Metric-Temporal Access Methods

Anabella De Battista, Andrés Pascal
Universidad Tecnológica Nacional, F.R. Concepción del Uruguay
Ing. Pereira 676, Concepción del Uruguay, Entre Ríos, Argentina
{debattistaa, pascala}@frcu.utn.edu.ar
Norma Edith Herrera
Universidad Nacional de San Luis
Ejército de los Andes 950, San Luis, Argentina
nherrera@unsl.edu.ar
Gilberto Gutierrez
Facultad de Ciencias Empresariales Universidad del Bio Bio
Avenida Andrés Bello s/n, Chillán, Chile
nherrera@unsl.edu.ar

ABSTRACT

Metric-temporal databases are a new database model that combines metric spaces with temporal databases to process similarity queries within a time interval or snapshot. The Historical FHQT is a metric-temporal index which has shown to be competitive answering this type of queries. This index store a list of valid snapshots where each one contains an Fixed Height Queries Tree that indexes all objects existing at that instant. In this paper we present an improvement to this access method that consists in using different sets of pivots for the Fixed Height Queries Tree that correspond to consecutive time instants. The experimental results show this modification improves the filtering capacity of the index.

Keywords: *metric spaces, temporal databases, indexes, metric-temporal databases.*

1 INTRODUCTION

Traditional databases are built around the concept of structured data and exact searching: the database is divided in records, each record having a fully comparable key; querying the database return all the records whose keys are exactly equal to the search key. Actually, the databases have included the ability to store new data types such as images, audio, video, text. The traditional models of databases are not useful in this new framework because of three reasons: first, the data are unstructured, this means that it is not possible anymore to organize them in records and fields; second, the exact searching has not interest; third, conventional databases capture a single state of the reality and it is insufficient for applications that requieres the support of past, current or even future data.

The metric spaces model [1, 2, 4, 5, 6, 10] formalizes the similarity search concept in nontraditional databases. This type of search is frequently found in diverse topics of computer science such as voice and image recognition, text compression, computational biology, artificial intelligence, data mining, etc. Temporal Databases [13, 9] is a database model that supports time-dependent data. While in traditional databases deal with time as any other type of data, this model incorporates time as a dimension. The Metric-Temporal Databases model [7, 8, 11] combines features of both models mentioned above allowing similarity query that considers too, the temporal aspect. This involves searching for objects similar to a given query within an interval whose duration overlaps the range provided by the query. As an example, consider a collection of photographic record of people entering a bank for a given period of time; it could be interesting to find people with similar faces to a given one that were in the bank within a time interval. In this paper we are interested in access methods (indexes) for this kind of database.

In [8] the Historical Fixed Height Queries Tree (H-FHQT) is proposed. This is a metric-temporal access method that store a list of valid instants of time $t_1, t_2, \cdots, t_k$ where each t_i contains a metric index, the Fixed Height Queries Tree (FHQT), with all valid objects at the instant t_i. In this paper we present an improvement to this index that is mainly suitable for interval metric-temporal queries.

The paper is organized as follows. Section 2 describes the related work, defining the concepts necessary for understanding this work. Section 3 presents our contribution, the Pivot H-FHQT index. Section 4

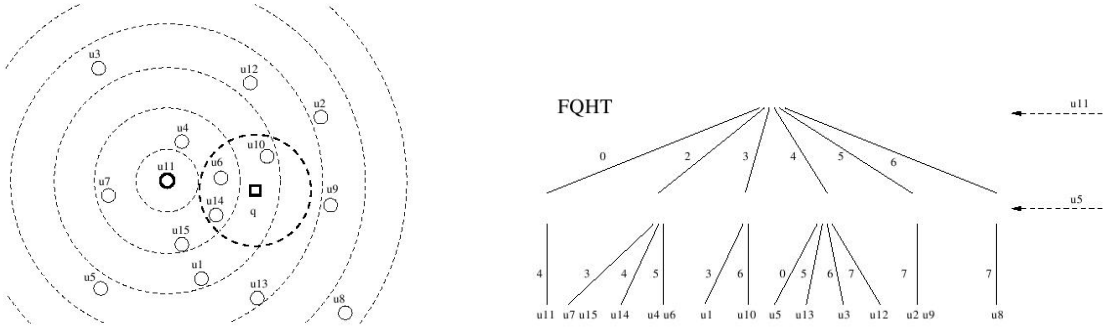

Figure 1: On the left, a set of points and the division of the space when u_{11} is taken as a pivot and the distances have been discretized. On the right, a FHQT for this set of points. We also show a query q over the set of points.

shows the experimental evaluation of the index and finally, Section 5 presents the conclusions and future work.

