The Frequency Median Sequence and its Construction

Frank Hadlock
Computer Science Department
Tennessee Tech University
Cookeville TN, USA

Brandon Malone
Computer Science Department
Tennessee Tech University
Cookeville TN, USA

Abstract *- A graph model based on single character operations is developed for constructing the median of two strings X and Y. Vertices represent stages of median construction and edges correspond to processing one or both characters of the two strings. In this paper, the median is defined as a string having the median character frequency of each character over the two strings and which minimizes the maximum distance to either string. While the graph has O(mn) vertices (product of the lengths of the two strings), an O(d·p·lg(p)) algorithm is presented for finding a median of length p where d = D_E(X, Y) is the Levenstein or edit distance between X and Y. The algorithm uses a character frequency based bound and a priority queue to find a median.*

Keywords: sequence, median, edit distance, algorithm, priority queue.

1 Introduction

The problem of finding the median of a string set S is defined in terms of some distance measure between strings, such as Levenstein or edit distance. Given two strings X and Y, the edit distance $D_E(X,Y)$ is defined as the cost of the minimum cost transformation T of X → Y, where T is a sequence of single character edit operations (insertion, deletion, equal and unequal replacement). Operations have uniform cost except equal replacement has zero cost.

1.1 Applications

Applications of edit distance and the median string arise in pattern recognition. Speech, EKG signal and other waveforms can be mapped into strings by discretizing frequency or slope and replacing the original object by a string of interval character tags. Mapping these objects into character strings enables the use of edit distance as a basis for classification or recognition. Given a granule or string class Ç corresponding to a phoneme, ectopic heart beat, or pen stroke in handwriting recognition, there is a need for a representative *R* of granule Ç which can be used, along with the distance measure, to classify instances from Ç using the criteria that τ belongs to Ç if τ lies within an interval (based on the distance measure) of Ç's representative *R*. Different candidates for class representative *R* are the longest common subsequence (LCS), the shortest common supersequence (SCS), and the median (*M*). The general problem of determining the absolute optimum candidate, LCS, SCS or *M*, has been shown to be intractable [1],[2],[3] except in the "trivial" case in which Ç contains only two strings, X and Y. In this case, polynomial time algorithms based on dynamic programming can produce an optimum candidate.

1.2 Edit Distance

Faster algorithms than dynamic programming have been developed for edit distance, including a priority queue based algorithm which models the transformation of string X into Y by a graph. For prefixes $X_i = x_1x_2 \ldots x_m$ and $Y_j = y_1y_2 \ldots y_n$, the graph employs a vertex to represent prefix X_i has been transformed into prefix Y_j. The start vertex corresponds to the transformation of the empty prefix of X into the empty prefix of Y, while the goal vertex corresponds to the transformation of X into Y. Directed edges correspond to edit operations weighted with the edit cost with the weight of replacement edges being zero or one, depending on whether the last character x_i of the X prefix equals the last character y_j of the Y prefix or is different. Transformation cost becomes the length of the corresponding path from start vertex to goal, and edit distance becomes the length of the shortest path from start to goal. The shortest path is found by a priority queue algorithm with priorities based on relative character frequencies of the suffixes and takes O(d·n·log(n)) time where n is the length of the longer string, d is the edit distance, and log(n) is overhead due to the priority queue. A similar algorithm [3] has been developed for finding the LCS of two strings with the same runtime requirements.

2 Median String

In this paper we define the median with both a distance and a character frequency constraint so that a lower bound on the distance between the median suffix and each of X and Y suffixes can be calculated and used to prioritize vertices of the construction graph.

