An Index Data Structure for Searching in Metric Space Databases

Roberto Uribe[1], Gonzalo Navarro[2], Ricardo J. Barrientos[1], and Mauricio Marín[1,3]

[1] Computer Engineering Department, University of Magallanes, Chile
[2] Computer Science Department, University of Chile
[3] Center for Quaternary Studies, CEQUA, Chile

Abstract. This paper presents the Evolutionary Geometric Near-neighbor Access Tree (EGNAT) which is a new data structure devised for searching in metric space databases. The EGNAT is fully dynamic, i.e., it allows combinations of insert and delete operations, and has been optimized for secondary memory. Empirical results on different databases show that this tree achieves good performance for high-dimensional metric spaces. We also show that this data structure allows efficient parallelization on distributed memory parallel architectures. All this indicates that the EGNAT is suitable for conducting similarity searches on very large metric space databases.

1 Introduction

Searching for similar objects into a large collection of objects stored in a metric-space database has become an important problem. For example, a typical query for these applications is the *range query* which consists on retrieving all objects within a certain distance from a given query object. From this operation one can construct other ones such as the nearest neighbors. Applications can be found in voice and image recognition, and data mining problems.

Similarity can be modeled as a metric space as stated by the following definitions.

Metric Space. A metric space is a set X in which a distance function is defined $d : X^2 \rightarrow R$, such that $\forall\, x, y, z \in X$,

1. $d(x, y) \geq 0 \, and \, d(x, y) = 0 \text{ iff } x = y$.
2. $d(x, y) = d(y, x)$.
3. $d(x, y) + d(y, z) \geq (d(x, z)$ (triangular inequality).

Range query. Given a metric space *(X,d)*, a finite set $Y \subseteq X$, a query $x \in X$, and a range $r \in R$. The results for query x with range r is the set $y \in Y$, such that $d(x, y) \leq r$.

The distance between two database objects in a high-dimensional space can be very expensive to compute and in many cases it is certainly the relevant performance metric to optimize; even over the cost secondary memory operations.

For large and complex databases it then becomes crucial to reduce the number of distance calculations in order to achieve reasonable running times. This makes a case for the use of parallelism.

The distance function encapsulates the particular features of the application objects which makes the different data structures for searching general purpose strategies [4]. Well-known data structures for metric spaces are BKTree [3], MetricTree [9], GNAT [2], VpTree [12], FQTree [1], MTree [5], SAT [6], Slim-Tree [8]. Some of them are based on clustering and others on pivots. The EGNAT proposed in this paper is based on clustering [10].

In the case of pivots based strategies a set of (usually random) objects are selected from the database and distances are calculated to organize the pivots in a, for example, tree fashion. A search query is executed by calculating distances between the query object and the pivots so that the search space is reduced by applying the triangular inequality to discard tree branches.

The strategies based on clustering divide the space in areas, where each area has a center point. Information is stored in each area so that it allows easy discarding of the whole area by just comparing the query with the center point. The strategies based on clustering are better suited than pivots ones for high-dimensional metric spaces.

Voronoi Diagrams: Consider a set of point $\{c_1, c_2, \ldots, c_n\}$(centers). A Voronoi Diagram is defined as the subdivision of the plane in n areas, one for each c_i, such that q is in the area c_i if and only if the euclidean distance holds $d(q, c_i) < d(q, c_j)$ for each c_j, with $j \neq i$.

The EGNAT is based on the concepts of Voronoi Diagrams and is an extension of the GNAT proposed in [2], which in turn is a generalization of the *Generalized Hyperplane Tree* (GHT) [9]. Basically the tree is constructed by taking two selected points (the two children of the root) and distributing the remaining points according with how close in distance they are to one of the two points. This is repeated recursively in each sub-tree.

In the GNAT k points, instead of two, are selected to divide the space $\{p_1, p_2, \ldots, p_k\}$, where every remaining point is assigned to the closet one among the k points.

