

CS 860 Topics in Database Systems
Prof: David Toman
Stud: Feng Xue
ID: 00418997

Efficient Similarity Search In Sequence Databases

-Paper Presentation Summary

Background Information

In order to understand the concepts and methods proposed in this paper, it would be a good idea to get some background knowledge regarding the following two topics:

1. Discrete Fourier Transform (DFT)

In mathematics, the Discrete Fourier Transform (DFT), which is also called the finite Fourier transform is a Fourier transform widely employed in signal processing and related fields to analyze the frequencies contained in a sample signal, solve partial differential equations, and to perform other operations such as convolutions. The DFT can be computed efficiently using a fast Fourier transform algorithm within $O(n \log n)$ time. The algorithm takes a sequence N of complex numbers and transform them into another sequence of N complex numbers using formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N - 1$$

Where e is the base of the natural logarithm and i is the imaginary unit ($i^2 = -1$).

2. R* tree

R* tree basically is a R tree with some variations for indexing spatial information. R tree is tree data structure that are similar to B-tree, but are used for spatial access methods. The fundamental difference between R* tree and R tree is that R* tree uses a combination of a revised node split algorithm and the concept of forced reinsertion at node overflow. R* tree attempts to reduce both coverage and overlap, and is likely to achieve significant improvement over other R tree variants. It's able to support both point and spatial data at the same time.

Problem Generalization

While other people are trying to design languages to query time-sequenced data and to develop access structures to efficiently process such queries, authors of this paper

come up with the question of how to successfully enhance the database in order to process “similarity” queries. They argued the value of this new functionality by presenting possible examples of the queries over sequence databases: “Identify companies with similar pattern of growth”, “Determine products with similar selling patterns”, etc.

In order to be able to process these queries, two generalized query types have to be solved: “Range Query” and “All-Pairs Query”. For Range Query, it means that given a query sequence, find similar sequences in the database; and for All-Pairs Query, it means that given N sequences, find the pair of sequences in the database that are similar to each other.

Direct Issues

We now observe two direct issues that we need to solve before we can move forward. One is how to index features from all sequences, and the other one is how to determine two sequences are similar.

Facing the first issue, R* tree is used as the multi-dimensional indexing method. We extract k features from every sequence, mapping it to k-dimensional space, and then using the R* tree to store and search these points.

The similarity measure is application-dependent. In this paper, the Euclidean distance is used to provide easy calculation. To express the formula in text format, it is the square root of the sum of squared differences.

It seems like we now have the power to process the similarity queries. We just need to follow the steps described below:

1. Index k features of each time sequence in a multi-dimensional space using R* tree
2. For a range query, obtain the corresponding point in the k-dimensional space and use the index to retrieve all sequences that are at most ϵ distance away from the query sequence

3. For an all-pair query, do a spatial join using the index. The result of the join will be the answer set

And hopefully we can have the result right away. Sadly, there're two other subtle, but important issues we need to solve before we can accomplish our task.

Subtle Issues

These two issues actually all come from step 1 from the proposed algorithm. During the mapping of each sequence into the corresponding multi-dimensional space, we could encounter the following issues.

1. Completeness of feature extraction: How to extract features so that we won't miss any qualifying object during search
2. Dimensionality curse: How to handle the extremely high running time when the dimensions of features extracted from each sequence is high

Apparently, with the algorithm proposed in previous section, it seems like we would definitely run into these two issues. In order to guarantee the completeness of feature extraction, we have to make sure that the Euclidean distance in the k -dimensional space is less than or equal to the real distance between the two objects. R^* tree works well for up to 20 dimensions, and we have to figure out a way to make sure the feature extraction would use limited number (less than 20) of features to distinguish each sequences.

A new approach with DFT

Now is the time to introduce the Discrete Fourier Transform into our algorithm. Since our data set would be a bunch of time sequences, the DFT would thus transform them from the time domain into frequency domain. As mentioned in the background section that DFT has a lot of nice features. It is easy and fast to compute. It preserves the distance between two objects since it's an orthonormal transform. This is also backed up by Parsval's Theorem. It also concentrates the energy of the signal in few coefficients.