2.1 Frequency Median

Given a set of strings C, the *set median* is a string from C which minimizes the sum of the distances to the other strings and can be computed in polynomial time. The *median* is a string which may or may not belong to C and which minimizes the sum of the distances to all strings in C. For the purpose of developing a character frequency based priority for suffix pairs, we define f(X,c) = number of occurrences of character c in string X. The *frequency median* of two strings X,Y is defined be a string M_f such that

- f(M, c) = median{ f(X,c), f(Y,c) } for each character c in Σ
- if $D_E(X,Y)$ is even and $D_E(M_f, X) + D_E(M_f, Y) \leq D_E(X,Y)$ then $D_E(M_f, X) \leq D_E(X,Y)/2$ and $D_E(M, X) \leq D_E(X,Y)/2$

2.2 Uniqueness

Neither the median nor the frequency median are unique. For example, if X = aacc, Y = ccaa, there are 4 frequency medians: acac, acca, caca and caac. All have median frequencies, are 2 distant from X and Y and so are frequency medians. They are also medians of X,Y. X and Y are themselves medians as well as frequency medians. This example is an extreme case with two strings of length 4 and of edit distance 4. The more usual case would be strings close together for which the frequency median would be more well defined. Consider X = acgta, Y = aactta of edit distance 3 apart. A frequency median is acatt which has median frequency and is 2 distant from X and Y and is unique.

2.3 Assumptions

To construct a frequency median M, the following assumptions are made.

- Every character c of M occurs in X or Y or both (the converse is not true). As a consequence, median construction involves scanning X and Y asynchronously, character by character, from left to right. Single character edit operations, {p – pause, i = insert, d = delete} are used to scan the strings and construct the median. The pause operation does not advance the scan of the respective string and does not alter the median candidate under construction. The insert (or copy) advances the scan and appends the scanned character to the median candidate. The delete (or discard) advances the scan and discards the scanned character.
- Any character c of X or Y can be included in M_{f+} or deleted meaning that character scan can advance independently in each string. As a consequence, there are eight transitions in median construction consisting of the operation pairs {<i,i>, <i,p>,<i,d>,<p,i>,<p,d>,<d,i>,<d,p>,<d,d>} where {<p,p>} is omitted from the operation pairs on X,Y.
- If $x_{i+1} = y_{j+1}$, then operation pair <i,i> only is performed on the median candidate; this is modeled by an edge from vertex $v_{i,j}$ representing prefixes X_i, Y_j to $v_{i+1,j+1}$ representing prefixes X_{i+1}, Y_{j+1}, advancing the scan of X and Y.
- If $x_i \neq y_j$ then each of the other seven transitions is tested and applied. If the resulting median candidate satisfies the bound constraints, a queue element is created, a priority based on distance and character frequency is assigned, and the element is inserted into a priority queue.

2.4 Construction Graph

The median construction graph is similar to the edit distance graph [3] in that the vertex set has a vertex $v_{k,j}$ corresponding to each prefix pair X_i , Y_j of X, Y. The start vertex corresponds to the empty prefix pair. Unlike the edit distance graph, the vertex corresponding to the complete prefix pair X_m , Y_n , is not a goal vertex unless X = Y in which case the median would coincide with X and Y. Edges of the construction graph correspond to operation pairs as follows. From vertex $v_{i,j}$, the following edges are defined:

- if $x_{i+1} = y_{j+1}$, then there is a single edge (arc) to vertex $v_{i+1,j+1}$ corresponding to the <i,i> operation
- if $x_i \neq y_j$, then there are
  - three edges to vertex $v_{i+1,j+1}$ corresponding to the <d,i>, <i,d> and <d,d> operations
  - two edges to vertex $v_{i,j+1}$ corresponding to the <p,i> and <p,d>
  - two edges to vertex $v_{i+1,j}$ corresponding to the <i,p> and <d,p>

A typical vertex $V_{i,j}$ with originating edges is shown in Figure 1.

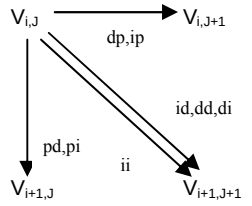

Figure 1 Median Construction Graph
Vertices correspond to prefix pairs X_i , Y_j
Edges correspond to character operation pairs on X and Y : d for delete, I for insert, p for pause.

