Distributed Spatial Approximation Tree - SAT*

Jose Penarrieta[1], Patricio Morriberon[1], and Ernesto Cuadros-Vargas[2]

[1] San Pablo Catholic University, Arequipa, Peru,
{jpenarrieta,pmorriberon}@inf.ucsp.edu.pe
[2] Peruvian Computer Society
ecuadros@spc.org.pe

Abstract. The problem of classifying elements by similarity has many applications. In this paper we propose a new Metric Access Method (MAM) called "Distributed Spatial Approximation Tree (SAT*)" based on the "Spatial Approximation Tree (SAT)" [12] which is based on approaching "spatially" the searched objects. However, this MAM cannot assure an optimal distribution because it chooses its root randomly. For example, it can choose an extreme of the dataset as the root, so the remaining objects would be at the same side of the root, heading to very inefficient queries. We present as a possible solution to this problem, an algorithm called "Centroid Selection Algorithm (CSA)" which is based on the idea of choosing the center of the dataset as the root. The advantage of SAT* is that it assures a much better distribution of the data structure, heading to more efficient queries. Experiments show that distance calculations are reduced up to 48% compared with SAT.

1 Introduction

Searching objects by similarity is a problem when we consider a very large dataset and high dimensional spaces. There are many examples of this problem such as multimedia databases, text retrieval where the most similar word to a query is returned allowing an error or searching documents by similarity. The similarity evaluation could be too expensive, compared with CPU time and disk accesses, so the challenge is to minimize similarity evaluations.

To perform queries we need a set of objects and a distance function. So, we can denote $\mathbb{X}$ as a set of valid objects, $\mathbb{U}$ (our dataset) as a finite subset of $\mathbb{X}$. The function

$$d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$$

denotes the distances between two objects. When the function $d()$ satisfies the following properties:

- Positiveness : $\forall x, y \in \mathbb{X}, d(x, y) \geq 0$
- Symmetry : $\forall x, y \in \mathbb{X}, d(x, y) = d(y, x)$

– Reflexivity : $\forall x \in \mathbb{X}, d(x,x) = 0$

– Triangular inequality : $\forall x, y, z \in \mathbb{X}, d(x,z) \leq d(x,y) + d(y,z)$

it is called a Metric Distance and the pair $(\mathbb{X}, d)$ is called a Metric Space. In Metric Spaces it is possible to perform similarity queries, such as:

1. **Range Query** $RQ(q,r)$ Retrieve all elements which are within distances r to q $\{x \in \mathbb{U} / d(q,x) \leq r\}$
2. **Nearest Neighbor Query** $NN(q)$ Retrieve the closest element to q $\{x \in \mathbb{U} / \forall y \in \mathbb{U}, d(q,x) \leq d(x,y)\}$
3. **k-Nearest Neighbor Query** $kNN(q,k)$ Retrieve the k closest elements to q. A set $\mathbb{N} \subseteq \mathbb{U}$ satisfying $|\mathbb{N}| = k$ and $\forall x \in \mathbb{N}, y \in \{\mathbb{U} - \mathbb{N}\}, d(q,x) \leq d(q,y)$.

A particular case of Metric Spaces is a Vector Space where the elements are indeed tuples $(x_1, x_2, x_3, ..., x_n)$ where $x_i \in \mathbb{R}$. Many methods were presented in this topic, such as Kd-Trees [3] and R-Trees [9]. The main problem of vector spaces is related with high dimensions, this problem is well known as the curse of dimensionality.

Many techniques to perform similarity queries have been presented in the literature [7]. One of them is SAT [12] which can be improved by selecting an optimal root for the tree. In this work we present a modification of SAT called SAT* that uses the CSA algorithm to select an optimal root for SAT.

The rest of the paper is organized as follows. In section 2 a review of the previous work is presented. Section 3 discusses SAT. In section 4 we present an algorithm to select SAT's root called the "Centroid Selection Algorithm (CSA)". Section 5 shows the experimental results. Finally, section 6 presents the conclusions of this work.