Parsval's Theorem:
$$\sum_{t=0}^{n-1} |x_t|^2 = \sum_{f=0}^{n-1} |X_f|^2$$

All the x are from the original domain, and all the X are in the frequency domain. This theorem guarantees that the energy in the original domain is the same as the one in the frequency domain.

Parsval's Theorem also gives:
$$\| \vec{x} - \vec{y} \|^2 \equiv \| \vec{X} - \vec{Y} \|^2$$

This formula implies that the Euclidean distance between two signals in the time domain is the same as their Euclidean distance in the frequency domain.

With all the features of DFT, we could finally solve the two subtle issues:

1. Solution to "Completeness of feature extraction"

With the Parseval's theorem, it guarantees the distance between two objects (sequences in this case) in the frequency domain is the same as the distance between them in the time domain. Thus it satisfies the requirement to guarantee the completeness of feature extraction.

2. Solution to "Dimensionality curse"

Since R* tree only works well for up to 20 dimensions, we have to find a feature extraction that uses limited number of features. As one attribute of Discrete Fourier Transform, the result sequence exhibit strong amplitudes for the first few frequencies. Thus, we could use the first few frequencies which will help us to avoid the dimensionality problem without introducing too many false hits. This is backed up with the experiment which compares the 1/f line between a Fourier coefficients generated by a manual random walk and Fourier coefficients generated with real Swiss-franc exchange rate. The result of this experiment demonstrated that the first few coefficients of the Fourier transform are far more important than the rest ones which allows us to use it as the approximation in the indexing method. Since we only use the first few coefficients, this will introduce a few false hits, but luckily it would not cause any "false dismissal" since the less constraint transformation would actually return a superset of the answer set.

Final Algorithm

After solving the outstanding two issues, we now have the final algorithm ready:

1. Obtain the coefficients of the Discrete Fourier Transforms of each sequence in the database
2. Construct $2*fc$ dimensional index with the first fc result coefficients using R* tree. Each sequence becomes a point in that space
3. For a range query, obtain the first fc Fourier coefficients of the query sequence, and then find matching sets that are at most ϵ distance away
4. For an all-pairs query, we do a spatial join using the index
5. No matter which type of query it is, we need to re-compute the distance in time-domain with result set to eliminate any false hits

Performance Experiments

As we can see from the algorithm that The fc variable has to be inputted manually. High fc reduce false hits, but increase dimensionality or R* tree which results in longer running time. Therefore, the easy way to determine the optimal fc is through some experiments. Also, we want to test if the algorithm we designed is going to meet our goal or not. Having these in mind, the paper targets at the following objectives during the performance experiments:

1. Determine fc
2. Figure out the relation between the search time of proposed algorithm and total sequences in database
3. Figure out the relation between the search time of proposed algorithm and sequence length

The experiments would test four values of fc which are 1, 2, 3 and 4 to determine the optimal value. To figure out the relation between the search time and total sequences, it would test one chosen fc on databases with 50, 100, 200, and 400 sequences. The experiment would also be set to test the various lengths of sequences with values of 256,

512, 1024, and 2048 for fixed fc . All tests will be conducted with both types of queries (range and all-pairs) and all the comparison tests (for objective 2 and 3) are conducted between the proposed algorithm and the sequential scanning algorithm.

Experiment Result

From the figures in the paper, we can see that $fc=2$ achieves the lowest running time for both types of queries. The proposed method demonstrated a significantly gain in performance over the traditional sequential scanning algorithm. With the number of sequences or length of sequences increase, the gain of the performance increases.

Conclusion and Future work

This paper presents us with a brand-new idea of using the DFT to extract the features from a sequence. The DFT successfully reserves the distance, overcomes the dimensional problem and finishes the feature extraction with fast processing time. This paper also shows us the consistent performance with the R^* tree being used with medium dimensionalities.

On the down side, this paper provides little information regarding the implementation detail since it doesn't mention anything about the type of database or hardware system used. During the comparison experiments, it only presents the result with the sequential scanning, which is pretty much the worst possible method. Also, the pattern matching problem considered in this paper is whole matching. It would have a lot more practical use if they could expand this into a subsequence matching method. In the future, we could also apply the same technique into finding the similar data streams. Since the method is fast enough, it would not impose any timing issue when we're trying to compute on the fly.