Fast Retrieval of Electronic Messages That Contain Mistyped Words or Spelling Errors

Jason Tsong-Li Wang, *Member, IEEE*, and Chia-Yo Chang

***Abstract*—This paper presents an index structure for retrieving electronic messages that contain mistyped words or spelling errors. Given a query string (e.g., a search key), we want to find those messages that *approximately* contain the query, i.e., certain inserts, deletes and mismatches are allowed when matching the query with a word (or phrase) in the messages. Our approach is to store the messages sequentially in a database and hash their "fingerprints" into a number of "fingerprint files." When the query is given, its fingerprints are also hashed into the files and a histogram of votes is constructed on the messages. We derive a lower bound, based on which one can prune a large number of nonqualifying messages (i.e., those whose votes are below the lower bound) during searching. The paper presents some experimental results, which demonstrate the effectiveness of the index structure and the lower bound.**

I. Introduction

IN A NETWORKED environment, a large body of information is transmitted through electronic messages [5], [20], [29]. These messages may contain both attributes (e.g., sender, receiver, date, etc.) and text. Much research has been performed in the past for efficient attribute-oriented retrieval [30] and text access methods [10], [11], [15], [17]. While these techniques are good for records and paper documents, they are insufficient when dealing with electronic messages. Very often, an electronic message contains misspelled or mistyped words (e.g., "meeting" is misspelled as "metimg"). Such a situation may arise when the sender of a message types letters carelessly, or he/she cannot memorize the exact words and does not have time (or even is reluctant) to find out them by checking a dictionary.

Spelling errors are also common in on-line airline reservation systems [8], text edit systems [1], [7], [22], [23], [24], program preparation [18], [21] and bibliography databases [3], [9], [13]. Past work for handling the spelling errors has focused on approximate "word-level" retrieval. That is, a set of keywords is first extracted from the original documents (or programs) and stored in a database. Then, the database is searched for the words that most "resemble" a given string. (The given string may be a search key, or a reserved keyword, which itself may be misspelled and may or may not occur in the database.) Word resemblance is generally measured by the edit distance (or simply the *distance*) between two strings, i.e., the minimum number of edit operations (insert, delete and change a character) needed to transform one string to the other [19], [31].[1] (For example, one needs at least two edit operations to obtain the correct spelling "meeting" from the misspelled "metimg.") From the retrieved words, the user is able to locate the original documents from which the words are extracted.

One key difference between the above systems and message retrieval systems is that in a message, there may not exist any keyword. Consider, for example, the following electronic message sent from an advisor to his/her students: "John will present his paper in this week's meeting." Here, the entire message as a whole informs the receivers what to do in the meeting. Since no word in the message appears more frequently than others, it's unlikely to extract keywords from the message based on existing techniques such as term-occurrence counting methods [27]. Published spelling correction methods (e.g., [22], [24], [26], [39]) are not suitable here either since the corrected words may not correspond to what the sender really wants to type. Thus, how to find the entire messages containing words (or phrases) that resemble a given string becomes an important topic in developing networked information retrieval systems.

This paper proposes an index structure, called *fingerprint files*, for retrieving electronic messages containing mistyped words or spelling errors. Fingerprints have been used in many domains (e.g., biology) for sequence alignment [4] and classification [33], [34]. In this paper we extend the fingerprint technique to help speed up database searching. In building the index structure, we first segment each message in the database. Then we generate fingerprints from each segment and hash the fingerprints into the index structure. When a query string (e.g., a search key) is given, to find the messages containing words that resemble the query, we process the query using the same hash functions as for the messages in the database. A histogram of votes on the messages is then constructed. Based on the histogram, we can discard a large number of nonqualifying messages during searching.

The rest of the paper is organized as follows. In Section II, we describe the index structure and the algorithms used to construct the index. Section III presents basic properties of the index structure. Section IV presents a practically useful lower bound suitable for pruning the nonqualifying mes-

Manuscript received April 29, 1995; revised March 8, 1996. This work was supported in part by the National Science Foundation under Grants IRI-9531548 and IRI-9224602, by the New Jersey Institute of Technology under Grant SBR-421280, and by a Grant from the AT&T Foundation.

The authors are with the Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102 USA.

Publisher Item Identifier S 1083-4419(97)02921-X.

[1] A generalized definition allows one to associate a cost function with each edit operation. For the purpose of this work, we assume that all edit operations have unit cost.

sages. Section V discusses experimental results. Section VI concludes the paper.

II. Fingerprint Files

The problem we intend to solve can be formalized as follows: Given a query string Q, a database of messages $\mathcal{D}$ and an integer d, find all messages E in $\mathcal{D}$ where Q occurs in E within distance d. Let the asterisk $*$ represent a variable length don't care (VLDC) [32]. In matching the regular expression $*Q*$ with a message E, the VLDC's may substitute for zero or more characters in E at no cost. The distance between $*Q*$ and E, denoted $\mathrm{dist}(*Q*, E)$, is the minimum number of edit operations used to transform Q to E after an optimal substitution for the VLDC's [32].[2] Thus, our problem can be stated alternatively as follows: find all messages E in $\mathcal{D}$ where $\mathrm{dist}(*Q*, E) \le d$.

A naive approach to solving this problem would be to compare Q with each message in $\mathcal{D}$ using an approximate regular expression matching algorithm (e.g., agrep [38]). However, when the database becomes large, this approach would be very time-consuming. Our approach instead is to build an index structure (or more precisely, a number of *fingerprint files*) as described below, so that when a query string is given, we can quickly eliminate nonqualifying messages from consideration and examine only the promising messages that may possibly contain the query string within the allowed distance.

Let E be a message in $\mathcal{D}$. We take every contiguous substring (or *segment*), denoted by Seg, of length n from E and generate fingerprints from Seg. Each fingerprint is a substring of Seg that always begins with the segment's first character. The lengths of the fingerprints range from 2 to $n-1$. Thus, there are totally $n-2$ fingerprints generated from Seg.

Next, for each fingerprint f of length k, $2 \le k \le n-1$, we use a hash function h_k to hash f into a fingerprint file $\mathcal{F}_k$. In $\mathcal{F}_k$, f is associated with a pair of integers (x, y). This pair serves as the position marker for f, where x indicates that f is generated from a segment of the xth message in $\mathcal{D}$ and y means that the first character of f occurs at the yth position in that message.[3]

1) Example 1: Consider the database $\mathcal{D}$ composed of the following three messages: $E_1 =$ BGENTVIELSEPS, $E_2 =$ SBGENTMENPYMA and $E_3 =$ SPNTVMEENTVSQ.[4] Suppose the length of segments is 6. Then, for example, we obtain the following segments from E_1: BGENTV, GENTVI, ENTVIE, NTVIEL, TVIELS, VIELSE, IELSEP, ELSEPS.