2 Previous Work

The first Metric Access Method (MAM), the Burkhard Keller Tree (BKT), was proposed in 1973 [5]. After that, a lot of new MAMs have been presented based on the same idea. Some methods were proposed for discrete distance functions and others for continuous distance functions.

BKT was designed for discrete distance functions. It takes an arbitrary element p to be the root of the tree, and group all the elements by their distance to p. Then each subset of elements is classified recursively with the same criterion.

The Fixed Queries Tree (FQT) [1] uses the same idea but it takes only one pivot per level of the tree. The Fixed Height Queries Tree (FHQT) [1] [6] extends each branch so all the leaves are at the same level. The Fixed Queries Array (FQA) [6] is an array containing the trajectory described by FHQT to go from the root to each element.

For continuous distance functions the Vantage Point Tree (VPT) [16] takes any element p as a root of a binary tree taking the median m , of all distances.

All the elements u that satisfy $d(p,u) \leq m$ are inserted at the left subtree and the other elements are inserted at the right subtree.

Another important MAM is the Bisector Tree (BST) [10] which takes two elements $c1$ and $c2$ for each node, the elements which are closer to $c1$ are inserted at the left subtree and the elements closer to $c2$ at the right subtree. In the search algorithm it uses a covering radius for each node.

The Generalized Hyperplane Tree (GHT) [10] is identical in construction but it uses an hyperplane between $c1$ and $c2$. The Geometric Near-neighbor Access Tree (GNAT) [4] is an extended GHT, but it selects m centers at each level.

In order to provide dynamic capabilities, balance, and good I/O performance, M-Tree was presented in [8]. It uses a construction process similar to GNAT, but the search algorithm is closer to BST because it uses a covering radius. When there is a new element, it is inserted at the best subtree, the subtree that causes less overlapping regions. When an overflow is produced, the node is splited in two and one node element is promoted upwards as in a B-Tree [2] or R-Tree [9].

Slim-Tree [15] was presented as an efficient M-tree [8]. It uses the same idea and also introduces the Fat-factor, which is an overlapping measure, and the Slim-Down algorithm in purpose to reduce overlapping regions.

There are other techniques such the Approximating Eliminating Search Algorithm (AESA) [13] which provides a very good query performance but it has to create a matrix with n(n-1)/2 precomputed distances. At search time it selects an arbitrary element $p \in \mathbb{U}$ and measure $r_p = d(p,q)$ so it only considers as candidates the elements u satisfying $r_p - r \leq d(u,p) \leq r_p + r$. The Linear Approximating Eliminating Search Algorithm (LAESA) [11] is other technique very similar to LAESA but it only uses a set of k pivots reducing considerably the construction cost.

The Omni Family [14] considers that LAESA's k pivots are randomly selected so they can be strategically selected to reduce the number of false candidates so it uses an algorithm to select the k pivots, called the Hull-Foci Algorithm (HF) [14].

These techniques have a very good query performance but they work dividing the space. There are other kind of MAMs such as SAT which uses the idea of approaching "spatially" the searched objects.

3 The Spatial Approximation Tree (SAT)

This MAM was presented in [12]. Instead of dividing the space, it retrieves the "spatially closest" element to a query. This is possible considering that each element $a \in \mathbb{U}$ has a set of neighbors $N(a)$, and we can only move to those neighbors.

3.1 Construction

SAT's algorithm takes an arbitrary element $a \in \mathbb{U}$ as the root of the tree. Then it connects a with all the elements that have a as their spatially closest element.

This produces a set of neighbors $N(a)$ where:

$$\forall x \in \mathbb{U}, x \in N(a) \Leftrightarrow \forall y \in \{N(a) - \{x\}\}, d(x,a) < d(x,y) \quad (1)$$

Algorithm 1 shows the building process [12]. It starts taking an element a (the root node) and its "queue" of nodes Q,containing the rest of $\mathbb{U}$. Then we order all Q elements by their distance to a. After that we add the elements satisfying (1) to the set $N(a)$ (which is initially empty), then we add all the elements $x \notin \{\{a\} \cup N(a)\}$ in the queue of its closest element in $N(a)$. The process continues recursively considering each element in $N(a)$ as the root of a new sub-tree.

