Iterative Incremental Clustering of Time Series

Jessica Lin, Michail Vlachos, Eamonn Keogh, and Dimitrios Gunopulos

Computer Science & Engineering Department
University of California, Riverside
Riverside, CA 92521
{jessica, mvlachos, eamonn, dg}@cs.ucr.edu

Abstract. We present a novel anytime version of partitional clustering algorithm, such as k-Means and EM, for time series. The algorithm works by leveraging off the multi-resolution property of wavelets. The dilemma of choosing the initial centers is mitigated by initializing the centers at each approximation level, using the final centers returned by the coarser representations. In addition to casting the clustering algorithms as anytime algorithms, this approach has two other very desirable properties. By working at lower dimensionalities we can efficiently avoid local minima. Therefore, the quality of the clustering is usually better than the batch algorithm. In addition, even if the algorithm is run to completion, our approach is much faster than its batch counterpart. We explain, and empirically demonstrate these surprising and desirable properties with comprehensive experiments on several publicly available real data sets. We further demonstrate that our approach can be generalized to a framework of much broader range of algorithms or data mining problems.

1 Introduction

Clustering is a vital process for condensing and summarizing information, since it can provide a synopsis of the stored data. Although there has been much research on clustering in general, most classic machine learning and data mining algorithms do not work well for time series due to their unique structure. In particular, the high dimensionality, very high feature correlation, and the (typically) large amount of noise that characterize time series data present a difficult challenge. Although numerous clustering algorithms have been proposed, the majority of them work in a batch fashion, thus hindering interaction with the end users. Here we address the clustering problem by introducing a novel anytime version of partitional clustering algorithm based on wavelets. Anytime algorithms are valuable for large databases, since results are produced progressively and are refined over time [11]. Their utility for data mining has been documented at length elsewhere [2, 21]. While partitional clustering algorithms and wavelet decomposition have both been studied extensively in the past, the major novelty of our approach is that it mitigates the problem associated with the choice of initial centers, in addition to providing the functionality of user-interaction.

The algorithm works by leveraging off the multi-resolution property of wavelet decomposition [1, 6, 22]. In particular, an initial clustering is performed with a very

coarse representation of the data. The results obtained from this "quick and dirty" clustering are used to initialize a clustering at a finer level of approximation. This process is repeated until the "approximation" is the original "raw" data. Our approach allows the user to interrupt and terminate the process at any level. In addition to casting the clustering algorithm as an anytime algorithm, our approach has two other very unintuitive properties. The quality of the clustering is often better than the batch algorithm, and even if the algorithm is run to completion, the time taken is typically much less than the time taken by the batch algorithm.

We initially focus our approach on the popular k-Means clustering algorithm [10, 18, 24] for time series. For simplicity we demonstrate how the algorithm works by utilizing the Haar wavelet decomposition. Then we extend the idea to another widely used clustering algorithm, EM, and another well-known decomposition method, DFT, towards the end of the paper. We demonstrate that our algorithm can be generalized as a framework for a much broader range of algorithms or data mining problems.

The rest of this paper is organized as follows. In Section 2 we review related work, and introduce the necessary background on the wavelet transform and k-Means clustering. In Section 3, we introduce our algorithm. Section 4 contains a comprehensive comparison of our algorithm to classic k-Means on real datasets. In Section 5 we study how our approach can be extended to other iterative refinement method (such as EM), and we also investigate the use of other multi-resolution decomposition such as DFT. In Section 6 we summarize our findings and offer suggestions for future work.

2 Background and Related Work

Since our work draws on the confluence of clustering, wavelets and anytime algorithms, we provide the necessary background on these areas in this section.

2.1 Background on Clustering

One of the most widely used clustering approaches is hierarchical clustering, due to the great visualization power it offers [12]. Hierarchical clustering produces a nested hierarchy of similar groups of objects, according to a pairwise distance matrix of the objects. One of the advantages of this method is its generality, since the user does not need to provide any parameters such as the number of clusters. However, its application is limited to only small datasets, due to its quadratic (or higher order) computational complexity.

A faster method to perform clustering is k-Means [2, 18]. The basic intuition behind k-Means (and in general, iterative refinement algorithms) is the continuous reassignment of objects into different clusters, so that the within-cluster distance is minimized. Therefore, if $\boldsymbol{x}$ are the objects and $\boldsymbol{c}$ are the cluster centers, k-Means attempts to minimize the following objective function:

$$F = \sum_{m=1}^{k} \sum_{i=1}^{N} \left\| x_i - c_m \right\| \quad (1)$$

The k-Means algorithm for N objects has a complexity of O(*kNrD*) [18], where *k* is the number of clusters specified by the user, *r* is the number of iterations until convergence, and *D* is the dimensionality of the points. The shortcomings of the algorithm are its tendency to favor spherical clusters, and its requirement for prior knowledge on the number of clusters, *k*. The latter limitation can be mitigated by attempting all values of *k* within a large range. Various statistical tests can then be used to determine which value of k is most parsimonious. However, this approach only worsens k-Means' already considerable time complexity. Since k-Means is essentiality a hill-climbing algorithm, it is guaranteed to converge on a local but not necessarily global optimum. In other words, the choices of the initial centers are critical to the quality of results. Nevertheless, in spite of these undesirable properties, for clustering large datasets of time-series, k-Means is preferable due to its faster running time.

Table 1. An outline of the k-Means algorithm

Algorithm k-Means	
1	Decide on a value for k.
2	Initialize the k cluster centers (randomly, if necessary).
3	Decide the class memberships of the N objects by assigning them to the nearest cluster center.
4	Re-estimate the k cluster centers, by assuming the memberships found above are correct.
5	If none of the N objects changed membership in the last iteration, exit. Otherwise goto 3.

