# Efficiently Finding Arbitrarily Scaled Patterns in Massive Time Series Databases

Eamonn Keogh

University of California - Riverside
Computer Science & Engineering Department
Riverside, CA 92521,USA
eamonn@cs.ucr.edu
www.cs.ucr.edu/~eamonn/

**Abstract.** The problem of efficiently finding patterns in massive time series databases has attracted great interest, and, at least for the Euclidean distance measure, may now be regarded as a solved problem. However in recent years there has been an increasing awareness that Euclidean distance is inappropriate for many real world applications. The limitations of Euclidean distance stems from the fact that it is very sensitive to distortions in the time axis. A partial solution to this problem, Dynamic Time Warping (DTW), aligns the time axis before calculating the Euclidean distance. However, DTW can only address the problem of *local* scaling. As we demonstrate in this work, *uniform* scaling may be just as important in many domains, including applications as diverse as bioinformatics, space telemetry monitoring and motion editing for computer animation. In this work, we demonstrate a novel technique to speed up similarity search under uniform scaling. As we will demonstrate, our technique is simple and intuitive, and can achieve a speedup of 2 to 3 orders of magnitude under realistic settings.

## 1 Introduction

The problem of efficiently finding patterns in massive time series databases has attracted great interest in the database and data mining communities, and, at least for the Euclidean distance measure, may now be regarded as a solved problem [2, 5, 11, 12]. However in recent years there has been an increasing awareness that Euclidean distance is inappropriate for many real world applications [1, 6]. The limitations of Euclidean distance stems from the fact that it is very sensitive to distortions in the time axis. A partial solution to this problem, Dynamic Time Warping (DTW), essentially aligns the time axis before calculating the Euclidean distance. Because of its well-documented lethargy, DTW was deemed impractical for large databases until a recent breakthrough demonstrated that DTW can be indexed [10]. DTW can only address the problem of *local* scaling, however uniform scaling may be just as important in many domains, including applications as diverse as bioinformatics, space telemetry monitoring and motion editing for computer animation.

N. Lavrač et al. (Eds.): PKDD 2003, LNAI 2838, pp. 253–265, 2003.

There exists a handful of techniques that can support similarity search under uniform scaling if the scaling factor is known in advance [3, 9]; however, in most domains it is unlikely that we know the scaling factor. In such instances we must resort to multiple queries, one for each possible scaling factor. Clearly, this is untenable for even moderately large databases. What we really need is a technique that can perform a single efficient query to retrieve all qualifying time series with *any* scaling. This is exactly the contribution of this paper.

The rest of this paper is organized as follows. Section 2 carefully motivates the need for similarity search under uniform scaling, and reviews related work. In Section 3 we introduce our approach to the problem. Section 4 contains an extensive empirical evaluation on 5 real world datasets. Finally, Section 5 contains conclusions and directions for future work.

## 2 Motivating the Need for Uniform Scaling

In addition to the classic Euclidean and Dynamic Time Warping distance measures, the last decade has seen the introduction of dozens of new similarity measures for time series. Recent empirical studies, however, suggest that the majority of these measures are of dubious utility for real world problems [13]. We will therefore take the time to motivate the absolute need for uniform scaling in several real world applications.

### 2.1 Space Shuttle Telemetry Monitoring

The Space Shuttle transmits thousands of sensor readings to Earth at 1mhz or greater during flight. With over 100 missions, averaging 8.6 days in orbit, this massive repository of data constitutes a potential goldmine for engineers wishing understand and predict in-flight anomalies [4]. Consider an engineer wishing to discover all occurrences of a "dipping" event. This event consists of a sudden positive change in yaw, followed by an auto correction by the Shuttle's onboard flight guidance system. Such events can easily be visually located in a small time series, as they form a 'V' pattern. However, in a massive dataset we must resort to a computerized similarity search.

If we create a 'V' shaped query that is 4 minutes long, and search using the Euclidean distance, we correctly find one true event as shown in Fig. 1 A. However, the second and third best matches fail to find the other two "dips". In contrast, if we issue a query for all 'V' shaped patterns in the range of 4 minutes to 6 minutes, we can correctly discover all three such events as shown in Fig. 1 B.