Algorithm 1 BUILD (Node a , Queue of nodes Q)

```
1: N ⟵ ∅ /* neighbors of a*/
2: Sort Q by distance to a (closer first)
3: for v ∈ Q do
4:    if ∀b ∈ N, d(v,a) < d(v,b) then
5:       N ⟵ N ∪ {v}
6:    end if
7: end for
8: for b ∈ N do
9:    Q(b) ⟵ ∅ /* subtrees queues*/
10: end for
11: for v ∈ Q − N − {a} do
12:    Let b ∈ N be the one minimizing d(v,b)
13:    Q(b) ⟵ Q(b) ∪ {v}
14: end for
15: for b ∈ N /*build subtrees*/ do
16:    Add b as child of a
17:    BUILD(b, Q(b) )
18: end for
```

3.2 Searching

Range queries are performed as follows. Having a query $q \in \mathbb{X}$ and its respective closest element $q' \in \mathbb{U}$. We do not know q' but we can go to q' using q : by the triangular inequality it holds that for any $x \in \mathbb{U}, d(x,q) - r \leq d(x,q') \leq d(x,q) + r$ where r is the tolerance radius of our search.

4 The Centroid Selection Algorithm (CSA)

When SAT is built, its random root could be an extreme of the dataset, so the queries are going to compute more distance calculations. Algorithm 3 shows the

Algorithm 2 SEARCH (Node a , Query q , Radius r)

```
1: if d(a, q) ≤ r then
2:    Report a
3: end if
4: N ⟵ children nodes of a
5: mind ⟵ min_{c∈{a∪N}} d(c, q)
6: for b ∈ N do
7:    if d(b, q) ≤ mind + 2r then
8:       SEARCH(b, q, r)
9:    end if
10: end for
```

CSA process which selects an element that is near to the ideal centroid of the dataset to be the root of SAT.

Based on the HF algorithm presented in the Omni Family [14], CSA algorithm first selects a random element s. Then it finds the farthest element e_1 from s.After that it finds the farthest element e_2 from e_1. We consider c the element that minimizes (2) and (3).

$$|d(e_1, c) - d(e_2, c)| \tag{2}$$

$$|d(e_1, e_2) - (d(e_1, c) + d(e_2, c))| \tag{3}$$

We need to minimize (2) in purpose to assure that e_1 and e_2 are almost at the same distance from c. Also it is necessary to minimize (3) in order to have c close to the middle of e_1 and e_2. CSA's complexity is linear ($\theta(n)$)

Algorithm 3 CSA (Dataset $\mathbb{U}$)

1: Select a random element $s \in \mathbb{U}$
2: Find the farthest element e_1 from s, $\forall x \in \mathbb{U} - \{e_1\}, d(s, e_1) > d(s, x)$
3: Find the farthest element e_2 from e_1, $\forall x \in \mathbb{U} - \{e_2\}, d(e_1, e_2) > d(e_1, x)$
4: Let c the element that minimizes $|d(e_1, c) - d(e_2, c)|$ and $|d(e_1, e_2) - (d(e_1, c) + d(e_2, c))|$
5: $Q \longleftarrow \mathbb{U} - \{c\}$
6: $BUILD(c, Q)$

5 Experimental Results

Each experiment consists in 500 nearest neighbor queries, and three groups of 500 range queries with different radius. The elements for the queries were obtained randomly from the same datasets and we used the same query elements for both MAMs. We used the following datasets:

ABALONE: This dataset contains 4177 vectors in 8-*d*, describing different forms. The file was obtained from *UCI-Irvine repository of machine learning databases and domain theories*[3];

COVERT: This dataset contains 72446 vectors in 54-*d*, known as Forest Covertype data. The file was obtained from *UCI-Irvine repository of machine learning databases and domain theories*[4];

IMAGES: This dataset contains 80000 vectors in 1215-*d*, containing the feature extraction of different images. We used a subset of 20000 vectors. The file was obtained from The *Informedia* Project at *Carnegie Mellon University.*

As we can see in Table 1, the query performance is much better on SAT*, with bigger radii queries it gets a better performance. Also, distance calculations with smaller radii are better than those obtained using SAT.