In order to scale the various clustering methods to massive datasets, one can either reduce the number of objects, *N*, by sampling [2], or reduce the dimensionality of the objects [1, 3, 9, 12, 13, 16, 19, 25, 26]. For time-series, the objective is to find a representation at a lower dimensionality that preserves the original information and describes the original shape of the time-series data as closely as possible. Many approaches have been suggested in the literature, including the Discrete Fourier Transform (DFT) [1, 9], Singular Value Decomposition [16], Adaptive Piecewise Constant Approximation [13], Piecewise Aggregate Approximation (PAA) [4, 26], Piecewise Linear Approximation [12] and the Discrete Wavelet Transform (DWT) [3, 19]. While all these approaches have shared the ability to produce a high quality reduced-dimensionality approximation of time series, wavelets are unique in that their representation of data is intrinsically multi-resolution. This property is critical to our proposed algorithm and will be discussed in detail in the next section.

2.2 Background on Wavelets

Wavelets are mathematical functions that represent data or other functions in terms of the averages and differences of a prototype function, called the analyzing or mother wavelet [6].

In this sense, they are similar to the Fourier transform. One fundamental difference is that wavelets are localized in time. In other words, some of the wavelet coefficients represent small, local subsections of the data being studied, as opposed to Fourier coefficients, which always represent global contributions to the data. This property is very useful for multi-resolution analysis of data. The first few coefficients contain an overall, coarse approximation of the data; additional coefficients can be perceived as "zooming-in" to areas of high detail. Figs 1 and 2 illustrate this idea.

The Haar Wavelet decomposition is achieved by averaging two adjacent values on the time series function at a given resolution to form a smoothed, lower-dimensional signal, and the resulting coefficients at this given resolution are simply the differences between the values and their averages [3]. As a result, the Haar wavelet decomposition is the combination of the coefficients at all resolutions, with the overall average for the time series being its first coefficient. The coefficients are crucial for reconstructing the original sequence, as they store the detailed information lost in the smoothed signal.

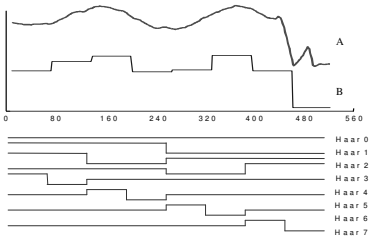

Fig 1. The Haar Wavelet representation can be visualized as an attempt to approximate a time series with a linear combination of basis functions. In this case, time series A is transformed to B by Haar wavelet decomposition, and the dimensionality is reduced from 512 to 8.

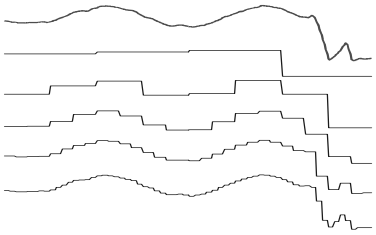

Fig 2. The Haar Wavelet can represent data at different levels of resolution. Above we see a raw time series, with increasing faithful wavelet approximations below.

2.3 Background on Anytime Algorithms

Anytime algorithms are algorithms that trade execution time for quality of results [11]. In particular, an anytime algorithm always has a best-so-far answer available, and the quality of the answer improves with execution time. The user may examine this answer at any time, and choose to terminate the algorithm, temporarily suspend the algorithm, or allow the algorithm to run to completion.

The utility of anytime algorithms for data mining has been extensively documented [2, 21]. Suppose a batch version of an algorithm takes a week to run (not an implausible scenario in mining massive, disk-resident data sets). It would be highly desirable to implement the algorithm as an anytime algorithm. This would allow a user to examine the best current answer after an hour or so as a "sanity check" of all assumptions and parameters. As a simple example, suppose the user had accidentally set the value of k to 50 instead of the desired value of 5. Using a batch algorithm the mistake would not be noted for a week, whereas using an anytime algorithm the mistake could be noted early on and the algorithm restarted with little cost. This motivating example could have been eliminated by user diligence! More generally, however, data mining algorithms do require the user to make choices of several parameters, and an anytime implementation of k-Means would allow the user to interact with the entire data mining process in a more efficient way.

2.4 Related Work

Bradley et. al. [2] suggest a generic technique for scaling the k-Means clustering algorithms to large databases by attempting to identify regions of the data that are compressible, that must be retained in main memory, and regions that may be discarded. However, the generality of the method contrasts with our algorithm's explicit exploitation of the structure of the data type of interest.

Our work is more similar in spirit to the dynamic time warping similarity search technique introduced by Chu et. al. [4]. The authors speed up linear search by examining the time series at increasingly finer levels of approximation.

3 Our Approach – the I-kMeans Algorithm

As noted in Section 2.1, the complexity of the k-Means algorithm is O($kNrD$), where D is the dimensionality of data points (or the length of a sequence, as in the case of time-series). For a dataset consisting of long time-series, the D factor can burden the clustering task significantly. This overhead can be alleviated by reducing the data dimensionality.

Another major drawback of the k-Means algorithm is that the clustering quality is greatly dependant on the choice of initial centers (i.e., line 2 of Table 1). As mentioned earlier, the k-Means algorithm guarantees local, but not necessarily global optimization. Poor choices of the initial centers, therefore, can degrade the quality of clustering solution and result in longer execution time (See [10] for an excellent discussion of this issue). Our algorithm addresses these two problems of k-Means, in addition to offering the capability of an anytime algorithm, which allows the user to interrupt and terminate the program at any stage.

We propose using a wavelet decomposition to perform clustering at increasingly finer levels of the decomposition, while displaying the gradually refined clustering results periodically to the user. Note that any wavelet basis (or any other multi-

resolution decomposition such as DFT) can be used, as will be demonstrated in Section 5. We opt for the Haar Wavelet here for its simplicity and its wide use in the time series community.