### 2.2 Gene Expression Data

Recent advances in bioinformatics technology have resulted in an explosion of gene expression data to be analyzed [1]. Several of the most important tasks, such as clus-

tering, classification and missing value reconstruction, require similarity matching as a first step. Both Euclidean distance and DTW are used; however, we argue that uniform scaling may be more useful for some tasks and datasets. Consider the two sequences shown in Fig. 2.

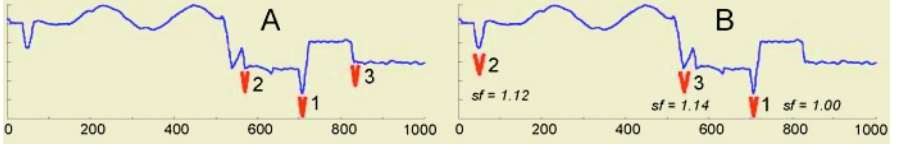


**Fig. 1.** Eight hours of STS-57 Space Shuttle Inertial Sensor Data: A) A 'V' shaped query correctly matches one steep valley in the data, but the second and third best matches fail to find the two other valleys because they happen more slowly. B) A 'V' shaped query that is allowed to rescale itself by up to 50% correctly finds the three valleys. The second and third best matches have a scaling factor (*sf*) of 1.12 and 1.14 respectively

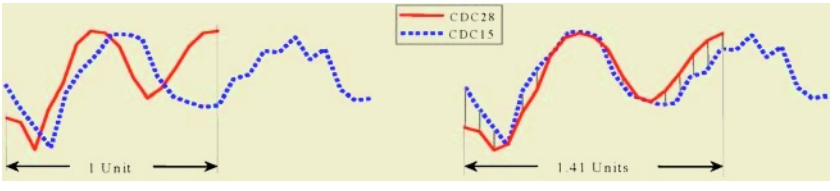


**Fig. 2.** Two yeast cell-cycle gene expression time series, from genes known to be functionally related. (*Left*) Using the original scale, the genes appear to be a poor match. (*Right*) If the shorter time series is rescaled by a scaling factor of 1.41, it becomes a high quality match to the "prefix" of the longer time series

Although the two genes are known to be functionally related [1], the raw time series subjectively appear to be a poor match. Simply rescaling the shorter time series by a factor of 1.41 allows the underlying similarity to be more readily discovered.

We considered other approaches for this problem. Euclidean distance is a very commonly used technique, but it is only defined for time series of the same length. One solution is to normalize the lengths with interpolation; another is to truncate the longer time series. Although DTW is defined for time series of different lengths, interpolation and truncation can also be useful here. In Fig. 3. we show all combinations of possibilities, none of them succeeds in capturing the underlying similarity of the data.

### 2.3 Motion Capture Editing

Motion capture data is increasingly used in video games, movie special effects and gait analysis [6]. The following is a classic problem in this domain. Given two examples of a human performing a task, once slowly, and once quickly, interpolate the motion at any desired speed [20]. Figure 4 shows an example. The problem is non-

trivial because of non-linear effects in human dynamics. Nevertheless correctly aligning the two time series from each instance is a critical first step in solving the problem. This can be achieved manually for a simple movie special effect, but for real time video games, or complex effect shots (i.e, the battle scenes in The Lord of the Rings), automation is required.

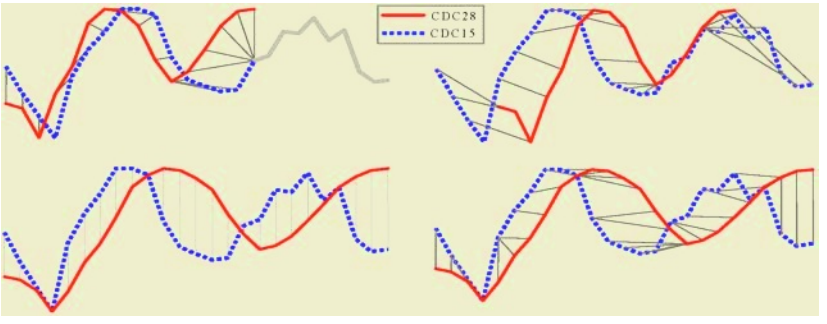


**Fig. 3.** None of the published alternatives to uniform scaling produce intuitive alignments between the two gene expression time series introduced in this section. Clockwise from the top left, DTW after truncating the longer time series, classic DTW, DTW after length normalization, Euclidean distance after length normalization

