

# Twisting the Metric Space to Achieve Better Metric Trees

César Feijó Nadvorny

Carlos Alberto Heuser

Instituto de Informática  
Universidade Federal do Rio Grande do Sul (UFRGS)

nadvorny@inf.ufrgs.br, heuser@inf.ufrgs.br

## Abstract

*In order to index text fields in XML databases for similarity queries, M-Trees may be applied. However, some datasets experiments using M-Trees for this purpose resulted in low query performance. The problem is that the arrangement of the objects in these datasets over the metric space seems to be inappropriate for grouping. This can jeopardize query performance because the M-Tree structure is based upon grouping of similar objects. We propose applying a twisting function over the metric space to generate a new space, the “twisted space”. In this new space, objects are better arranged enabling a more appropriate grouping. However, this twisted space cannot be used for querying, because it does not have the metric properties required by M-Trees. Therefore, we need to use both, twisted and metric space. The paper presents such modified M-Trees as well as experiments that show performance improvements.*

## 1. Introduction

XML is being adopted not only as a WEB data exchange language but also as a model for data storage. In this paper, we consider the problem of similarity querying for XML databases. Similarity querying appears in some application, like databases that collect data from the WEB. Consider as an example a database that collects bibliographic data from the WEB. Suppose someone wants to find papers written by the author “Edgard Codd”. In a large database, many aliases for this same author, like “Codd, Edgard” or “E. Codd” may appear. Actually the problem is not specific to XML databases, and could appear also in relational databases, when querying small text fields. However, in structured relational databases usually there is no data redundancy and instances are identified by primary keys reducing the problem.

With the growth of the size of XML databases the need for indexing techniques arises. Several index techniques specifically designed with the purpose of indexing XML were proposed [12, 18, 9, 19, 14, 11, 22]. However these techniques focus on querying XML for exact matches.

In this paper we discuss the problem of indexing the contents of XML text fields for similarity queries.

Similarity queries are used to find objects in the database that are within a certain degree of similarity of the object the user wants. The object to be found is called *query object*. To evaluate this degree of similarity, a similarity function  $s$  is used, that compares the query object with each database object and returns a value between zero(0) and one(1), zero(0) meaning completely different objects, and one(1) meaning the same objects.

Given a set of all possible objects  $\mathbb{O}$ ,

$$s : \mathbb{O}^2 \rightarrow [0, 1]$$

is the similarity function that measures how similar one object is to another (the value one(1) means 100% similarity).

For the purpose of similarity querying of images several access methods have been proposed [15, 24, 1, 2, 16, 23, 21, 20]. In this paper we discuss the adaptiveness of one of them, the M-Tree [10], for the purpose of similarity querying short text field as appearing in XML databases.

We provide experiments showing the performance of M-Trees for this purpose. M-Tree’s performance depends on how well the objects it is indexing are organized in the metric space. This distribution depends on two factors, the data values themselves and the distance function that is applied. In the case of small text fields and the distance function used to compare them (edit distance [17], n-grams [6]) our experiments show that the dataset is not well disposed for indexing, so the performance is not as good as expected. To solve this problem, we propose twisting the space the objects are immersed, so that they are re-arranged in a better disposition for indexing. Experiments we provide here show that this indeed improves the performance of the index.

Section 2 briefly presents some existing indexing methods. Section 3 presents experiments of the application of M-Tree for XML. Section 4 introduces a technique to improve existing indexing methods. Section 5 presents experiments showing performance improvements using this technique. Section 6 presents the conclusion and suggestions for future work.

## 2. Related Work

One method that could be used to index a dataset is the vector space [13]. Each relevant feature of an object is mapped to a number which represents how much of this feature is present in the object. Then, if there are  $n$  relevant features, there will be a vector of size  $n$  for each object. The distance of two objects is given by the distance of the vectors. However, the vector space cannot be used for text fields in XML documents because it is not possible to map text to meaningful numbers.

Nevertheless, it is still possible to compute a number that represents the distance as the similarity between two text fields. One way to do this is computing how many letters it would be necessary to transform one name into another and have this as the distance between them (edit distance). This is called a distance function. A distance function compares two objects, attributing a number to their distance, even though it does not map each of them to a vector space.

