Indexing Time-Series Databases for Inverse Queries *

Alexandros Nanopoulos Yannis Manolopoulos
Dept. of Informatics, Aristotle University, Thessaloniki 54006, Greece
tel: ++3031-996363, fax: ++3031-998419,
email: {*ananopou,manolopo*}*@athena.auth.gr*

Abstract. In this paper we examine the problem of indexing time sequences in order to answer inverse queries. An inverse query computes all the time points at which the sequence contains values equal to the query value. The presented method is based on [7] in order to represent each time sequence with a few ranges of values, which are in fact one dimensional minimum bounding rectangles. We compare the proposed method with the IP-index which has been presented in [8] as an indexing mechanism for answering inverse queries. As it is shown, the proposed method outperforms the IP-index for very large time sequences.

1 Introduction

Time series databases consist of discrete sequences representing the values of one or several variables as a function of time. This function may be discrete or continuous. In the case of a continuous function, the values of the sequence are produced by a sampling procedure and, if it is desirable, the intermediate values can be obtained through an interpolation function (linear, step-wise constant, splines).

Time sequences have found applications in temporal and historical databases [10] and in scientific databases which store several phenomena measurements. In data mining applications, time sequences are used in order to discover sequential patterns [3]. All these applications require the time sequences to be dynamically updated and that all the past values should be maintained.

The two basic categories of queries for time series data are the similarity queries and the retrieval queries. The first category has been already studied thoroughly [1, 7, 2]. Retrieval queries ask for the value at a particular time point, or for the range of values for a time span. These queries can be handled efficiently by several type of indexes [5].

Lin et al. [8] have proposed a method for answering inverse queries, i.e. queries that ask for the time points where the sequence contains values equal to the query value. Figure 1a illustrates an example of an inverse query. The approach of [8] is based on the association of each inserted value with all its covering segments.

* Work supported by the European Union's TMR program ("Chorochronos" project, contract number ERBFMRX-CT96-0056 (DG 12 - BDCN)).

G. Quirchmayr et al. (Eds.): DEXA'98, LNCS 1460, pp. 551–560, 1998.
© Springer-Verlag Berlin Heidelberg 1998

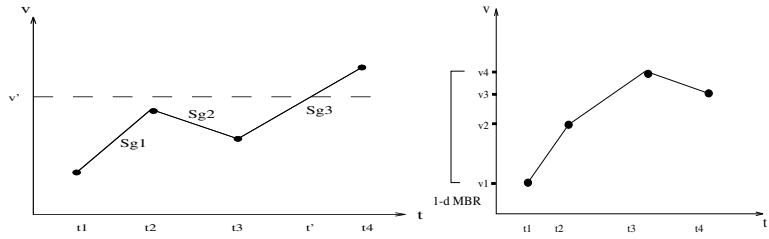

Fig. 1. a. An example of an inverse query : value v' is equal to the value at t'. **b.** An 1-d MBR of 4 successive values.

A segment is defined as the line segment that joins two consecutive values (in Figure 1a all segments are represented as Sg_i). In the same figure, we remark that the value at the time point t_2 is covered by the segment Sg_3 (a segment Sg_i covers a value v, if v is between the values of the bounding points of Sg_i). This indexing scheme is called IP-index, and compared to sequential scanning results in a performance improvement of more than two orders of magnitude.

As it is mentioned in [8], the insertion time and the space requirements are heavily depended on the number of covering segments which, as it is shown in the sequel, for very large time series grows rapidly, resulting in an increase in the above requirements. Therefore, for very large time sequences, which are stored in the secondary memory, we have to take into consideration the fact that successive values are likely to be covered by a large number of common segments. Thus, we propose the representation of successive values of the time sequence by their covering range of values, which is in fact their one dimensional minimum bounding rectangle (1-d MBR). Figure 1b presents a number of successive values and the corresponding MBR. The collection of the resulting MBRs can be efficiently stored in an R*-tree [4]. This idea was successfully used in [7], where the values of the Fourier Transformation of time series are indexed, in order to answer subsequence matching similarity queries. It will be shown that this approach correctly finds all the qualifying time points. As a result, we have to store a much smaller number of entries, yielding to reduced space overhead and processing costs as it is validated by the experimental results.