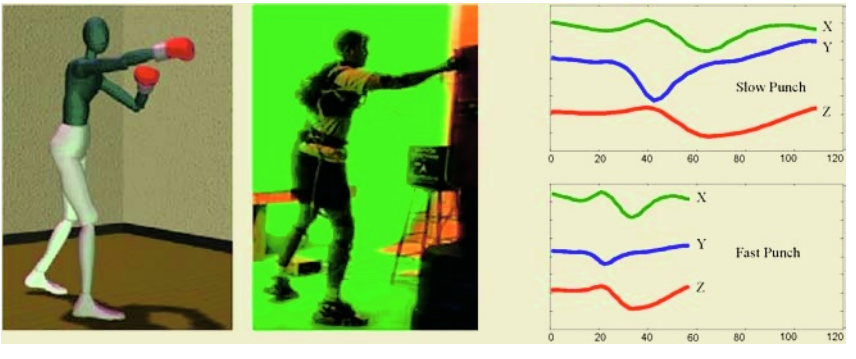


**Fig. 4.** (*left*) A computer animation of a boxer, driven by a motion capture system (*center*). Given that we have captured an example of a fast moment and a slow movement (*right*), an important problem in motion capture editing is to interpolate the movement at any desired speed. Aligning the signals with uniform scaling is a important first step in this process

Having motivated the need for uniform scaling in several domains, we will next consider related work.

## 2.4 Related Work

The past decade has seen literally hundreds of papers on similarity search using the Euclidean distance [2, 5, 11, 12]; useful surveys can be found in [8] and [17]. How-

ever recent years have seen an increasing awareness that the Euclidean distance may be unsuitable for many applications [1, 10, 18, 19].

Many non Euclidean distance measures for time series have been introduced, however, a recent empirical study suggests that most of them are of questionable utility [10]. The only non-Euclidean distance measure that has been forcefully shown to be superior to Euclidean distance is DTW, it's utility has been demonstrated in domains as diverse as bioinformatics [1], chemical engineering, gait analysis [6], speech recognition, meteorology, and robotics. However DTW only considers local stretching and shrinking of the time axis. As we demonstrated in the previous section, uniform scaling may be equally important in many domains.

The utility of uniform scaling has been noted before [9, 14, 15]. However, all previous work has focused on speeding up similarity search, when the scaling factor is *known*. For example, there are systems that can index data of length 200, and support queries of any length from 150 to 200. However the user must specify what length query they wish to run, perhaps a query of length 175. If the user wishes to find the best matching time series, at any length from 150 to 200, they would have to run every possible query, of length 150, 151 ,…, 200 to find the answer. This is clearly untenable. As all these systems claim about one order of magnitude speed up, placing them in a loop and running them 50 times is clearly going to be self defeating. The feature that differentiates our work from all the rest is that we allow a user to issue a single query, and find the best match at *any* scaling. Our proposed technique is unique in this aspect.

## 3 Uniform Scaling

We begin by formally defining the uniform scaling problem.

Suppose we have two time series, a query Q and a candidate match C, of length $n$ and $m$ respectively, where:

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n \tag{1}$$

$$C = c_1, c_2, \ldots, c_j, \ldots, c_m \tag{2}$$

For clarity of presentation we will assume that $n \leq m$, that is to say, C is always longer than or equal to Q, and thus we are only interested in stretching the query to match some prefix of C. This assumption is only to simplify notion and does not preclude matching a time series by shrinking, since we can always reverse the roles of the sequences.

If we wish to compare the two time series, and it happens that $n = m$, we can use the ubiquitous Euclidean distance:

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2} \tag{3}$$

Since the square root function is monotonic and concave, we can remove the square root step and get identical rankings, clustering and classifications. This measure is called the squared Euclidean distance:

$$D(Q,C) \equiv \sum_{i=1}^{n} (q_i - c_i)^2 \tag{4}$$

In addition to the utility of slightly speeding up the calculations, working with this distance measure makes other optimizations possible [13].

