K-Nearest Neighbor Search in KMeD2 Temporal Indexing Module

Sanghyun Park

December 27, 2000

Similarity search in sequence databases can be classified into *range search* and *k-nearest neighbor search*. To perform range search, a user specifies a query sequence $\vec{q}$ with an appropriate distance tolerance ε. Then, the query processing system retrieves all the (sub)sequences whose distances to $\vec{q}$ are within ε. However, it is not easy to determine the appropriate value of ε. For this reason, most of applications support k-nearest neighbor search for which a user specifies a query sequence with the required number of answers. This article explains the k-nearest neighbor searching algorithm of KMeD2 temporal indexing module.

Arrays of `ans` and `dst` are maintained for the nearest k subsequences encountered so far, and their distances to $\vec{q}$, respectively. These are sorted in order of increasing distance. A subsequence becomes a candidate whenever less than k subsequences have been encountered, or whenever its lower-bound distance to $\vec{q}$ is less than the distance of the k-th subsequence currently in the answer set. For a candidate, we extract the actual subsequence $\vec{x}$ from the database and compute the distance between $\vec{x}$ and $\vec{q}$ using the original similarity measure (i.e. time warping distance). Then, the arrays of `ans` and `dst` are updated accordingly. The algorithm is described in Algorithm 1.

Algorithm 1: k-nearest neighbor searching algorithm NNS

Input : query sequence $\vec{q}$, the number of answers k
Output: array of subsequences, array of distances

ans := new array of k-subsequences;
dst := new array of k-distances;

for *each subsequence* $\vec{x}^C$ *found in the suffix tree* **do**
    lb := GetLowerBoundDistance($\vec{x}^C$, $\vec{q}$) ;
    if *less than* k *subsequences are encountered so far or* lb *is less than the largest distance in* dst **then**
        Extract the corresponding subsequence $\vec{x}$ from the database;
        d := GetDistance($\vec{x}$, $\vec{q}$);
        if *less than* k *subsequences are encountered so far or* d *is less than the largest distance in* dst **then**
            Insert $\vec{x}$ into ans and d into dst;
            Sort ans and dst in order of decreasing distance;
        if *more than* k *subsequences are now stored in* ans **then**
            Kick out the subsequence with the largest distance;

return ans and dst;