Given a set of all possible objects  $\mathbb{O}$ ,

$$d : \mathbb{O}^2 \rightarrow \mathbb{R}^+$$

is a distance function that compares objects in  $\mathbb{O}$ . Zero(0) means that the objects are the same. The greater the value the greater the difference between them.<sup>1</sup>

Distance functions have the following properties [7]:

$$\forall o_1, o_2, o_3 \in \mathbb{O},$$

---

<sup>1</sup>Note that distance functions return low values for similar objects while similarity functions returns low values for different objects. Also note that the range of similarity functions is  $[0, 1]$  while the range of the distance functions is  $[0, \infty)$ , although in our experiments we have normalized it to  $[0, 1]$ .

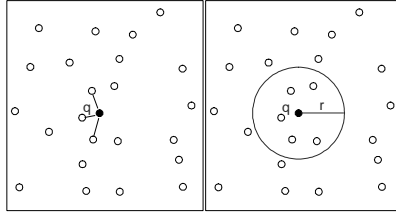


Figure 1: Examples of a  $k$ -nearest neighbor (left) and a range (right) query.

**positiveness:**  $d(o_1, o_2) \geq 0$

**symmetry:**  $d(o_1, o_2) = d(o_2, o_1)$

**reflexivity:**  $d(o_1, o_1) = 0$

**strict positiveness:**  $o_1 \neq o_2 \Rightarrow d(o_1, o_2) > 0$

If the distance function  $d$  also satisfies the triangle inequality propriety:

**triangle inequality:**  $d(o_1, o_2) + d(o_2, o_3) \geq d(o_1, o_3)$

then it is called *metric* and the pair  $(\mathbb{O}, d)$  defines a metric space. Metric spaces are useful for indexing objects that cannot be mapped to vector spaces.

Two types of queries are mainly used in the metric space [7]. They are shown in Figure 1:

**Range query**  $(q, r)$ : retrieve all objects that are within distance  $r$  to  $q$ .

**k-Nearest neighbor query**  $NN_k(q)$ : retrieve the  $k$  closest objects to  $q$  in the database.

In order to index objects embedded in a metric space, several metric access methods (MAMs) have been proposed [5, 26, 8, 3, 28, 4]. The M-Tree has introduced important features standing out from its predecessors. M-Tree uses fixed size nodes allowing the use secondary memory. It is dynamic, because it is possible to insert and remove objects from the tree without the need of a costly re-organization of the tree structure. Finally, an M-Tree is constructed bottom-up, to keep its structure balanced, so there will not be a drastic performance degradation as the tree grows (scalability).

The M-Tree organizes the database objects into fixed size nodes. Leaf nodes store the objects or pointer to the objects of the database, whereas non-leaf nodes store a *routing object*, a pointer to the sub-tree, the covering radius of the routing object, and the distance of the routing object from its parent.

**Routing objects:** objects from the database that *guides* the query on the M-Tree. Also known as *representatives*.

**Pointer to the sub-tree:** each routing object has a pointer to a sub-tree that stores other objects that are within a certain distance from the routing object.

**Covering radius:** represents the maximum distance from the routing object and any object in its sub-tree.

**Distance from parent:** the distance between the routing object and its parent (another routing object).

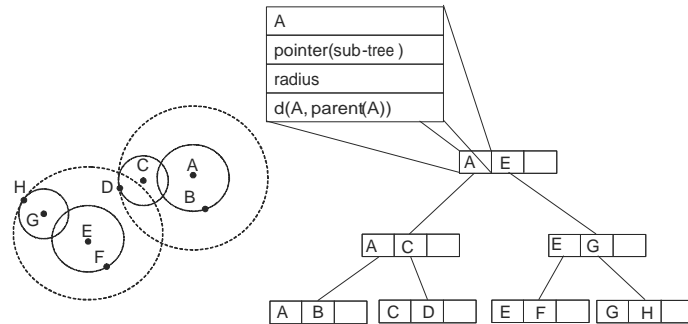


Figure 2: Example of an M-Tree structure.

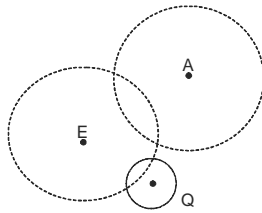


Figure 3: Pruning nodes in the M-Tree.

An example of an M-Tree structure is presented in Figure 2. The structure takes advantage of the triangle inequality propriety of the distance function to avoid node access and distance computations during the query process. Suppose a range query  $(Q, R_Q)$ , a routing object  $O$  and its radius  $R_O$ . It is possible to discard the sub-tree of the routing object  $O$  if  $d(Q, O) > R_Q + R_O$  because any object in  $O$ 's sub-tree would be inside  $O$ 's radius thus not satisfying the query parameters. In the example of Figure 3,  $Q$  is the query object,  $A$  and  $E$  are routing objects of an M-Tree. The sub-tree of object  $A$  can be discarded while  $E$ 's sub-tree have to be traversed because  $d(Q, E) \leq R_Q + R_E$ .

