Partitioning Event Sequences into Regular and Accelerating Sections

February 2019

1 Definitions

An *event sequence* is a set $X = \{x_1, \dots, x_n\}$ of real numbers where $x_1 < \dots < x_n$. Each element $x_i \in X$ is called a *timestamp*. Define $d_i \coloneqq x_{i+1} - x_i$ and $a_i \coloneqq d_{i+1} - d_i$, and denote subsequences of X by $X[i:j] \coloneqq \{x_i, \dots, x_j\}$.

A *partition* of X is an ordered set $P = \{p_1 = 1, p_2, \dots p_k = n\}$ of *split indices* with $1 \le p_i \le n$. Let $\mathcal{P}_X$ be the set of all partitions of X.

2 Objective

Given an event sequence X, we would like to detect sections where the intervals between consecutive timestamps are either regular or accelerating/decelerating in length. In other words, we want to find the most reasonable way to partition the sequence and label each section appropriately.

Define the cost functions

$$r(j,k) \coloneqq \min_{d \in \mathbb{R}} \sum_{i=j}^{k-1} |d_i - d|,$$

$$a(j,k) \coloneqq \min_{a \in \mathbb{R}} \sum_{i=j}^{k-2} |a_i - a|$$

which measure the errors of $X[j:k]$ against the best fitting regular and constant accelerating sequences, respectively. Then, we can define the overall cost of a partition P as

$$f(X,P) \coloneqq \sum_{j=1}^{|P|-1} \min\left(r(p_j, p_{j+1}), a(p_j, p_{j+1})\right).$$

In order to prevent the cost function from always assigning a lower cost to more complex partitions, we can also penalize exceedingly fine partitions by adding a regularization term. Formally then, our goal is to compute

$$\operatorname*{arg\,min}_{P \in \mathcal{P}_X} f(X,P) + \lambda |P|$$

where λ is a parameter which controls the strength of the regularization.

3 Algorithm

We will outline an algorithm to find the optimal partition in polynomial time, using a dynamic programming approach.

Let $D[i]$ be the minimum cost of a partition of $X[1 : i]$, which can be computed as

$$D[i] = \min_{k<i}\{D[k] + \min\left[r(k, i), a(k, i)\right]\} + \lambda. \tag{1}$$

We start by setting $D[1] = D[2] = 0$, since a subsequence of fewer than 3 timestamps cannot form a partition segment. Then for each $i > 2$, we compute $D[i]$ according to (1) and store the partition which achieves this cost. The final solution will be the partition which achieves a cost of $D[n]$.

4 Complexity

We will show that the algorithm described above runs in $O(n^2)$ time and $O(n^2)$ space.

At each iteration of the algorithm, we must compute $D[i]$, which involves computing $r(k, i)$ and $a(k, i)$ for each $k < i$. If we precompute all the d_i's and a_i's beforehand, then this can be done in constant time for each k, so each iteration of the algorithm requires $O(i)$ time overall. Thus, over all n iterations the algorithm takes $O\left(\sum_{i=1}^{n} i\right) = O(n^2)$ time.

Similarly, at each iteration we must store the optimal partition of $X[1 : i]$ found at that iteration, which can contain no more than i split points. Thus, the overall space required is also $O(n^2)$.