2.5 Frequency Bounds and Distance Constraints

Let X^i and Y^j be the suffixes $x_{i+1}..x_m$ and $y_{j+1}..y_n$ of X and Y. For any character c, the suffix frequencies, $f(X^i,c)$ and $f(Y^j,c)$ can be calculated and stored in a table [2] by a right to left scan. Character frequencies are assigned to the frequency median M_f for any character c as the median frequency over X and Y. If m is a median candidate as a prefix of M_f, its character frequencies can be computed from m. An extension m' of m is any string over the alphabet which supplements the frequencies of m with respect to M_f. Thus the suffix frequency of character c of any extension m' of m can be calculated as $f(M_f,c) - f(m,c)$. The edit distances $D_E(X_i,m)$ and $D_E(Y_j,m)$ are calculated in the construction of m and stored in the priority queue element containing m. If m' is any extension of m, a lower bound $d_E(X^i,m')$ on the distance $D_E(X^i,m')$ is calculated as in [2] from the character frequencies (maximum of the sum of the excesses versus the sum of the deficiencies over the alphabet). A lower bound $d_E(Y^j,m')$ is calculated in the same fashion. A distance constraint on a median candidate m is that $D_E(X_i,m) + d_E(X^i,m') + D_E(Y_j,m) + d_E(Y^j,m') \leq D_E(X,Y)$. This ensures that M_f satisfies the distance constraint when it is found. By assigning a priority of $D_E(X_i,m) + d_E(X^i,m') + D_E(Y_j,m) + d_E(Y^j,m')$ and selecting elements in order of increasing priority, the selection process favors median candidates with minimum total distance to X and Y.

3 A Priority-based Algorithm for the Median

3.1 Queue Elements

The algorithm uses a queue Q with elements<i,j, b, dx, dy, m>. where i is the index of the last X character scanned, j is the index of the last Y character scanned, b is the suffix bound on the distance of any extension of median prefix m to either X or Y suffix, and dx and dy are the distances of m from X and Y. An element of Q is assigned a priority of $b + \max\{dx,dy\}$.

3.2 Algorithm Pseudocode

- Input = string X of length m, string Y of length n
- Compute Frequency Tables $f(X_i,c)$, $f(Y_j,c)$ for i = 1..m, j = 1..n, and character c.
- Set frequencies of frequency median : $f(M_f,c) = (f(X,c) + f(Y,c)) / 2$ if $f(X,c) + f(Y,c)$ is even else alternate between X and Y as to closer frequency of M_f for odd sum
- Initial suffix bound b_0 for the frequency median of X,Y is h(0,0) = maximum over X and Y of the sum of absolute frequency differences over the alphabet.
- Initialize queue Q with $<0,0,b_0,0,0,\lambda>$
- Done = false
- while not done
  - v = Q.GetHead
  - cur_i = v.i, cur_j = v.j
  - xOK = i < m, yOK = j < n
  - if xOK then xchr = X_{i+1} ; if yOK then ychr = Y_{j+1}
  - if xOK and yOK then
    - if xchr = ychr then
      - u = Expand(v, "ii")
      - SaveOrDiscard(u)
  - else
    - for each operation pair p in {pi,pd,dd,dp,di,id,ip}
      - u = Expand(v, p)
      - SaveOrDiscard(u)