Now consider the segment Seg = BGENTV taken from E_1. We can generate the following fingerprints from Seg: BG, BGE, BGEN, BGENT. Table I shows all fingerprints generated from the segments of E_1.[5] Let $f = XYZ$ be a fingerprint of length 3. Suppose the hash function h_3 is $h_3(f) = (\mathrm{num}(X) \times 26^2 + \mathrm{num}(Y) \times 26^1 + \mathrm{num}(Z)) \bmod 7$, where $\mathrm{num}(X)$ is X's ASCII value minus 64. Fig. 1 shows $\mathcal{F}_3$ for the messages in $\mathcal{D}$.

When the query string Q is given, we segment Q in the same way as for the messages and generate fingerprints from the resulting segments. We then hash the fingerprints, using the same hash functions as for the messages. When a match between Q's fingerprint and a message's fingerprint occurs, we give a vote to an appropriate position on the message. The result is a histogram of votes on the messages in the database. Fig. 2 shows the algorithm Voting used to calculate the votes.

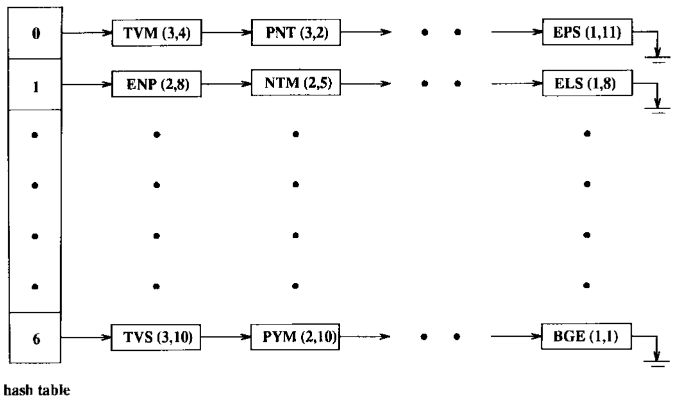

Fig. 1. The fingerprint file $\mathcal{F}_3$ for the database of messages in Example 1.

TABLE I
Fingerprints (of Lengths 2, 3, 4, and 5, Respectively) Generated From the Segments of E_1

2-letter fingerprints		3-letter fingerprints		4-letter fingerprints		5-letter fingerprints	
BG	IE	BGE	IEL	BGEN	IELS	BGENT	IELSE
GE	EL	GEN	ELS	GENT	ELSE	GENTV	ELSEP
EN	LS	ENT	LSE	ENTV	LSEP	ENTVI	LSEPS
NT	SE	NTV	SEP	NTVI	SEPS	NTVIE	
TV	EP	TVI	EPS	TVIE		TVIEL	
VI	PS	VIE		VIEL		VIELS	

```
Input: A query string $Q$, a database of messages $\mathcal{D}$ and $\mathcal{D}$'s fingerprint files.
Output: A histogram of votes on the messages in $\mathcal{D}$.

/* Let $\mathcal{F}$ contain all fingerprints generated from $Q$. */
for each fingerprint $f$ in $\mathcal{F}$ do
  begin
    /* Let the length of $f$ be $k$. */
    hash $f$ using $h_k$ and probe into the fingerprint file $\mathcal{F}_k$;
    for each match between $f$ and a fingerprint $\hat{f}$ in $\mathcal{F}_k$ do
      begin
        /* Let the position marker associated with $\hat{f}$ be $(i, q)$. */
        /* Suppose the first character of $f$ occurs at the $p$th position in $Q$. */
        add one vote to the position $q - p + 1$ in the $i$th message in $\mathcal{D}$;
      end (for);
  end (for);
```

Fig. 2. Algorithm Voting.

[2]For example, in matching $*$TGI$*$ with a string MYALTIHKR, the first asterisk would substitute for MYAL and the second asterisk would substitute for HKR. The distance is 1 (representing the cost of deleting G).

[3]The fingerprint files are built based on the hashing scheme. Therefore one can maintain them just as for hash data structures.

[4]For exposition purposes, we shall represent a message simply as a short string in the paper.

[5]Our algorithms also take into account the fingerprints generated from the last several segments (i.e., LSEPS, SEPS, EPS, PS) of lengths less than n (6 in this example) in E_1. For the segments of lengths $k < n$, the fingerprints generated from them have lengths ranging from 2 to k.

TABLE II
NOTATION AND TERMINOLOGY

- Σ is an alphabet and Λ is a special character not in Σ.
- Σ^* is the Kleene closure of Σ [16].
- $Q \in \Sigma^*$ is the query string and $\mathcal{D}$ is a database of messages $E \in \Sigma^*$; d is the allowed distance between $*Q*$ and the messages in $\mathcal{D}$.
- n denotes the length (i.e., the number of characters) of a segment from which fingerprints are generated.
- $|X|$ denotes the length of a character string X.
- $X[p, q]$ represents the substring of X starting at the pth position and ending at the qth position in X, inclusively; $X[p]$ represents the character at the pth position in X; $X[p, p] = X[p]$.
- $X \bullet Y$ represents the concatenation of two character strings X and Y.

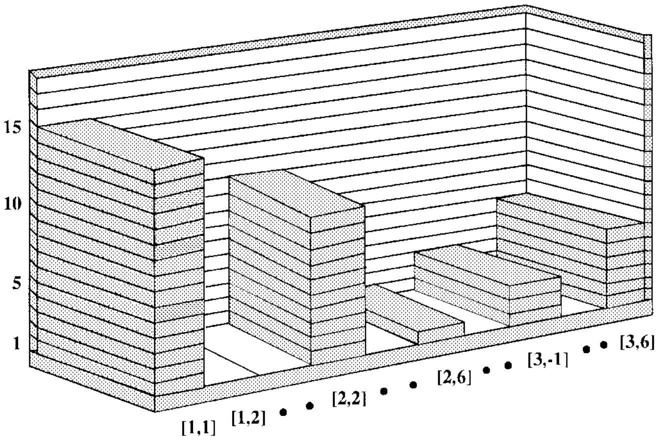

Fig. 3. The histogram obtained after processing the query string Q in Example 2.The y-axis shows votes. Each $[i, q]$ on the x-axis represents the qth position in the ith message in the database in Example 1. The figure displays only the positions with nonzero votes.

2) Example 2: Suppose the given query string Q is BGENTVQEL. Fig. 3 shows the histogram obtained after matching Q's fingerprints with the fingerprints of the messages in Example 1.