Slim-Tree [25] is a MAM similar to the M-Tree, but the Slim-Tree post-process its structure avoiding intersections of the covering radius achieving better query performance. Also, it uses a faster splitting algorithm improving the construction performance.

The DBM-Tree [27] is an unbalanced tree. It uses higher sub-trees to cover more dense areas in the metric space, and shorter sub-trees to cover areas where there are fewer objects. This improves performance by minimizing overlap of the covering radius, thus avoiding node access (=disk page requests).

### 3. Applying M-Trees to textual fields in XML databases

In this section we present some experiments to evaluate metric trees applied to XML databases. We have used an M-Tree as the metric access method. As distance functions we have used the edit distance [17] and n-grams [6] (actually n-grams is a similarity function so it has been adapted to a distance function).

We have used real data sources for our experiments: citations from BibTeX files. The data originated from The Collection of Computer Science Bibliographies (<http://dblp.unitrier.de/xml/>) and from BibTeX files from members of UFRGS Database Group (

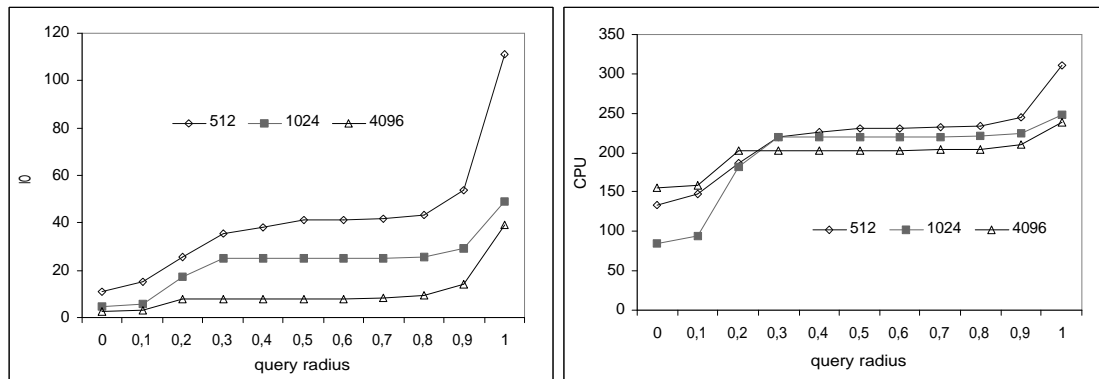


Figure 4: Block size comparison for the <LAST> field.

`inf.ufrgs.br/DBGroup/`) totalizing more than 16.000 citation entries. In the first source, the files used were the following: ACM/SIGMOD, Information Systems, Bibliography on database systems, Bibliography on Semistructured Data, VLDB journal, VLDB Conference and ACM Transactions on Information Systems. As the sources are originally in BibTeX format, we have converted them to XML using bib2XML (<http://www.cs.duke.edu/sprenkle/bibtex2html/>). The XML files can be found at <http://metropole.inf.ufrgs.br/~dorneles/xml> bibsources.

In the experiments we have used a subset of 200, 500 and 1000 objects of this data repository. Each XML document represents a publication, and contains information about the author, title, year, etc.

To measure the distance, Levenshtein (edit distance) and n-grams functions have been used. On the presented experiments, the dataset was indexed by the title of the paper, represented by the <TITLE> field and the last name of the author, represented by the <LAST> field (each of them with its own index).

To evaluate the performance of the indexes, 14 range queries have been run. For each query the radius (distance) varies from 0.0 to 1.0 in a 0.1 scale (0.0, 0.1, 0.2...1.0). Therefore, 154 queries have been run for each evaluated index. The average costs of those 14 queries for each radius are plotted in the charts we provide here. We have evaluated the IO cost (i.e. the number of page requests) and the CPU cost (number of distance calculations).

Figure 4 shows the different costs when querying trees of 512 Bytes, 1024 Bytes and 4096 Bytes of disk page size. Those three trees are indexing the last name text field.

Increasing the page size, the IO cost is reduced, because with an increased page size it is possible to store more objects per page. Hence, less pages are required to store all the objects in the dataset leading to a lower IO cost for the query.