2 RELATED WORK

In this section we describe the metric spaces model, the metric-temporal model and the indexes that form the basis of this work.

2.1 Metric Spaces

Searching in non-traditional databases (e.g. images, fingerprints, audio clips, etc.) deter the concept of exact searching. There is no point in searching for a bitwise equal object when presenting a query. We search instead for *similar* objects.

In [6] is shown that the similarity search problem can be expressed as follows: given a set X of objects and a distance function d, defined among them, that quantifies their similitude, the aim is to retrieve all the elements similar to a given one. This function d satisfies the properties required to be a distance function: positivity ($d(x, y) \geq 0$), simetry ($d(x, y) = d(y, x)$) and triangle inequality ($d(x, y) \leq d(x, z) + d(z, y)$).

The smaller the distance between two objects, the more similar. The pair (X, d) is called ***metric space***. A finite subset $U \subseteq X$, which will be called ***database***, is the set of objects where we search.

One of the typical queries over this new database model is the ***range query*** denoted by $(q, r)_d$. Given a query $q \in X$ and a tolerance radius r, a range query retrieve all elements within a distance r from q in the the database U , this is:

$$(q, r)_d = \{u \in \mathcal{U} : d(q, u) \leq r\}$$

The total query time T can be calculated as T = #evaluations of d $\times$ complexity(d) + extra CPU time + I/O time. In many applications, the evaluation of function d is so costly that the other terms in the formulae can be neglected. This is the complexity model we used in this work; therefore, our complexity measure will be the number of evaluations of the distance function d.

A range query can be trivially answered by an exhaustive examination of the database. Unfortunately, this is generally very costly in real applications, ($O(n)$ distance evaluations where $n = |U|$). To avoid this situation, the database is preprocessed using an indexing algorithm whose aim is to build a ***data structure*** or ***index***, designed to save distance evaluations at query time.

In [6] the authors present a unifier development for all the existing solutions in this topic. In that survey the authors state that the metric spaces indexing algorithms are based on, first, partitioning the space into equivalence classes and, second, a subsequent indexation of each class. Afterwards, at query time, some of these classes can be discarded using the index and an exhaustive search is maked only on the remaining classes. The main difference between the existing indexing algorithms is how they build the equivalence classes. Basically there are two groups: pivots-based algorithms and compact partitions-based algorithms.

The Fixed Height Queries Tree (FHQT) [1] belongs to the pivot-based algorithms group and it is basically a variant of the Fixed Queries Tree[1] where all the leaves are at the same depth. Originally these structures were proposed for discrete distance functions, but they can be suitable for continuous distances [6, 12].

The FHQT is a tree built from an element p, called ***pivot***, that can be chosen arbitrarily or by some pivot selection techniques [3]. For each distance i, we define $C_i = \{u \in U/(d(p, u) = i\}$ as the set of all elements at distance i to the root p. For any nonempty C_i, we build a child of p labeled i where we recur-

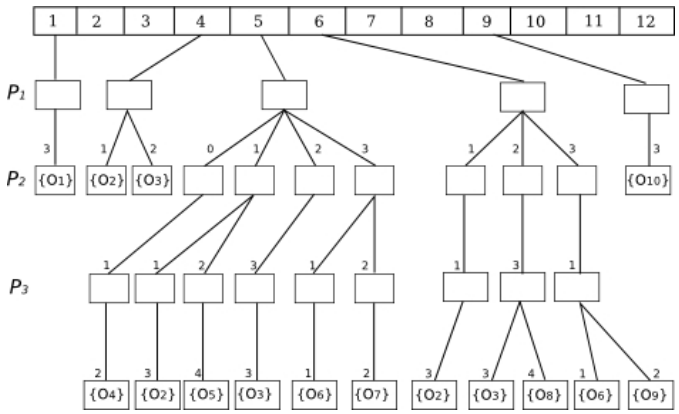

Figure 2: Example H-FHQT built on the time interval $[1..12]$.

sively build the FHQT for C_i. This process is repeated until the leaves are at the same depth h and have no more that b elements. All the pivots stored in the nodes of the same level are the same and, of course, not necessarily belong to the set stored in the subtree. Note that the total number of pivots is determined by the height of the tree.

