NTNU
Det skapende universitet

Metric Indexing in Time Series

Espen Ekornes Rekdal

Master i datateknikk
Oppgaven levert: Juni 2008
Hovedveileder: Magnus Lie Hetland, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Oppgavetekst

Determine the application of metric indexing methods for similarity retrieval on high dimensional data, in this case time series. As opposed to the traditional approach, where the data is mapped on to lower dimensional signatures. Specifically determine which kind of data sets metric indexing will outperform the traditional approach. The main suspicion is that metric indexing will achieve better results on data sets with high intrinsic dimensionality, while mapping does better at data sets with low intrinsic dimensionality.

Oppgaven gitt: 25. januar 2008
Hovedveileder: Magnus Lie Hetland, IDI

Metric Indexing of Time Series

Espen Ekornes Rekdal

June 6, 2008

Abstract

Background: Similarity retrieval has seen massive growth of interest lately. Doing this on high dimensional data, while maintaining the dimensionality is difficult, but has its uses in areas like classification and data mining. Metric indexing does not deal with the actual dimensionality of the data, only the distances between the data objects, and thus maintains the actual dimensionality of the data.

Objectives: Determine the application of metric indexing methods for similarity retrieval on high dimensional data, in this case time series. As opposed to the traditional approach, where the data is mapped on to lower dimensional signatures. Specifically determine which kind of data sets metric indexing will outperform the traditional approach. The main suspicion is that metric indexing will achieve better results on data sets with high intrinsic dimensionality, while mapping does better at data sets with low intrinsic dimensionality.

Methods: Two metric indexing methods will be implemented. One based on pivot-space filtering (LAESA) and the other based on compact partitioning (LC). These methods' performance will be measured in performance by the number of actual distance calculations done. Several real world data sets will be used as testing material. The results will then be compared with the results gained from doing piecewise aggregate approximation signature extraction on the same data sets.

Contents

1 Introduction **3**
1.1 Applications 3

2 Background **4**
2.1 Basic Concepts 4
2.2 Time Series 5
2.3 Dimensionality Reduction 6
2.3.1 Decompositions 6
2.3.2 Piecewise Aggregate Approximations 7
2.4 Metric Indexing Methods 7
2.4.1 LAESA 8
2.4.2 List of Cluster (LC) 9

3 Implementation **11**
3.1 Environment 11
3.2 Framework 11
3.3 Brute Force 12
3.4 PCA 12
3.5 LC 13
3.6 LAESA 14
3.7 Benchmark 14

4 Results and Findings **16**
4.1 The Time Series 16
4.2 The Queries 18
4.3 The Results 19
4.4 Summary and Further Work 23

1 Introduction

This thesis will compare the performance of *Metric Indexing* methods compared to the traditional *Dimensionality Reduction* methods for proximity searching in *Time Series* data.

Dimensionality reduction in essence is the removal of details resulting in smaller amount of data to be processed at the cost of accuracy. Metric indexing uses only the distance between the data not the actual data, thus making the size of the data irrelevant.

1.1 Applications

The purpose here is to determine if metric indexing methods can provide better performance for searching in high dimensionality data. This could lead to improvements in such areas as data mining and classification; text and information retrieval; searching in DNA strings; function prediction, i.e. predicting the value of stock, and many other fields where effective and efficient searching in large amounts of complex data is necessary.

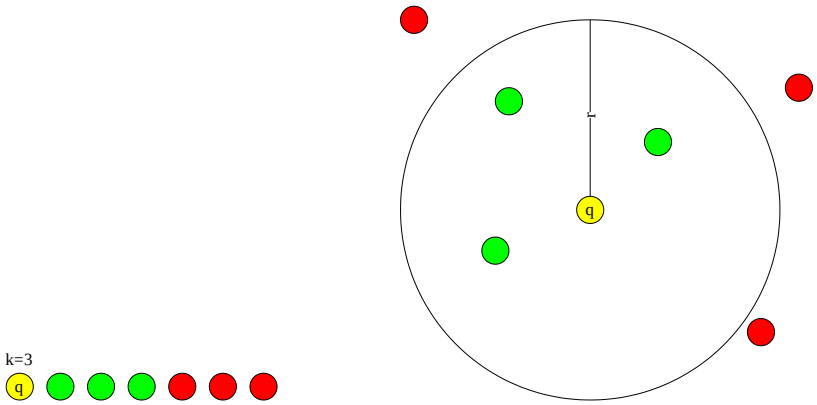
2 Background

This section will introduce the basic building blocks for this project. Firstly some basic concepts, later followed by an introduction to the indexing methods, both metric and signature based, that will be implemented for testing.

2.1 Basic Concepts

First off are some of the basic concepts needed to give a decent foundation for this project. First there is the proximity search.

There are two basic types of proximity searches. *Range queries* and *Nearest Neighbour queries*. Each of these searches takes two variables as input. Both of them take a query object, q. This is the center of the query, and the results are the objects which are similar to q. q is not necessarily an object in the database. In addition, the Range query takes a range, r. The result of a range query is all objects within r distance of q. Nearest Neighbour takes a number k and results in the k closest objects to q. Nearest Neighbour query is often called KNN for short.



(a) K-Nearest Neighbour (b) Range Query

Figure 1: Proximity Searches

Several properties that a good indexing scheme should fulfill are presented in [5] and extended further in [10]. These are:

1. Speed. The method should at least be faster then sequential search, otherwise it would serve no practical purpose.
2. Correctness. The method should not result in any *false dismissals*. (*False alarms* are acceptable, as these can be easily detected and removed.)
3. Size. The method should only require a reasonable amount of overhead.
4. Dynamic. Changes, i.e. insertions and deletion, in the data, should not require the index to be reconstructed from scratch.
5. Query Size. The method should be able to handle both data and queries of varying length.