ABALONE (8-d)				
Query type		Accumulated distances		% gained
		SAT	SAT*	
NN		31 896	28 127	11.81%
RQ	r=0.005	571 029	439 032	23.11%
	r=0.01	903 123	604 386	33.07%
	r=0.05	5 226 120	2678 205	48.75%

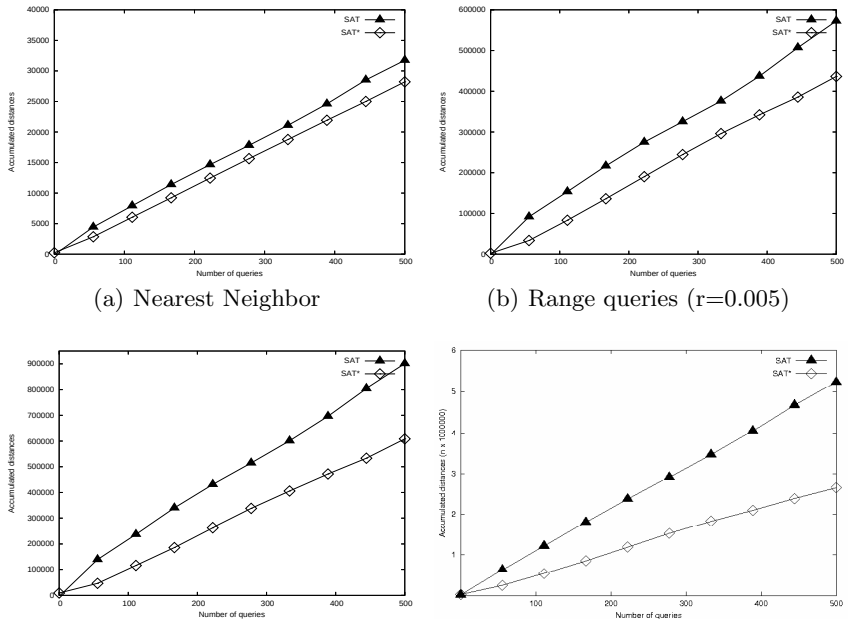
Table 1. Experiments using ABALONE dataset

Table 2 shows that the advantage in queries depends of the dataset but SAT* was better than SAT. As we can see in table 3 even in high dimensional spaces CSA algorithm chooses always an optimal root for the tree.

COVERT (54-d)				
Query type		Accumulated distances		% gained
		SAT	SAT*	
NN		33 702	30 034	10.88%
RQ	r=0.1	10 604 150	8 478 337	20.04%
	r=0.2	49 433 886	37 727 292	23.68%
	r=0.3	76 367 292	64 402 742	15.66%

Table 2. Experiments using COVERT dataset

[3] `ftp://ftp.ices.Uzi.ed/pub/machine-learning-databases/abalone/`
[4] `ftp://ftp.ices.Uzi.ed/pub/machine-learning-databases/covtype/`



(a) Nearest Neighbor (b) Range queries (r=0.005)

(c) Range queries (r=0.01) (d) Range queries (r=0.05)