A search for a query element q proceeds down the tree by computing at each level the distance between q and the pivot p_i for that level. We call ***signature*** of the query q to the vector $(d(q, p_1), d(x, p_2), \cdots, d(q, p_k))$ where k is the number of pivots used. The triangle inequality is used in order to filter out elements in the database without measuring their distance to the query q. If we are at internal node v with pivot p_i then all children of v labeled j such that $|d(q, p_i) - j| > r$ can be discarded. Items that can not be discarded form part of a list of candidates. These candidates are compared with the query q to determine whether they belong or not to the answer.

Figure 1 shows an example of a FHQT with two pivots. On the left is showed the space division that produces the selection of u_{11} as pivot and on the right the FHQT resulting of choose u_5 and u_{11} as pivots.

2.2 Metric-Temporal Database

This type of database allows searching for non-structured objects that have an associated validity interval. A ***metric-temporal space*** is defined as a pair (U, d), where $U = O \times N \times N$ denote the universe of valid objects and $d : O \times O \rightarrow R$ is a distance function.

Each element $u \in U$ is a 3-tuple (obj, t_i, t_f) where obj is an object (i.e a picture, sound, string, etc.) and $[t_i, t_f]$ is the validity period of obj. The distance function d, which measures the similarity between two objects, satisfies the properties of a metric (positivity, symmetry, reflexivity and triangle inequality). A metric-temporal query is defined as a 4-tuple $(q, r, t_{iq}, t_{fq})_d$, such that: $(q, r, t_{iq}, t_{fq})_d = \{o/(o, t_{io}, t_{fo}) \in X \wedge d(q, o) \leq r \wedge \ t_{io} \leq t_{fq} \ \wedge t_{iq} \leq t_{fo})\}$.

A trivial way to answer a metric-temporal query, avoiding the sequential scan, is to build a metric index adding a time interval of validity to each object. Given a query $(q, r, t_{iq}, t_{fq})_d$,, firstly we use a metric index to discard objects obj that are farther than r from q, and then perform a review on the set of elements non-discarded in the first step in order to determine which objects form the answer to the query. They are those whose range overlaps with $[t_{iq}, t_{fq}]$.

The main drawback of this trivial solution is that the time component is not used to improve the filtering ability of index, in this process only the metric component is used. A better strategy is to use both metric and temporal components to discard objects at query time.

The Historical-FHQT (H-FHQT) [8] is a metric-temporal index that uses both metric and temporal components to efficiently answer metric-temporal queries. This index is composed by a list of valid snapshots where each one contains an FHQT that indexes all objects existing at that instant. The FHQT of different instants have different depths, i.e. different numbers of pivots, depending on the amount of objects to be indexed. The number of pivots used in each tree is calculated as $\lceil \log_2(|o_i|) \rceil$, where $|o_i|$ is the number of objects alive at instant i. In this way, trees with greater depth than necessary are avoided, and consequently the storage cost is reduced. Although the number of pivots varies in different instants, it always works with the same pivots and at the same order. This means that if the instant i needs k_i pivots and the instant j needs k_j pivots, where $k_i < k_j$, then the first k_i pivots are equal in both instants. This avoids that a query is compared with different sets of pivots in different instants, implying increase the number of distance function evaluations to calculate the signature of q at each time.