We compute the Haar Wavelet decomposition for all time-series data in the database. The complexity of this transformation is linear to the dimensionality of each object; therefore, the running time is reasonable even for large databases. The process of decomposition can be performed off-line, and needs to be done only once. The time series data can be stored in the Haar decomposition format, which takes the same amount of space as the original sequence. One important property of the decomposition is that it is a lossless transformation, since the original sequence can always be reconstructed from the decomposition.

Once we compute the Haar decomposition, we perform the k-Means clustering algorithm, starting at the second level (each object at level i has $2^{(i-1)}$ dimensions) and gradually progress to finer levels. Since the Haar decomposition is completely reversible, we can reconstruct the approximation data from the coefficients at any level and perform clustering on these data. We call the new clustering algorithm I-kMeans, where I stands for "interactive." Fig 3 illustrates this idea.

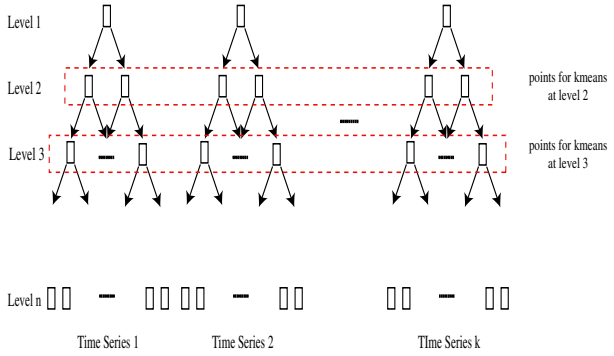

Fig 3. k-Means is performed on each level on the reconstructed data from the Haar wavelet decomposition, starting with the second level

The intuition behind this algorithm originates from the observation that the general shape of a time series sequence can often be approximately captured at a lower resolution. As shown in Fig 2, the shape of the time series is well preserved, even at very coarse approximations. Because of this desirable feature of wavelets, clustering results typically stabilize at a low resolution, thus saving time by eliminating the need to run at full resolution (the raw data). The pseudo-code of the algorithm is provided in Table 2.

The algorithm achieves the speed-up by doing the vast majority of reassignments (Line 3 in Table 1) at the lower resolutions, where the costs of distance calculations are considerably lower. As we gradually progress to finer resolutions, we already start with good initial centers (the choices of initial centers will be discussed later in this

section). Therefore, the number of iterations *r* until convergence will typically be much lower.

Table 2. An outline of the I-kMeans algorithm

Algorithm I-kMeans	
1	Decide on a value for k.
2	Initialize the k cluster centers (randomly, if necessary).
3	Run the k-Means algorithm on the $level_i$ representation of the data
4	Use final centers from $level_i$, as initial centers for $level_{i+1}$. This is achieved by projecting the k centers returned by k-Means algorithm for the 2^i space in the 2^{i+1} space.
5	If none of the N objects changed membership in the last iteration, exit. Otherwise goto 3.

The I-kMeans algorithm allows the user to monitor the quality of clustering results as the program executes. The user can interrupt the program at any level, or wait until the execution terminates once the clustering results stabilize. One surprising and highly desirable finding from the experimental results is that even if the program is run to completion (until the last level, with full resolution), the total execution time is generally less than that of clustering on raw data.

As mentioned earlier, on every level except for the starting level (i.e. level 2), which uses random initial centers, the initial centers are selected based on the final centers from the previous level. More specifically, the final centers computed at the end of level i will be used as the initial centers on level i+1. Since the length of the data reconstructed from the Haar decomposition doubles as we progress to the next level, we project the centers computed at the end of level i onto level i+1 by doubling each coordinate of the centers. This way, they match the dimensionality of the points on level i+1. For example, if one of the re-computed centers at the end of level 2 is (0.5, 1.2), then the initial center used for this cluster on level 3 is (0.5, 0.5, 1.2, 1.2). This approach resolves the dilemma associated with the choice of initial centers, which is crucial to the quality of clustering results [10]. It also contributes to the fact that our algorithm often produces better clustering results than the k-Means algorithm. More specifically, although our approach also uses random centers as initial centers, it's less likely to be trapped in local minima at such a low dimensionality. In addition, the results obtained from this initial level will be refined in subsequent levels. Note that while the performance of k-Means can be improved by providing "good" initial centers, the same argument applies to our approach as well[1].

The algorithm can be further sped up by *not* reconstructing the time series. Rather, clustering directly on the wavelet coefficients will produce identical results. However, the projection technique for the final centers mentioned above would not be appropriate here. Instead, we can still reuse the final centers by simply padding the

[1] As a matter of fact, our experiments (results not shown) with good initial centers show that this is true – while the performance of k-Means improves with good initial centers, the improvements on I-kMeans, in terms of both speed and accuracy, are even more drastic.

additional dimensions for subsequent levels with zeros. For brevity, we defer further discussion on this version to future work.

4 Experimental Evaluation

To show that our approach is superior to the k-Means algorithm for clustering time series, we performed a series of experiments on publicly available real datasets. For completeness, we ran the I-kMeans algorithm for all levels of approximation, and recorded the cumulative execution time and clustering accuracy at each level. In reality, however, the algorithm stabilizes in early stages and can automatically terminate much sooner. We compare the results with that of k-Means on the original data. Since both algorithms start with random initial centers, we execute each algorithm 100 times with different centers. However, for consistency we ensure that for each execution, both algorithms are seeded with the same set of initial centers. After each execution, we compute the error (more details will be provided in Section 4.2) and the execution time on the clustering results. We compute and report the averages at the end of each experiment. By taking the average, we achieve better objectiveness than taking the best (minimum), since in reality, it's unlikely that we would have the knowledge of the correct clustering results, or the "oracle," to compare with (as was the case with one of our test datasets).