CPU costs tend to follow the IO costs. Nevertheless, that not necessarily applies to every query. CPU cost is given by the number of distance calculations that have to be computed for each object inside the node. Therefore, it is not the amount of nodes (disk pages) but the total number of objects in those nodes that have to be compared to the query object that determines the CPU cost. A greater number of nodes would probably indicate a greater number of objects, but not necessarily, because the node “density” has also to be considered. For instance, the CPU cost for querying a tree with twenty nodes that stores two objects per node is lower than the CPU cost for querying a tree with three nodes that stores fifty objects each. For that reason, the CPU curve tends to follow the IO curve, but not always do.

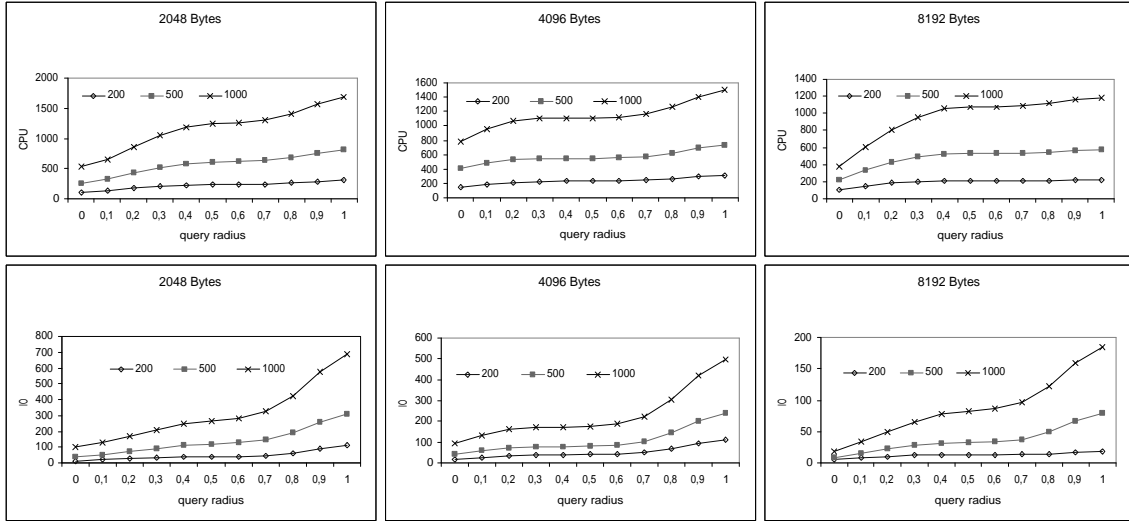


Figure 5: Number of objects comparison for the <TITLE> field.

Figure 5 compares trees indexing the <TITLE> field, focusing on the performance difference of trees indexing different amount of objects. We can notice that IO and CPU costs have a linear increase regarding the amount of objects indexed. This means that it is possible to increase the dataset being indexed without having a huge performance decrease, as expected for M-Trees.

Observing the trees generated over the XML databases, we could notice that the objects were not well grouped. Some nodes had a very low occupation ratio. In addition, some representatives presented a radius=1, which means that this representative's node will always be accessed. The value one(1) is the greatest possible distance between two objects using the distance calculation we did. When a representative has a radius=1, this means that this representative could be covering any object, therefore, any query would have to access its node. Therefore, query performance is decreased due to excessive access to the nodes.

#### 4. Twisting the Metric Space

Consider that the objects in the space have a different distribution and that the tree in this space is better grouped, i.e. does not present the problem mentioned above. Such a tree would lead to lower query costs.

As we have mentioned earlier, the objects distribution in the space depends on the objects themselves and on the distance function used in the metric space. It is not possible to change the objects of the space, but it is possible to change the distance function, so that it leads to a space more suited for grouping the objects.

Therefore, applying a twisting function  $t$  over the distance function  $d$  twists the metric space  $(\mathbb{D}, d)$  and may lead to a more suitable space. We will refer to this new space as *twisted space*  $(\mathbb{D}, t(d))$  and the tree that accesses this space as the *TM-Tree (Twisted and Metric Tree)*. In this new space, the objects are re-arranged so that they can be more adequately grouped.

An example of a twisting function could be the division of each distance by a constant value. However, this is not a good twisting function, because it would cause an uniform approximation of every object, and the undesired distribution of the objects would be the same.