3.3 Examples

X and Y	X : acgta, Y :aactta
X Frequencies	a 2 1 1 1 1 0 c 1 1 0 0 0 0 g 1 1 1 0 0 0 t 1 1 1 1 0 0
Y Frequencies	a 3 2 1 1 1 1 0 c 1 1 1 0 0 0 0 g 0 0 0 0 0 0 0 t 2 2 2 2 1 0 0
Initial Queue	<0,0,2,0,0, λ >
Expand <0,0,2,0,0,λ>	Insert : ii -> <1, 1, 2, 0,0,a> in queue
Expand <1,1,2,0,0,a>	Nbr <2, 1, 2, 1,0,a> already generated, Insert : ip -> <2, 1, 2, 0,1,ac> in queue Nbr <2, 2, 2, 0,1,ac> already generated, Insert : di -> <2, 2, 2, 1,0,aa> in queue Nbr <1, 2, 2, 1,0,aa> already generated, Nbr <1, 2, 2, 0,1,a> already generated Nbr <2, 2, 2, 1,1,a> already generated
Expand <2,1,2,0,1,ac>	Nbr <3, 1, 2, 1,1,ac> already generated, Nbr <3, 1, 2, 0,2,acg> exceeded character frequency constraints, Nbr <3, 2, 2, 0,1,acg> exceeded character frequency constraints Insert : di -> <3, 2, 2, 1,1,aca> in queue, Nbr <2, 2, 2, 1,1,aca> already generated Nbr <2, 2, 2, 0,1,ac> already generated, Nbr <3, 2, 2, 1,1,ac> already generated
Expand <2,2,2,1,0,aa>	Nbr <3, 2, 2, 1,0,aa> already generated, Nbr <3, 2, 2, 1,1,aag> exceeded character frequency constraints, Nbr <3, 3, 2, 1,1,aag> exceeded character frequency constraints Insert : di -> <3, 3, 2, 1,0,aac> in queue, Nbr <2, 3, 2, 2,0,aac> has exceeded distance bounds, Nbr <2, 3, 2, 1,1,aa> already generated, Nbr <3, 3, 2, 1,1,aa> already generated
Expand <3,2,2,1,1,aca>	Nbr <4, 2, 2, 1,1,aca> already generated Nbr <4, 2, 1, 1,2,acat> has exceeded distance bounds Insert : id -> <4, 3, 1, 1,1,acat> in queue Nbr <4, 3, 2, 1,1,acac> exceeded character frequency constraints Nbr <3, 3, 2, 2,1,acac> exceeded character frequency constraints Nbr <3, 3, 2, 1,1,aca> already generated Nbr <4, 3, 2, 1,1,aca> already generated
Expand <4,3,1,1,1,acat>	Nbr <5, 3, 1, 1,1,acat> already generated, Nbr <5, 3, 1, 1,2,acata> exceeded character frequency constraints, Nbr <5, 4, 1, 1,1,acata> exceeded character frequency constraints **<5, 4, 0, 1,1,acatt> is median**

Figure 2. Example of frequency median construction

String x	String y	$D_E(x,y)$	steps	Median	m · n	steps
acgta	aactta	2	6	acatt	30	6
acgtacgta	aacttaactta	4	13	aagcttaact	99	12
acgtacgtacgta	aacttaacttaactta	6	23	aagtctgtaacttaac	208	92
acgtacgtacgta	gacggtacggtacggta	4	17	gcgtacggtacgata	221	48

Figure 3. Sample string pairs, Edit distance by priority queue calculation with number of steps, frequency median by priority queue calculation with number of steps versus mn = product of string lengths.

4 References

[1] David Epstein, Zvi Galil, Raffaele Giancarlo, Giuseppe F. Italiano. *Sparse dynamic programming I: linear cost functions.* JACM, v.39,n.3,p.519-545,July 1992

[2] Frank Hadlock. *Minimum Detour Computation of Least Common Subsequence and Shortest Common Supersequence*. Congressus Numerantium, 65:171-180, 1988**.**

[3] Frank Hadlock. *An edit distance based algorithm for string/sequence classification*. 39th Annual SE ACM conference, 2001, pp 18-21.

[4] D.S. Hirschberg. *A linear space algorithm for computing maximal common subsequences.*CACM, v.18 n.6, p341-243, June 1975.

[5] D.S. Hirschberg. *Algorithms for the longest common subsequence problem.* JACM, v.24, n.4, p.665-675, Oct. 1977.

[6] D.Sankoff and J.B. Kruskal, Editors. *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1983.