4.1 Datasets and Methodology

We tested on two publicly available, real datasets. The dataset cardinalities range from 1,000 to 8,000. The length of each time series has been set to 512 on one dataset, and 1024 on the other.

• **JPL**: This dataset consists of readings from various inertial sensors from Space Shuttle mission STS-57. The data is particularly appropriate for our experiments since the use of redundant backup sensors means that some of the data is very highly correlated. In addition, even sensors that measure orthogonal features (i.e. the X and Y axis) may become temporarily correlated during a particular maneuver; for example, a "roll reversal" [8]. Thus, the data has an interesting mixture of dense and sparse clusters. To generate data of increasingly larger cardinality, we extracted time series of length 512, at random starting points of each sequence from the original data pool.

• **Heterogeneous**: This dataset is generated from a mixture of 10 real time series data from the UCR Time Series Data Mining Archive [14] (see Fig 4). Using the 10 time-series as seeds, we produced variation of the original patterns by adding small time shifting (2-3% of the series length), and interpolated Gaussian noise. Gaussian noisy peaks are interpolated using splines to create smooth random variations. Fig 5 illustrates how the data is generated.

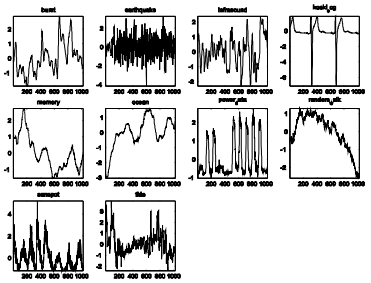

Fig 4. Real time series data from UCR Time Series Data Mining Archive. We use these time series as seeds to create our Heterogeneous dataset.

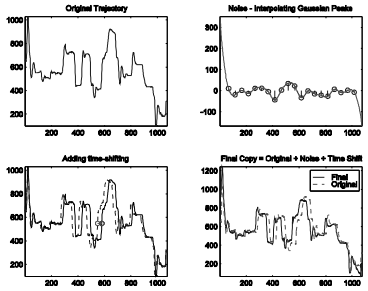

Fig 5. Generation of variations on the heterogeneous data. We produced variation of the original patterns by adding small time shifting (2-3% of the series length), and interpolated Gaussian noise. Gaussian noisy peaks are interpolated using splines to create smooth random variations.

In the Heterogeneous dataset, we know that the number of clusters is 10. However, for the JPL dataset, we lack this information. Finding k is an open problem for the k-Means algorithm and is out of scope of this paper. To determine the optimal k for k-Means, we attempt different values of k, ranging from 2 to 8. Nonetheless, our algorithm out-performs the k-Means algorithm regardless of k. In this paper we only show the results with k equals to 5. Fig 6 shows that our algorithm produces the same results as the hierarchical clustering algorithm, which is in generally more costly.

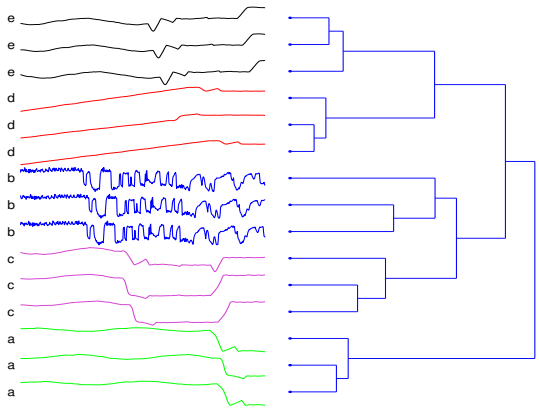

Fig 6. On the left-hand side, we show three instances taken from each cluster of the JPL dataset discovered by the I-kMeans algorithm. We can visually verify that our algorithm produces intuitive results. On the right-hand side, we show that hierarchical clustering (using average linkage) discovers the exact same clusters. However, hierarchical clustering is more costly than our algorithm.

4.2 Error of Clustering Results

In this section we compare the clustering quality for the I-kMeans and the classic k-Means algorithm.

Since we generated the heterogeneous datasets from a set of given time series data, we have the knowledge of correct clustering results in advance. In this case, we can simply compute the clustering error by summing up the number of incorrectly classified objects for each cluster c and then dividing by the dataset cardinality. This is achieved by the use of a confusion matrix. Note the accuracy computed here is equivalent to "recall," and the error rate is simply (1-accuracy).

The error is computed at the end of each level. However, it's worth mentioning that in reality, the correct clustering results would not be available in advance. The incorporation of such known results in our error calculation merely serves the purpose of demonstrating the quality of both algorithms.

For the JPL dataset, we do not have prior knowledge of correct clustering results (which conforms more closely to real-life cases). Lacking this information, we cannot use the same evaluation to determine the error.

Since the k-Means algorithm seeks to optimize the objective function by minimizing the sum of squared intra-cluster error, we evaluate the quality of clustering by using the objective functions. However, since the I-kMeans algorithm involves data with smaller dimensionality except for the last level, we have to map the cluster membership information to the original space, and compute the objective functions using the raw data in order to compare with the k-Means algorithm. We show that the objective functions obtained from the I-kMeans algorithm are better than those from the k-Means algorithm. The results are consistent with the work of Ding et. Al. [5], in which the authors show that dimensionality reduction reduces the chances of the algorithm being trapped in a local minimum. Furthermore, even with the additional step of computing the objective functions from the original data, the I-kMeans algorithm still takes less time to execute than the k-Means algorithm.

In Figs 7-8, we show the errors/objective functions from the I-kMeans algorithm as a fraction of those obtained from the k-Means algorithm. As we can see from the plots, our algorithm stabilizes at early (i.e. 2nd or 3rd) stages and consistently results in smaller error than the classic k-Means algorithm.