Figure 2 shows an example of an H-FHQT built on the time interval $[1..12]$. As can be see, the object o_2

is alive at the interval $[4..6]$, $d(o_2, p_1) = d(o_2, p_2) = 1$ and $d(o_2, p_3) = 3$. We denote $fhqt_i$ to the FHQT that correspond the instant i, and k_i to its number of pivots. A metric-temporal query $(q, r, t_{iq}, t_{fq})_d$ is answered by the H-FHQT in the following way: for each instant i included in the query interval, a similarity range query is made on the corresponding $fhqt_i$ and then the resulting sets are joined. Figure 3 shows the search algorithm. The *extend* process calculate the signature of the query on as-needed basis. If k_i is greater than the maximum number of pivots used so far, the signature is extended to k_i pivots. Otherwise the signature already contains all the necessary information to perform a range search over $fhqt_i$.

3 IMPROVING THE H-FHQT PERFORMANCE: PIVOT H-FHQT

As described on the previous section, the aim of using the same group of pivots for different $fhqt$ involved in an H-FHQT is to avoid calculation of different signatures for each time instant. But, let us suppose that an object o such as $o \notin (q, r, t_{iq}, t_{qf})_d$, is valid in several moments of time i included in the interval of the query. If the object o could not be discarded by $fhqt_i$, then the only chance to be discarded by the $fhqt_{i+1}$ is that the number of pivots k_{i+1} is greater than k_i, so these additional pivots could eliminate o (let us remember the elimination rule of the search algorithm seen in section 2.1). This means that the filtering ability of $fhqt_{i+1}$ against the object o is reduced to the filtering ability of the additional pivots, if they exist. One solution to this problem is to use disjoint sets of pivots for consecutive $fhqt$, although increases the number of distance evaluations when calculating the signature of the query q, also increases the likelihood of reducing the number of candidates which should be compared with q.

These ideas were the basis for design changes to improve the H-FHQT performance. We call this new version ***Pivot H-FHQT*** (PH-FHQT).

In a PH-FHQT the $fhqt_i$ is built with different pivots that the $fhqt_{i-1}$ and the $fhqt_{i+1}$. To achieve this goal, each $fhqt_i$ take the first k_i available pivots from a global list of pivots $(p_0, p_1, \cdots, p_{m-1})$, which is handled as a circular list: the $fhqt_1$ use the first k_1 pivots, $fhqt_2$ uses the k_2 following pivots, and so on, until reach the last available pivot from the list. Then, it is resumed from the beginning. The construction of consecutive $fhqt$ with different set of pivots gives to the index more filtering power from the metric point of view.

Figure 4 shows an example of a PH-FHQT built on the interval time $[1, 12]$. It has valid objects only at the instants 5, 7, 8 and 9. As can be seen $fhqt's$ having different depths depending on the number of objects indexed.

To query the PH-FHQT we proceed in a similar way that the used for the H-FHQT, i.e. the instants i included in the query interval are selected, then each of the corresponding $fhqt_i$ are queried for similarity and finally the resulting sets are joined in a non-trivial way. For each $fhqt_i$, a set C_i of candidates and a set D_i of rejected elements are obtained. For an object o to be a final candidate, must belong at least to one of the C_i involved and to none of the D_i, i.e. the object o should have survived at all pivots of the involved $fhqt_i$. Then, since the filtering capacity of the $fhqt_i$ is independent of the filtering capacity of the $fhqt_{i-1}$, the filtering is more efective.

Figure 5 shows the search algorithm on PH-FHQT. The *compute* process calculate the signature of the query q using the pivots of $fhqt_i$. This signature is used por *range* process to make a range search on the $fhqt_i$, returning the set of candidates C_i and the set of rejected elements D_i. At the end, all objects belonging to C_i and not to D_i, make the final list of candidates which are compared with q to determine the final result.

H-FHQT $(q, r, t_{iq}, t_{fq}, d)$ **: set**
1. $R = \emptyset$
2. $last = 0$
3. for $t_{iq} \leq i \leq t_{fq}$
4. if $k_i > last$
5. $signature_q = \texttt{extend}(sigature_q, last, k_i)$
6. $last = k_i$
7. end if
8. $R = R \cup$ Range$(q, r, d, fhqt_i, signature_q, k_i)$
9. end for
10. return(R)

*$fhqt_i$ is the FHQT at instant i and k_i is the number of pivots of $fhqt_i$
*Range process make $(q, r)_d$ on $fhqt_i$ using the first k_i elements of $signature_q$.