6. Time. The index should be constructable within reasonable time.

7. Distance measures. The indexing method should be independent of distance measure.

Spatial indexing methods, like R-trees, suffer the so called *Curse of Dimensionality*. Originally conceived by Richard E. Bellman, the dimensionality curse is simply the fact that for each dimension added to a mathematical space, it's size grows exponentially. As a result, most spatial indexing methods will be crippled when the dimensionality of the space approaches high numbers. [2] showed that most methods loses performance after 8 to 12 dimensions, both in the time needed and the space required to construct the index. This makes most spatial methods useless for time series, as this form of data often has as many as 1000 dimensions, if not more.

Distance Measures, simply a method to measure the distance between two data objects, and in essence defining the metric space. The most commonly used and most known distance measure is the *Euclidean* distance, or the 2-norm distance viewed from the more general *Minkowski* distance. In layman terms, the euclidean distance is the distance you would measure with a ruler between two points on a piece of paper.

This simple distance measure does not work in all cases. For some data, the Euclidean distance can be unintuitive, and for where transformations like scaling, translation or warping is required, it fails [7]. Also, some domains might require tailored distance measure. There are some more robust measures, but Euclidean distance will suffice for this project.

2.2 Time Series

A time series is a sequence of points measured at some (usually uniform) interval. A time series is usually represented by a list of pair values (x, y), where x is the time and y is the value at that time. In cases where the time interval between values is uniform, the x can be neglected as it is implicitly known. Time series are usually very long, usually 1000 data points per time series, sometimes 10000 or even higher magnitudes. There are also shorter time series as short as 10 or 20 points, but handling these become rather trivial, so the focus will be on time series that are 1000 or more points long, as these make the normal spatial indexing methods break down, i.e. require an unreasonable amount of time and/or space.

Some typical examples of time series are:

- Stock data, typically the stock value of one or more companies over time.
- Seismic data, movements in the earth's crust recorded by a seismic sensor.
- Weather data: Temperature, air pressure, rainfall or humidity etc. recorded by a weather station or similar apparatus.
- Cardiology data, the activity of the human heart.
- Sunspot data, the measurement of solar activity.
- Sound and music data. Sounds are usually represented as amplitude over time, and music can also be considered as time series by assigning a value to each note.

- DNA, is also similar in structure to time series.

In general anything that can be sampled over time, or represented as such (list of (x, y) data pairs. x could for instance be distance instead of time or some other dimension).

One important aspect to note is how time series are matched in regard to length. That is, is the whole sequence of interest or are the subsequences interesting. With subsequence matching, the query is shorter sequence than the data objects. The query sequence is tested against all parts of a data sequence to find the best fitting match, like sliding a window the same length as the query sequence across the data sequence. A common example of this would be work done with DNA, where the subsequences and similar subsequences across different data sequences are of great interest. In respect to this project only whole sequences will be matched. All data sets used will contain equal length sequences, and the queries performed on these will also be of the same length. This is mainly to keep things simple, and thus reducing the possibilities of errors introduced by added complexity.

Another interesting feature of time series, is the correlation between the data values. Note however, that this is not necessarily true for all kinds of time series data, but is in most cases. Take for example a time series representing the speed of a automobile, sampled at 1 seconds intervals. The values will always be dependent on the previous value, as there is a limit to how much the automobiles speed can change in the course of 1 second, and will therefore not jump wildly up and down. This is mentioned as it lends an advantage to piecewise aggregate methods, with which the metric indexing methods will be compared.

2.3 Dimensionality Reduction

Here we will cover some of the basic methods of dimensionality reduction.

2.3.1 Decompositions

Spectral decomposition, also know as the *Discrete Fourier Transform*, is commonly used in the fields of signal analysis and image processing, and is a very well documented technique. The essence of the discrete Fourier transform is that any signal can be represented by the combination of a finite number of sine (or cosine) waves [12]. Each wave is represented by a complex number called the *Fourier coefficient*, and the data when transformed is often said to be in the frequency domain. A time series can easily be considered a signal for the purpose of Fourier transformation. The dimensionality reduction, or in this case, data reduction, comes from the fact that some frequencies in the transformed version contribute very little to the original "signal" and can be discarded with little loss, a "Frequency reduction". Parseval's law implies that the Euclidian distance is preserved in the frequency domain, and that the removal of coefficients is guaranteed to be an underestimate of the original, satisfying the lower bounding lemma. It should be noted that this might not be true for other distance measures.

Another alternative, somewhat similar to the Fourier approach, is the *Wavelet decomposition*. Instead of being a sum of waves, as with the Fourier, wavelets are mathematical functions, and the sum and difference of these. The main difference being that the wavelets are localized in time, as opposed to frequency.

Work has been done to try and utilize wavelet decomposition for indexing, see [6], but [10] notes that the Piecewise Aggregate Approximation method is inherently better, more about this method later.

Singular Value decomposition should also be mentioned, as it has the novelty of being a global method as opposed to the above two which are strictly local methods. Local in the sense that they consider one data object exclusively at a time, while this global approach considers the data set as a whole. This method is implemented and tested in [10] where they use the approach presented in [11].

2.3.2 Piecewise Aggregate Approximations

Piecewise Constant Approximation (PCA) is the method that will be used for comparison with the metric indexing methods. Probably the simplest and most intuitive signature methods, but despite is shown to have one of the best performances, especially on time series [10]. The basic principal behind this method is simply: Divide the data into n equal length parts, and calculate the mean of each part. The signature is simply these n means.

There is also a variant to PCA, APCA or Adaptive Piecewise Constant Approximation [9]. The difference is that the parts no longer need to be of equal length. That is to say, if there exist in the data a long flat area with little change, the entire area can be summed up into one mean, while other parts with many changes and high variance can be covered by several means. This results in an increase in both accuracy (complex areas can be covered in more detail), and compression (non-changing areas can be covered by a single value). The drawback is that this method requires special distance methods to work. Two have been suggested, one which guarantees to underestimate the Euclidian distance, and one more efficient one which does not guarantee that and can produce false dismissals.

Another variant is Piecewise Linear Approximation where instead of representing each area with a single mean value, it is represented by a linear approximation ($aX + b$). This results in even better accuracy and compression possibilities, but at the cost of having to use even more complex distance measures.

