9 Sequence Matching

Sequences are lists of values $S = (x_1, x_2, \ldots, x_k)$, although we shall often think of the same sequence as a continuous function defined on the interval 0-to-1. That is, the sequence S can be thought of as sample values from a continuous function $S(t)$, with $x_i = S(i/k)$.

9.1 Sequence-Matching Problems

In the simplest case, we are given a collection of sequences $\{S_1, S_2, \ldots, S_n\}$, and a query sequence Q, each of the same length. Our problem is to find that sequence S_i whose *distance* from Q is the minimum, where "distance" is defined by the "energy" of the difference of the sequences; i.e., $D(S,T) = \int_0^1 \big(S(t) - T(t)\big)^2 dt$.

For instance, the S_i's might be records of the prices of various stocks, and Q is the price of IBM stock, delayed by one day. If we found some S_i that was very similar to Q, we could use the price of the stock S_i to predict the price of IBM stock the next day, Notes:

- Do not try this at home. Anything easy to mine about stock prices is already being done, and the market has adjusted to whatever knowledge can be gleaned.

- Sequence matching is a great opportunity to violate the Bonferroni principal, since there has to be a "closest sequence." For instance, a famous mistake was looking in the UN book of world statistics to find the statistic that best predicted the Dow-Jones average. It was "cotton production in Bangladesh."

9.2 Fourier Transforms as Indexes for Sequences

We could treat sequences of length k as points in a k-dim vector space, but doing so is not likely to be useful. Usually, k will be so high, that spacial index techniques like kd-trees or R trees will be useless.

The trick adopted by Faloutsos and his colleagues is to map sequences to the first few terms of their Fourier transforms. Formally, the jth term of the *Fourier Expansion* of the function $S(t)$ is $\int_0^1 S(t)e^{2\pi jit} dt$. Recall that the imaginary exponential $e^i x$ is defined to be $\sin x + i \cos x$. Thus, the real and imaginary parts of X_j tell how well $S(t)$ matches sine and cosine functions that have j periods within the interval 0-to-1, i.e., $\sin 2\pi jt$ and $\cos 2\pi jt$.

Example 9.1: Figure 25 suggests a simple function $S(t)$ (solid) and compares it to the single-period sine function (dotted) and single-period cosine function (dashed). The integral of the product $S(t)\sin 2\pi t + iS(t)\cos 2\pi t$ is the complex number X_1. □

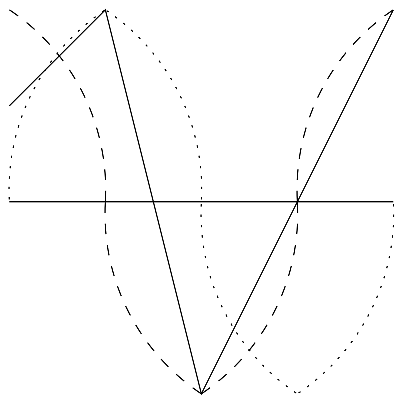

Figure 25: Function $S(t)$ matches the sine only slightly; the cosine better but not perfectly

- Key point: *Parsival's Theorem* states that the energy in a signal $S(t)$ is the same as the energy in its Fourier transform: $\sum_{i \geq 0} |X_i|^2$. Note $|X|$ is the magnitude of a complex number X.

- Key application: If $S(t)$ is actually the difference of two sequences, then the distance between those sequences, which is $\int_0^1 S^2(t)dt$ is the same as the sum of the squares of the magnitudes of the differences of the Fourier coefficients of the two sequences.
- Since the square of a magnitude is always positive, we can find all the sequences whose distance from a given query sequence Q is no more than ϵ^2 by finding all those sequences S the sum of whose magnitudes of the differences of the first m Fourier coefficients is no more than ϵ^2.
  - We shall retrieve some "false drops" — sequences that are close to Q in their first m coefficients, but differ greatly in their higher coefficients.
  - There is some justification for the rule that "all the energy is in the low-order Fourier coefficients," so retrieval based on the first few terms is quite accurate in practice.

9.3 Matching Queries to Sequences of the Same Length

As long as stored sequences and queries are of the same length, we can arrange the sequences in a simple, low-dimension data structure, as follows:

1. Compute the first m Fourier coefficients of each sequence S_i, for some small, fixed m. It is often sufficient to use $m = 2$ or $m = 3$. Since Fourier coefficients are complex, that is the equivalent of a 4- or 6-dim space.
2. Place each sequence S_i in this space according to the values of its first m Fourier coefficients. The points themselves are stored in some suitable multidimensional data structure.
3. To search for all sequences whose distance from query Q is no more than ϵ^2, compute the first m Fourier coefficients of Q, and look for points in the space no more distant than ϵ from the point corresponding to Q.
4. Since there are false drops, compare each retrieved sequence with Q to be sure that the distance is truly no greater than ϵ^2.

In some applications, it may make sense to "normalize" sequences and queries to make the 0th Fourier coefficient (which is the average value) 0 and perhaps to make the variance (i.e., the "energy") of each sequence the same.

Example 9.2: In Fig. 26(a) are two sequences that have a large difference, but are in some sense the same sequence, shifted and scaled vertically. We could shift them vertically to make their averages be 0, as in Fig. 26(b). However, that change still leaves one sequence always having twice the value of the other. If we also scaled them by multiplying the more slowly varying by 2, they would become the same sequence. □

9.4 Queries That are Shorter than the Sequences

Assume that queries are at least of length w. The *sequential scan* method for finding close (within distance ϵ^2) matches to a query Q is as follows:

1. Store the first m coefficients of the Fourier transform of each subsequence of length w for each sequence. As a result, if sequences are of length k, we store $k - w + 1$ points for each sequence.
2. Given a query Q, take the first w values of Q and compute the Fourier transform of that sequence of length w. Retrieve each sequence S and position p that matches within ϵ.
3. For each such S and p, compare the entire query Q with sequence S starting at position p. If the distance is no more than ϵ^2, report the match.