3) Remark: The technique of segmenting a string and then generating fingerprints from the segments was originated from Califano and Rigoutsos [4]. The authors applied the technique to find the best alignment position between two DNA sequences. A base sequence is first segmented and its fingerprints are generated. Then a reference sequence is processed and its fingerprints are matched with those of the base sequence. The position with the highest votes on the base sequence indicates the best position at which the first letter of the reference sequence should align. The proposed fingerprint files differ from Califano's technique in two significant ways. First, Califano's technique is based on the table look-up scheme whereas the fingerprint files are based on hashing. There are four characters (A, C, T, G) in a DNA sequence. Califano and Rigoutsos construct a table, in which each entry represents a combination (with a fixed length) of the four characters. If the first character of a fingerprint (with the same length) occurs at the pth position in the base sequence, then store p in the corresponding entry in the table. Using look-up tables is efficient when fingerprints are short and the character alphabet is small. However, when the fingerprints are long and the character alphabet is large (such as that for messages which may contain lower case and upper case English letters as well as special characters such as @), the table look-up scheme becomes infeasible. Second, Califano's technique is designed for aligning two DNA sequences whereas our scheme is aimed to provide a lower bound to accelerate searching a database of messages.

III. BASIC PROPERTIES

A. Terminology and Background

Table II defines the terms and notation we use. We assume $|Q| > 2d + 1$; otherwise our lower bound is simply 0. We first review some definitions for approximate string matching and introduce terms needed in deriving our lower bound. Throughout the section, we let X and Y be two strings in $(\Sigma \cup \{\Lambda\})^*$.

1) Definition 1: [31] A *trace* from X to Y is a triple (T, X, Y) (or simply T when the context is clear) where T is any set of ordered pairs of integers (i, j) satisfying the following conditions:

1. $1 \leq i \leq |X|$ and $1 \leq j \leq |Y|$;
2. for any two distinct pairs (i_1, j_1) and (i_2, j_2) in T, (a) $i_1 \neq i_2$ and $j_1 \neq j_2$; (b) $i_1 < i_2$ iff $j_1 < j_2$.

Let $(i, j) \in T$. We call $Y[j]$ the *image* of $X[i]$. Intuitively, a trace is a graphical specification of which edit operations apply to each character in the two strings. Fig. 4 shows a trace between two strings X and Y. The trace is $\{(2, 3), (3, 8), (4, 10), (5, 11), (7, 15), (12, 16)\}$. Thus, for example, $Y[3] = B$ is the image of $X[2] = B$, and $Y[11] = L$ is the image of $X[5] = N$.

Let T be a trace from string X to string Y. The cost of T, denoted $\gamma(T)$, is the cost of deleting characters of X not touched by a trace line plus the cost of inserting characters of Y not touched by a trace line plus the cost of changing characters in those pairs related by trace lines with differing characters. In [31], Wagner and Fischer showed that the edit

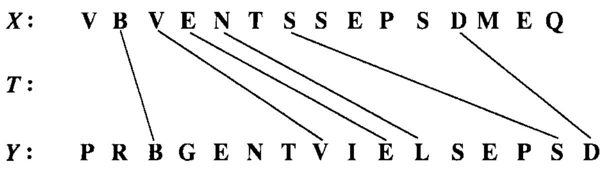
X: V B V E N T S S E P S D M E Q
T:
Y: P R B G E N T V I E L S E P S D

Fig. 4. A trace from string X to string Y; the characters in X not touched by a trace line are to be deleted, the characters in Y not touched by a trace line are to be inserted, and the distinct characters touched by a trace line are to be changed.

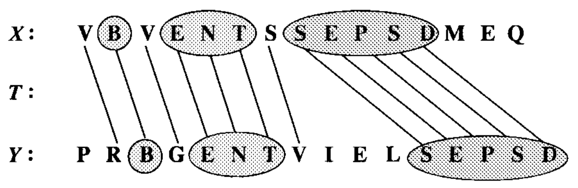
X: V B V E N T S S E P S D M E Q
T:
Y: P R B G E N T V I E L S E P S D

Fig. 5. Illustration of partitions; shaded portions represent the partitions.

distance between the string X and the string Y equals the cost of a minimum cost trace from X to Y. The edit distance between a regular expression $*X*$ and Y equals the cost of a minimum cost trace from X to Y after an optimal substitution for the VLDC's in $*X*$ [32].

2) Definition 2: Let T be a minimum cost trace from string X to string Y. A substring $X[p, q]$, where $1 \le p \le q \le |X|$, is said to be a *partition* on X with respect to T if there exists a substring $Y[r, r+q-p]$, where $1 \le r \le r+q-p \le |Y|$, satisfying the following conditions:

1. $(p+u, r+u) \in T$ and $X[p+u] = Y[r+u]$, for $0 \le u \le q-p$, i.e., $X[p, q] = Y[r, r+q-p]$;
2. $(p-1, r-1) \notin T$ or $X[p-1] \ne Y[r-1]$;
3. $(q+1, r+q-p+1) \notin T$ or $X[q+1] \ne Y[r+q-p+1]$.

Here, $Y[r, r+q-p]$ is said to be a *partition* on Y with respect to T that corresponds to $X[p, q]$.

3) Example 3: Consider again the two strings X and Y in Fig. 4. Fig. 5 shows a minimum cost trace T from X to Y and the resulting partitions. Thus $X[2]$, $X[4, 6]$ and $X[8, 12]$ are partitions on X with respect to T and $Y[3]$, $Y[5, 7]$ and $Y[12, 16]$ are their corresponding partitions on Y.

Note that X and Y must have the same number of partitions with respect to T. Suppose each has l partitions. Then we can represent X as

$$G_T^0 \cdot P_T^1 \cdot G_T^1 \cdot P_T^2 \cdot \cdots \cdot G_T^{l-1} \cdot P_T^l \cdot G_T^l$$

and represent Y as

$$H_T^0 \cdot R_T^1 \cdot H_T^1 \cdot R_T^2 \cdot \cdots \cdot H_T^{l-1} \cdot R_T^l \cdot H_T^l$$

where P_T^u, $1 \le u \le l$, is a partition on X with respect to T, R_T^u is a partition on Y with respect to T that corresponds to P_T^u and $P_T^u = R_T^u$. We refer to the substrings G_T^v, $0 \le v \le l$, as *gaps* on X and the substrings H_T^v as gaps on Y. Note that $|G_T^v| \ge 0$ and $|H_T^v| \ge 0$.

4) Definition 3: Let x be a position in string X. The votes contributed by $X[x]$ after applying the algorithm Voting to the query string X and the database $\{Y\}$ are defined as the votes obtained from the matches between the fingerprints of the segment $X[x, x+n-1]$ and the fingerprints of Y.

5) Definition 4: Let T be a minimum cost trace from string X to string Y. Let $X[p, q]$ be a partition on X with respect to T. Let i be a position in $X[p, q]$, $p \le i \le q$, and let j be a position in Y such that $(i, j) \in T$. The vote contributed by $X[i]$ with respect to T, denoted $\text{vote\_by}_T(X[i])$, is defined as the sum of the votes contributed by $X[i]$ that are added to the position $j - i + 1$ in Y. (These votes are obtained from the matches between the fingerprints of the segment $X[i, i+n-1]$ and the fingerprints of the segment $Y[j, j+n-1]$.) The vote contributed by $X[p, q]$ with respect to T, denoted $\text{vote\_by}_T(X[p, q])$, is defined as $\sum_{p \le i \le q} \text{vote\_by}_T(X[i])$.