For this project PCA will be used for its simplicity.

2.4 Metric Indexing Methods

Metric space is defined solely by the distance between the objects it contains. This distance can be based on any given distance function fulfilling the metric requirements:

- The distance can not be negative. (non-negativity)
- The distance between two objects can not be 0 unless the objects are identical. (identity of indiscernibles)
- The distance from object A to object B is the same as the distance from B to A. (symmetry)
- The distance from A to C is greater or equal to the sum of the distances from A to B and B to C. (triangle inequality)

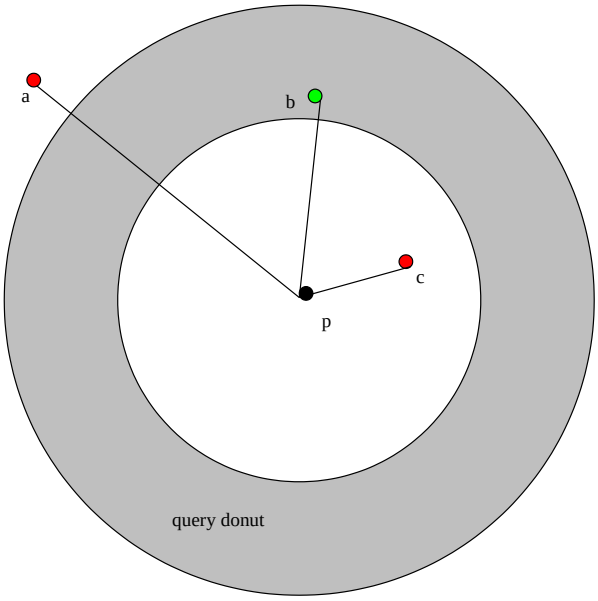


Figure 2: LAESA filtering

Metric indexing methods can be divided into two main approaches [4]. *Compact Partitioning* methods and *Pivot-based* methods.

Partitioning indexing methods partition the space into spatial zones. Compactness of the zones are important. Whole zones can be discarded by a single distance calculation between the query object and, in most cases, the zone centroid.

Pivot-based indexing methods utilize a number of pivots and compare the distances between these pivots and the data objects with the distance from the pivots to the query object. More pivots provides usually provides better performance, but at the cost of space.

These two metric indexing methods that will be implemented and benchmarked in this project. LAESA and LC, below we will examine them in greater detail.

2.4.1 LAESA

Based on AESA by Vidal, E. AESA uses the pre-calculated distance between every object in the data set to every other object in the data set to speed up searches. For any reasonably sized data sets, this becomes unfeasible. LAESA was developed to counter this. Instead of using the distance between every object, LAESA compromises by using a subset, the pivots, and pre-calculates the distance between these objects and every other object. The construction cost is then limited to kn, where k is the number of pivots and n is the number of objects. Instead of n^2 that AESA results in. As can be gathered, LAESA is a pivot based method.

2.4.2 List of Cluster (LC)

The List of Clusters presented in [3] is a simple metric indexing technique that is shown to have one of the best performances on high-dimensionality data. LC is based on partitioning, so both base methods will be represented in the results.

The basic idea is that you partition off some of the data elements, then you put them aside, so they're no longer considered part of the original set of elements, you then select a new partition on the original data set. This is done by choosing a center and a radius, how to select these will be discussed later. All elements with the selected radius of the selected center is considered part of the first partition, these are then put aside, and a new center is selected and the remaining elements within the radius of this new center becomes the next partition, which is put aside. This is repeated until all elements are part of a partition, and thus the original data set is empty.

When searching in the LC, it's important to traverse the list of partitions in the same order as they were created. The idea here too is simple, for each partition, check if there it intersects the query ball, if it doesn't, all elements in this partition can be disregarded, if it does intersect there is a possibility that it contains elements that should be part of the result of the search. The next partition in the list is then considered in the same way until the end of the list, or if a partition completely engulfs the query ball, no further partitions need to be considered, as these will not contain any relevant elements. These would have been removed from the original data set when the current partition was created.

A more formal description can be found in [3]. A detailed description of how it was implemented can be found in the implementation part of this paper.

[3] discusses several methods for center selection and what the radius should be. These methods will be presented briefly here; first, center selection methods.

- By far the simplest method is to choose one of the data elements at random as the center. Being trivial to implement, the resulting index is more often then not, not optimal.
- Another simple method that provides somewhat better results, is to choose the from the remaining elements the one that is closest to the previous center, in effect traversing the metric space, but with the side effect of producing a lot of overlap.
- Opposite to the above, one can instead choose the data element furthest from the previous center. This approach will result in very little overlap of the partitions.
- The two above methods can be further extended to be more global. First, instead of closest to the previous center, the element with the smallest distance sum to all previous centers can be selected.
- Likewise the element with the largest distance sum to all previous centers can be selected

For the radius, two different approaches are discussed.

- The first approach is to have the radius a set constant value. I.e. all elements within r of the center is included in the partition.

- The second approach is to have the radius be dynamic, but keep the number of elements in each partition constant. I.e. the n closest elements to the center are included in the partition.

The differences between these two approaches become apparent towards the end of the index construction when there are few data elements remaining in the data set. With the first approach, the last clusters in the list will contain few elements (often just one), as the elements are likely scattered across a big part of the metric space, as a result they will also be numerous. By using the second approach, there will be few partitions towards the end of the list, but these will cover a huge area of the metric space; worst case scenario, two opposite outer elements remain, causing the last cluster to contain the entire metric space.

3 Implementation

3.1 Environment

The test application is implemented in Java. The main reason for this is the author's experience with the language. It brings with it the advantage of being platform independent and rich API libraries reducing implementation time.

The main drawback of using Java in this setting, is the bad memory management, which is mostly automated, and thus, could result in lots of unnecessary swapping to the hard drive. Luckily, the data sets that will be used for these experiments are not large enough for this to be an issue. If at a later time, it would be necessary for this implementation to handle significantly large data sets, all the currently implemented algorithms are done so, so that they process the data sets linearly, thus extending the application to process the data set from disc instead of memory is feasible. As it stands, Java's lack of memory management facilities should have no impact on actual performance.