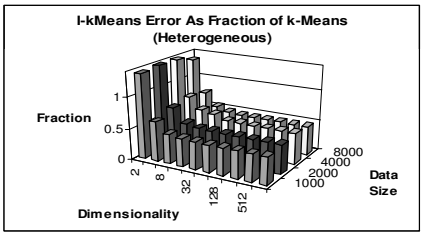

Fig 7. Error of I-kMeans algorithm on the Heterogeneous dataset, presented as fraction of the error from the k-Means algorithm.

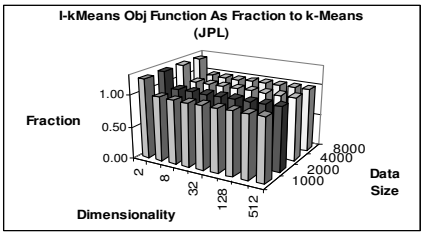

Fig 8. Objective functions of I-kMeans algorithm on the JPL dataset, presented as fraction of error from the k-Means algorithm.

4.3 Running Time

In this section, we present the cumulative running time for each level on the I-kMeans algorithm as a fraction to the k-Means algorithm. The cumulative running time for any level *i* is the total running time from the starting level to level *i*. In most cases, even if the I-kMeans algorithm is run to completion, the total running time is still less than that of the k-Means algorithm. We attribute this improvement to the good choices of initial centers, since they result in very few iterations until convergence. Nevertheless, we have already shown in the previous section that the I-kMeans algorithm finds the best result in relatively early stage and does not need to run through all levels. The time required for I-kMeans is therefore less than 50% of time required for k-Means for the Heterogeneous datasets. For the JPL datasets, the running time is less than 20% of time for k-Means, and even if it is run to completion, the cumulative running time is still 50% less than that of the k-Means algorithm.

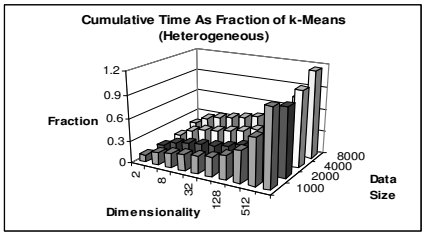

Fig 9. Cumulative running time for the Heterogeneous dataset. Our algorithm cuts the running time by more than half.

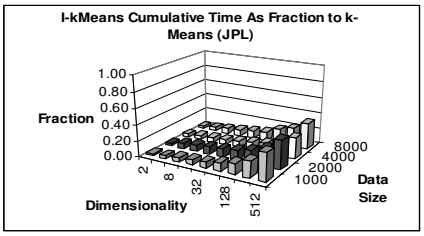

Fig 10. Cumulative running time for the JPL dataset. Our algorithm typically takes only 20% of the time required for k-Means.

While the speedup achieved is quite impressive, we note that these results are only for the main memory case. We should expect a much greater speedup for more realistic data mining problems. The reason is that when performing k-Means on a massive dataset, every iteration requires a database scan [2]. The I/O time for the scans dwarfs the relatively inexpensive CPU time. In contrast, our multi-resolution approach is able to run its first few levels in main memory, building a good "approximate" model[2] before being forced to access the disk. We can therefore expect our approach to make far fewer data scans.

4.4 I-kMeans Algorithm vs. k-Means Algorithm

In this section, rather than showing the error/objective function on each level, we present only the error/objective function returned by the I-kMeans algorithm when it out-performs the k-Means algorithm. We also present the time taken for the I-kMeans algorithm to stabilize (i.e. when the result does not improve anymore). We compare

[2] As we have seen in Figs 7 and 8, the "approximate" models are typically better than the model built on the raw data.

the results to those of the k-Means algorithm. The running time for I-kMeans remains small regardless of data size because the algorithm out-performs k-Means at very early stages.

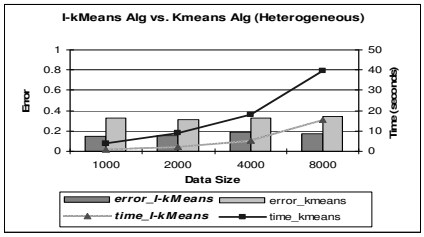

Fig 11. The I-kMeans algorithm is highly competitive with the k-Means algorithm. The errors (bars) and execution time (lines) are significantly smaller.

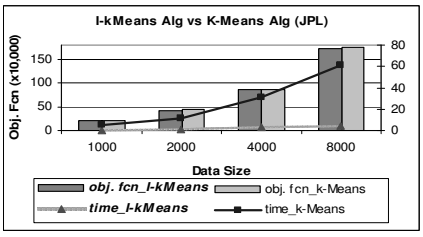

Fig 12. I-kMeans vs. k-Means algorithms in terms of objective function (bars) and running time (lines) for JPL dataset.

Below we present additional results on a number of additional datasets from the UCR time-series archive. The running time recorded for I-kMeans is the time required to achieve the *best* result. Therefore, the speedup measured here is pessimistic, since I-kMeans typically outperforms k-Means in very early levels. We observe an average speedup of 3 times against the traditional k-Means and a general improvement in the objective function. Only in the *earthquake* dataset the cumulative time is more than k-Means. This happens because the algorithm has to traverse the majority of the levels in order to perform the *optimal* clustering. However, in this case the prolonged execution time can be balanced by the significant improvement in the objective function.