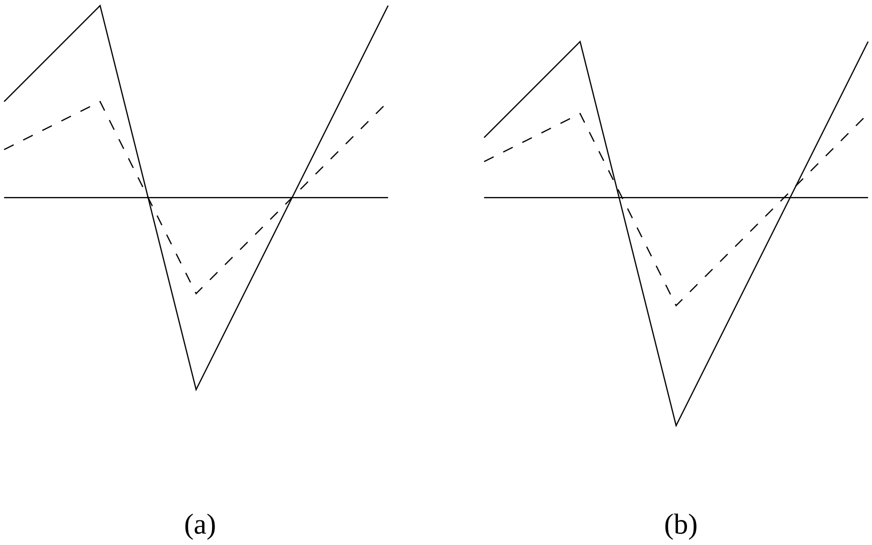

Figure 26: Shifting and scaling can make two sequence that look rather different become the same

9.5 Trails

The problem with sequential-scan is that the number of points that must be stored is almost the product of the number of sequences and the length of the sequences. The FRM paper takes advantage of the fact that as we slide a window of length w across a sequence S, the low-order Fourier coefficients will not vary too rapidly. We thus get a *trail* if we plot the points corresponding to consecutive windows of length w, as suggested in Fig. 27.

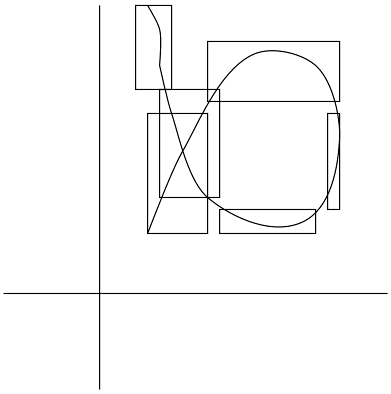

Figure 27: Trials and the rectangles that bound them

Instead of storing individual points, we store rectangles (in the appropriate number of dimensions) that bound the points in one segment of the trail, which is also suggested by Fig. 27. A rectangle can be stored using two opposite vertices, for example. Thus, if rectangles tend to represent many points, then the space ued to store the rectangles is much less than the space needed to store individual points.

- To retrieve the points whose distance from a query Q of length w is no more than ϵ, find the rectangles that are within ϵ of Q. Match Q only against the sequences that correspond to at least one retrieved rectangle, and only at begining positions represented by those rectangles.
- Partitioning trails into rectangles is an opimization problem. FRM uses as the cost of storing a rectangle whose side in dimension i is L_i (as a fraction of the length of the total distance along the ith axis): $\prod_i (L + i + 0.5)$. For example, a rectangle of fractional sides 1/4 and 1/3 would have cost

(3/4)(5/6) = 5/8. Start from the beginning of the trail, and form rectangles by adding points as we treavel along the trail. When deciding whether or not to add another point to the current rectangle, make sure that the cost of the new rectangle per point covered decreases. If it increases, then start a new rectangle instead.

9.6 Matching Queries of Arbitrary Length

If the query Q has length w, just find the rectangles within distance ϵ from Q, as discussed in Section 9.5. Note that Q may actually be within a rectangle, and even so, Q may be distant from actual points on the trail, as suggested in Fig. 28.

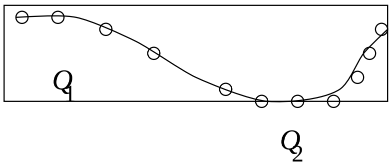

Figure 28: Two examples of queries near a rectangle; note that the query inside the rectangle is actually further from the points on the trail than the query outside the rectangle, but both need to be considered

However, if the query Q has length greater than w, say pw for some constant $p > 1$, there are several things we can do:

1. Search for just the first w values of Q, retrieve the matching sequences and their positions, and then compare the entire Q with the sequence starting at that position. This method is like sequential scan, but it takes advantage of the storage efficiency of trails.
2. Search for the rectangles that are sufficiently close (within ϵ) to the first w values of Q, the next w values of Q, and so on. Only a sequence that has at least one rectangle that is within ϵ of *each* of these subsequences of Q is a candidate match. Check the candidate matches.
3. Like (2), but check a sequence S only if it has a rectangle within distance $\epsilon/\sqrt{p}$ from *at least one* of the p subsequences of Q. Note that the distance of sequences must be at most ϵ^2, and if all p subsequences of Q are at least $\epsilon/\sqrt{p}$ from any point on the trail of S, then $p(\epsilon/\sqrt{p})^2 = \epsilon^2$ is a lower bound on the distance between Q and any subsequence of S.