An effective twisting function is, for example,  $y = x^2$  (other twisting functions are presented

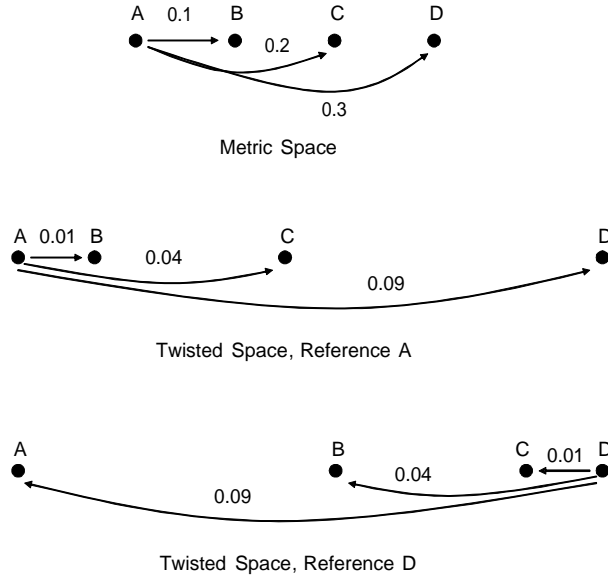


Figure 6: Relativeness of the distances in the twisted space.

in Section 5). This function brings every object closer to each other (the  $x$  value is between zero(0) and one(1)). However, objects get closer to each other in a different ratio, depending on their distance on the metric space (see Figure 6). This is important because that is what causes the space to change significantly.

We also would like to point out that the distances in the twisted space are relative to each reference object. In the example of Figure 6, if object  $A$  is the reference, then the distance between object  $C$  and  $D$  would be  $0.09 - 0.04 = 0.05$ . In spite of that, if object  $D$  is the reference, the distance between object  $C$  and  $D$  would be 0.01, since the distance between them in the metric space is 0.1 ( $(0.1)^2 = 0.01$ ). Therefore, each object in this twisted space “sees” every other object getting closer toward itself.

#### 4.1. Triangle Inequality Problem

The change on the distance between the objects depending on the reference causes a serious problem. In the M-Tree, there is a radius assigned to each representative. Any object that is in the representative’s sub-tree, should be covered by this radius, meaning that the distance between the representative and the objects in its sub-tree should not be greater than the representative radius.

However, this applies to the twisted space only if the representative is the reference. Despite that, when the M-Tree is being queried, the reference is the query object and not the representative. Therefore, it would be possible that an object is outside the representative radius boundaries if the reference is the query object, but inside if the reference is the representative. Figure 7 points this situation out. Object  $P$  is the representative,  $O$  is an object inside  $P$ ’s radius, and  $Q$  is the query object. Suppose that  $O$  is in  $P$ ’s sub-tree. Figure 7(a) shows the metric space. Object  $O$  is covered by  $Q$ ’s radius thus being retrieved through the representative  $P$ . If this space is twisted applying the  $y = x^2$  function, the resulting space is shown in Figure 7(b) considering  $P$  as the reference. Object  $O$  is not covered by  $Q$ ’s radius anymore,