3.2 Framework

The application itself is designed as a framework for implementing a variety of methods for indexing time series (or similar formated data) and doing range queries on them. The main features are independence between indexing method and distance measure, allowing these to be mixed and matched. A simple template are provided for implementing either of these. This framework is similar to GEMINI as described in [10].

The signature for the distance measure consists of just one method: *double calculateDistance(time series a, time series b)*. It returns the distance between the two specified elements. Due to the nature of the application, it is important that the distance measure meet the metric space requirements.

The template for the indexing methods are slightly more complex, but not overly so. There are two main methods: *construct* and *rangeQuery*. The former takes as input a data set and a distance measure (mentioned above), and builds an index from these. The latter method takes as input a time series object and a search radius and returns all elements in the data set that are within the search radius distance from the inputed time series object, also called the query object. Most implementations will also return false positives. That is, objects that are outside the actual search radius, since most methods are underestimates of the actual distance in some form or another. The main reason for this is that false positives are much more acceptable then missed positives, as the false ones can be filtered out later, while the missed ones are lost and the query result thus incomplete. Two additional methods are introduced in this template: *resetDistanceCounter* and *getDistanceCounter*. These two are present for the benchmarking. The first one will set the counter to zero, while the second one returns the value of the counter. The framework will call the reset before a range query call and use the get method after the query recording the result.

Another important element of this framework is the configuration file. All variable settings are set in a configuration file which is globally available in the application. Examples of settings would be the number of pivots to use for LAESA or the number of segments to use for the signatures in PCA. The use of a configuration file allows for easily setting up different environments for

testing. To make it convenient, the application takes as an input parameter which configuration file to use. On the subject of input parameters; in addition to the configuration file, the application also needs to have a data set file, a query set file and an output file. The data set file and the query set file must be a perfect matrix with one time series per row. Any inconsistencies in time series length or if any of the elements contains invalid data, the application will terminate, thus preventing the errors this would cause during execution.

In the following sections, the four methods that were implemented for this thesis will be presented and discussed in some detail.

3.3 Brute Force

Being the very reason behind this thesis, the brute force approach is just to computational expensive to be feasible in any sizable real world implementation. The size of the available data sets do however not hinder the use of this approach to determine the "correct answers" to the queries. The results this method provides will be used to determine the results of the other methods, primarily for correctness. If any of the other methods misses any of the results that the brute force approach found, it will be an indication of it being faulty.

There is no magic behind the implementation of this method: Each element in the data set gets its distance to the query object calculated, if this distance is lower then the search radius, add it to the result. The approach has no construction, all computation is done at runtime. For benchmarking the get counter simply returns the size of the data sets.

3.4 PCA

PCA has already been presented in this thesis. This part will focus on how it was implemented.

PCA is implemented in a straight forward way. During construction it fetches from the configuration file how many segments long the signatures should be. All the time series gets their signature generated in order. For each, all but the last segment is processed in the main loop, adding up all the values in that segment from the source, dividing it on number of elements, and inserts the result into the signature. The last segment is handled independently from the rest. This is because it needs some special logic to account for the fact that it is not necessarily as long as the other segments. This removes any conditional statements form the main loop, allowing it to process faster. As for the handling of the last segment, it is similar to the others, except that the number of elements is different. This is counted as the values are summed.

One noteworthy feature is that the signatures themselves are saved as *time series* objects, just shorter then their parent ones. All methods that then are applicable on the time series can also be used on the signatures. This advantage becomes apparent when calculating the distance between the signatures and the signature of the query object by the same method used for time series.

For the range query, the first step is to generate the signature of the query object. The same internal method that produces the signatures from the time series is recycled for this using the same number of segments as specified in the configuration file. The method then proceeds to use the brute force method on the signatures with one exception: In [10] Keogh shows that signatures in their

pure state are a much larger underestimation than necessary, mostly due to the reduced number of dimensions. They proceed to show that you can optimize this underestimation by multiplying the distance between two signatures by a factor. This factor is proven in their paper to be: $\sqrt{N/n}$ where N is the length of the original time series and n is the number of segments it is divided into, i.e. the length of the signature.

For benchmarking, this method does no real distance calculation at search time, so the distance counter returns zero. Most of the work is done during construction. One should note however, that the statement about doing no real distance calculations at runtime is only true when the signature length is a lot shorter then the actual data series length. In some of the test cases this is not true, and will be noted. Here lies one of the faults of the application. If one sets the number of segments to the time series length, the result is a brute force approach at apparently zero cost, which is not true.

3.5 LC

The theory and workings of LC, or List of Clusters has also been explained in the previous sections, as with PCA focus here will be on the actual implementation.

This implementation is using a constant number of elements per cluster, as opposed to the other approach where cluster radius is constant and number of elements varies. The first cluster center is chosen amongst all data elements at random. As mentioned earlier, all implemented methods' construction processes the data sets in a linear fashion. The first step is to take the n first elements in the data set, and put them in the candidate set. n here is the cluster size. The element in the candidate set that is the furthest away from the center is found. The remaining elements in the data set are then processed, i.e. the distance from them to the center is calculated. If this distance is smaller than the biggest in the candidate set, that element is replaced by the new element, and again, the element furthest from the center in the candidate set is determined. The comparison between this element and the remaining elements in the data set continues in this fashion. Any elements in the data set marked as already used in a cluster is skipped.

While processing the clusters, the cumulative distance from each center to each element is recorded. The non-used element with the biggest cumulative distance from all previous centers is chosen to be the center for the next cluster. If the number of remaining elements is smaller than the size of a cluster, the process is terminated and the remaining elements stored in a special center-less cluster. In many cases the radius of this final cluster would encompass almost the entire data set space, making normal processing of it useless.

The query processes each cluster in the order they were created. If there is overlap between the query ball and the cluster, all elements in the cluster are added to the result and the next cluster is processed. If the cluster envelopes the query ball, its elements are added to the result and no further clusters are processed. Finally all the elements in the last center-less cluster mentioned above are automatically added to the result. Each time a cluster is processed, the distance counter is incremented.