Most data structures and algorithms for searching in metric-space databases were not defined to be dynamic ones [4]. However, some of them allow insertion operations in an efficient manner once the whole tree has been constructed from an initial set of points (objects). Deletion operations, however, are particularly complicated because in this strategies the invariant that supports the data structure can be easily broken with a sufficient number of deletions, which makes it necessary to re-construct from scratch the whole tree from the remaining points. Experimental results about these issues can be found in [7].

When we consider the use of secondary memory we find in the literature a few strategies which are able to cope efficiently with this requirement. Among the well-know strategies are the *M-Tree* [5] which has a similar performance to the GNAT in terms of number of accesses to disk and overall size of the data structure.

2 Evolutionary Geometric Near-neighbor Access Tree

The construction of the initial EGNAT is performed using the GNAT method proposed by [2], that is

1. Select k points called *centers*, $p_1, \ldots, p_k$.
2. Associate every remaining point with the nearest center. The set of points associated with every center p_i is denoted by D_{p_i}.
3. For each pair of centers (p_i, p_j), the following range is calculated,

$$\text{range}\left(p_i, D_{p_j}\right) = \left[\min\{d\left(p_i, D_{p_j}\right)\}, \max\{d\left(p_i, D_{p_j}\right)\}\right].$$

4. The tree is constructed recursively for each element in D_{p_i}.

Every set D_{p_i} represents a sub-tree whose root is p_i.

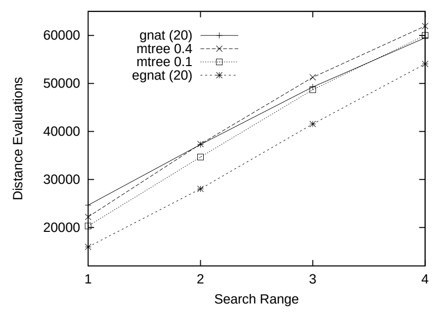
Additionally, the EGNAT [10] is data structure in which the nodes are created as buckets in which the only information is the distance to their father. This allows a significant reduction in space used in disk and also allows good performance in terms of number of distance evaluations. When a node becomes full of objects it evolves from a bucket to a GNAT node by re-inserting all objects in the bucket to the new GNAT sub-tree node.

Searching in the EGNAT is performed recursively as follows,

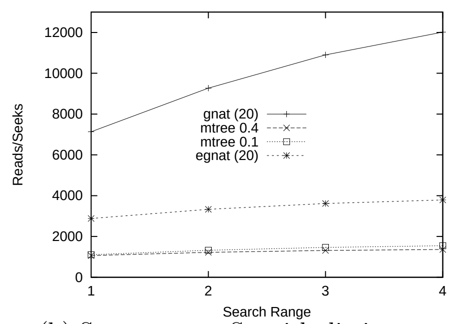
1. Assume that we are interested in retrieving all objects with distance $d \leq r$ to the query object q (range query). Let P be the set of centers of the current node in the search tree.
2. Choose randomly a point p in P, calculate the distance $d(q,p)$. If $d(q,p) \leq r$, add p to the output set result.
3. $\forall\, x \in P$, if $[d(q,p) - r, d(q,p) + r] \cap \text{range}(p, D_x)$ is empty, the remove x from P.
4. Repeat steps 2 and 3 until processing all points (objects) in P.
5. For all points $p_i \in P$, repeat recursively the search in D_{p_i}.

3 EGNAT performance (sequentially and in parallel)

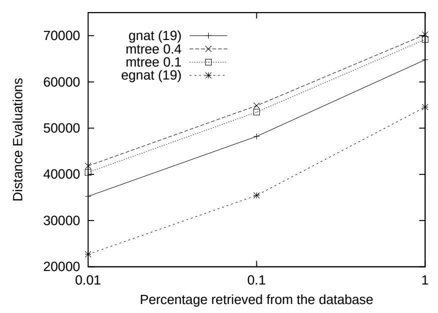
In figure 1 we show results comparing the EGNAT with the M-Tree for both number of distance calculations and access to secondary memory. These results show that EGNAT is more efficient than the M-Tree. The results shown in the figure were obtained for the best tunning parameters for each data structure (19, 20, 0.4, 0.1, details in references).