The rest of this paper is organized as follows. In Section 2 we present a description of the IP-index and the way it can be implemented in secondary storage. Section 3 presents the proposed approach and Section 4 describes the experimental evaluation of both approaches. We summarize in Section 5 by giving some brief conclusions.

2 Related Work and Motivation

In this section, we present the implementation of the IP-index [8] for secondary storage. Lin et al. have proposed that for time sequences of one value, each pair of a time point t_i and a value v_i (a point in the two-dimensional t-v plane)

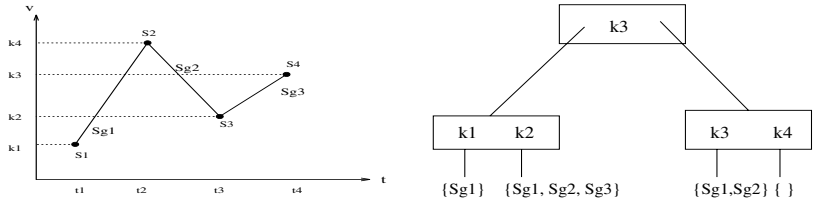

Fig. 2. a. A sequence of 4 time points and the projections of the values on the v-axis. **b.** The corresponding B^+-tree and the attached segment lists.

defines a state S_i. Two consecutive states S_i and S_{i+1} define a line segment Sg_i. Then the problem of answering an inverse query for a value v' is equivalent to finding all the segments Sg_i that are intersected by the line $v = v'$ (see Figure 1a). The corresponding time points can be found by applying any interpolation function (e.g. linear interpolation is used in the example of Figure 1a).

In order to answer an inverse query without scanning the hole time sequence, each line segment Sg_i is projected on the v-axis. The collection of all the projections for all states S_i forms a set of non-overlapping intervals $[k_j, k_{j+1})$ (see Figure 2a). Each k_j corresponds to one (more than one in case of equal values) value v_i. Then, all the values that belong in the interval $[k_j, k_{j+1})$ are covered by the same segments. The IP-index associates each interval $[k_j, k_{j+1})$ with the corresponding list of covering segments $< Sg_i >$. Since intervals are successive, each interval can be represented only with its starting value, i.e. k_j.

In [8], the IP-index was implemented in main memory by using an AVL-tree. For secondary storage, the IP-index has to be implemented as a B^+-tree, where every leaf node entry contains a pointer to its corresponding segment list. Figure 2b gives the resulting B^+-tree for the example of Figure 2a. To answer an inverse query for a value v' using the IP-index, we have to search the B^+-tree (or any other implementation) to find the interval $[k_j, k_{j+1})$ that v' belongs to. This is achieved by finding the maximum value k_j that is less than or equal to v'. Then, the list of the covering segments for that interval, contains all the segments that intersect the line $v = v'$.

The performance of the IP-index method is affected by the total number of entries and especially the number of covering segments, which in very large sequences tend to increase rapidly. In [8], this problem is addressed by limiting the precision of the measured values. This results into the reduction of the number of index entries due to the fact that several time points will have the same value for the reduced precision, while this is not the case of the original precision. Apparently, the reduction of the precision is application depended and cannot be applied in general.

Since the IP-index has been proposed as a main memory index, the search time is linear with respect to the length of the segment lists. However, for secondary storage, extra disk access cost is required to traverse these lists. For the implementation of those lists in secondary storage, we follow the approach proposed in [6], where the problem of the efficient handling of posting records

in a B^+-tree is examined. With this approach, we have a number of covering segments stored in consecutive positions in secondary memory. These segments are stored in buckets, the contents of which one can be retrieved with one logical disk access. The cost measures that are considered in [6] for posting lists are similar with those ones in the case of segment lists. More precisely, we have to take into account the fragmentation which occurs when we choose a large bucket size and the increased search time when we choose a small bucket size. We have to note that we use a constant bucket size. Although the other method described in [6] achieves improved search time, it requires total reorganization of the bucket entries. This would had a great impact on the insertion time due to the large number of bucket entries which the IP-index requires.