Table 3. Performance of I-kMeans on additional datasets. Smaller numbers indicate better performance

Dataset	Obj. k-Means	Obj. I-kMeans	Time k-Means	Time I-kMeans	Speed Up
Ballbeam	6328.21	**6065.61**	5.83	**4.30**	1.36
earthquake	110159	**108887**	**12.9**	15.19	0.85
sunspot	$4377E^{6}$	**$4361E^{6}$**	7.45	**3.07**	3.36
spot_exRates	**2496.66**	2497.47	6.83	**2.71**	2.52
powerplant	$9783E^{8}$	**$9584E^{8}$**	10.33	**3.07**	3.36
evaporator	$6303E^{3}$	**$6281E^{3}$**	21.33	**6.59**	3.24
memory	$1921E^{4}$	**$1916E^{4}$**	19.48	**5.91**	3.29

5 Extension to a General Framework

We have seen that our anytime algorithm out-performs k-Means in terms of clustering quality and running time. We will now extend the approach and generalize it to a

framework that can adapt to a much broader range of algorithms. More specifically, we apply prominent alternatives on the frame of our approach, the clustering algorithm, as well as its essence, the decomposition method.

We demonstrate the generality of the framework by two examples. Firstly, we use another widely-used iterative refinement algorithm – the EM algorithm, in place of the k-Means algorithm. We call this version of EM the I-EM algorithm. Next, instead of the Haar wavelet decomposition, we utilize an equally well-studied decomposition method, the Discrete Fourier Transform (DFT), on the I-kMeans algorithm. Both approaches have shown to outperform their k-Means or EM counterparts. In general, we can use any combination of iterative refining clustering algorithm and multi-resolution decomposition methods in our framework.

5.1 I-EM with Expectation Maximization (EM)

The EM algorithm with Gaussian Mixtures is very similar to k-Means algorithm introduced in Table 1. As with k-Means, the algorithm begins with an initial guess to the cluster centers (the "E" or Expectation step), and iteratively refines them (the "M" or maximization step). The major distinction is that k-Means attempts to model the data as a collection of k spherical regions, with every data object belonging to exactly one cluster. In contrast, EM models the data as a collection of k Gaussians, with every data object having some degree of membership in each cluster (in fact, although Gaussian models are most common, other distributions are possible). The major advantage of EM over k-Means is its ability to model a much richer set of cluster shapes. This generality has made EM (and its many variants and extensions) the clustering algorithm of choice in data mining [7] and bioinformatics [17].

5.2 Experimental Results for I-EM

Similar to the application of k-Means, we apply EM for different resolutions of data, and compare the clustering quality and running time with EM on the original data. We use the same datasets and parameters as in k-Means. However, we have to reduce the dimensionality of data to 256, since otherwise the dimensionality-cardinality ratio would be too small for EM to perform well (if at all!). The EM algorithm presents the error as the negative log likehood of data. We can compare the clustering results in a similar fashion as in k-Means, by projecting the results obtained at a lower dimension to the full dimension and computing the error on the original raw data. More specifically, this is achieved by re-computing the centers and the covariance matrix on the full dimension, given the posterior probabilities obtained at a lower dimension. The results are similar to those of k-Means. Fig 13 shows the errors for EM and I-EM algorithms on the JPL datasets. The errors for EM are shown as straight lines for easy visual comparison with I-EM at each level. The results show that I-EM outperforms EM at very early stages (4 or 8 dimensions).

Fig 14 shows the running time for EM and I-EM on JPL datasets. As with the error presentation, the running times for EM are shown as straight lines for easy visual

comparison with I-EM. The vertical dashed line indicates where I-EM starts to out-perform EM (as illustrated in Fig 13, I-EM out-performs EM at every level forward, following the one indicated by the dashed line).

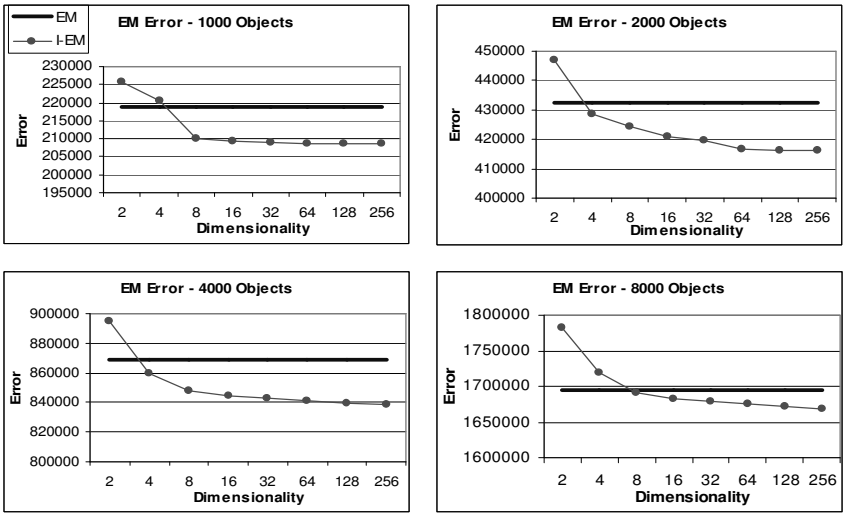

Fig 13. We show the errors for different data cardinalities. The errors for EM are presented as constant lines for easy visual comparison with the I-EM at each level. I-EM out-performs EM at very early stages (4 or 8 dimensions)