Figure 3: Metric-temporal search algorithm on a H-FHQT

PH-FHQT $(q, r, t_{iq}, t_{fq}, d)$ **: set**
1. $C = D = R = \emptyset$
2. for $t_{iq} \leq i \leq t_{fq}$
3. $signature_{i_q} = \texttt{compute}(q, pivots(fhqt_i))$
4. $(C_i, D_i) = \texttt{range}(q, r, d, fhqt_i, signature_{i_q}, i, k_i)$
5. $C = C \cup C_i$
6. $D = D \cup D_i$
7. end for
8. $C = C - D$
9. for all $o \in C$
10. if $d(q, o) \leq r$ then $R = R \cup \{o\}$
11. end for
12. return(R)

Figure 5: Metric-temporal search algorithm on a PH-FHQT

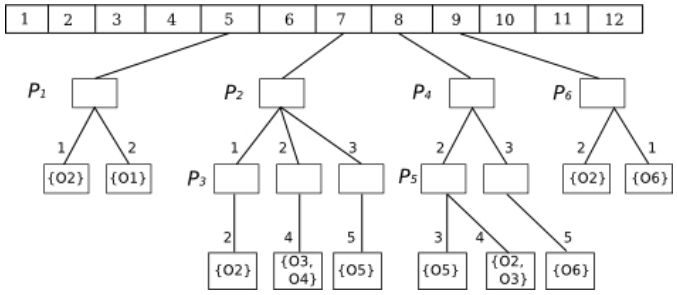

Figure 4: Example Pivot H-FHQT built on the time interval $[1..12]$.

4 EXPERIMENTAL EVALUATION

For our experiments, we used two vectorized image databases widely used by the metric spaces community: ***Colors***, containing 761-dimensional vectors and ***Nasa***, with 20-dimensional vectors, available at http://www.sisap.org/library/dbs/vectors.

From both, batches of sizes 5,000, 10.000 and 15.000 were randomly generated. An identifier and a validity interval were assigned to each object. The interval indicates the validity time of the object, within the range [1..1000]. The Euclidean distance was used as distance function in both cases. From now on, well refer the metric-temporal databases generated, as ***NasaMT*** and ***ColorsMT***.

For each of the six databases created, 100 metric-temporal range queries were generated by randomly taking 100 elements from each batch, varying the query radii and interval.We have considered queries retrieving on average 1%, 5% and 10% of the dataset, this means radii 5, 9 and 11 for *ColorsMT* y radii 7, 9 and 11 for *NasaMT*. The query intervals used the values: *instant,* 10%, 25% and 50% of the total range.

In the H-FHQT, each $fhqt$ was constructed taking pivots from the same list of randomly chosen elements from the database. In the case of PH-FHQT, a global list of pivots with a cardinality ranging from $\lceil log_2(|O_i|)\rceil$ to $\lceil log_2(|O_i|)\rceil * 2$ was generated.

Because of space reasons, only most relevant results will be included; intermediate results will not be exposed but are available on demand.

Figure 6 shows the results obtained for the ColorsMT database with 5000 objects for instant and interval queries with the H-FHQT and the PH-FHQT. The X-axis represent the search radii and the Y-axis, the average number of distance evaluations. As can be seen, the PH-FHQT index has better performance than the H-FHQT performing 12% less distance evaluations. This improvement is more significant in the case of interval queries, reaching up to 26% less distance function evaluations. In all cases the percentage of improvement decreases as the search radius increase, reaching 7% in the case of instant queries and a 10% for interval queries. The results obtained with batches of 10,000 and 15,000 elements followed the same pattern.

The results obtained for the NasaMT with 5.000 objects as show in Figure 7. In this case, we can see that for instant queries the PH-FHQT outperforms the H-FHQT only when $r = 7$, achieving an improvement of 1%.In contrast, for $r = 9$ and $r = 11$ the H-FHQT is more competitive than the PH-FHQT, getting improvements of 2% and 3% respectively. It is noted for these two cases that the improvement rate increase when the search radius increases. Instead, when analyzing the results for the interval queries, we see that the PH-FHQT index is the most efficient in all cases considered. The improvements vary between 2% and 14%, decreasing as increasing the search radii. These results obtainded with batches of 10,000 and 15,000 objects folloved the same pattern.