Let r be the position in Y such that $(p, r) \in T$. By Definition 4, we know that $\text{vote\_by}_T(X[p, q])$ must be added to the position $r - p + 1$ in Y.

Let $X = G_T^0 \cdot P_T^1 \cdot G_T^1 \cdot P_T^2 \cdot \cdots \cdot G_T^{l-1} \cdot P_T^l \cdot G_T^l$ where P_T^u, $1 \le u \le l$, is a partition on X with respect to T. We note that the positions to which $\text{vote\_by}_T(P_T^u)$, $1 \le u \le l$, are added may overlap. For example, consider again the trace T and strings X and Y in Fig. 5. $\text{vote\_by}_T(X[2])$ is added to the position $3 - 2 + 1 = 2$ in Y; $\text{vote\_by}_T(X[4, 6])$ is also added to the position $5 - 4 + 1 = 2$ in Y.

6) Definition 5: Let p_v, $1 \le v \le m$, $m \le l$, represent the *distinct* positions in string Y to which $\text{vote\_by}_T(P_T^u)$, $1 \le u \le l$, are added. Let $\text{vote}(p_v, T, X)$ be the total votes among $\text{vote\_by}_T(P_T^u)$, $1 \le u \le l$, that are added to the position p_v. The vote of Y with respect to the trace T and string X, denoted $\text{vote}(Y, T, X)$, is defined as $\text{vote}(Y, T, X) = \max_{1 \le v \le m} \text{vote}(p_v, T, X)$.

In deriving our lower bound for pruning the nonqualifying messages in the database $\mathcal{D}$, we shall focus on only the votes contributed by the partitions resulted from a minimum cost trace between the query string Q and a message in $\mathcal{D}$. The following subsection formalizes the notion of votes on the messages and presents techniques for finding the lower bound.

B. A Theoretical Lower Bound

1) Definition 6: Let E be a message in $\mathcal{D}$ and let p be a position in E, $1 \le p \le |E|$. Let $\text{vote}(E[p])$ represent the total votes added to the position p after applying the algorithm Voting to Q and $\mathcal{D}$. The vote of E, denoted $\text{vote}(E)$, is defined to be $\max_{1 \le p \le |E|} \text{vote}(E[p])$.

We will show that if the query string Q approximately occurs in a message E within the allowed distance d (i.e., $\text{dist}(*Q*, E) \le d$), then $\text{vote}(E)$ must be greater than or equal to our lower bound provided $|Q| > 2d + 1$. Thus, suppose we are searching for messages in $\mathcal{D}$ that approximately contain Q within distance d. We can eliminate a message E from consideration if $\text{vote}(E)$ is smaller than the lower bound. It suffices to examine only the messages with votes greater than or equal to the lower bound.

Let S_E represent the remaining substring of a message E after an optimal substitution for the VLDC's when matching $*Q*$ and E. (We call S_E the *stub* of E.) Thus, $\text{dist}(*Q*, E) = \text{dist}(Q, S_E)$.

2) Definition 7: Let $\mathcal{B}$ contain S_E, for all $E \in \mathcal{D}$. Let p, $1 \le p \le |S_E|$, be a position in S_E and let $\text{vote}(S_E[p])$ be the total votes added to the position p after applying the algorithm Voting to Q and $\mathcal{B}$. The vote of S_E, denoted vote(S_E), is defined to be $\max_{1\le p\le |S_E|}$ vote$(S_E[p])$.

Because of the fact that S_E is a substring of E and the way algorithm Voting works, we obtain

3) Lemma 1: vote$(E) \ge$ vote(S_E) for all $E \in \mathcal{D}$.

In the rest of this subsection, let T be a minimum cost trace from the query string Q to a message stub $S_E; \gamma(T) = \mathrm{dist}(Q, S_E) = d$.

4) Lemma 2: vote$(S_E) \ge$ vote(S_E, T, Q).

Proof: Since T is a minimum cost trace from the string Q to the string S_E, we can represent Q as $G_T^0 \cdot P_T^1 \cdot G_T^1 \cdot P_T^2 \cdot \cdots \cdot G_T^{l-1} \cdot P_T^l \cdot G_T^l$ where P_T^u, $1 \le u \le l$, is a partition on Q with respect to T. Let p_v, $1 \le v \le m$, $m \le l$, be the distinct positions in S_E to which vote_by$_T(P_T^u)$, $1 \le u \le l$, are added. From Definition 5, vote(p_v, T, Q) represents the total votes among vote_by$_T(P_T^u)$, $1 \le u \le l$, that are added to p_v. From Definition 7, vote$(S_E[p_v])$ represents the total votes (including those contributed by the partitions P_T^u, $1 \le u \le l$, and possibly other positions in Q) added to p_v. Thus vote$(S_E[p_v]) \ge$ vote(p_v, T, Q). Therefore,

$$\begin{aligned} \text{vote}(S_E) &= \max_{1\le p\le |S_E|} \text{vote}(S_E[p]) \quad \text{(Definition 7)} \\ &\ge \max_{1\le v\le m} \text{vote}(S_E[p_v]) \\ &\ge \max_{1\le v\le m} \text{vote}(p_v, T, Q) \\ &= \text{vote}(S_E, T, Q) \quad \text{(Definition 5).} \end{aligned}$$ □

In deriving the lower bound for pruning the messages in $\mathcal{D}$, our strategy is to find a lower bound for their stubs in $\mathcal{B}$. In doing so, we focus on a minimum cost trace T from Q to each message stub S_E and consider only the votes contributed by the partitions on Q with respect to T. We will show that if $\mathrm{dist}(*Q*, E) = d$ (i.e., $\mathrm{dist}(Q, S_E) = d$), then vote$(S_E, T, Q)$ must be greater than or equal to the lower bound. By Lemmas 1 and 2, vote(E) must be greater than or equal to the lower bound as well.

The votes contributed by different partitions on Q may be added to the same position in S_E when calculating vote(S_E, T, Q). As a result, it's difficult to estimate the total votes a position in S_E may obtain. Our strategy here is to transform T to a new trace so that in the new trace, the positions obtaining the partitions' votes are all distinct. Let $Q = G_T^0 \cdot P_T^1 \cdot G_T^1 \cdot P_T^2 \cdot \cdots \cdot G_T^{l-1} \cdot P_T^l \cdot G_T^l$ and $S_E = H_T^0 \cdot R_T^1 \cdot H_T^1 \cdot R_T^2 \cdot \cdots \cdot H_T^{l-1} \cdot R_T^l \cdot H_T^l$ where P_T^u, $1 \le u \le l$, are partitions on Q with respect to T and R_T^u are their corresponding partitions on S_E. The following details our transformation procedures.