If *n* is smaller than *m*, then the distance measures introduced above are not defined. To compare the two time series in this case, we have several choices; we can truncate C, and compare Q to $[c_1,c_2,\ldots, c_n]$, or we can somehow *stretch* Q to be of length *m*, or more generally we can *stretch* Q to be of length *p*, ($n \le p \le m$), truncate off the last *m-p* values of Q, then use squared Euclidean distance. The informal idea behind *stretching* can be captured in the more formal definition of scaling. To scale time series Q to produce a new time series QP of length *p*, the formula is:

$$QP_j = Q_{\lceil j * n/p \rceil}, \ 1 \le j \le p \tag{5}$$

Note that we can quickly obtain any scaling in O(*p*) time. We call the ratio *p*/*n* the *scaling factor* or *sf*. Slightly different definitions of scaling do exist, but they do not affect the results that follow. Fig. 5. visually summarizes the above definitions.

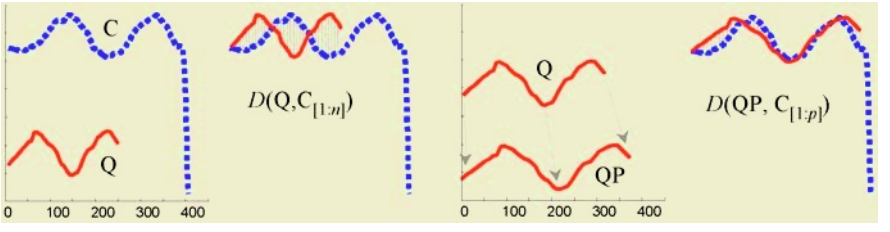


**Fig. 5.** A visual summary of the notation introduced in this section. From (*left*) to (*right*) A candidate time series C, and a shorter query Q. The squared Euclidean distance between Q and the first *n* datapoints in C can be visualized as the sum of the squared lengths of the gray hatch lines. The query Q can be stretched to length *p*, producing a new time series QP. In this case, QP is a good match to the first *p* datapoints in C

### 3.1 Brute Force Search under Uniform Scaling

If we wish to find the best scaled match between Q and C, we can simply test all possible scalings, as illustrated in Table 1.

**Table 1.** An algorithm to find the best scaled match between two time series

```
Algorithm: Test_All_Scalings(Q,C)
  best_match_val      = inf;
  best_scaling_factor = null;
  for p = n to m
       QP = rescale(Q,p);
       distance = squared_Euclidean_distance(QP, C[1..p]);
       if distance <  best_match_val
              best_match_val = distance;
              best_scaling_factor = p/n;
       end;
  end;
return(best_match_val, best_scaling_factor)
```

The algorithm takes only O($p$*($m$-$n$)) time and seems unworthy of any optimization effort. However, when mining real world datasets, rather than having a single candidate time series C, we are typically confronted with massive collection of possible candidate time series, which will denote as **C**. As a motivating example, the MACHO dataset, a collection of star light curve microlensing events, has over 40 million time series [7]. To find the best scaled match to a query Q, in data collection **C**, we can use a brute force algorithm as shown in Table 2.

Note that the time complexity for this algorithm is O(|**C**| * ($m$-$n$)), this is simply untenable for large datasets.

**Table 2.** An algorithm to find the best scaled match to query from a set of possible matches

```
Algorithm: Search_Database_for_Scaled_Match(Q,C)
  overall_best_time_series = null;
  overall_best_match_val   = inf;
  overall_best_scaling     = null;
  for i = 1 to number_of_time_series_in_(C)
       [dist, scale] = Test_All_Scalings(Q,Ci)
       if dist <  overall_best_match_val
              overall_best_time_series = i;
              overall_best_match_val   = dist;
              overall_best_scaling     = scale;
       end;
  end;
return(overall_best_time_series, overall_best_match_val, overall_best_scaling)
```