3.6 LAESA

The fourth and final method implemented is LAESA.

This implementation of LAESA creates a distance map, a two dimensional matrix with number of rows equal to number of pivots and columns equal to number of elements in the data set. The distance map is filled with the distance between the pivot and the data element matching in the matrix, one pivot at a time. The first pivot is picked randomly from the entire data set. A similar method to what was used in the implementation of LC for choosing the next cluster center is used for picking the next pivot. As the distances between the current pivot and the data elements are calculated, they are accumulated in an array. By using the cumulative distance between all current pivots and all data elements, the next pivot is chosen to be the most distant data element, as per the LC implementation. As will be indicated later, this approach seems to be a bad choice. The implementation contains most of an alternative and much more successful pivot selection method, albeit time did not permit its completion and its details moved to the conclusion and further work sections.

For the query, the distance map is traversed one pivot at a time using the distance from the query object to the pivot and the distance from the pivot to each element to determine if the element falls inside the query donut. The simple check used here is: $p < d + r$ and $p > d - r$, where p is the distance between pivot and element, d is the distance between query and pivot while r is the query range. Elements that satisfy both these conditions are added to the result. Elements that qualify but are already in the results due to being added when processing a previous pivot are not added a second time. This implementation does not account for guaranteed positives. It's possible with LAESA, when the query ball engulfs the pivot and the distance between the pivot and an element is smaller then the engulfment, to say that that element is guaranteed to be a positive. In most big data sets, this should not happen sufficiently often to have major impact on the results, justifying the change it would imply on the actual framework and the work involved expanding it to handle more then one class of results, but should be noted. Number of search time distance calculations is equal to the number of pivots.

3.7 Benchmark

As mentioned a few times before the metric that is the focus of this thesis is the number of actual distance calculations. That is, distance calculation between two full length time series. Only the ones done at the time of the search and the ones needed to filter out all the false positives are relevant. To filter out the false positives, all of the elements returned as the result of the query needs to be compared against the query object in a brute force manner. This makes the total number of actual distance calculations *search time distance calculations + number of results returned*. As mentioned in the framework section and subsequent method sections, search time distance calculations is recorded and available through a method invocation. The framework prints the results to the log file specified at startup.

For validating the methods the results of each of them are compared with the results achieved by the brute force approach. This was a vice choice as it made a flaw in the LC method apparent, where the elements in the final center-less

cluster were not handled properly. After this flaw being resolved, all queries have been correct and complete. This data has been removed from the results tables to simplify them. For determining the number of false positives in each test, compare the results with the brute force result that is listed at the top.

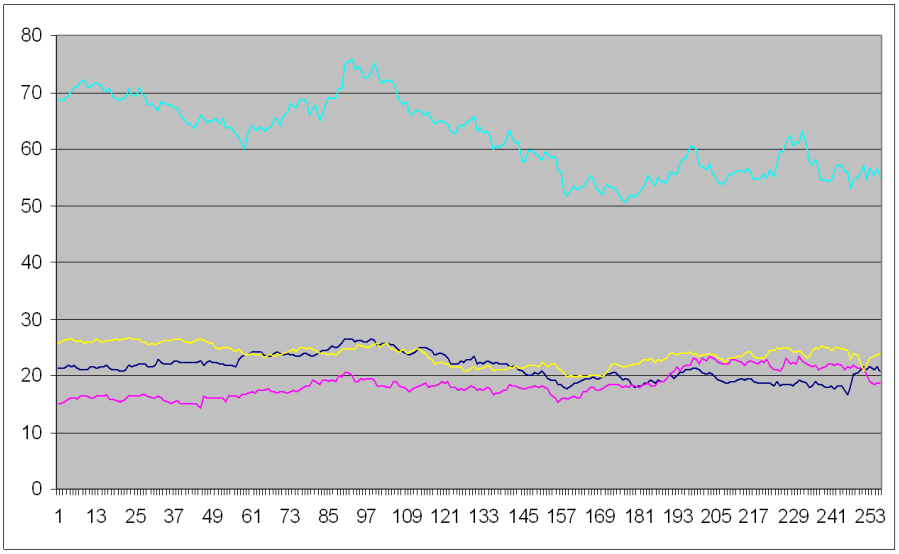


Figure 3: Stock Data Sample

4 Results and Findings

4.1 The Time Series

Four data sets were picked from the UCR Time series data [8]. The prime criterion was their size. A lot of the time series in the collection contained either very few, very long time series, or the opposite. The chosen series were big enough both in number and length to provide reasonable results. The second criterion for picking the data sets was shape. Testing two data sets with almost identical data would not provide as many insights as two data sets with different data. The following figures were generated by choosing four time series from each set at random. Only the first 255 data points of the time series are shown due to some technical limitations, and trying to display more points would lead to the graphs being unnecessarily compressed. All the time series continues in the same pattern as shown in the graphs.

Figure 3 shows typical stock data, a company's value varying over time with small local fluctuations and a overall global trend, which the aquamarine line shows well. The data set contains 400 time series each of them consists of 6481 data points.

As for figure 4, the data here came with no apparent documentation except its name "hockey". Characteristic of this data as can be seen form the figure is the data is flat and almost unchanging for its entire length. Unlike the other three data sets, the figure here shows the time series in its entirety. The data set contains 10000 time series, but they are just 257 data points long. This data set is far larger then the other sets in terms of number of time series at the same time it also has the shortest time series; points of note.