5) Definition 8: Let $\bar{Q} \in (\Sigma \cup \{\Lambda\})^*$ and $\bar{S} \in \Sigma^*$ be two strings. Let $\bar{T}$ be a minimum cost trace from $\bar{Q}$ to $\bar{S}$ such that $\bar{Q} = \bar{G}_{\bar{T}}^0 \cdot \bar{P}_{\bar{T}}^1 \cdot \bar{G}_{\bar{T}}^1 \cdot \bar{P}_{\bar{T}}^2 \cdot \cdots \cdot \bar{G}_{\bar{T}}^{m-1} \cdot \bar{P}_{\bar{T}}^m \cdot \bar{G}_{\bar{T}}^m$ and $\bar{S} = \bar{H}_{\bar{T}}^0 \cdot \bar{R}_{\bar{T}}^1 \cdot \bar{H}_{\bar{T}}^1 \cdot \bar{R}_{\bar{T}}^2 \cdot \cdots \cdot \bar{H}_{\bar{T}}^{m-1} \cdot \bar{R}_{\bar{T}}^m \cdot \bar{H}_{\bar{T}}^m$, where $\bar{P}_{\bar{T}}^v$, $1 \le v \le m$, are partitions on $\bar{Q}$ with respect to $\bar{T}$, and $\bar{R}_{\bar{T}}^v$ are their corresponding partitions on $\bar{S}$. $\bar{T}$ is called a *pure delete dual* of T if the following conditions hold:

- All the characters in $\bar{S}$ are also in S_E, all the characters (except Λ) in $\bar{Q}$ are also in Q, $|\bar{Q}| = |Q|$ and $|\bar{S}| \le |S_E|$.
- For any two characters touched by a trace line in $\bar{T}$, they must also be touched by a trace line in T.
- For each gap $\bar{G}_{\bar{T}}^v$, $0 \le v \le m$, either $|\bar{G}_{\bar{T}}^v| = 0$ or the gap is solely composed of Λs. Furthermore, $|\bar{H}_{\bar{T}}^v| = 0$ for all $0 \le v \le m$, i.e., all the Λs in the gaps in $\bar{Q}$ are to be deleted.
- $\gamma(\bar{T}) = \mathrm{dist}(\bar{Q}, \bar{S}) = d$, $\sum_{v=1}^m |\bar{P}_{\bar{T}}^v| = \sum_{v=1}^m |\bar{R}_{\bar{T}}^v| = |Q| - d$, and $\sum_{v=0}^m |\bar{G}_{\bar{T}}^v| = d$.

We require that any match between the fingerprints containing Λ does not yield a vote. The following lemma shows the existence of a pure delete dual $\bar{T}$ for a minimum cost trace T and establishes the relationship between the votes of $\bar{T}$ and T.

6) Lemma 3: If $|Q| > 2d + 1$, then there exists a trace $\bar{T}$ from a string $\bar{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\bar{S} \in \Sigma^*$ such that $\bar{T}$ is a pure delete dual of T. Furthermore,

$$\text{vote}(S_E, T, Q) \ge \text{vote}(\bar{S}, \bar{T}, \bar{Q}).$$

Proof: First, we prove the existence property by actually constructing $\bar{T}$, $\bar{Q}$ and $\bar{S}$. Since $\gamma(T) = d$, there must exist a deletes, b inserts, and c mismatches such that $a + b + c = d$ and $a, b, c \ge 0$.

Now for the a deletes in Q, replace them by Λ. For the c characters in Q that are mismatches, replace them by Λ; remove their images from S_E. For the b inserts in S_E, we transform them as follows. Let H_T^s be a gap of characters between the two partitions R_T^s and R_T^{s+1} on S_E that are to be inserted. There are two cases to be considered:

Case 1. $|P_T^s| \ge |H_T^s|$. In this case, remove all the characters in H_T^s from S_E. Let $\mathcal{C}$ contain the $|H_T^s|$ rightmost characters in P_T^s. Replace the characters in $\mathcal{C}$ by Λ; remove their images in R_T^s from S_E.

Case 2. $|P_T^s| < |H_T^s|$. In this case, remove $|P_T^s|$ characters in H_T^s from S_E. Replace all the characters in P_T^s by Λ;[6] remove their images in R_T^s from S_E.

Since $|Q| > 2d+1$, $|P_T^1| + |P_T^2| + \cdots + |P_T^l| = |Q| - d > 2d+1-d > d \ge b$. Therefore, we can always find b characters in Q and replace them by Λ. Call the resulting query string $\bar{Q}$, the resulting stub $\bar{S}$, and the resulting trace $\bar{T}$. It's easy to see that $\bar{T}$, $\bar{Q}$ and $\bar{S}$ satisfy the conditions stated in Definition 8.

Let $\bar{P}_{\bar{T}}^s$ and $\bar{P}_{\bar{T}}^t$, $s \ne t$, be two distinct partitions on $\bar{Q}$. The positions to which vote_by$_{\bar{T}}(\bar{P}_{\bar{T}}^s)$ and vote_by$_{\bar{T}}(\bar{P}_{\bar{T}}^t)$ are added must be different. Furthermore, any fingerprint containing Λ does not yield a vote. Therefore, vote$(S_E, T, Q) \ge$ vote$(\bar{S}, \bar{T}, \bar{Q})$. This completes the proof. □

7) Example 4: Fig. 6(a) shows a minimum cost trace T from a query string Q to a message stub S_E. Fig. 6(b) shows a pure delete dual $\bar{T}$ of T from $\bar{Q}$ to $\bar{S}$. This pure delete dual is obtained by applying the transformation procedure described in Lemma 3; the characters crossed by a slanted line in $\bar{S}$ represent the removed characters during the transformation.

8) Definition 9: Let $\hat{Q} \in (\Sigma \cup \{\Lambda\})^*$ and $\hat{S} \in \Sigma^*$ be two strings. Let $\hat{T}$ be a minimum cost trace from $\hat{Q}$ to $\hat{S}$ such that $\hat{Q} = \hat{P}_{\hat{T}}^1 \cdot \hat{G}_{\hat{T}}^1 \cdot \hat{P}_{\hat{T}}^2 \cdot \cdots \cdot \hat{G}_{\hat{T}}^d \cdot \hat{P}_{\hat{T}}^{d+1}$ and $\hat{S} = \hat{R}_{\hat{T}}^1 \cdot \hat{R}_{\hat{T}}^2 \cdot \cdots \cdot \hat{R}_{\hat{T}}^{d+1}$, where $\hat{P}_{\hat{T}}^v$, $1 \le v \le d+1$, are partitions on $\hat{Q}$ with respect

[6] If $s = 1$ and there is no character left in the partition P_T^s (i.e., $|P_T^s| = 0$) during the transformation, we find an arbitrary remaining partition on Q and replace the characters in it by Λ.