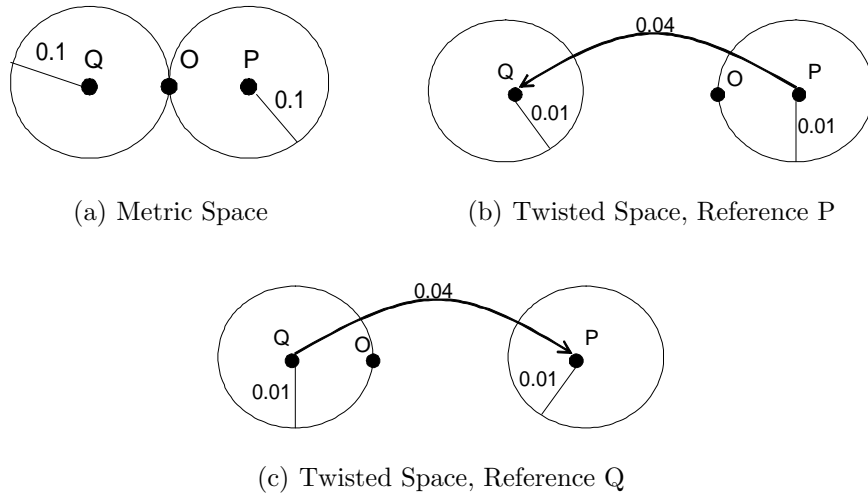


Figure 7: Triangle Inequality Problem

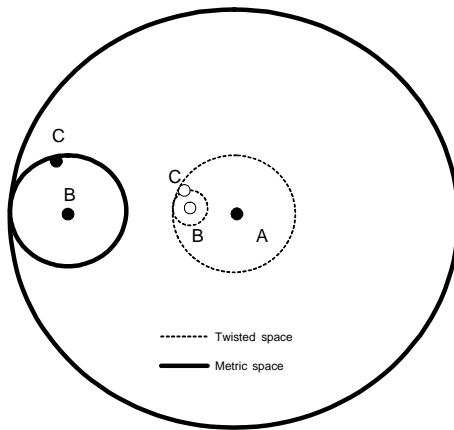


Figure 8: A TM-Tree

thus, in this space,  $O$  should not be retrieved by  $Q$ . Despite that, in Figure 7(c), where  $Q$  is the reference, object  $O$  should be retrieved by  $Q$  and should not be in object  $P$ 's sub-tree. This means that if object  $Q$  were being queried or any object at 0.01 from  $Q$ ,  $R$ 's sub-tree would not have been accessed, thus object  $O$  would not have been retrieved even though it respects the query's criteria because when the tree is being created, the references for the twisted space are the representatives.

This problem occurs because when the function  $y = x^2$  is applied to the distances, the triangle inequality does not apply anymore. For instance,  $0.1 + 0.2 \geq 0.3$ . Although,  $(0.1)^2 + (0.2)^2 (= 0.05)$  is less than  $(0.3)^2 (= 0.09)$ . This means that the twisted space is not a metric space.

To solve this problem, the radius from the twisted space is used just to *create* the tree. When *querying* the tree, the radius from the metric space is used instead, which is also calculated when constructing the tree. Therefore, the objects have a better grouping because the space is twisted disposing the objects in a more suitable way for grouping them. In the other hand, the triangle inequality problem does not occur because we are using the distances from the metric space when querying. Therefore, both spaces, metric and twisted are used as shown in Figure 8.

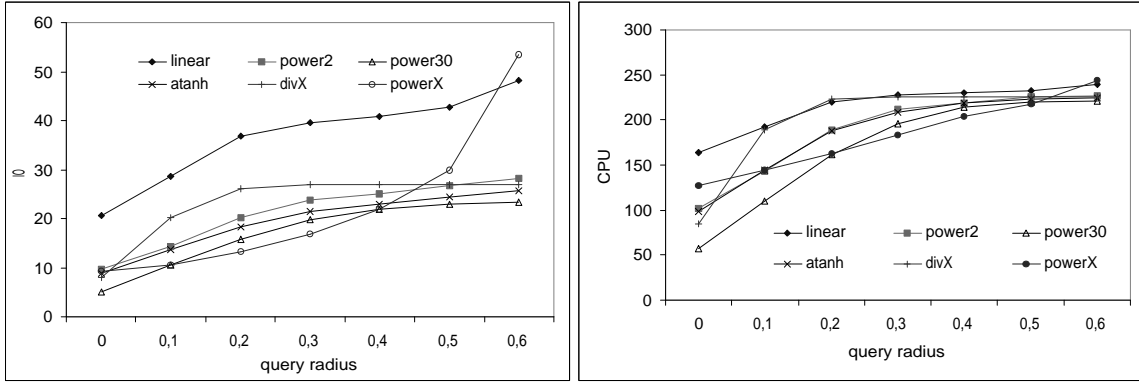


Figure 10: Performance comparison of the standard metric tree and the TM-Trees, indexing the <TITLE> field, using n-grams distance.

## 5. Twisted Space Experiments

To evaluate TM-Trees, we made some experiments, showing the improvements TM-Trees achieve when the objects distribution in the metric space is not suited for grouping. Several trees were constructed using several twisting functions. A 200-object XML dataset was indexed by the <TITLE> field, using 4096 Bytes of disk page, and range queries were performed on them. The distance function used is n-grams. In this paper, we will compare only these twisting functions:

Hyperbolic arctangent:  $atanh(x) = \frac{1}{2} \cdot \ln\left(\frac{1+x}{1-x}\right)$

power2:  $x^2$

power30:  $x^{30}$

divX:  $\frac{-1}{x-1} - 1$

powerX:  $4^x - 1$

Figure 9 is a plotting of those functions. All those functions causes a varying deformation of the linear curve depending on the radius size. It is important to twist the space non-uniformly, so that the objects have their distribution considerably modified.

Figure 10 shows the performance comparison between the regular metric tree (linear function) and the trees using those function to twist the metric space concerning IO and CPU costs. The <TITLE> field was indexed using n-grams distance function. It is restricted to a 0.6 radius to show more clearly the comparison, because of scale issues. Function *powerX* drastically loses performance for greater query radius. However that is not a problem since usually a query would not have a radius greater than 0.4 (60% similarity).