In figure 5 you see the current of an electric motor. The result of this is a smooth curve with a somewhat reoccurring pattern, unlike the following, most of the information here leans towards being global. The set consists of 420 time

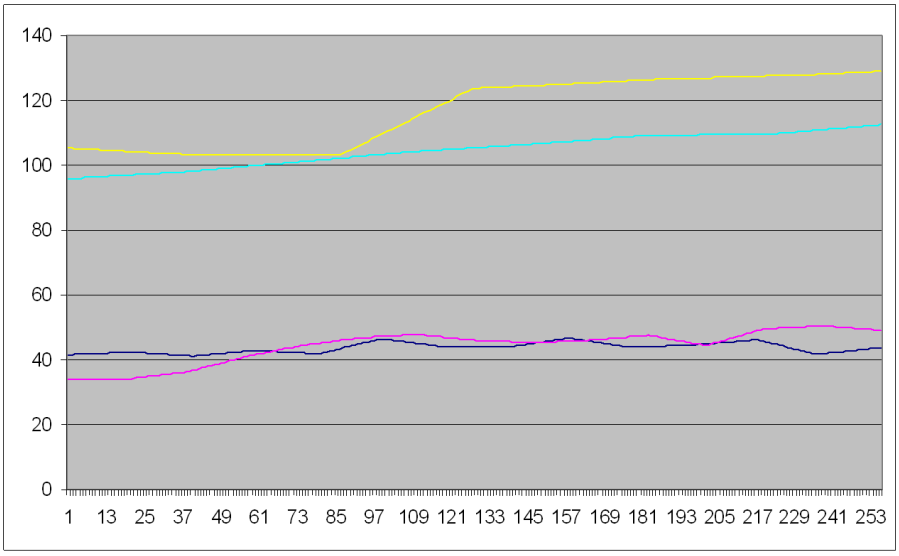


Figure 4: Hockey Data Sample

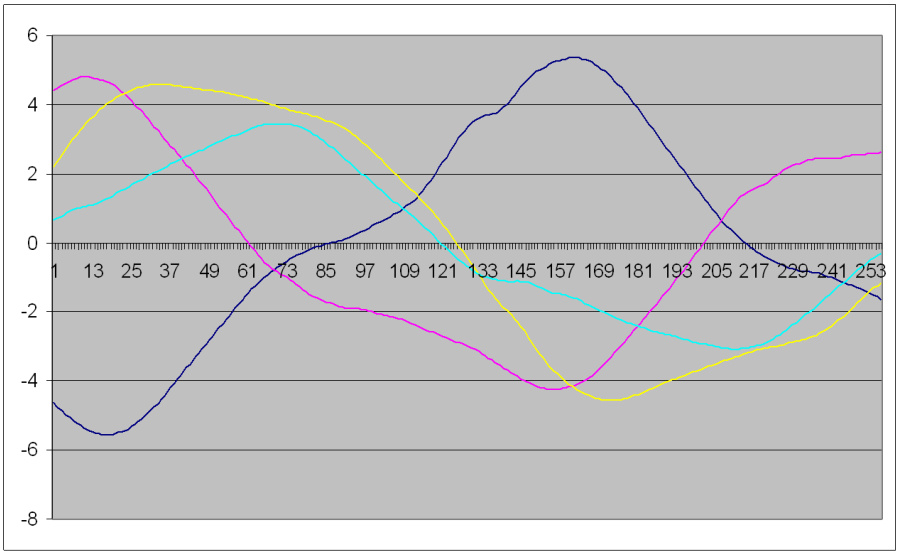


Figure 5: Motor Current Data Sample

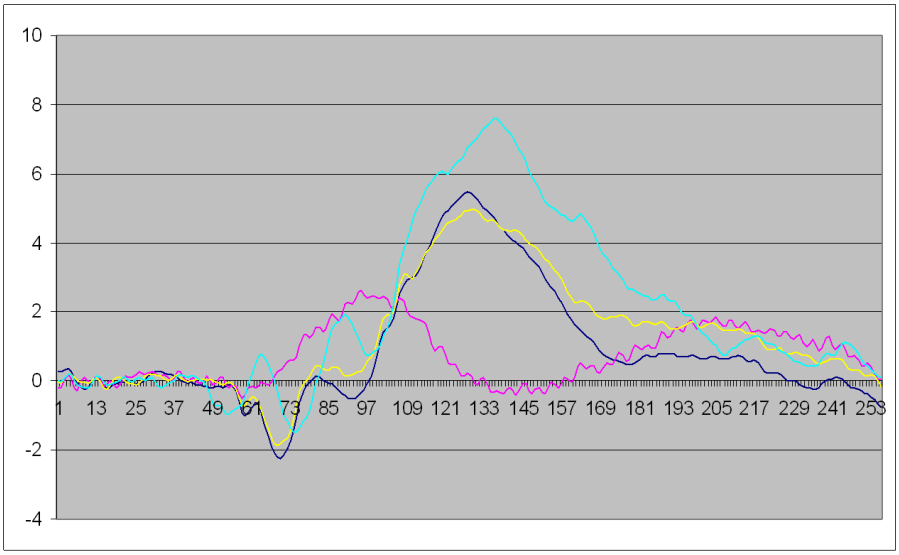


Figure 6: ERP Data Sample

series that are 1501 data points long.

The forth and final set, figure 6, also came without any documentation other then it's name: "Converted ERP". The nature of the data is not crucial to the success of the tests, but the shape of it is. The time series seems to be constructed by overlapping reoccurring waves of different frequencies atop of each other. The parts not shown by the graph are similar to what is, reoccurring peaks with constant local "noise". This set is similar to the stock data in size, there are 496 time series each of which are 6401 data points in length.

4.2 The Queries

Five queries were generated for each of the data sets. This was done by using a module in the framework called *QueryMaker*. Its functions are simple; it picks a specified number of elements from a specified data set, adds to it the specified amount of noise and writes the results to a specified query file. The syntax of a query file is identical to that of a data set file.

For all queries generated, additive Gaussian noise was added to 5 randomly picked time series.

- For the stock data, the average absolute data point value was 50, the standard deviation on the Gaussian noise was set to 10.
- The average absolute hockey data point value was 96, the standard deviation for the noise was set to 25.
- Motor current's average was 2.6, noise set to 0.5.
- ERP's average was 0.6, noise set to 0.1.

