS (Seção 2), em relação a uma análise da inferência sobre a consulta KNN Neural. Finalmente, deve-se comprovar a robustez e a capacidade de generalização do algoritmo KNN Neural em bases de diferentes domínios e com variações de dimensionalidade.

Referências

Benrazek, A.; Kemouguette, I.; Kouahla, Z.; Farou, B.; Seridi, H. (2026). Optimizing Overlap in Tree-Based Indexing Structures for Enhanced K-NN Search Efficiency. Journal of Eng.and Technology for Industrial Applications. 10.5935/jetia.v12i57.2744.

Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In Proc. Int. Conf. VLDB, pages 426–435, USA.

Daulay, R. S. A.; Efendi, S.; Suherman (2023). Review of literature on improving the KNN algorithm. Transactions on Engineering and Computing Sciences, [S. l.], v. 11, n. 3, p. 63–72, 2023. 10.14738/tecs.113.14768.

Eclipse Foundation (2026). Deeplearning4j: Open-source, distributed deep learning for the JVM. Disponível: deeplearning4j.konduit.ai. Acesso: mai/2026.

Gou, J.; Du, L.; Zhang, Y.; Xiong, T. (2012). A New Distance-weighted k-nearest Neighbor Classifier. Journal of Information and Computational Science. 1429-1436.

Hjaltason, G. R.; Samet, H. (2003). Index-driven similarity search in metric spaces. ACM Transactions on Database Systems, 28(4), 517-580. 10.1145/958942.958948.

Halder, R.K.; Uddin, M.N.; Uddin, M.A.; Aryal S.; Khraisat, A.(2024) Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications. *J Big Data* **11**, 113 (2024). 10.1186/s40537-024-00973-y.

Carvalho, L. O., Seraphim, T. F. P., Traina, C., and Seraphim, E. (2013). Obinject: a noodmg persistence and indexing framework for object injection. J. Inf. Data Manag., 4:220–235.

Procházka, D.; Slanináková, T.; Čerňanský, J.; Olha, J.; Antol, M.; Dohnal, V. (2025). Scaling Learned Metric Index to 100M Datasets. Similarity Search and Applications. SISAP 2024. Lecture Notes in Computer Science, vol 15268. Springer.

Schubert, E.; Zimek A. (2010). ELKI Multi-View Clustering Data Sets Based on the Amsterdam Library of Object Images. Zenodo, 2010, 10.5281/zenodo.6355684

Tian, Y.; Yan, T.; Zhao, X.; Huang, K.; Zhou, X. (2022). A Learned Index for Exact Similarity Search in Metric Spaces. IEEE Transactions on Knowledge and Data Engineering (TKDE). 10.1109/TKDE.2022.3206441

Traina Jr.; C., Traina, A. J. M.; Faloutsos, C.; Seeger, B. (2002). Fast Indexing and Visualization of Metric Data Sets using Slim-Trees. IEEE Transactions on Knowledge and Data Engineering (TKDE). 14(2). 10.1109/69.991715

Silva, W. Z.; Traina A. J. M.; Traina Jr., C. (2026), Include-Slim: Supporting Similarity Retrieval Variants with a Metric Access Method in IEEE Open Journal of the Computer Society, pp. 1-13, PrePrints 5555. 10.1109/OJCS.2026.3687006.

Wu, X.; Kumar, V.; Quinlan, J. R.; Ghosh, J.; Yang, Q.; Motoda, H.; Mclachlan, G. J.; Ng, A.; Liu, B.; Yu, P. S. (2008). Top 10 algorithms in data mining. Knowledge and Information Systems, [S. l.], v. 14, n. 1, p. 1–37, 2008. 10.1007/s10115-007-0114-2.

Wang, Y.; Wang B.; Zhu R.; Sun W.; Yang X.; (2026). LM-Tree: A Hybrid Learned Index for Similarity Search in Metric Spaces. Proc. ACM Manag. Data 4, 1, Article 51 (2026), 25 pages. 10.1145/3786665.