### 3.2 Speeding up Search with Lower Bounding

To speed up matching under uniform scaling we will rely on the classic idea of lower bounding. The intuition is this: given some technique for quickly calculating the minimum possible distance between the query and a candidate sequence at any possible scaling, we can prune off many calculations. In more detail, we maintain a variable that contains the distance of the best-scaled match encountered thus far. Before calling the subroutine `Test_All_Scalings` on the next candidate time series, we first perform the quick lower bounding test. If the lower bound distance between the candidate and the query is greater than the distance of the best-scaled match already seen, we can simply discarded the candidate from consideration. For clarity, the idea is formalized in Table 3, although the algorithm differs from the algorithm in Table 2 only in the addition of the lower bounding test as a precondition to the subroutine `Test_All_Scalings`.

There are only two important properties of a lower bounding measure:

- It must be fast to compute. A measure that takes as long to compute as `Test_All_Scalings` is of little use. We would like the time complexity to be at most linear in the length of the time series.
- It must be a relatively tight lower bound. A function can achieve a trivial lower bound by always returning zero as the lower bound estimate. However, in order for the algorithm in Table 3 to be effective, we require a method that tightly bounds the value of the best match.

**Table 3.** A modified algorithm for searching for the best match under uniform scaling

```
Algorithm: Faster_Search_Database_for_Scaled_Match(Q,C)
   overall_best_time_series = null;
   overall_best_match_val   = inf;
   overall_best_scaling     = null;
   for i = 1 to number_of_time_series_in_(C)
        if lower_bound_distance(Q,Ci) < overall_best_match_val
               [dist, scale] = Test_All_Scalings(Q,Ci)
                      if dist <  overall_best_match_val
                             overall_best_time_series = i;
                             overall_best_match_val   = dist;
                             overall_best_scaling     = scale;
                      end;
        end;
   end;
return(overall_best_time_series, overall_best_match_val, overall_best_scaling)
```

The idea of speeding up search using lower bounding is not new; in fact, it is the cornerstone of virtually every time series similarity search algorithm. However, while dozens of lower bounding measures are known for Euclidean distance [2, 5, 9, 11, 12], and 3 lower bounding measures known for DTW [10], there are no lower bounding measures in the literature for uniform scaling. In the next section we introduce the first such measure.

### 3.3 Lower Bounding Uniform Scalings

To create a lower bounding distance measure for uniform scaling we will generate a bounding envelope. Bounding envelopes were introduced in [10] to lower bound DTW, and since then they have sparked a flurry of research activity [16, 18, 19]. While the principle is the same here, the definitions of the envelope are very different. In particular, we create two sequences U and L, such that:

$$U_i = \max( c_{\lfloor (i-1)*m/n \rfloor +1}, \ldots, c_{\lfloor i*m/n \rfloor} ) \quad (6)$$

$$L_i = \min( c_{\lfloor (i-1)*m/n \rfloor +1}, \ldots, c_{\lfloor i*m/n \rfloor} ) \quad (7)$$

These sequences can be visualized as bounding the first $n$ points of the time series C. Fig. 6. shows some examples.

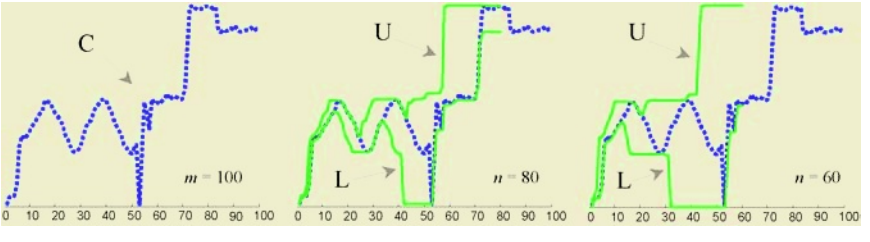


**Fig. 6.** (*Left*) A time series C of length 100. (*Center*) The time series shrouded by upper and lower envelopes U and L with lengths 80. (*Right*) The same time series shrouded by upper and lower envelopes U and L with lengths 60

Having defined the U and L, we can now introduce the lower bounding function, it was originally introduced in [10] for the problem of DTW.

$$LB\_Keogh(Q,C) = \sum_{i=1}^{n} \begin{cases} (q_i - U_i)^2 & if\ q_i > U_i \\ (q_i - L_i)^2 & if\ q_i < L_i \\ 0 & otherwise \end{cases} \quad (8)$$

This function can be visualized as the squared Euclidean distance between any part of the query time series not falling within the envelope and the nearest (orthogonal) corresponding section of the envelope. Fig. 7. illustrates the idea.

