Optimización de Búsquedas en Bases de Datos Métricas [*]

Cristian Bustos, Verónica Ludueña, Nora Reyes
Departamento de Informática, Universidad Nacional de San Luis.
{cjbustos,vlud,nreyes}@unsl.edu.ar
y
Gonzalo Navarro
Departamento de Ciencias de la Computación, Universidad de Chile.
gnavarro@dcc.uchile.cl

Resumen

Con la evolución de las tecnologías de información y comunicación, han surgido depósitos no estructurados de información. No sólo se consultan nuevos tipos de datos tales como texto libre, imágenes, audio y vídeo; sino que además, en algunos casos, ya no se puede estructurar más la información en claves y registros. Estos escenarios requieren un modelo más general tal como *bases de datos métricas*.

La necesidad de una respuesta rápida y adecuada, y un eficiente uso del espacio disponible, hace necesaria la existencia de estructuras de datos especializadas que incluyan estos aspectos. En particular, nos vamos a dedicar a cómo resolver eficientemente no sólo las búsquedas, sino también a algunos otros temas de interés en el ámbito de las bases de datos métricas. Así, la investigación apunta a poner estos nuevos modelos de bases de datos a un nivel de madurez similar al de las bases de datos tradicionales.

Palabras Claves: bases de datos, espacios métricos, consultas, algoritmos, estructuras de datos.

1. Introducción y Motivación

Con la evolución de las tecnologías de información y comunicación, han surgido depósitos no estructurados de información. No sólo se consultan nuevos tipos de datos tales como texto libre, imágenes, audio y vídeo; sino que además ya no se puede estructurar más la información de la manera tradicional. Aún cuando esta estructuración sea posible, nuevas aplicaciones tales como la minería de datos requieren acceder a la base de datos por cualquier campo y no sólo por los marcados como "claves".

Los escenarios anteriores requieren un modelo más general tal como *bases de datos métricas* (bases de datos que incluyan objetos de un espacio métrico), entre otros; y contar con herramientas capaces de realizar búsquedas eficientes sobre estos tipos de datos. Las técnicas que emergen desde estos campos muestran un área de investigación propicia para el desarrollo de herramientas que resuelvan eficientemente los problemas involucrados en la administración de estos tipos de bases de datos no convencionales.

Un concepto unificador es el de búsqueda por "similitud" o "proximidad", que se expresa como: dado un conjunto de objetos de naturaleza desconocida, una función de distancia definida entre ellos, que en paticular es una métrica y mide cuán diferentes son, y otro objeto, llamado la *consulta*, encontrar todos los elementos del conjunto suficientemente similares a la consulta. El conjunto de objetos junto con la función de distancia se denomina espacio métrico [6, 16, 13].

En algunas aplicaciones, los espacios métricos resultan ser de un tipo particular llamado "espacio vectorial", donde los elementos consisten de D coordenadas de valores reales (ver [9] para más detalles); pero normalmente las soluciones en estos espacios no se pueden extender a los espacios métricos generales.

Por otra parte, una base de datos de texto es un sistema que debe proveer acceso eficiente a grandes volúmenes de texto no estructurado. Así, se necesitan construir índices que permitan realizar búsquedas eficientes de patrones ingresados por el usuario. Sin embargo, también la búsqueda puede consistir en encontrar todas las apariciones del patrón en el texto admitiendo un cierto número de errores, lo que se enmarca dentro del contexto de las bases de datos métricas.

La necesidad de una respuesta rápida y adecuada, y un eficiente uso de memoria, hace necesaria la existencia de estructuras de datos especializadas que incluyan estos aspectos. En particular, nos vamos a dedicar a optimizar las estructuras y los algoritmos de búsqueda en esos ámbitos. Además, como las estructuras y algoritmos para búsquedas en espacios métricos sufren de la así llamada "maldición de la dimensionalidad", nos interesa también estudiar la manera de estimar la dimensión intrínseca de un espacio métrico con el fin de elegir el índice más eficiente en cada aplicación particular.