Figure 11 points the same comparison for the edit distance function. The performance improvement is not as much intense because objects distribution in this metric space was not so bad. As the graph shows, the metric space using edit distance achieves a much better performance than the metric space using n-grams. Thus, the improvement of twisting the space is not so high.

Constructing the metric tree over a twisted space resulted in better performance than the regular metric tree construction.

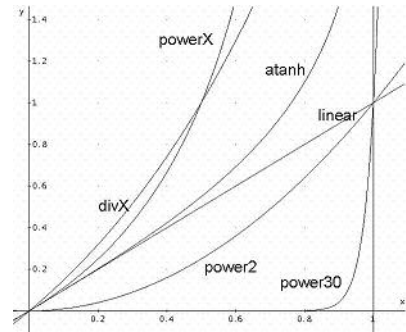


Figure 9: Plotting of the twisting functions used.

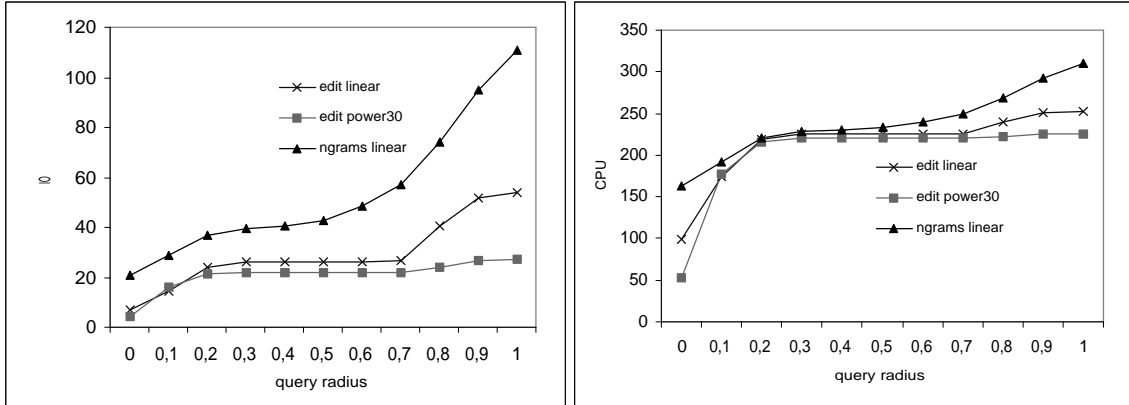


Figure 11: Performance comparison of the standard metric tree and the TM-Trees, indexing the  $\langle \text{TITLE} \rangle$  field, using edit distance.

Figure 12 provides a comparison of the indexes' volume (amount of nodes). All function except *powerX* generated smaller trees than the linear function (regular metric tree). When querying with a radius=1, it is necessary to access all tree nodes. This is the reason for the function *powerX* having a poor performance for greater radius. However, for smaller radius (that is the most usual case), this function outperformed almost all the others.

## 6. Conclusions and Future Work

M-Tree's performance is highly dependent on the metric space it is indexing. If the metric space arranges its objects so that they are appropriately distributed for grouping, the performance of the M-Tree is very good. However, if the objects are not well distributed, query performance can be poor.

We propose to twist the metric space introducing the TM-Tree, when the space is not adequate for the M-Tree, to re-arrange the objects in a twisted space where the objects can be more easily grouped, leading to a better query performance.

As we have pointed out, although this twisted space is better for indexing the objects, it does not work for querying, because twisted spaces may not respect the triangle inequality. Therefore, it is necessary to use the metric space to query, and the twisted space to index the objects.

We are interested in applying this technique to the Slim-Tree [25] and verify if its performance is also improved by the twisted space. Slim-Trees are metric trees similar to the M-Tree. Nevertheless, they enhance the tree structure avoiding intersection of the covered space of the representatives and has a more efficient algorithm for choosing the representatives. Probably, the twisted space would have a lower enhancing effect in this tree.

The DBM-Tree [27] approaches the same problem of unsuited metric space creating an unbalanced tree that has higher sub-trees to cover denser regions of the space. We are interested in comparing these two techniques and verify if they can be improved if joined.

We would like to investigate if it is possible to find an ideal twisting function given a metric space, automatically generating twisting functions based on the objects distribution.