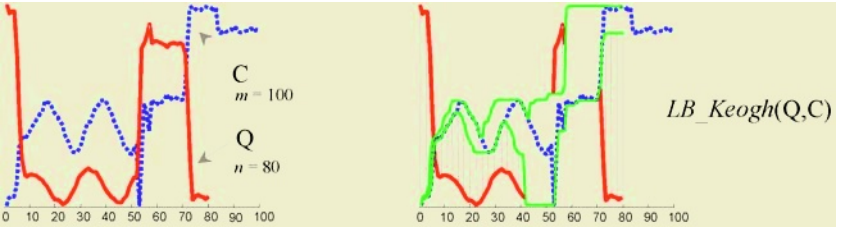


**Fig. 7.** (*Left*) A time series C and a shorter query Q. (*Right*) A visualization of the lower-bounding function *LB_Keogh*(Q,C). Note that any part of query time series Q that falls inside the bounding envelope is ignored. Otherwise the distance corresponds to the sum of the squared straight line distances from the query to the nearest point in the envelope (the gray hatch lines)

We have claimed that *LB_Keogh*(Q,C) lower bounds the squared Euclidean distance between any scaling of Q, and the appropriate prefix of C. The proof is straightforward, we omit it brevity.

### 3.4 Further Optimizations

While *LB_Keogh*(Q,C) is the optimal lower bound for uniform scaling, given only U and L, several further optimizations are possible in the context of similarity search. We will give one such example here, using concrete numbers for clarity. Suppose we are using the algorithm in Table 3 for similarity search, with $n = 100$, and $m = 200$. Further suppose that the best matching time series encountered thus far is at a distance of 10. If we test the lower bound of the next candidate time series and we find it to be 11, we can prune it from the search space. However, if the lower bound is 9 we must call the `Test_All_Scalings` subroutine.

We can observe, however, that although the lower bounding test did fail for the fairly drastic scaling factor of 2 (i.e. 200/100), it would be less likely to do so for smaller scaling factor, say 3/2. We could rescale the query to length 150, rebuild U and L and apply the lower bounding test again. If it happens that the lower bound is now 10 or greater, we could prune all possible scalings from length 150 to 200 from consideration, and only examine the scalings from 100 to 149. Of course, we could apply the above logic recursively to the scalings from 100 to 149, and more generally this suggests doing a binary search over all the scalings. We call this algorithm `Bi-`

nary_Test_All_Scalings, but omit a detailed description since it is rather obvious. Note that we cannot use binary search to speed up the brute force algorithm, since the squared Euclidean distance does not vary monotonically with the scaling factor (in general). We use this optimization in all our experiments below.

## 4 Experimental Results

In this section we test our proposed approach with a comprehensive set of experiments. We compare only to the brute force search algorithm defined in Table 2, because there are no other techniques in existence that support uniform scaling queries, with a single query. To eliminate the possibility of implementation bias [13], we will report the *Pruning Power*, the fraction of times that our approach must call the squared Euclidean distance function.

$$Pruning\ Power = \frac{Number\ of\ calls\ to\ distance\ function\ by\ proposed\ approach}{Number\ of\ calls\ to\ distance\ function\ by\ brute\ force\ search} \tag{13}$$

This measure depends only on the tightness of the lower bounds, and is independent of language, platform, caching or any other implementation details. As an additional sanity check we also measured the CPU time, however since it is almost perfectly correlated with the *Pruning Power*, we will omit it for brevity.

It has been forcefully demonstrated that the quality of lower bounding measures, and therefore the speed of search, can vary greatly depending on the data [13]. We therefore tested our approach on a variety of datasets. Fig 8. shows a sample of each.