3 Proposed Method

The problem of answering inverse queries translates to finding a way to index the values of the time sequence in order to discard irrelevant parts of the sequence and search only the ones that contain values satisfying the query condition and thus, avoiding the straightforward solution to search the entire sequence for finding the time points which answer the query.

As it has been mentioned previously, the IP-index divides the set of all values into intervals of values which contain values that are all covered by the same segments. This method gains at instances of time sequences that contain the same values at several time points. As it is mentioned in [8], the IP-index performs extremely well for periodic time sequences with values of reduced precision. However, in most cases, time sequences are not periodic and may contain values of high precision. This fact leads to a point where we have to store as many intervals as the number of points in the time sequence. Moreover, the problem is enhanced due to the fact that several intervals are covered by the same segments. As a consequence, there is a large number of duplicates stored in the segments lists. Since, as it will be shown later, for very large time series the overlapping between covering segments is also large, the IP-index leads to a large space overhead. This also affects the insertion and search times.

The inefficiency in terms of space and insertion time stems from the fact that every interval is explicitly stored. The proposed method is based on the property that consecutive time points will probably have similar values. This approach was followed in [7], in order to answer subsequence similarity queries. Therefore, we can divide the sequence into subsequences of consecutive time points that do not have values with large differences. The collection of these points corresponds to a range of values and all included values can be represented only with that range. An inverse query has to retrieve only the ranges that contain values satisfying the query condition. It is easy to show that if an inverse query intersects some of the values inside the range then it has to intersect the range itself. The time points of all the subsequences which take values inside the retrieved ranges have to be computed by examing the contents of each subsequence. As it will be shown in the sequel, the proposed method correctly finds all the time points which

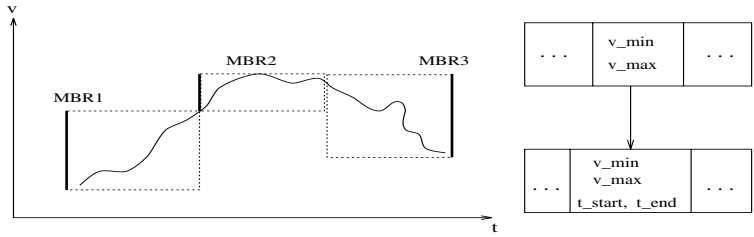

Fig. 3. a. A sequence divided into 3 parts and the corresponding MBRs. **b.** The structure of an internal and of a leaf node of the R*-tree.

satisfy the query condition. Such a range of values represents the one dimensional minimum bounding rectangle (MBR) of the values it includes. Therefore, the proposed index method can be implemented with an 1-d R*-tree, storing these MBRs.

Figure 3a presents an example of a time sequence which is divided into three subsequences. For each one of them the corresponding 1-d MBR is depicted with a solid line (for illustration reasons each subsequence is bounded by a rectangle). In the same figure it is shown that different MBRs may have overlapped projections on the v-axis. Notice that the R*-tree has to store at the leaf nodes the starting time and the ending time of each subsequence. Figure 3b illustrates the structure of an internal and an external node of the R*-tree.

3.1 Insertion

Following the previously described approach, we have to define a way such that several consecutive values will be included in the same MBR, forming one subsequence. As it is mentioned in [7], the criterion whether a value will belong to a subsequence or not is based on the fact that it does not increase the probability that the corresponding MBR is going to be accessed. As it can be easily shown for data normalized to the range $[0, 1]$, the probability P that an MBR (1-d in our case) of length L will be accessed by the average range query is equal to $P = L + 0.5$ (it is assumed that an average range query is of length equal to 0.5). Also, notice that this criterion requires that the values are normalized to the range $[0, 1]$. In [7] for an MBR consisting of k values the marginal cost mc of this MBR is defined as $mc = (L + 0.5)/k$ which means that the probability of accessing the MBR is divided by the number of values that this MBR includes.