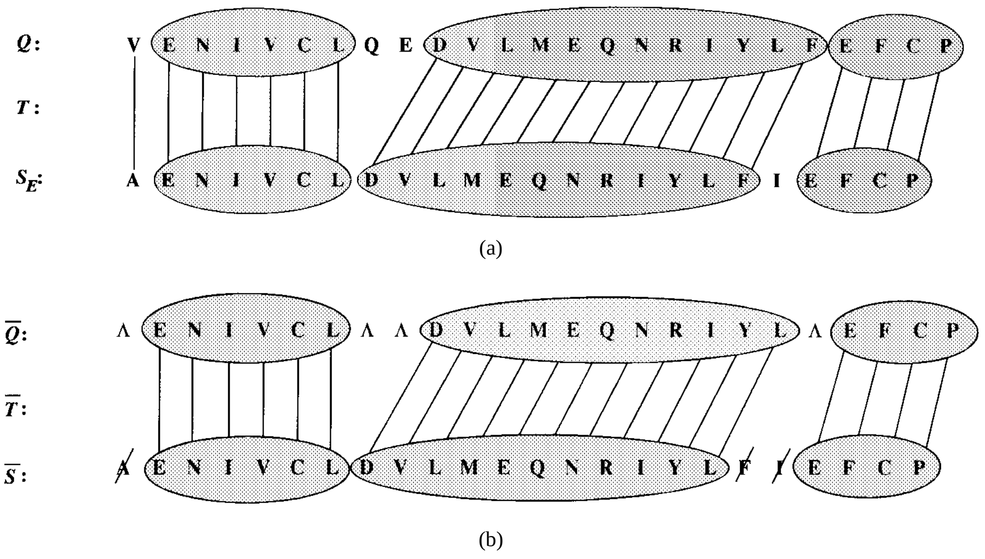
Q:
T:
S_E:
V E N I V C L Q E D V L M E Q N R I Y L F E F C P
A E N I V C L D V L M E Q N R I Y L F I E F C P
(a)
Q̄:
T̄:
S̄:
Λ E N I V C L Λ Λ D V L M E Q N R I Y L Λ E F C P
A E N I V C L D V L M E Q N R I Y L F I E F C P
(b)

Fig. 6. (a) A minimum cost trace T from Q to S_E and (b) a pure delete dual of T.

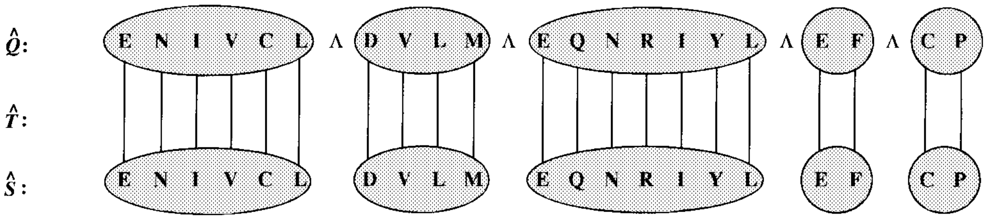
Q̂:
T̂:
Ŝ:
E N I V C L Λ D V L M Λ E Q N R I Y L Λ E F Λ C P
E N I V C L D V L M E Q N R I Y L E F C P

Fig. 7. An evenly distributed delete dual of the trace in Fig. 6(a).

to $\hat{T}$, and $\hat{R}^v_{\hat{T}}$ are their corresponding partitions on $\hat{S}$. $\hat{T}$ is called an *evenly distributed delete dual* of T if the following conditions hold:

- $\hat{T}$ is a pure delete dual of T.
- Each $\hat{G}^v_{\hat{T}}$, $1 \le v \le d$, is composed of a single Λ.

Let $\mathcal{J}$ contain all the evenly distributed delete duals of T. Let $\tilde{Q} \in (\Sigma \cup \{\Lambda\})^*$ and $\tilde{S} \in \Sigma^*$ be two strings. A trace $\tilde{T}$ from $\tilde{Q}$ to $\tilde{S}$ is called a *minimum vote dual* of T if

$$\text{vote}(\tilde{S}, \tilde{T}, \tilde{Q}) = \min_{\hat{T} \in \mathcal{J}} \{\text{vote}(\hat{S}, \hat{T}, \hat{Q})\}.$$

9) Example 5: Consider again the minimum cost trace T in Fig. 6(a). Fig. 7 shows an evenly distributed delete dual of T.

10) Theorem 1:

(i) If $|Q| > 2d+1$, then there exists a trace from a string $\tilde{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\tilde{S} \in \Sigma^*$ such that $\tilde{T}$ is a minimum vote dual of T.

(ii) For any minimum vote dual $\tilde{T}$ of T from a string $\tilde{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\tilde{S} \in \Sigma^*$,

$$\text{vote}(S_E, T, Q) \ge \text{vote}(\tilde{S}, \tilde{T}, \tilde{Q}).$$

Proof: By Lemma 3, there exists a pure delete dual $\bar{T}$ of T from a string $\bar{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\bar{S} \in \Sigma^*$ such that vote$(S_E, T, Q) \ge$ vote$(\bar{S}, \bar{T}, \bar{Q})$. By Definition 8, $|\bar{Q}| = |Q| > 2d+1$ and there are $d\Lambda$s in $\bar{Q}$, all of which are to be deleted. Thus, we can rearrange the partitions on $\bar{Q}$ and move the Λs around so that there are $d+1$ partitions and every two partitions are separated by a single Λ. Call the resulting string $\hat{Q}$ and the resulting trace $\hat{T}$. Let $\hat{S} = \bar{S}$. Clearly, $\hat{T}$, $\hat{Q}$ and $\hat{S}$ satisfy the conditions stated in Definition 9. Thus $\hat{T}$ is an evenly distributed delete dual of T. So, $\mathcal{J}$ is not empty, and therefore (i) is proved.

To prove (ii), let the trace $\tilde{T}$ from a string $\tilde{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\tilde{S} \in \Sigma^*$ be a minimum vote dual of T, i.e., vote$(\tilde{S}, \tilde{T}, \tilde{Q}) = \min_{\hat{T} \in \mathcal{J}}\{$vote$(\hat{S}, \hat{T}, \hat{Q})\}$. Consider the pure delete dual $\bar{T}$ of T and the strings $\bar{Q}$ and $\bar{S}$ found in (i). We will show that vote$(\bar{S}, \bar{T}, \bar{Q}) \ge$ vote$(\tilde{S}, \tilde{T}, \tilde{Q})$. Since vote$(S_E, T, Q) \ge$ vote$(\bar{S}, \bar{T}, \bar{Q})$, we obtain vote$(S_E, T, Q) \ge$ vote$(\tilde{S}, \tilde{T}, \tilde{Q})$.