Fig. 1. Abalone 8-d Database

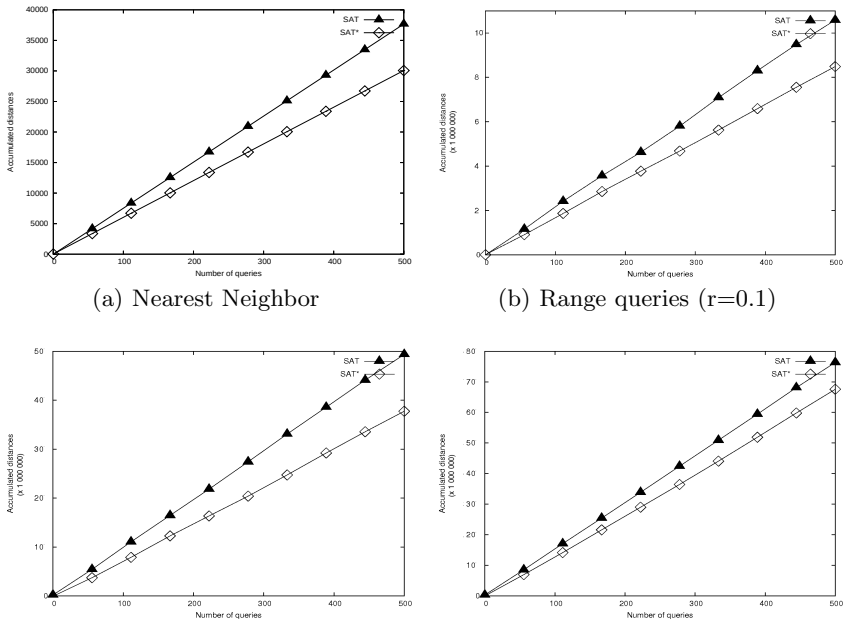
IMAGES (1215-d)				
Query type		Accumulated distances SAT	SAT*	% gained
NN		29 857	26 365	11.69%
RQ	r=0.05	645 377	552 494	14.39%
	r=0.08	1 216 116	881 142	27.54%
	r=0.1	5 1 782 392	1 222 090	31.43%

Table 3. Experiments using IMAGES dataset

Database	Accumulated distances SAT	SAT*
ABALONE (8-d)	308 989	325 239
COVERT (54-d)	5 923 510	4 671 473
IMAGES (1215-d)	1 204 957	1 221 315

Table 4. Number of distance calculations during the construction of SAT and SAT*

As we can see on Table 4, depending on the dataset, when we select an optimal root for SAT (SAT*), distance calculations are minimized, even if we consider the construction process.



(a) Nearest Neighbor (b) Range queries (r=0.1)

(c) Range queries (r=0.2) (d) Range queries (r=0.3)