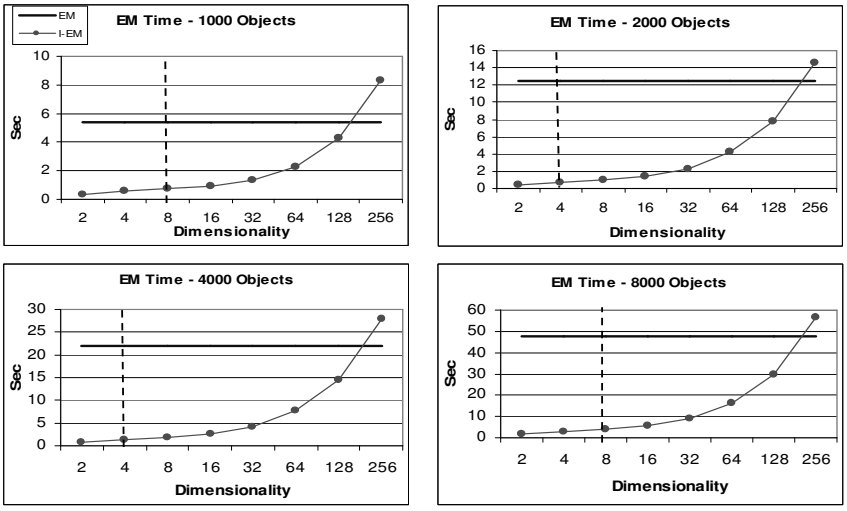

Fig 14. Running times for different data cardinalities. The running times for EM are presented as constant lines for easy visual comparison with the I-EM at each level. The vertical dashed line indicates where I-EM starts to out-perform EM as illustrated in Fig 13

5.3 I-kMeans with Discrete Fourier Transform

As mentioned earlier, the choice of Haar Wavelet as the decomposition method is due to its efficiency and simplicity. In this section we extend the I-kMeans to utilize another equally well-known decomposition method, the Discrete Fourier Transform (DFT) [1, 20].

Similar to the wavelet decomposition, DFT approximates the signal with a linear combination of basis functions. The vital difference between the two decomposition methods is that the wavelets are localized in time, while DFT coefficients represent global contribution of the signal. Fig 15 provides a side-by-side visual comparison of the Haar wavelet and DFT.

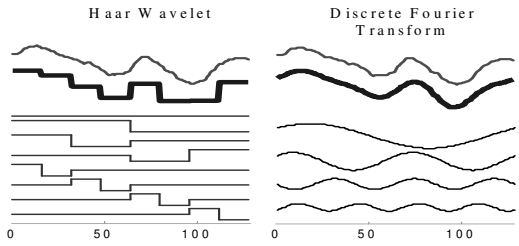

Fig 15. Visual comparison of the Haar Wavelet and the Discrete Fourier Transform. Wavelet coefficients are localized in time, while DFT coefficients represent global contributions to the signal

While the competitiveness of either method has been largely argued in the past, we apply DFT in the algorithm to demonstrate the generality of the framework. As a matter of fact, consistent with the results shown in [15], the superiority of either method is highly data-dependent. In general, however, DFT performs better for smooth signals or sequences that resemble random walks.

5.4 Experimental Results for I-kMeans with DFT

In this section we show the quality of the results of I-kMeans, using DFT as the decomposition method instead of the Haar wavelet. Although there is no clear evidence that one decomposition method is superior than the other, it's certain that using either one of these methods with I-kMeans outperforms the batch k-Means algorithm. Naturally it can be argued that instead of using our iterative method, one might be able to achieve equal-quality results by using a batch algorithm on higher resolution with either decomposition. While this is true to some extent, there is always a higher chance of the clustering being trapped in the local minima. By starting off at lower resolution and re-using the cluster centers each time, we minimize the dilemma with local minima, in addition to the choices of initial centers.

In datasets where the time-series is approximated more faithfully by using Fourier than wavelet decomposition, the quality of the DFT-based incremental approach is slightly better. This experiment suggests that our approach can be tailored to specific applications, by carefully choosing the decomposition that provides the least reconstruction error.

Table 4 shows the results of I-kMeans using DFT.

Table 4. Objective functions for k-Means, I-kMeans with Haar Wavelet, and I-kMeans with DFT. Smaller numbers indicate tighter clusters.

Dataset	Obj. k-Means	Haar I-kMeans	DFT I-kMeans
ballbeam	6328.21	**6065.61**	6096.54
earthquake	110159	108887	**108867**
Sunspot	$4377E^{6}$	$\mathbf{4361E^{6}}$	$4388E^{6}$
spot_exRates	2496.66	2497.47	**2466.28**
powerplant	$9783E^{8}$	$\mathbf{9584E^{8}}$	$11254E^{8}$
evaporator	$6303E^{3}$	$\mathbf{6281E^{3}}$	$6290E^{3}$
Memory	$1921E^{4}$	$\mathbf{1916E^{4}}$	$1803E^{4}$

6 Conclusions and Future Work

We have presented an approach to perform incremental clustering of time-series at various resolutions using multi-resolution decomposition methods. We initially focus our approach on the k-Means clustering algorithm, and then extend the idea to EM. We reuse the final centers at the end of each resolution as the initial centers for the next level of resolution. This approach resolves the dilemma associated with the choices of initial centers and significantly improves the execution time and clustering quality. Our experimental results indicate that this approach yields faster execution time than the traditional k-Means (or EM) approach, in addition to improving the clustering quality of the algorithm. Since it conforms with the observation that time series data can be described with coarser resolutions while still preserving a general shape, the anytime algorithm stabilizes at very early stages, eliminating the needs to operate on high resolutions. In addition, the anytime algorithm allows the user to terminate the program at any stage.

Our extensions of the iterative anytime algorithm on EM and the multi-resolution decomposition on DFT show great promise for generalizing the approach at an even wider scale. More specifically, this anytime approach can be generalized to a framework with a much broader range of algorithms or data mining problem. For future work, we plan to investigate the following:

- Extending our algorithm to other data types. For example, image histograms can be successfully represented as wavelets [6, 23]. Our initial experiments on image histograms show great promise of applying the framework on image data.

- For k-Means, examining the possibility of re-using the results (i.e. objective functions that determine the quality of clustering results) from the previous stages to eliminate the need to re-compute all the distances.