With the queries created, the search radius was tweaked until the brute force results contained a satisfying amount of elements. The same radius was used

across all 5 queries on the same data set, but it varied between data sets. The radiuses used were:

- Stock: 1700
- Hockey: 500
- Motor current: 40
- ERP: 60

4.3 The Results

Listed below is the numerical data for each of the test runs. The data is divided into three tables, one per data set. Each table is divided into one section per method. The top one being the brute force results for comparison. Each column is a query as mentioned in the above section. For each of the three tested methods, the leftmost column displays the main variable used for that method. For LAESA, this is the number of pivots; LC, the cluster size; PCA, the number of segments. The results that are in bold typeface are the best result for that method in that query.

Table 1: Stock - 400 time series of 6481 Length

query	1	2	3	4	5
BF	67	2	35	107	55
pivots	LAESA				
5	477	490	482	466	474
10	**458**	**190**	**384**	**423**	**459**
15	484	484	481	485	481
20	464	198	398	442	469
25	483	456	486	492	483
50	519	519	519	525	519
99	559	355	560	559	555
csize	LC				
5	544	**399**	494	424	539
10	509	419	479	389	509
15	503	473	503	383	503
20	484	484	484	404	464
25	494	494	494	444	494
50	**459**	459	459	459	**459**
99	505	406	**406**	**307**	505
segments	PCA				
5	334	139	266	184	353
10	300	48	241	179	315
15	285	34	224	173	290
20	277	21	206	171	281
25	265	9	198	166	272
50	228	3	183	162	255
99	**206**	**2**	**153**	**159**	**225**

Table 2: Hockey - 10000 time series of 257 Length

query	1	2	3	4	5
BF	146	445	196	126	115
pivots	LAESA				
5	3950	3356	4427	4538	**1226**
10	4503	2502	4215	3342	2351
15	**2212**	3788	3547	**2263**	1344
20	2601	1839	**2097**	2614	1745
25	2918	**1774**	2110	2908	2885
50	4454	2157	2514	4505	4695
99	2484	4024	3083	2410	1566
csize	LC				
5	2534	2634	2424	2409	2444
10	1669	1759	1519	1589	1559
15	1441	1456	1306	1336	1351
20	**1279**	1339	1159	1299	1279
25	1326	1224	1124	1249	**1149**
50	1349	1249	**1099**	1299	1249
99	1389	**1191**	1290	**1191**	1290
segments	PCA				
5	2853	4135	2400	2754	4378
10	2276	2148	1955	2209	2781
15	1837	1680	1616	1786	1917
20	1694	1561	1435	1585	1519
25	1503	1377	1282	1315	1319
50	1034	1126	861	821	865
99	**631**	**748**	**538**	**590**	**577**

Table 3: Motor - 420 time series of 1501 Length

query	1	2	3	4	5
BF	34	19	25	26	31
pivots	LAESA				
5	**87**	**162**	**168**	169	169
10	188	246	172	185	186
15	118	190	234	192	201
20	242	316	204	235	241
25	207	182	239	**112**	**114**
50	230	209	244	140	143
99	441	519	471	512	514
csize	LC				
5	178	173	168	173	173
10	141	141	141	151	151
15	132	**117**	132	147	147
20	120	120	**120**	**120**	**120**
25	**111**	132	132	136	136
50	128	125	125	126	126
99	226	225	225	125	125
segments	PCA				
5	420	420	420	420	420
10	86	82	83	82	82
15	74	74	69	75	75
20	70	69	65	70	70
25	69	64	59	70	69
50	55	38	48	60	58
99	**43**	**33**	**43**	**54**	**44**

Table 4: ERP - 496 time series of 6401 Length

query	1	2	3	4	5
BF	52	49	35	32	78
pivots	LAESA				
5	252	313	179	354	264
10	189	160	196	**93**	**136**
15	**148**	303	**177**	104	256
20	235	**153**	185	242	146
25	388	270	334	316	443
50	277	230	311	304	179
99	232	405	280	193	341
csize	LC				
5	235	225	200	175	255
10	**195**	185	185	155	**215**
15	199	199	184	169	244
20	200	**180**	**160**	140	220
25	215	190	190	102	240
50	205	205	205	150	305
99	303	204	303	**101**	303
segments	PCA				
5	496	496	496	496	496
10	496	496	496	496	496
15	496	496	496	496	496
20	496	496	494	496	496
25	496	496	474	496	496
50	496	481	397	463	495
99	**466**	**422**	**334**	**332**	**441**

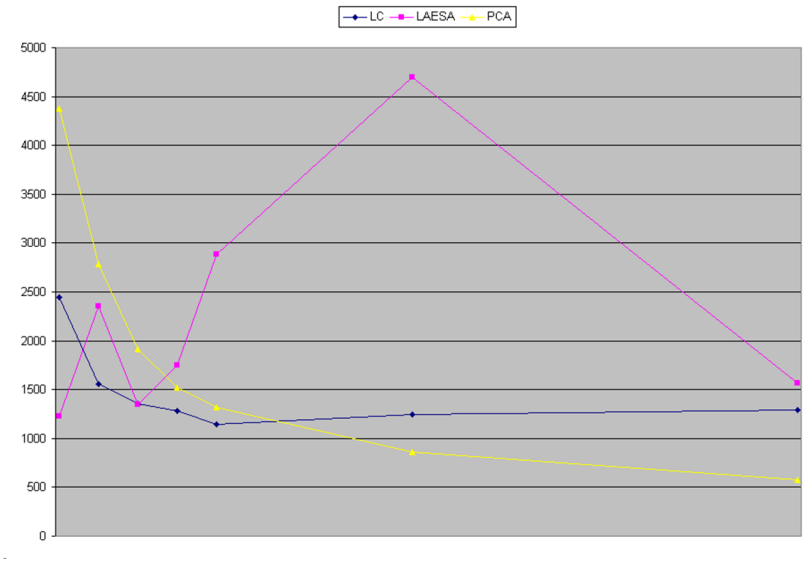


Figure 7: Hockey Query 5

4.4 Summary and Further Work

Due to the random nature of LAESA's pivot selection, its results vary greatly from one run to the other, and no clear pattern can be confidently established as to the relation between number of pivots and resulting performance. Figure 7 shows an example of the randomness LAESA displays. Implementing some of the methods in [1] for better pivot selection would most probably lead to both better and less random results. It might also be possible to adept these methods for LC as well by applying them to center selection. Although mostly based on luck, LAESA does sometimes produce very good results, as can be seen scattered in the various tables, particularly in table 1 on the second query. These seemingly random occurrences makes further testing with LAESA and optimized pivot selection an interesting venture.