[*]Este trabajo ha sido financiado parcialmente por el Núcleo Milenio Centro de Investigación de la Web, Proyecto P04-067-F, Mideplan, Chile (último autor).

2. Bases de Datos Métricas

El planteo general del problema en bases de datos métricas es: dado un conjunto $\mathcal{S} \subseteq \mathcal{U}$, recuperar los elementos de $\mathcal{S}$ que sean similares a uno dado, donde la similitud entre elementos es modelada mediante una función de distancia positiva d. El conjunto $\mathcal{U}$ denota el universo de objetos válidos y $\mathcal{S}$, un subconjunto finito de $\mathcal{U}$, denota la base de datos en donde buscamos. El par $(\mathcal{U}, d)$ es llamado *espacio métrico*. La función $d : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}^+$ cumple con las propiedades propias de una función de distancia.

Básicamente, existen dos tipos de búsquedas de interés en espacios métricos:

Búsqueda por rango: recuperar todos los elementos de $\mathcal{S}$ a distancia r de un elemento q dado.

Búsqueda de los *k* vecinos más cercanos: dado q, recuperar los k elementos más cercanos a q.

En muchas aplicaciones, la evaluación de la función d, suele ser una operación costosa y así en la mayoría de los casos la cantidad de evaluaciones de distancias necesarias para resolver la búsqueda, se usa como medida de complejidad.

Integrar objetos de un espacio métrico en un ambiente de bases de datos requiere principalmente poder resolver consultas por similitud eficientemente. En aplicaciones reales es posible tener grandes volúmenes de datos, ya sea por la cantidad de objetos o por el tamaño de cada objeto. Ello implica que debemos contar con estructuras adecuadas y eficientes para memoria secundaria, y en ese caso debemos optimizar también la cantidad de E/S.

Además en un escenario real de un ambiente de base de datos es necesario contar con herramientas que, además de ser capaces de trabajar con grandes volúmenes de datos, usualmente permitan insertar o eliminar objetos dinámicamente y de manera eficiente. Un objeto de un espacio métrico puede ser una imagen, una huella digital, un documento, o cualquier otro tipo de objeto. Ésta es una de las razones por las que los elementos de una base de datos métrica no se pueden almacenar en bases de datos relacionales, las cuales tienen tamaño fijo de tupla; lo que además implica que las operaciones sobre datos de un espacio métrico son, en general, más costosas que las operaciones relacionales estándar.

Existen índices que, en principio, resuelven estos tipos de problemas; pero aún están muy inmaduros para ser usados en la vida real por dos motivos importantes: *falta de dinamismo* y *necesidad de trabajar en memoria principal*. Estas características son sobreentendidas en los índices para bases de datos tradicionales, y la investigación apunta a poner los índices para estas nuevas bases de datos a un nivel de madurez similar.

3. Optimización de Estructuras

Existen numerosas estructuras para búsquedas por similitud en espacios métricos, sólo unas pocas de ellas trabajan eficientemente en espacios de alta o mediana dimensión. Además, la mayoría no admiten dinamismo, ni están diseñadas para trabajar sobre grandes volúmenes de datos; es decir, en memoria secundaria. Por lo tanto, estudiamos distintas maneras de optimizar algunas de las estructuras que han mostrado buen desempeño.

3.1. *SATD*

Hemos desarrollado una estructura para búsqueda por similitud en espacios métricos llamado *Árbol de Aproximación Espacial Dinámico (SATD)* [12] que permite realizar inserciones y eliminaciones, manteniendo un buen desempeño en las búsquedas. Muy pocos índices para espacios métricos son completamente dinámicos. Esta estructura se basa en el *Árbol de Aproximación Espacial* [11], el cual había mostrado un muy buen desempeño en espacios de mediana a alta dimensión, pero era completamente estático.

El *SAT* está definido recursivamente; la propiedad que cumple la raíz a (y a su vez cada uno de los siguientes nodos) es que los hijos están más cerca de la raíz que de cualquier otro punto de S. La construcción del árbol se hace también de manera recursiva. De la definición se observa que se necesitan de antemano todos los elementos para la construcción y que queda completamente determinado al elegirle una raíz, lo cual en el *SAT* original se realizaba al azar.