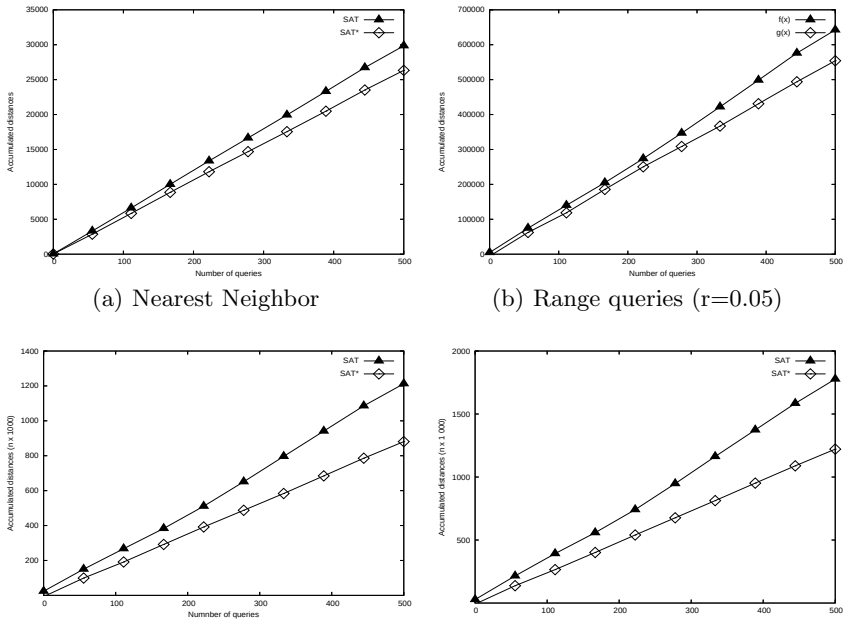
Fig. 2. Covert 54-d dataset

6 Conclusions

In this work we presented a modification of SAT called SAT* which is based on the idea that the best root for SAT is the center of the dataset. We proposed the CSA algorithm to select SAT's root as a solution to this problem. As we can see in our experiments, SAT* reduces distances calculations in both Range Queries and Nearest Neighbor queries, with only an irrelevant additional construction cost.

Considering Table 4 values, in every dataset the distance calculations are not equal to the linear CSA's cost + SAT's construction cost. That is because it simplifies the construction process only with an additional cost. We can observe that this cost is compensated with the efficiency gained during the construction. In the COVERT dataset SAT*'s construction computed less distances calculations than SAT. This is possible because SAT's random root was not near to the center of the dataset. But the element selected by CSA is a centroid of the dataset, making the construction process very efficient. It is important to know that CSA's performance decreases when space's dimension increases because objects are farther from each other and the centroid selected by CSA could not be centered at all.

As we demonstrate in the experiments SAT* has a very good query performance. Nearest Neighbor queries are much better performed by SAT*, but the



(a) Nearest Neighbor (b) Range queries (r=0.05)

(c) Range queries (r=0.08) (d) Range queries (r=0.1)

Fig. 3. IMAGES 1215-d dataset

difference is bigger for Range Queries. SAT* reduces more distance calculations compared with SAT. In the best case it gained more than 48% of efficiency.

References

1. Ricardo A. Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity Matching Using Fixed-Queries Trees. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 198–212, London, UK, 1994. Springer-Verlag.
2. Rudolf Bayer and Edward M. McCreight. Organization and Maintenance of Large Ordered Indexes. In *Record of the 1970 ACM SIGFIDET Workshop on Data Description and Access, November 15-16, 1970, Rice University, Houston, Texas, USA (Second Edition with an Appendix)*, pages 107–141. ACM, 1970.
3. Jon Louis Bentley. Multidimensional Binary Search Trees in Database Applications. *IEEE Transactions on Software Engineering*, 5(4):333–340, 1979.
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, September 11-15, 1995, Zurich, Switzerland*, pages 574–584. Morgan Kaufmann, 1995.
5. W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

6. Edgar Chávez, José L. Marroquín, and Gonzalo Navarro. Fixed Queries Array: A Fast and Economical Data Structure for Proximity Searching. *Multimedia Tools and Applications*, 14(2):113–135, 2001.
7. Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computer Survey*, 33(3):273–321, 2001.
8. Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 426–435. Morgan Kaufmann, 1997.
9. Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *ACM SIGMOD Conference*, pages 47–57, Boston, 1984.
10. Iraj Kalantari and Gerard McDonald. A Data Structure and an Algorithm for the Nearest Point Problem. *IEEE Transactions on Software Engineering*, 9(5):631–634, 1983.
11. María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.
12. Gonzalo Navarro. Searching in Metric Spaces by Spatial Approximation. *The VLDB Journal*, 11(1):28–46, 2002.
13. E. V. Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986.
14. Roberto Figueira Santos-Filho, Agma Traina, C. Traina Jr., and C. Faloutsos. Similarity Search without Tears: The OMNI Family of All-purpose Access Methods. *Proceedings of the 17th International Conference on Data Engineering*, pages 623–630, April 2001. Heidelberg, Germany.
15. Caetano Traina Jr., Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes. *Lecture Notes in Computer Science*, 1777:51–65, 2000.
16. Jeffrey K. Uhlmann. Satisfying General Proximity/Similarity Queries with Metric Trees. *Information Processesing Letters*, 40(4):175–179, 1991.