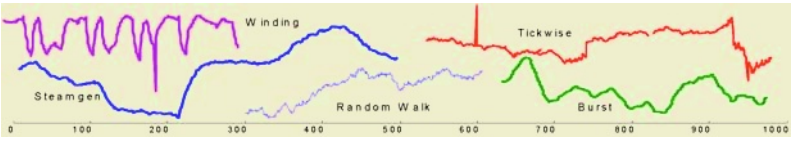


**Fig. 8.** Randomly extracted samples of the time series datasets

Since the speed-up obtained for our approach clearly depends on range of scaling factors and the length of the time series, we will test our approach for the cross product of scaling factors = {1.05, 1.10, 1.15, 1.20, 1.25} and time series candidate lengths of {16, 32, 64, 128, 256}.

We conducted our experiments as follows. We randomly removed a subsequence of the appropriate length from the data to use as a query, then we randomly choose 5,000 other subsequences to act as the database. We then searched for the best scaled match, noting the pruning power. We repeated this 100 times for every combination of scaling factors and candidate lengths. Fig 9. shows the results.

The results are quite impressive, the worst case is a single order of magnitude speed-up, more generally two to three orders of magnitude speedup are observed. Note that, the pruning power seems independent of the candidate time series lengths, but does get worse as the scaling factor increases. This is to be expected, since for large scaling factors the *LB_Keogh* function has relatively little information with which to calculate the lower bound.

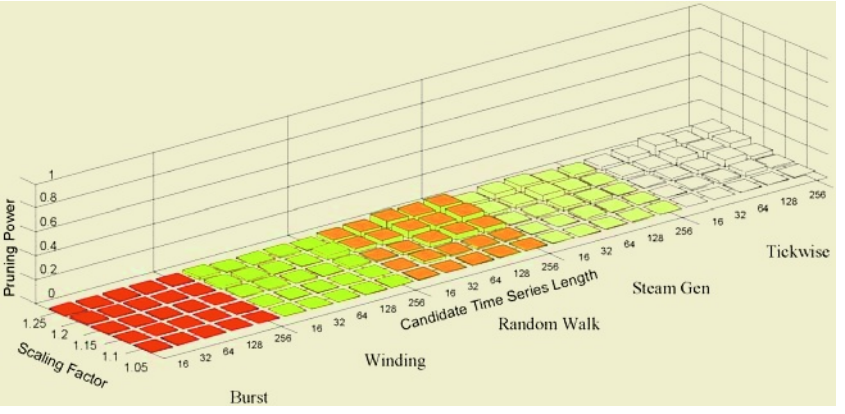


**Fig. 9.** The pruning power of *LB_Keogh* of 5 different datasets, over a range of scaling factors and candidate lengths

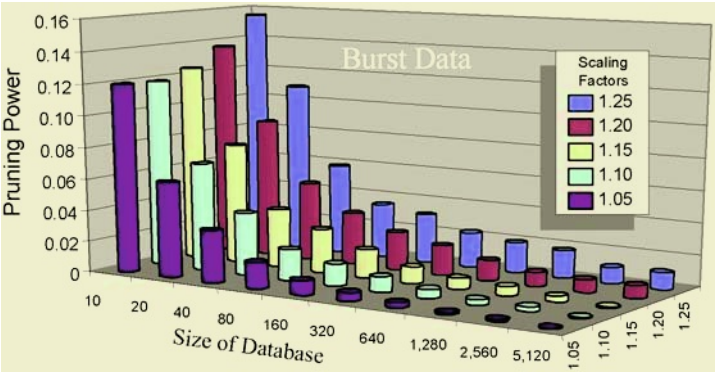


**Fig. 10.** The pruning power of *LB_Keogh* on the burst dataset, over a range of scaling factors and database sizes. Note the scale of the Z-axis is different from that of Fig. 9

As with many indexing techniques, the pruning power of our approach improves with the size of the dataset. The intuition behind this effect is that the larger the dataset, the more likely we are to find a very close match early on in the search, and thus derive the maximum benefit from the lower bound pruning test (the outermost `if` statement in Table 3). To demonstrate this, we repeated the previous experiment for different size datasets. The results for just the burst dataset are shown in Fig. 10.

The results clearly show that as the database size increases, the pruning power improves. This is a very desirable property when mining larger datasets.