3.1.1. Elección de la raíz en el *SATD*

Para optimizar las búsquedas en el *SAT*, ya hemos estudiado diferentes maneras de elegir la raíz para que esta selección refleje alguna de las características propias del espacio métrico a indexar, tal como su dimensión intrínseca. Este análisis nos permitió observar que alguna de ellas consiguen mejorar significativamente los costos de las búsquedas, mostrando que vale la pena tomarse el trabajo de elegir mejor la raíz.

El *SATD* se construye incrementalmente, tomando como raíz el primer elemento insertado. Sin embargo, dado que ya hemos visto se obtienen mejores resultados en las búsquedas eligiendo la raíz con más información sobre el espacio métrico considerado, pensamos adaptar y estudiar el comportamiento de los distintos métodos de selección de la raíz que mostraron buen desempeño en el *SAT*, a la versión dinámica. En este caso necesitaríamos demorar la selección de la raíz hasta conocer un número razonable de objetos de la base de datos, pero sin perder el dinamismo.

Esperamos con esto que, tal como sucedió en el *SAT*, logremos brindar una estructura dinámica más eficiente en las búsquedas gracias a tener una mejor raíz para el árbol.

3.1.2. Actualización de *SATD*

Para el desarrollo del *SATD* [12] se han estudiado distintas maneras de realizar incorporaciones de nuevos elementos sobre el árbol, pero se optó por un método que

inserta un nuevo elemento en un punto determinado del árbol, manteniendo la aridad acotada. Gracias a esos trabajos, quedó demostrado que existen otros posibles puntos de inserción válidos, aunque no se analizó cuál de ellos sería el mejor.

Por lo tanto, tiene sentido considerar si los otros puntos posibles de inserción para un elemento consiguen mejorar aún más los costos de búsqueda. Para ello, cada vez que un elemento elige un punto de inserción, evaluamos cómo se comportaría la estructura durante las búsquedas si insertáramos al elemento más profundo en el árbol. De esta manera es posible que el árbol resultante sea menos "ancho" y más "profundo" que el que se hubiera obtenido con el *SATD* original. Cabe destacar que en el *SATD*, a diferencia de los árboles de búsqueda tradicionales, se ha mostrado que mantener aridad baja en los primeros niveles en algunos casos mejora las búsquedas.

Como resultado se espera brindar una estructura que mejore el comportamiento durante las búsquedas gracias a elegir adecuadamente los puntos de inserción de los nuevos elementos.

3.1.3. *SATD* con Clustering

El *SATD* es una estructura que realiza la partición del espacio considerando la proximidad espacial, pero si el árbol lograra agrupar los elementos que se encuentran muy cercanos entre sí, lograría mejorar las búsquedas al evitarse el recorrerlo para alcanzarlos. Así, nos hemos planteado el estudio de una nueva estructura de datos que realice la aproximación espacial sobre clusters o grupos de elementos, en lugar de elementos individuales.

Podemos pensar entonces que construimos un *SATD*, con la diferencia que cada nodo representa un grupo de elementos muy cercanos ("clusters") y relacionamos los clusters por su proximidad en el espacio. La idea sería que en cada nodo se mantenga el centro del cluster correspondiente, y se almacenen los k elementos más cercanos a él; cualquier elemento a mayor distancia del centro que los k elementos, pasaría a formar parte de otro nodo en el árbol.

La búsqueda de un elemento q con radio r debería proceder de manera similar al *SATD*, es decir realizando aproximación espacial entre los centros de los nodos. Sin embargo, al tener clusters en los nodos debemos además verificar si hay o no intersección entre la zona consultada y el cluster. Más aún, si no la hay se pueden descartar todos los elementos del cluster, sin necesidad de compararlos contra q. Si el cluster no se pudo descartar para la consulta, es posible usar al centro de cada nodo como un pivote para los elementos x_i que se encuentran en el cluster, porque mantenemos las distancias $d(a, x_i)$ respecto del centro a. De esta manera es posible que, evitemos algunos cálculos de distancia entre q y los x_i, si $|d(q, a) - d(a, x_i)| > r$. Cabe destacar que si la zona consultada cae completamente dentro de un cluster, podemos estar seguros que en ninguna otra parte del árbol encontraremos elementos relevantes para esa consulta [1].