There are two cases to examine:

Case 1. There are $d+1$ partitions on $\bar{Q}$. Then $\bar{T} \in \mathcal{J}$. Therefore vote$(\bar{S}, \bar{T}, \bar{Q}) \ge$ vote$(\tilde{S}, \tilde{T}, \tilde{Q})$.

Case 2. There are k partitions on $\bar{Q}$, where $k < d+1$. Since (i) $\bar{Q}$ has less than $d+1$ partitions, (ii) $\bar{Q}$ contains $d\Lambda$s, and (iii) $|\bar{Q}| = |Q| > 2d+1$, there must exist a partition P on $\bar{Q}$ such that $|P| \ge 2$ and P is proceeded (or followed) by at least two contiguous Λs. We transform $\bar{Q}$ into a string $\bar{Q}_1$ with $k+1$ partitions by decomposing P into two partitions P_1 and P_2 and moving one of those contiguous Λs into the middle of P_1 and P_2. Call the resulting trace $\bar{T}_1$.

11) Claim: vote$(\bar{S}, \bar{T}, \bar{Q}) \ge$ vote$(\bar{S}, \bar{T}_1, \bar{Q}_1)$.

12) Proof of Claim: Without loss of generality, we assume P is followed by $z+1$, $z \ge 1$, contiguous Λs. Let the length of P be m and its starting position be i, i.e., $P = \bar{Q}[i, i+m-1]$.

Let the length of P_1 be m_1, where $m_1 < m$, and the starting position of P_1 be j, i.e., $P_1 = \bar{Q}_1[j, j+m_1-1]$, $P_2 = \bar{Q}_1[j+m_1+1, j+m]$ and $\bar{Q}_1[j+m_1] = \Lambda$. Thus P_1 is separated from P_2 by a Λ and P_2 is followed by z contiguous Λs. There are two cases to examine:

Case 1. vote_by$_{\bar{T}_1}(\bar{Q}_1[j, j+m_1-1]) \geq$ vote_by$_{\bar{T}_1}(\bar{Q}_1[j+m_1+1, j+m])$. By Definition 4 and the fact that $m > m_1$, vote_by$_{\bar{T}}(P) =$ vote_by$_{\bar{T}}(\bar{Q}[i, i+m-1]) =$ vote_by$_{\bar{T}}(\bar{Q}[i, i+m_1-1])+$vote_by$_{\bar{T}}(\bar{Q}[i+m_1, i+m-1])$. Since the fingerprints containing Λ do not yield votes, vote_by$_{\bar{T}}(\bar{Q}[i, i+m_1-1]) \geq$ vote_by$_{\bar{T}_1}(\bar{Q}_1[j, j+m_1-1])$. Consequently, vote_by$_{\bar{T}}(P) \geq$ vote_by$_{\bar{T}_1}(P_1)$ and vote_by$_{\bar{T}}(P) \geq$ vote_by$_{\bar{T}_1}(P_2)$. Therefore vote$(\bar{S}, \bar{T}, \bar{Q}) \geq$ vote$(\bar{S}, \bar{T}_1, \bar{Q}_1)$.

Case 2. vote_by$_{\bar{T}_1}(\bar{Q}_1[j, j+m_1-1]) <$ vote_by$_{\bar{T}_1}(\bar{Q}_1[j+m_1+1, j+m])$. Similarly we can prove that vote_by$_{\bar{T}}(\bar{Q}[i+m_1, i+m-1]) \geq$ vote_by$_{\bar{T}_1}(\bar{Q}_1[j+m_1+1, j+m])$ and therefore the claim holds. □

In this way, we can transform $\bar{Q} = \bar{Q}_0$ with k partitions into $\bar{Q}_1$ with $k+1$ partitions and transform $\bar{T} = \bar{T}_0$ into $\bar{T}_1$ such that vote$(\bar{S}, \bar{T}, \bar{Q}) \geq$ vote$(\bar{S}, \bar{T}_1, \bar{Q}_1)$. Thus, we perform a series of transformations, starting with $\bar{Q} = \bar{Q}_0$ having k partitions and ending with $\bar{Q}_{d-k+1}$ having $d+1$ partitions such that vote$(\bar{S}, \bar{T}, \bar{Q}) \geq$ vote$(\bar{S}, \bar{T}_{d-k+1}, \bar{Q}_{d-k+1})$. Clearly $\bar{T}_{d-k+1} \in \mathcal{J}$. Thus, vote$(\bar{S}, \bar{T}_{d-k+1}, \bar{Q}_{d-k+1}) \geq$ vote$(\tilde{S}, \tilde{T}, \tilde{Q})$. Therefore vote$(\bar{S}, \bar{T}, \bar{Q}) \geq$ vote$(\tilde{S}, \tilde{T}, \tilde{Q})$. This completes the proof of (ii). □

Theorem 1 implies that vote$(\tilde{S}, \tilde{T}, \tilde{Q})$ is a lower bound for vote(S_E, T, Q). By Lemmas 1 and 2, it is also a lower bound for vote(E). In the next section, we shall show how to calculate vote$(\tilde{S}, \tilde{T}, \tilde{Q})$ and obtain a practically useful lower bound for pruning nonqualifying messages during searching.

IV. Calculation of the Lower Bound

Throughout this section, we let Q be the query string and let E be a message in $\mathcal{D}$ where $|Q| > 2d+1$ and $\text{dist}(*Q*, E) = d$. Let T be a minimum cost trace from Q to S_E. Let $\hat{T}$ be an evenly distributed delete dual of T from a string $\hat{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\hat{S} \in \Sigma^*$ (again see Table II and the previous section for an explanation of the notation). We introduce a sequence of lemmas, needed in proving the main theorem for calculating our lower bound.

1) Lemma 4: Let $\hat{Q}[i, i+m-1]$ be a partition of length m on $\hat{Q}$, where $m \geq n$. Let $\hat{Q}[p, p+n-1]$ be a substring in $\hat{Q}[i, i+m-1]$ where $p \geq i$ and $p+n-1 \leq i+m-1$. Then

$$\text{vote\_by}_{\hat{T}}(\hat{Q}[p]) = n-2.$$

Proof: Let $\hat{S}[j, j+m-1]$ be the partition on $\hat{S}$ that corresponds to $\hat{Q}[i, i+m-1]$. Let $p = i+u$, for some $u \leq m-n$. By Definition 4, vote_by$_{\hat{T}}(\hat{Q}[p]) =$ vote_by$_{\hat{T}}(\hat{Q}[i+u])$ is the sum of votes added to the position $((j+u)-(i+u)+1) = (j-i+1)$ where the votes are obtained from matches between the fingerprints of the segments $\hat{Q}[i+u, i+u+n-1]$ and $\hat{S}[j+u, j+u+n-1]$. The result follows by observing that there are totally $n-2$ fingerprints generated from the two segments, and each of these fingerprints yields one vote. □