5 CONCLUSIONS AND FUTURE WORK

In this paper we presented an improvement to the H-FHQT metric-temporal index consisting in to allow the use of different groups of pivots for trees that correspond to consecutive instants. This originated a new index, the PH-FHQT, which proved to be more competitive than its predecessor, the H-FHQT, in all interval queries executed on the NasaMT and ColorsMT databases. For instant queries on ColorsMT, the PH-FHQT outperforms in all cases the H-FHQT, and for the NasaMT database, was more competitive in 66% of the tests. The improvements observed in this index are due to the greater power of filtering that is achieved by generating consecutive $fhqt_i$ with different sets of pivots. Regarding future work we intend to improve this index with respect to the storage space required. The aim is to detect equal subtrees at different instants in order to reuse such structures to avoid redundancy in storage.

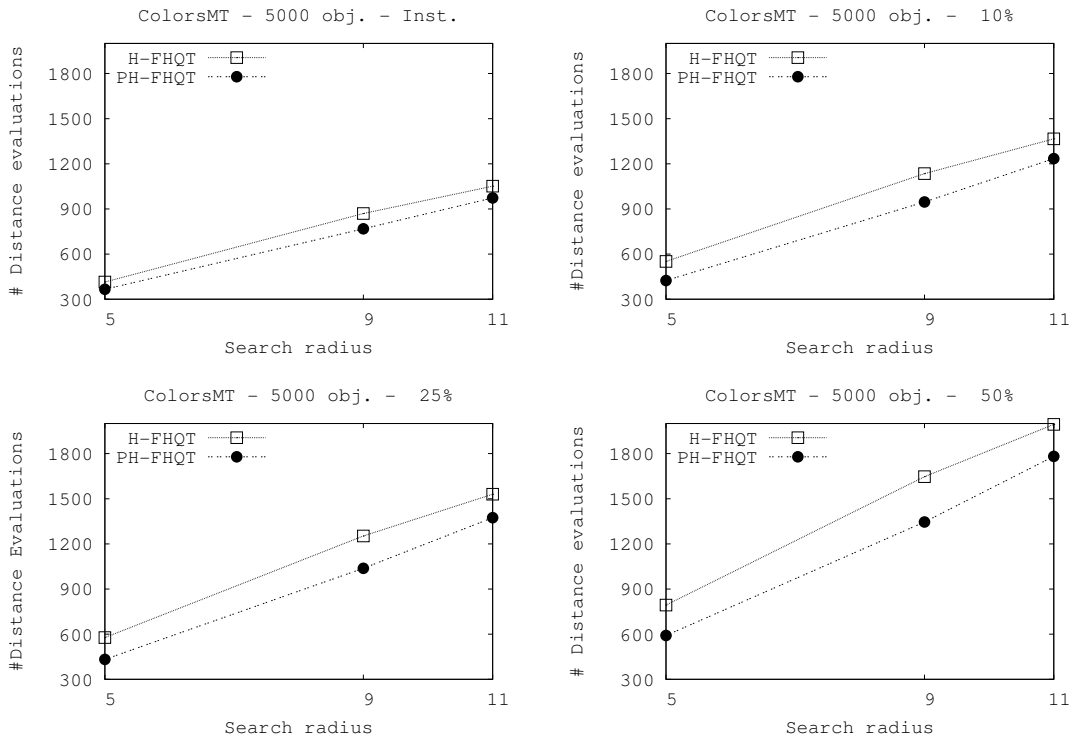

Figure 6: Distance evaluations for ColorsMT, using query intervals $instant$, 10%, 25% and 50%.