Hence, the criterion to include (or not) a new value in a subsequence is based on the comparison of the new marginal cost mc_a of the MBR produced in case of insertion of the new value into the specific subsequence, with the marginal cost mc_b of the MBR before the insertion. More precisely if the MBR length before the insertion was L_b, whereas after the insertion it changes to L_a (notice that L_a may be equal to L_b if there is no expansion), then we have to test if $L_a/(k+1) \leq L_b/k$

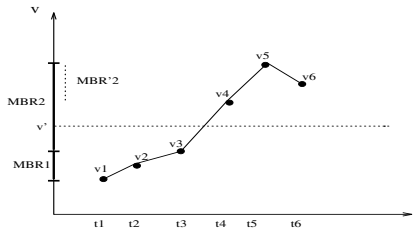

Fig. 4. MBR_2 is involved in inverse query v' while MBR'_2 (with dotted line) does not.

Therefore, we start with the first value of the whole sequence and we form a subsequence by including all the successive values, as long as these values preserve the above inequality. In case that the inequality does not hold, we have to start a new subsequence. At this point, our approach differs from that in [7] in order to include a special case. Figure 4 illustrates this case. In this figure values v_1, v_2 and v_3 belong to the same subsequence and they are represented by their 1-d minimum bounding rectangle MBR_1. Value v_4 is such that it has to be excluded from the previous subsequence and to start a new subsequence. In this new subsequence, values v_5 and v_6 will also be included. In order to be able to answer correctly an inverse query with a value v' such that $v_3 < v' < v_4$, we have to define as starting value of the second subsequence the value v_3 and not v_4. As a consequence, the 1-d minimum bounding rectangle MBR_2 of the second subsequence, as it is drawn in Figure 4, will be intersected by values of inverse queries like the value v'. Otherwise, if we started the new subsequence from the value v_4 (forming the MBR'_2) then the indexing method could not guarantee safe satisfaction of inverse queries. Finally, notice that the proposed method supports the dynamic appending of new values at the end of the sequence, since these new values can be the starting points of new subsequences.

3.2 Search

Here, we examine how we have to search the index for the time points that correspond to an inverse query. First, for a given inverse query that asks for all the time points that the time sequence takes values equal to v', we perform a point query within the R*-tree to retrieve all the leaf MBRs that totally enclose the value v'. The retrieved MBRs, as it was described before, represent ranges of values in subsequences of the original sequence. The fact that an MBR encloses the value v' means that the corresponding subsequence includes values, where some of them are equal to v'.

In the sequel we have to examine all the subsequences that are represented with an MBR that has been retrieved in the previous step. Thus, we perform sequential scanning, but only inside each subsequence, in order to find the exact time points that the time sequence takes values equal to v'. As it has been

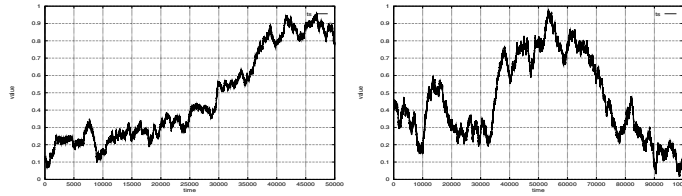

Fig. 5. Synthetic Time Sequences

described, the boundaries of each subsequence, i.e. its starting and its finishing time, are stored within the corresponding MBR. This way for each query we can quickly select only the interesting subsequences, and then we can exactly answer the query by searching only inside these subsequences.

3.3 Correctness

The following lemma proves the correctness of the proposed method.