## 5 Discussion and Conclusions

We have shown how to dramatically speed up similarity search under uniform warping, however, we have not considered *indexing* under uniform warping. Fortunately

the ability to index the data comes for free! A technique for indexing envelopes under *LB_Keogh* was introduced in [10]. Since then, many other researchers have used this technique and suggested extensions [16, 18, 19] (Note that paper [19] claims to *introduce* the "concept of envelopes", *introduce* must be a typo for *review*, since envelopes were introduced in [10]). This explosion of interest has ensured that indexing of time series envelopes has become a mature technology in only one year. We omitted empirical testing of indexing for brevity and clarity; we simply note that it works exceptionally well. We leave a full discussion for future work.

## Acknowledgements

The author would like to thank Victor Zordan for his help with the motion capture example, Dennis DeCoste at JPL for contributing the Space Shuttle data, and Jessica Lin and Michalis Vlachos for their suggestions.

## References

1. Aach, J. and Church, G. (2001). Aligning gene expression time series with time warping algorithms. Bioinformatics. Volume 17, pp 495-508
2. 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. pp 126-133
3. Chu, K., Lam., S. & Wong, M. (1998) An Efficient Hash-Based Algorithm for Sequence Data Searching. The Computer Journal 41 (6): 402-415
4. Dennis DeCoste and Marie Levine. (2000). Automated Event Detection in Space Instruments: A Case Study Using IPEX-2 Data and Support Vector Machines. *SPIE Conference Astronomical Telescopes and Instrumentation*.
5. Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. *In Proc. ACM SIGMOD Conf*., Minneapolis. pp. 419-429
6. Gavrila, D. M. & Davis,L. S.(1995). Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *International Workshop on Automatic Face- and Gesture-Recognition*
7. Hegland, M., Clarke, W. & Kahn, M. (2002). Mining the MACHO dataset, Computer Physics Communications, Vol 142(1-3), December 15. pp. 22-28
8. Hetland, M. (2003). A Survey of Recent Methods for Efficient Retrieval of Similar Time Sequences. To appear in an Edited Volume*, Data Mining in Time Series Databases*. Published by the World Scientific Publishing Company
9. Kahveci, T. & Singh, A. (2001). Variable length queries for time series data. In *proceedings of the 17th Int'l Conference on Data Engineering*. Heidelberg, Germany, pp 273-282
10. Keogh, E. (2002). Exact indexing of dynamic time warping. In 28th *International Conference on Very Large Data Bases*. Hong Kong. pp 406-417
11. Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra (2000). Dimensionality reduction for fast similarity search in large time series databases. Journal of Knowledge and Information Systems. pp 263-286

12. Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra (2001) Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc of ACM SIGMOD Conference on Management of Data*. pp 151-162
13. Keogh, E. and Kasetty, S. (2002). On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In *the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, Canada. pp 102-111.
14. Park, S., Chu, W. W., Yoon, J. & Hsu, C. (2000). Efficient searches for similar subsequences of different lengths in sequence databases. In *proceedings of the 16th Int'l Conference on Data Engineering*. San Diego, CA, pp 23-32
15. Perng, C., Wang, H., Zhang, S., & Parker, S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. In *proceedings of 16th International Conference on Data Engineering*. pp 33-42
16. Rath, T. & Manmatha, R. (2002): Lower-Bounding of Dynamic Time Warping Distances for Multivariate Time Series. Tech Report MM-40, University of Massachusetts Amherst.
17. Roddick, J. F. and Spiliopoulou, M. (2001). A Survey of Temporal Knowledge Discovery Paradigms and Methods. *IEEE Tran's on Knowledge and Data Engineering*. pp. 750-767
18. Vlachos, M., Kollios, G., & Gunopulos, G. (2002). Discovering similar multidimensional trajectories. In *Proc 18th International Conference on Data Engineering*
19. Zhu, Y. & Shasha, D. (2003). Query by Humming: a Time Series Database Approach. To appear in SIGMOD 2003.
20. Zordan, V. B., Hodgins, J. K., (2002). Motion capture-driven simulations that hit and react, ACM SIGGRAPH Symposium on Computer Animation.