3.1.4. *SATD* en Memoria Secundaria

Hemos desarrollado una versión del *SATD* que funciona adecuadamente en memoria secundaria y actualmente estamos evaluando su desempeño en las búsquedas y su dinamismo, comparándolo contra otras estructuras que poseen estas propiedades.

Así, eligiendo la estructura más apta, sería posible pensar en extender apropiadamente el álgebra relacional y diseñar soluciones eficientes para los nuevos operadores, teniendo en cuenta aspectos no sólo de memoria secundaria, sino también de concurrencia, confiabilidad, etc. Algunos ejemplos de las operaciones que podrían ser de interés resolver son: join espacial, operaciones de conjuntos y otras operaciones de interés en bases de datos espaciales tales como los operadores topológicos. Algunos de estos problemas ya poseen solución en las bases de datos espaciales, pero no en el ámbito de los espacios métricos.

En este caso no sólo se busca minimizar la cantidad de cálculos de la función de distancia, sino también la cantidad de operaciones de E/S, lo que permitiría utilizarla en aplicaciones reales, tales como búsquedas en la web.

3.2. *VP–forest* Desbalanceado

El *VP–forest* es una estructura para resolver búsquedas por similitud, presentada en [15], que forma una foresta de varios árboles binarios completamente balanceados *VP–trees* [14]. Sin embargo, en [5] se muestra que, en espacios métricos de alta o mediana dimensión, tener un índice balanceado no es sinónimo de búsquedas más eficientes, como lo era en las bases de datos tradicionales.

Por otra parte, es posible desbalancear el *VP–forest* forzando que las ramas izquierda y derecha de cada árbol binario posean una cantidad muy diferente de elementos y así cubran "zonas" de tamaño muy diferente. Actualmente hemos demostrado empíricamente en esta estructura que el desbalanceo produce mejores resultados en las búsquedas, pero aún nos resta encontrar cuál es el desbalanceo más recomendable si uno no conoce de antemano el espacio métrico a indexar.

Entonces, nos interesa optimizar esta estructura determinando no sólo el mejor desbalance, sino también si hay otras maneras de desbalancearla que resulten en búsquedas más eficientes.

4. Dimensión Intrínseca

En los espacios de vectores, la "maldición de la dimensionalidad" describe el fenómeno por el cual el desempeño de todos los algoritmos existentes se deteriora exponencialmente con la dimensión. En espacios métricos generales la complejidad se mide como el número de cálculos de distancias realizados, pero la ausencia de coordenadas no permite analizar la complejidad en términos de la dimensión.

En los espacios vectoriales existe una clara relación entre la dimensión (*intrínseca*) del espacio y la dificultad de buscar. Se habla de "intrínseca", como opuesta a "representacional". Los algoritmos más ingeniosos se compor-

tan más de acuerdo a la dimensión intrínseca que a la representacional. Existen varios intentos de medir la dimensión intrínseca en espacios de vectores, tales como la transformada de *Karhunen-Loève (KL)* y otras relacionadas, más fáciles de computar; medidas tales como *Fastmap* [8]. Otro intento útil para medir la dimensión intrínseca, de espacios de vectores no uniformemente distribuidos, es la *dimensión fractal* [2].

Existen sólo unas pocas propuestas diferentes sobre cómo estimar la dimensión intrínseca de un espacio métrico tales como el *exponente de la distancia* (basada en una ley de potencias empírica observada en muchos conjuntos de datos) [10], y la medida de dimensión intrínseca como una medida cuantitativa basada en el histograma de distancias [3]. Además, también parece posible adaptar algunos de los estimadores de distancia para espacios de vectores para aplicarlos a espacios métricos generales, como por ejemplo *Fastmap* y *dimensión fractal*.