Lemma 1. *The proposed method correctly finds all the time points that the time sequence contains values equal to the query value.*

Proof. Let an inverse query of value v'. Then for any interval of consecutive values $[v_1, v_2]$ which satisfies the condition : $v_1 \leq v' \leq v_2$, values v_1 and v_2 will either belong to the same or to consecutive subsequences. In the first case, the MBR that represents the subsequence they belong to, will enclose that interval and thus, the MBR will be retrieved by the query. In the second case, as it was explained in subsection 3.1, the MBR of the second subsequence will enclose the interval $[v_1, v_2]$ and so it will also be retrieved by the query. Since these cases are the only ones, the proposed method correctly answers inverse queries. □

4 Performance Evaluation

We implemented both the IP-index structure and the proposed one in C++ and we carried out experiments on a Ultra Sparc under Solaris 2.5.1. We used synthetic data of various lengths and characteristics. According to [8], the IP-index is by far better than the sequential method, henceforth we do not study the sequential scanning any more. We measured the performance of the IP-index and the proposed method, which we will refer to as SIQ-index (denoting indexing Sequences for Inverse Queries). More precisely we performed measurements of space overhead, insertion time and search time.

The synthetic data were modeled as *random walks* [9]. A random walk, also called brown noise, can be generated by using the following formula [1]:

$$x_t = x_{t-1} + z_t$$

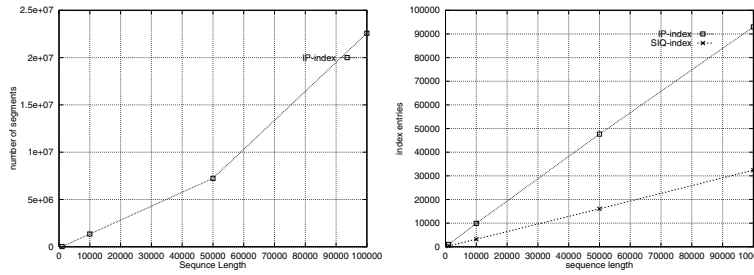

Fig. 6. a. Number of segments IP-stores versus sequence length. **b.** Number of index entries versus sequence length.

where z_t, $t = 0, 1, \ldots$ are IID random variables following uniform distribution. Figure 5 shows examples of the produced synthetic time sequences. These sequences have lengths equal to 50000 and 100000 respectively. Each value was represented with a float number and no limit to the precision was applied.

The page size for the B$^+$-tree and R*-tree access methods is equal to 4K. Hence, the B$^+$-tree has a larger fan-out because, as it was explained earlier, the SIQ-index requires some additional information to be stored in the leaf nodes of the R*-tree. On the other hand, for the implementation of the segment lists for the IP-index, we chose the buckets containing the elements of the lists (the covering segments) to be of constant size equal to 1K. As it is mentioned in [8], each segment is represented with only one integer denoting the time point of its staring value. Note that if we chose a larger bucket size, e.g. 4K, then this would result in severe fragmentation and, as a consequence, in large space waste. In all our experiments, a bucket of 1K was large enough to hold the average number of entries of the segment list for each entry in the B$^+$-tree.

4.1 Space overhead

We first measured the number of covering segments that the IP-index requires to store. This is the total number of covering segments for all index entries, including the duplicates which the IP-index requires to store. Figure 6a gives the results with respect to the time sequence length. As it is illustrated, the number of covering segments grows rapidly with respect to the size of the sequences.

For the same sequences, we measured the number of index entries for the IP and SIQ indexes. Since the IP-index requires the storage of almost every individual value, whereas the SIQ-index stores only ranges which include more than one values, the SIQ-index has much less index entries. Figure 6b illustrates the number of entries for each index. Therefore, the SIQ-index requires much less space overhead since it stores less index entries and moreover, it does not require the storage of the covering segments.

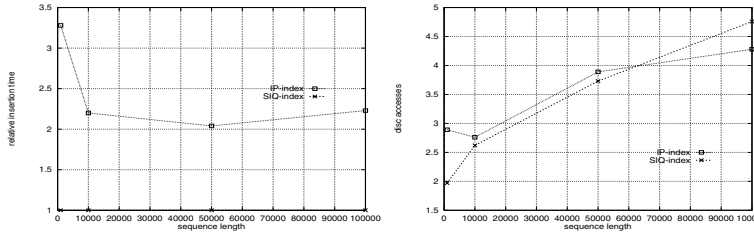

Fig. 7. a. Relative insertion time versus sequence length. **b.** Required disk accesses versus sequence length for the search operation.