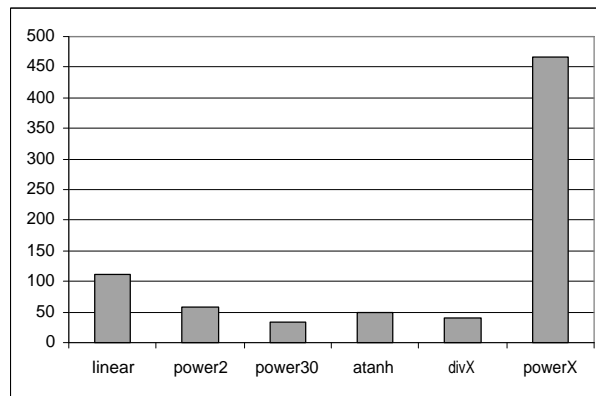


Figure 12: Index volume comparison.

## References

- [1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R\*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, 23–25 May 1990.
- [2] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An index structure for high-dimensional data. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, pages 28–39, Mumbai (Bombay), India, 3–6 September 1996. Morgan Kaufmann.
- [3] Tolga Bozkaya and Meral Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):357–368, 1997.
- [4] Sergey Brin. Near neighbor search in large metric spaces. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, pages 574–584, Zurich, Switzerland, 11–15 September 1995. Morgan Kaufmann.
- [5] Walter A. Burkhard and Robert M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, April 1973.
- [6] W. B. Cavnar. N-gram-based text filtering for TREC-2. In National Bureau of Standards, editor, *Proceedings of TREC-2: Text Retrieval Conference 2*, August 1993.
- [7] Edgar Chavez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and Jose L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
- [8] Tzi-cker Chiueh. Content-Based Image Indexing. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 582–593, Santiago, Chile, 1994.
- [9] Chin-Wan Chung, Jun-Ki Min, and Kyuseok Shim. APEX: an adaptive path index for XML data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 121–132. ACM Press, 2002.
- [10] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

- [11] Brian Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 341–350. Morgan Kaufmann Publishers Inc., 2001.
- [12] Alberto Mendelzon Flavio Rizzolo. Indexing XML data with toxin. In *International Workshop on the Web and Databases (WebDB'2001)*, Santa Barbara, California, May 2001.
- [13] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.
- [14] Roy Goldman, Jason McHugh, and Jennifer Widom. Lore: A database management system for XML. *Dr. Dobb's Journal of Software Tools*, 25(4):76, 78–80, April 2000.
- [15] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 14(2):47–57, 1984.
- [16] H. V. Jagadish. Spatial search with polyhedra. In *Proc. IEEE Int'l. Conf. on Data Eng.*, pages 311–319, Los Angeles, CA, February 1990.
- [17] V. I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Russian Problemy Peredachi Informatsii*, 1:12–25, January 1965.
- [18] Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *Proceedings of the Twenty-seventh International Conference on Very Large Data Bases: Roma, Italy, 11–14th September, 2001*, pages 361–370, Los Altos, CA 94022, USA, 2001. Morgan Kaufmann Publishers.
- [19] Tova Milo and Dan Suciu. Index structures for path expressions. *Lecture Notes in Computer Science*, 1540:277–295, 1999.
- [20] Y. Ohsawa and M. Sakauchi. A new tree type data structure with homogeneous nodes suitable for a very large spatial database. In *Proc. IEEE CS Intl. Conf. No. 6 on Data Engineering*, February 1990.
- [21] Beng Chin Ooi. Spatial kd-tree: An indexing mechanism for spatial databases. In *Proceedings of the IEEE Computer Software and Applications Conference*, pages 433–438, 1987.
- [22] Leela Krishna Poola and Jayant R. Haritsa. SphinX: Schema-conscious XML indexing.
- [23] M. Schiwietz. *Speicherung und anfragebearbeitung komplexer geo-objekte*. PhD thesis, Ludwig-Maximilians-Universität München, Germany, 1993.
- [24] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *The VLDB Journal*, pages 507–518, 1987.
- [25] Caetano Traina Jr., Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-Trees: High performance metric trees minimizing overlap between nodes. In *EDBT: International Conference on Extending Database Technology, EDBT*, pages 51–65, Konstanz, Germany, 2000. LNCS, Springer-Verlag.
- [26] Uhlmann. Satisfying general proximity / similarity queries with metric trees. *IPL: Information Processing Letters*, 40, 1991.
- [27] Marcos Rodrigues Vieira. DBM-Tree: Método de acesso métrico sensível à densidade local. Master's thesis, Universidade de São Paulo, 2004.
- [28] Peter N. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *The Sixth DIMACS Implementation Challenge Workshop: Near Neighbor Searches*, Baltimore, Maryland, January 1999.