References

1. Agrawal, R., Faloutsos, C. & Swami, A. (1993). Efficient Similarity Search in Sequence Databases. In *proceedings of the 4th Int'l Conference on Foundations of Data Organization and Algorithms*. Chicago, IL, Oct 13-15. pp 69-84.
2. Bradley, P., Fayyad, U., & Reina, C. (1998). Scaling Clustering Algorithms to Large Databases. In *proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining*. New York, NY, Aug 27-31. pp 9-15.
3. Chan, K. & Fu, A. W. (1999). Efficient Time Series Matching by Wavelets. In *proceedings of the 15th IEEE Int'l Conference on Data Engineering*. Sydney, Australia, Mar 23-26. pp 126-133.
4. Chu, S., Keogh, E., Hart, D., Pazzani, M. (2002). Iterative Deepening Dynamic Time Warping for Time Series. In *proceedings of the 2002 IEEE International Conference on Data Mining*. Maebashi City, Japan. Dec 9-12.
5. Ding, C., He, X., Zha, H. & Simon, H. (2002). Adaptive Dimension Reduction for Clustering High Dimensional Data. In *proceedings of the 2002 IEEE Int'l Conference on Data Mining*. Dec 9-12. Maebashi, Japan. pp 147-154.
6. Daubechies, I. (1992). Ten Lectures on Wavelets. *Number 61, in CBMS-NSF Regional Conference Series in Applied Mathematics*, Society for Industrial and Applied Mathematics, Philadelphia.
7. Dempster, A., Laird, N., & Rubin, D. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society, Series B. Vol. 39, No. 1, pp. 1-38.
8. Dumoulin, J. (1998). NSTS 1988 News Reference Manual. http://www.fas.org/spp/civil/sts/
9. Faloutsos, C., Ranganathan, M. & Manolopoulos, Y. (1994). Fast Subsequence Matching in Time-Series Databases. In *proceedings of the ACM SIGMOD Int'l Conference on Management of Data*. Minneapolis, MN, May 25-27. pp 419-429.
10. Fayyad, U., Reina, C. &. Bradley. P (1998). Initialization of Iterative Refinement Clustering Algorithms. In *proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. New York, NY, Aug 27-31. pp 194-198.
11. Grass, J. & Zilberstein, S. (1996). Anytime Algorithm Development Tools. Sigart Artificial Intelligence. Vol 7, No. 2, April. *ACM Press*.
12. Keogh, E. & Pazzani, M. (1998). An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback. In *proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining*. NewYork, NY, Aug 27-31. pp 239-241.
13. Keogh, E., Chakrabarti, K., Pazzani, M. & Mehrotra, S. (2001). Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *proceedings of ACM SIGMOD Conference on Management of Data*. Santa Barbara, CA. pp 151-162.
14. Keogh, E. & Folias, T. (2002). The UCR Time Series Data Mining Archive. [http://www.cs.ucr.edu/~eamonn/TSDMA/index.html].

15. Keogh, E. & Kasetty, S. (2002). On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In *proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. July 23 - 26, 2002. Edmonton, Alberta, Canada. pp 102-111.
16. Korn, F., Jagadish, H. & Faloutsos, C. (1997). Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences. In *proceedings of the ACM SIGMOD Int'l Conference on Management of Data*. Tucson, AZ, May 13-15. pp 289-300.
17. Lawrence, C. & Reilly, A. (1990). An Expectation Maximization (EM) Algorithm for the Identification and Characterization of Common Sites in Unaligned Biopolymer Sequences. *Proteins*, Vol. 7, pp 41-51.
18. McQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observation. L. Le Cam and J. Neyman (Eds.), In *proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, CA. Vol. 1, pp 281-297.
19. Popivanov, I. & Miller, R. J. (2002). Similarity Search over Time Series Data Using Wavelets. *In proceedings of the 18th Int'l Conference on Data Engineering*. San Jose, CA, Feb 26-Mar 1. pp 212-221.
20. Rafiei, Davood & Mendelzon, Alberto. (1998). Efficient Retrieval of Similar Time Sequences Using DFT. In *proceedings of the FODO Conference*. Kobe, Japan, November 1998.
21. Smyth, P,. Wolpert, D. (1997). Anytime Exploratory Data Analysis for Massive Data Sets. In *proceedings of the 3rd Int'l Conference on Knowledge Discovery and Data Mining*. Newport Beach, CA. pp 54-60
22. Shahabi, C., Tian, X. & Zhao, W. (2000). TSA-tree: a Wavelet Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries. In *proceedings of the 12th Int'l Conference on Scientific and Statistical Database Management*. Berlin, Germany, Jul 26-28. pp 55-68.
23. Struzik, Z. & Siebes, A. (1999). The Haar Wavelet Transform in the Time Series Similarity Paradigm. In *proceedings of Principles of Data Mining and Knowledge Discovery*, 3rd European Conference. Prague, Czech Republic, Sept 15-18. pp 12-22.
24. Vlachos, M., Lin, J., Keogh, E. & Gunopulos, D. (2003). A Wavelet-Based Anytime Algorithm for K-Means Clustering of Time Series. In *Workshop on Clustering High Dimensionality Data and Its Applications, at the 3rd SIAM Int'l Conference on Data Mining*. San Francisco, CA. May 1-3.
25. Wu, Y., Agrawal, D. & El Abbadi, A. (2000). A Comparison of DFT and DWT Based Similarity Search in Time-Series Databases. In *proceedings of the 9th ACM Int'l Conference on Information and Knowledge Management*. McLean, VA, Nov 6-11. pp 488-495.
26. Yi, B. & Faloutsos, C. (2000). Fast Time Sequence Indexing for Arbitrary Lp Norms. In *proceedings of the 26th Int'l Conference on Very Large Databases*. Cairo, Egypt, Sept 10-14. pp 385-394.l Database Management. Berlin, Germany, Jul 26-28. pp 55-68.