As expected PCA's performance only improves as the length of the signature increases. Figure 7 gives a very nice example of this. It also clearly shows that as the length increases, performance gained becomes less and less, while the price in performance takes huge leaps. In figure 8 no noticeable gain is achieved for increasing the signature size from the lowest length tested.

LC is also affected by some randomness in its construction, but not to the degree that LAESA is. As expected, in most cases performance improved as the number of clusters increased up to a certain point, where the price of more clusters exceeded the performance gained. Both figure 7 and 8 illustrates this.

The results of the different approaches vary greatly depending on the data they are applied to. PCA fails miserably on the ERP data where oscillations reoccur and local means will wash out the data. This is clearly show in table 4 where the PCA finds all objects as possible results for almost all the tested signature sizes, i.e. no objects gets pruned. Both LAESA and LC offer much improved performance on this kind of data.

On the opposite side of that specter we have the stock data, these results can

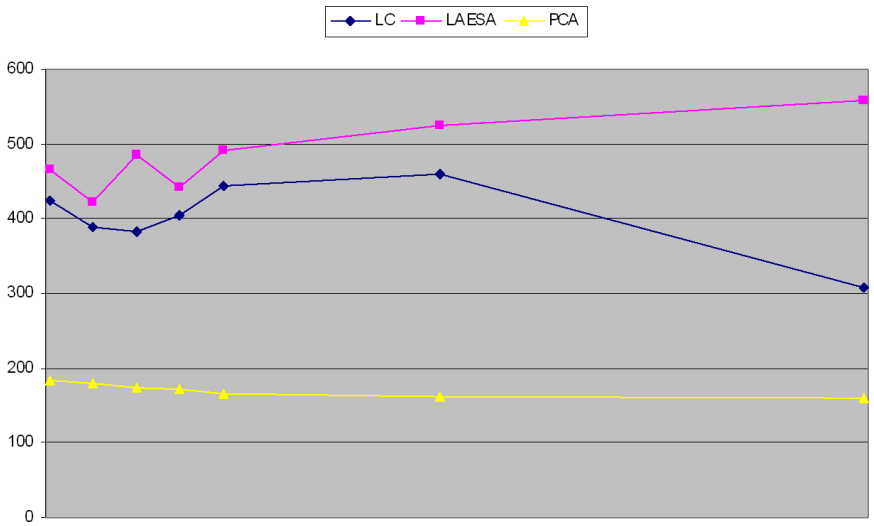


Figure 8: Stock Query 4

be seen in table 1. Here PCA vastly outperforms the two metric methods. This kind of data is very well suited for PCA, as local details are not as significant as a stock's growth over time, which the means capture nicely. As an added bonus, both the stock data and the ERP data is of small size but long length, showing that these two factors pale in comparison to the nature of the data.

Combining the motor data (table 3) with the ERP data (table 4), both of which exhibits cyclic behaviour, it might be probable that PCA needs to reach a critical signature size before it can effectively function. The motor data is short, while the ERP data is long. For a signature length of 5 on the motor data, no objects were pruned, when signature length was increased to 10, the performance improved greatly. On the ERP data PCA only started showing any effect for the longest tested signatures. This is probably credited to the locality of the data, the signature length, or rather, the size of each segment over which a mean is calculated needs to be smaller then the locality of the data, so that the discriminating features are not "washed out".

Overall the performance of the metric methods are good, considering no optimizations were implemented. On data where the discriminating factors are to be found in the local data rather then the global data, metric methods will outperform piecewise approximation methods. It's conceivable that with general optimizations regarding pivot or center selection the metric methods will provide a satisfiable approach to indexing complex data. The methods will be less sensitive to the nature of the data providing a more robust method applicable to almost any data set without more adaptation then determining the optimal number of pivots/clusters. One weakness to the tests applied here is the sizes of the data sets available. No truly large data set of complex data was available, i.e. a set on the scale of 10000-100000 objects, each object as longs as 10000-100000, maybe even larger. Not many real world data sets of these sizes are available, on the other hand artificial ones do exist or can be created. The

second problem with such sets is computer performance, the implementation will also have to be expanded to encompass memory management and disk caching. The current implementation of the methods do process the data sequentially, making the upgrade for it to handle this size of data possible.

List of Figures

1 Proximity Searches 4
2 LAESA filtering 8
3 Stock Data Sample 16
4 Hockey Data Sample 17
5 Motor Current Data Sample 17
6 ERP Data Sample 18
7 Hockey Query 5 23
8 Stock Query 4 24

References

[1] B. Bustos, G. Navarro, and E. Ch'avez. Pivot selection techniques for proximity searching in metric spaces, 2001.

[2] Kaushik Chakrabarti and Sharad Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *ICDE*, pages 440–447. IEEE Computer Society, 1999.

[3] Edgar Chávez and Gonzalo Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[4] Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[5] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In Richard T. Snodgrass and Marianne Winslett, editors, *SIGMOD Conference*, pages 419–429. ACM Press, 1994.

[6] Ykä Huhtala, Juha Kärkkäinen, and Hannu Toivonen. Mining for similarities in aligned time series using wavelets.

[7] Eamonn Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 239–241, New York City, NY, 1998. ACM Press.

[8] Eamonn J. Keogh. The ucr time series classification/clustering page. [`http://www.cs.ucr.edu/~eamonn/time_series_data/`].

[9] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, 2001.

[10] Eamonn J. Keogh, Kaushik Chakrabarti, Michael J. Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3):263–286, 2001.

[11] Sam Roweis. EM algorithms for PCA and SPCA. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[12] Hagit Shatkay. The fourier transform - A primer. Technical Report CS-95-37, 1995.