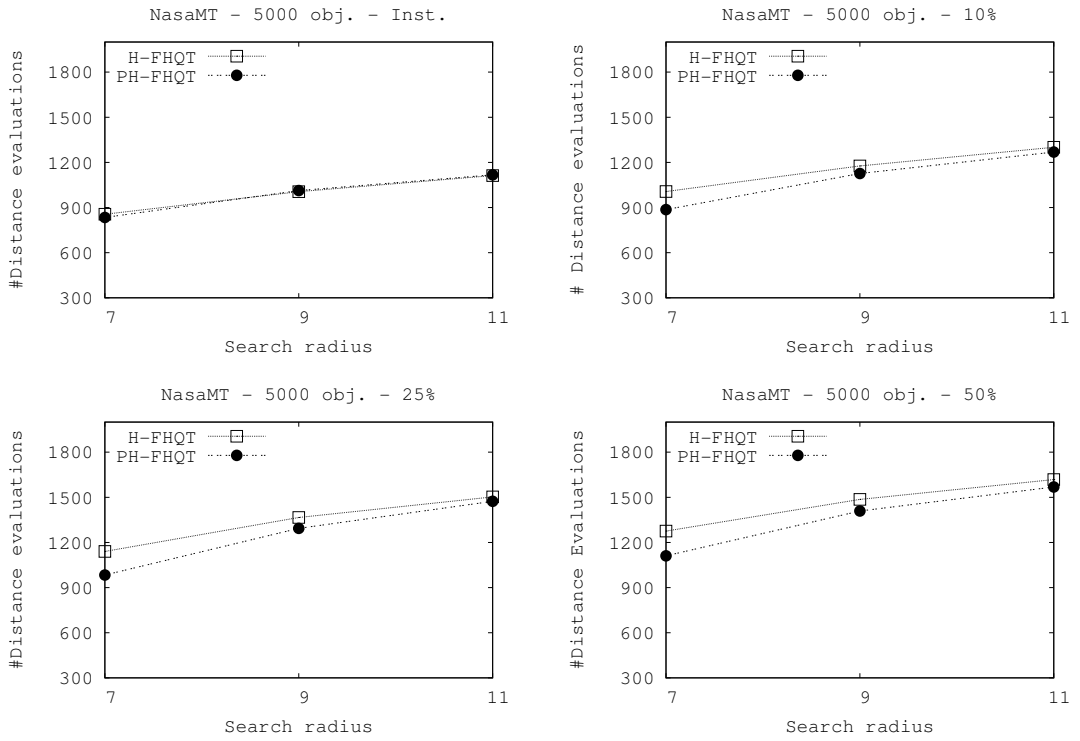

Figure 7: Distance evaluations for NasaMT, using query intervals $instant$, 10%, 25% and 50%.

REFERENCES

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

[2] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)*, pages 574–584, 1995.

[3] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)*, pages 33–40. IEEE CS Press, 2001.

[4] E. Chávez and K. Figueroa. Faster proximity searching in metric data. In *Proceedings of MICAI 2004. LNCS 2972, Springer*, Cd. de México, México, 2004.

[5] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2):113–135, 2001.

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

[7] A. De Battista, A. Pascal, G. Gutierrez, and N. Herrera. Búsquedas en bases de datos métricas-temporales. In *Actas del VIII Workshop de Investigadores en Ciencias de la Computación*, Buenos Aires, Agentina, 2006.

[8] A. De Battista, A. Pascal, G. Gutierrez, and N. Herrera. Un nuevo índice métrico-temporal: el historical-fhqt. In *Actas del XIII Congreso Argentino de Ciencias de la Computación*, Corrientes, Agentina, 2007.

[9] C. Jensen. A consensus glossary of temporal database concepts. *ACM SIGMOD Record*, 23(1):52–54, 1994.

[10] G. Navarro. Searching in metric spaces by spatial approximation. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, pages 141–148. IEEE CS Press, 1999.

[11] A. Pascal, A. De Battista, G. Gutierrez, and N. Herrera. Procesamiento de consultas métrico-temporales. In *XXIII Conferencia Latinoamericana de Informática*, pages 133–144, San José de Costa Rica, 2007.

[12] C. Ruano, E. Chávez, and N. Herrera. Discretización binaria para el ftrie. In *Actas del X Congreso Argentino de Ciencias de la Computación (CACIC'04)*, pages 100–111, Buenos Aires, Argentina, 2004.

[13] B. Salzberg and V. Tsotras. A comparison of access methods for temporal data. *ACM Computing Surveys*, 31(2), 1999.