En aplicaciones reales de búsqueda en espacios métricos, sería muy importante contar con un buen estimador de la dimensión intrínseca porque esto nos permitiría decidir el índice adecuado a utilizar en función de la dimensión del espacio. Además, al conocer una buena estimación de la dimensión nos permitiría, en algunas ocasiones, elegir la función de distancia de manera tal que se obtenga una menor dimensión.

5. Búsqueda por Similitud en Textos

El problema de búsqueda por similitud en textos es: dado un gran texto T de longitud n y un patrón P de longitud m (comparativamente más corto) y un umbral r, para el número de "errores" o "diferencias" permitidas, devolver todas las ocurrencias del patrón, es decir, todos los substrings del texto cuya *distancia de edición* (o *distancia de Levenshtein*) a P sea a lo sumo r. Tanto el texto como el patrón son secuencias de un alfabeto de símbolos de tamaño σ.

Los esquemas de indexación para este problema están todavía inmaduros y la mayoría de los esfuerzos se orientan a la búsqueda en textos de lenguaje natural y las soluciones no se pueden extender a casos generales como ADN, proteínas, símbolos orientales, etc. Teniendo en cuenta que la distancia de edición es una métrica, se puede definir un *espacio métrico* sobre el conjunto de substrings de T, en la aproximación [4] se replantea el problema de búsqueda aproximada como uno de búsqueda por rango sobre este espacio métrico.

El principal aporte de la aproximación [4] es que permite proveer un método que colapsa significativamente el $O(n^2)$ de diferentes substrings en un texto a indexar en $O(n)$ conjuntos, al encontrar una forma de construir un espacio métrico con esos conjuntos. La idea es indexar los n sufijos del texto, cada sufijo $[T_{j...}]$ representa todos los substrings que comienzan en la posición j. En esta propuesta, el espacio métrico formado por $O(n)$ conjuntos de strings se indexa por *pivotes*. Cabe destacar que no necesitamos guardar los sufijos, ya que ellos se pueden obtener e indexar directamente del texto. Así, la única estructura necesaria sería el *índice métrico*. Además se consideran políticas de selección de pivotes que aseguran buen desempeño.

Si se considera la búsqueda de un patrón dado P con a lo sumo r errores se está ante un query por rango de radio r en el espacio métrico de sufijos. Entonces, comparamos el patrón P contra los k pivotes y obtenemos una coordenada k-dimensional $(ed(P, p_1), \ldots, ed(P, p_k))$. Ahora, dado cualquier pivote p_i y los sufijos $[T_{j...}]$ de T se cumple que $ed(P, p_i) + r < \min(ed([T_{j...}], p_i))$, por la desigualdad triangular se sabe que $ed(P, [T_{j...}])) > r$ para todo substring que esté representado por $[T_{j...}]$. Los nodos que no puedan ser eliminados usando algún pivote deberán ser directamente comparados contra P y para aquellos cuya distancia mínima a P sea a lo sumo r, se reportarán todas sus ocurrencias.

El conocimiento de que los pivotes con distancias mínimas grandes permiten la eliminación de una mayor cantidad de sufijos puede ayudar a mejorar el comportamiento del índice permitiendo descartar más elementos utilizando el mismo espacio.

6. Trabajos Futuros

Como trabajo futuro de esta línea de investigación se considera:

Elección de la raíz del SATD: Por el momento estamos adaptando los distintos métodos de selección de la raíz, que mostraron mejor desempeño en el *SAT*, para la versión dinámica. Sin embargo, vamos a analizar también si existen otros métodos diferentes, con el fin de contrastarlos con los ya planteados. Además, es posible considerar si alguno de estos métodos podría hacer que el árbol obtenido posea un almacenamiento eficiente en memoria secundaria.

Actualización del SATD: Se espera poder adquirir un conocimiento más acabado sobre el *SATD* a través de la experimentación y proponer finalmente una nueva versión que mejore aún más los costos tanto de las operaciones de actualización, como también la construcción y las búsquedas.