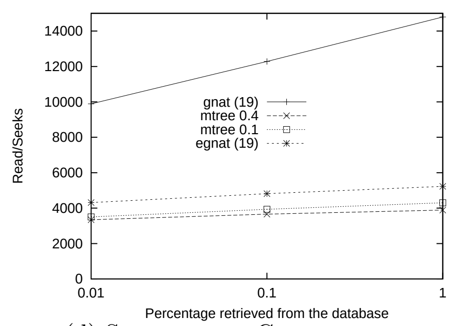
(a) Avg cost for search, Spanish dictionary



(b) Sec. memory, Spanish dictionary



(c) Avg cost for search, Gauss vectors



(d) Sec. memory, Gauss vectors

Fig. 1. Results for the local index approach.

We also tested the suitability of the EGNAT data structure for supporting query processing in parallel. We evenly distributed the database among P processors of a 10-processors cluster of PCs. Queries are processed in batches as we assume an environment in which a high traffic of queries is arriving to a broker machine. The broker routes the queries to the processors in a circular manner. We takes batches as we use the BSP model of computing for performing the parallel computations [11].

In the bulk-synchronous parallel (BSP) model of computing [11], any parallel computer (e.g., PC cluster, shared or distributed memory multiprocessors) is seen as composed of a set of P processor-local-memory components which communicate with each other through messages. The computation is organized as a sequence of *supersteps*. During a superstep, the processors may only perform sequential computations on local data and/or send messages to other processors. The messages are available for processing at their destinations by the next superstep, and each superstep is ended with the barrier synchronization of the processors.

We used two approaches to the parallel processing of batches of queries. In the first case, an independent EGNAT is constructed in the piece of database stored in each processors. Queries in this case start at any processor at the beginning of each superstep. The first step in processing a particular query is to send a copy of it to all processors including itself. At the next superstep the searching algorithms is performed in the respective EGNAT and all solutions found are reported to the processor that originated the query. We call this strategy the *local index* approach.

In the second case, we assume that a single EGNAT has been constructed considering the whole database. The first levels of the tree are kept duplicated in every processor. The size of this tree is large enough to fit in main memory. Downwards the tree branches or sub-trees are evenly distributed onto secondary memory of the processors. A query stars at any processor and the sequential algorithms is used for the first levels of the tree. The copies of the query "travel" to other processors to continue the search in the sub-trees stored in remote secondary memory. Note that queries can divided in multiple copies according to the tree paths that contains valid results. Thus these copies are processed in parallel when are sent to different processors. We call this strategy the *global index* approach.

Results for databases formed by natural language text, a large set of points formed with Gaussian distribution and a collection of images are shown in the figure 2. A total of 10,000 queries are processed. The results show that the local index approach achieves good efficiency in parallel. In particular, because of the relatively larger cost of disk access with respect to communication cost, we observed super-linear speedups in the results. The speedups for the global index approach were very similar.

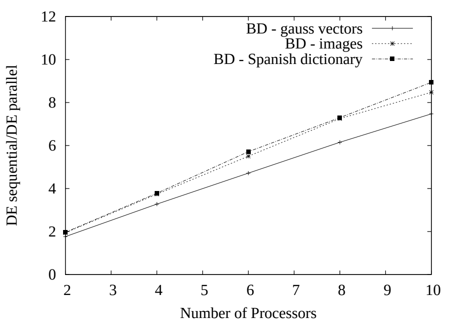
4 Conclusions

We have described the EGNAT data structure and shown its performance both sequentially and in parallel.