4.2 Insertion time

Next we measured the insertion time for both indexing methods. Figure 7a illustrates the results with respect to the sequence length. In Figure 7a, the insertion time of the IP-index is given relatively to the insertion time of the SIQ-index. As it is clearly shown by these results, the SIQ-index outperforms the IP-index due to the fact that it involves the insertion of a much smaller number of entries and moreover, it does not require the manipulation of the covering segments that the IP-index does.

4.3 Search time

Finally, we measured the search time which both methods require in order to answer inverse queries. We carried out the measurement by producing 100 random query values. As a cost measure we choose the number of required disk accesses. Since both the IP-index and SIQ-index involve a post-processing phase (the former in order to do the interpolation and the latter in order to discard all intervals that are not intersected by the query value), this measure does not include the post-processing time which is considered to be negligible since it is a main memory operation.

Figure 7b illustrates the results of this measurement (the vertical axis depicts the average disk accesses for one query). As it shown in this figure, in most cases the SIQ-index is slightly better than the IP-index, except some cases where IP-index performs a little better. This is due to the fact that in these cases, the overlapping between the MBRs is quite large, so the R*-tree requires more than one leaf MBR to be searched in order to answer the query. On the other hand, the IP-index can answer the query by accessing only one leaf node because it divides all values into non-overlapping intervals. The price paid for this is a very large space overhead.

5 Conclusions

We have presented a method to index time series databases in order to efficiently answer inverse queries. This method is based on [7], where the problem of indexing time series databases for subsequence similarity queries is examined. The

proposed method represents successive values of the sequence with their range of values which is in fact an one dimensional minimum bounding rectangle (MBR). Therefore, the collection of these MBRs can be stored in an R*-tree.

The proposed method is compared with the IP-index [8] which organizes a time sequence by keeping track of each individual interval of consecutive values. As it was shown, the proposed method achieves clearly a much less space overhead and better insertion times. At the same time, the approximation of time values by ranges of values does not lead to worse search time.

Future work may include the examination of the proposed method for indexing collections of time sequences. Also, one other interesting topic is the examination of the method for time sequences of more than one dimensions. Finally, future research could involve the generalization of the proposed method for queries which find the intersection points between time sequences.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. "Efficient Similarity Search in Sequence Databases". *Proc. of the Fourth International Conference on Foundations of Data Organization and Algorithms"*, Chicago, USA, October 1993.

[2] R. Agrawal, K. Lin, H. Sawhney, and K. Shim. "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases". *Proc. of the 21st VLDB Conference*, Zurich, Switzerland, 1995.

[3] R. Agrawal, and R. Srikant. "Mining Sequential Patterns". *Proc. of the 11th Int'l Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles". *Proc. of the ACM SIGMOD Conference on Management of Data*, Atlantic City, USA, May, 1990.

[5] R. Elmasri, G. Wuu, and Y. Kim. "The Time Index: An access structure for temporal data". *Proc. of the 16th VLDB Conference*, Brisbane, Australia, 1990.

[6] C. Faloutsos and H. Jagadish. "On B-tree Indices for Skewed Distributions". *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.

[7] C. Faloutsos, M. Raganathan, and Y. Manolopoulos. "Fast Subsequence Matching in Time-Series Databases". *Proc. of the ACM SIGMOD Conference on Management of Data*, Minnesota, USA, May 1994.

[8] L. Lin, T. Risch, M. Skold, and D. Badal. "Indexing Values of Time Sequences". *Proc. of the Int'l Conference on Information and Knowledge Management (CIKM)*, Rockville, USA, 1996.

[9] M. Schroeder. "Fractal, Chaos, Power Laws : Minutes from an Infinite Paradise". *W.H. Freeman and Company.* New York, USA, 1991.

[10] R. Stam and R. Snodgrass. "A bibliography on temporal databases". *IEEE Bulletin on Data Engineering, 11(4)*, December 1998.