Clustering+SATD: Esperamos poder evaluar el comportamiento de la estructura propuesta experimentando sobre distintos espacios métricos. Además pretendemos hacer comparaciones contra otras estructuras tal como la *Lista de Clusters* propuesta en [5].También queremos analizar, entre otras, cuestiones tales como: ¿esta estructura sería eficiente en memoria secundaria?, ¿es posible mantener el cluster separado del nodo para obtener una mejor opción para memoria secundaria?, ¿sería más adecuado mantener los clusters con cantidad fija de elementos o con radio fijo?, ¿existen otras maneras de combinar estas técnicas para búsquedas en espacios métricos?.

SATD en Memoria Secundaria: Estamos actualmente comparando empíricamente nuestra versión de memoria secundaria del *SATD* con las únicas dos estructuras de datos que actualmente permiten dinamismo y están diseñadas especialmente para memoria secundaria: *M-tree* [7] y *D-index* [16]. Con la estructura que resulte más competitiva en memoria secundaria se estudiará la manera de utilizarla como base para implementar otras operaciones de interés sobre bases de datos métricas tal como el join por similitud.

VP–forest desbalanceado: Ya hemos probado empíricamente que si mantenemos desbalanceados los *VP–trees* que componen la foresta podemos obtener mejores resultados no sólo en las búsquedas, sino también en algunos casos en la construcción del índice. Actualmente estamos analizando si podemos determinar cuál es el mejor desbalance, desde el punto de vista de las búsquedas, como así también otra manera de desbalancear el *VP–forest* que nos permita además explotar la localidad espacial de los datos.

Dimensión Intrínseca: Hemos adaptado algunas medidas de dimensionalidad intrínseca a espacios métricos y estamos actualmente evaluando cuál de ellas permite predecir más adecuadamente la facilidad o dificultad que se encontraría al realizar búsquedas sobre un espacio métrico. Esto nos permitiría no sólo predecir de alguna manera los costos de búsqueda sobre un espacio métrico, sino también elegir el índice más adecuado a la dimensionalidad del espacio métrico considerado.

Aplicación a Bases de Datos de Texto: Habiendo ya realizado la implementación de la aproximación presentada, se está analizando su competitividad por medio de una intensa experimentación. Además existe la posibilidad de extender esta técnica a otras funciones de distancias entre strings que son difíciles de manejar con otras aproximaciones. También se pretende tratar de reducir el número de substrings a indexar como también explorar otros métodos de selección de pivotes.

Referencias

[1] M. Barroso, G. Navarro, and N. Reyes. Combinando clustering con aproximación espacial para búsquedas en espacios métricos. In *Proc. del XI Congreso Argentino de Ciencias de la Computación (CACIC 2005)*:447–458, 2005.

[2] Francesco Camastra. Data dimensionality estimation methods: a survey. *Pattern Recognition*, 36(12):2945–2954, 2003.

[3] E. Chávez and G. Navarro. Towards measuring the searching complexity of metric spaces. In *Proc. International Mexican Conference in Computer Science (ENC'01)*, volume II:969–978, 2001.

[4] E. Chávez and G. Navarro. A metric index for approximate string matching. In *Proc. of the 5th Latin American Symposium on Theoretical Informatics (LATIN'02)*, LNCS 2286:181–195, 2002.

[5] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. of the 23rd Conference on Very Large Databases (VLDB'97)*:426–435, 1997.

[8] Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data, May 22-25, 1995*:163–174. ACM Press, 1995.

[9] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[10] Caetano Traina Jr., Agma J. M. Traina, and Christos Faloutsos. Distance exponent: A new concept for selectivity estimation in metric trees. In *ICDE*:195, 2000.

[11] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

[12] G. Navarro and N. Reyes. Fully dynamic spatial approximation trees. In *Proc. of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, LNCS 2476:254–270. Springer, 2002.

[13] H. Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., USA, 2005.

[14] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*:311–321, 1993.

[15] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.

[16] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, Vol. 32 of *Advances in Database Systems*. Springer, 2006.