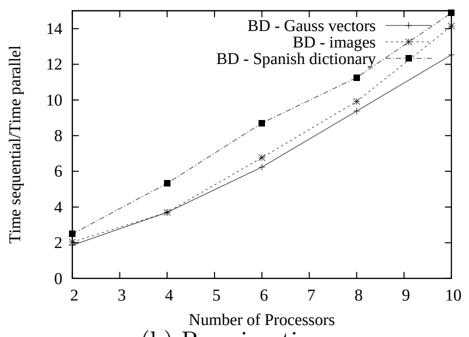
The results show that this data structure is a good choice for systems large enough that tree nodes has to be stored in secondary memory. It also allows insert and delete operations to take place once the tree has been constructed.

For the sequential case the results with different databases show that EGNAT is more efficient than the well-known M-Tree both in number of distance evaluations required to solve range queries and amount of accesses to secondary memory.

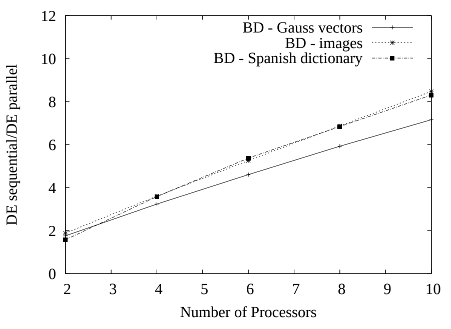
For the parallel setting, the results show that the EGNAT admits an efficient parallelization. The results for running time show that it is feasible to significantly reduce the running time by the inclusion of more processors. This because of distance calculations takes in parallel during solutions of batches of queries. We emphasize that for use of parallel computing to be justified we must put ourselves in a situation of a very high traffic of user queries. The results show that in practice just with a few queries per unit time it is possible to achieve



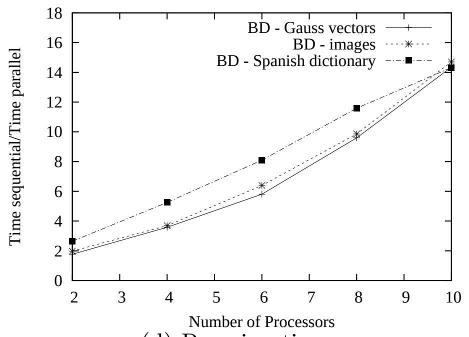
(a) Distance calculations,
1 new query/superstep



(b) Running time,
1 new query/superstep



(c) Distance calculations,
10 new queries/superstep



(d) Running time,
10 new queries/superstep

Fig. 2. Results for the local index approach.

good performance. That is, the combined effects of good load balance in both distance evaluations and accesses to secondary memory across the processors, are quickly enough to achieve good performance.

Acknowledgements

This work has been partially funded by research project FONDECYT 1040611, Chile.

References

1. R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixedqueries trees. In *5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

2. Sergei Brin. Near neighbor search in large metric spaces. In *the 21st VLDB Conference*, pages 574–584. Morgan Kaufmann Publishers, 1995.
3. W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communication of ACM*, 16(4):230–236, 1973.
4. Edgar Chvez, Gonzalo Navarro, Ricardo Baeza-Yates, and Jos L. Marroqun. Searching in metric spaces. In *ACM Computing Surveys*, pages 33(3):273–321, September 2001.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree : An efficient access method for similarity search in metric spaces. In *the 23st International Conference on VLDB*, pages 426–435, 1997.
6. Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.
7. Gonzalo Navarro and Nora Reyes. Fully dynamic spatial approximation trees. In *the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, pages 254–270, Springer 2002.
8. Caetano Traina, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *VII International Conference on Extending Database Technology*, pages 51–61, 2000.
9. J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters*, pages 40:175–179, 1991.
10. R. Uribe. A space-metric data structure for secondary memory. Master's thesis, Computer Science Department, University of Chile, Santiago, Chile, Abril 2005.
11. L.G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33:103–111, Aug. 1990.
12. P. Yianilos. Data structures and algoritms for nearest neighbor search in general metric spaces. In *4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.