2) Lemma 5: Let $\hat{Q}[i, i+m-1]$ be a partition of length m on $\hat{Q}$, where $m < n$. Then

$$\text{vote\_by}_{\hat{T}}(\hat{Q}[i, i+m-1]) = \frac{m(m-1)}{2}.$$

Proof: By Definition 4,

$$\begin{aligned}\text{vote\_by}_{\hat{T}}(\hat{Q}[i, i+m-1]) &= \sum_{u=i}^{i+m-1} \text{vote\_by}_{\hat{T}}(\hat{Q}[u]) \\ &= \sum_{u=0}^{m-1} \text{vote\_by}_{\hat{T}}(\hat{Q}[i+u]).\end{aligned}$$

Among the $n-2$ fingerprints generated from each segment $\hat{Q}[i+u, i+u+n-1]$,[7] $0 \leq u \leq m-1$, only $(i+m-1)-(i+u) = m-u-1$ fingerprints yield votes; the other $(n-2)-(m-u-1) = n-m+u-1$ fingerprints contain Λ and therefore do not yield votes. Thus,

$$\begin{aligned}\text{vote\_by}_{\hat{T}}(\hat{Q}[i,\ i+m-1]) &= \sum_{u=0}^{m-1} \text{vote\_by}_{\hat{T}}(\hat{Q}[i+u]) \\ &= \sum_{u=0}^{m-1}(m-u-1) \\ &= \frac{m(m-1)}{2}.\end{aligned}$$ □

3) Lemma 6: Let $\hat{Q}[i, i+m-1]$ be a partition of length m on $\hat{Q}$, where $m \geq n$. Then

$$\begin{aligned}\text{vote\_by}_{\hat{T}}(\hat{Q}[i,\ i+m-1]) &= (m-n+1) \times (n-2) \\ &\quad + \frac{(n-1)(n-2)}{2}.\end{aligned}$$

Proof: Note that

$$\begin{aligned}&\text{vote\_by}_{\hat{T}}(\hat{Q}[i,\ i+m-1]) \\ &= \sum_{u=0}^{m-n} \text{vote\_by}_{\hat{T}}(\hat{Q}[i+u]) \\ &\quad + \text{vote\_by}_{\hat{T}}(\hat{Q}[i+m-n+1,\ i+m-1]).\end{aligned}$$

For each position $i+u$ on $\hat{Q}$, $0 \leq u \leq m-n$, the segment $\hat{Q}[i+u, i+u+n-1]$ is a substring in $\hat{Q}[i, i+m-1]$. By Lemma 4,

$$\begin{aligned}\sum_{u=0}^{m-n} \text{vote\_by}_{\hat{T}}(\hat{Q}[i+u]) &= \sum_{u=0}^{m-n}(n-2) \\ &= (m-n+1) \times (n-2).\end{aligned}$$

By substituting $n-1$ for m in Lemma 5,

$$\text{vote\_by}_{\hat{T}}(\hat{Q}[i+m-n+1,\ i+m-1]) = \frac{(n-1)(n-2)}{2}.$$

Therefore, the result follows. □

(See Theorem 2 at the bottom of the next page).

[7] If $i+u+n-1 > |\hat{Q}|$, then there are only $|\hat{Q}|-i-u-1$ fingerprints generated from the segment $\hat{Q}[i+u, |\hat{Q}|]$ (cf. footnote 5).

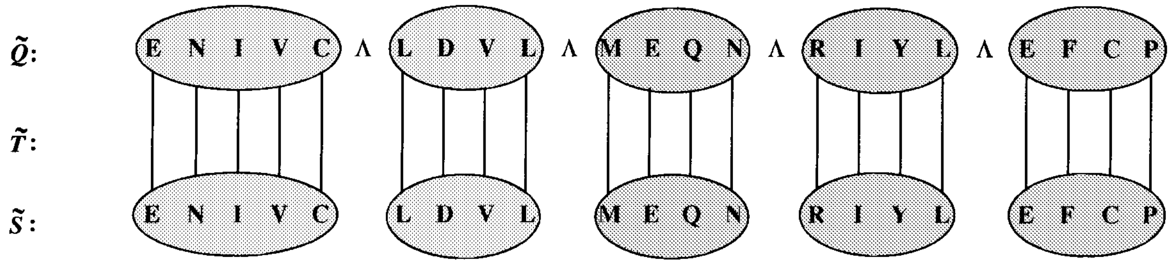
$\tilde{Q}$:
$\tilde{T}$:
$\tilde{S}$:
E N I V C Λ L D V L Λ M E Q N Λ R I Y L Λ E F C P
E N I V C L D V L M E Q N R I Y L E F C P

Fig. 8. A minimum vote dual of the trace in Fig. 6(a).

Proof of Theorem 2: Let $\tilde{T}$ be an evenly distributed delete dual of T from a string $\tilde{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\tilde{S} \in \Sigma^*$ such that for each partition P on $\tilde{Q}$, $|P| = \lfloor N \rfloor$ or $|P| = \lfloor N \rfloor + 1$.

4) Claim: $\tilde{T}$ is a minimum vote dual of T.

5) Proof of Claim: Suppose to the contrary that $\tilde{T}$ is not a minimum vote dual of T. Let $\vec{T}$ be an evenly distributed delete dual of T from a string $\vec{Q} \in (\Sigma \cup \{\Lambda\})^*$ to a string $\vec{S} \in \Sigma^*$ such that vote$(\vec{S}, \vec{T}, \vec{Q}) <$ vote$(\tilde{S}, \tilde{T}, \tilde{Q})$. Note $|\vec{Q}| = |\tilde{Q}| = |Q|$.

Let $\vec{P}_u$, $1 \leq u \leq d+1$, be the $d+1$ partitions on $\vec{Q}$ and let $\tilde{P}_u$, $1 \leq u \leq d+1$, be the $d+1$ partitions on $\tilde{Q}$. From Definition 9, the positions in $\vec{Q}(\tilde{Q}$, respectively) to which the vote_by$_{\vec{T}}(\vec{P}_u)$(vote_by$_{\tilde{T}}(\tilde{P}_u)$, respectively), $1 \leq u \leq d+1$, are added must be distinct. Let $\vec{P}_s$ for some s, $1 \leq s \leq d+1$, be the partition on $\vec{Q}$ that yields vote$(\vec{S}, \vec{T}, \vec{Q})$ and let $\tilde{P}_t$ for some t, $1 \leq t \leq d+1$, be the partition on $\tilde{Q}$ that yields vote$(\tilde{S}, \tilde{T}, \tilde{Q})$. By assumption and from Definition 5, vote_by$_{\vec{T}}(\vec{P}_s) <$ vote_by$_{\tilde{T}}(\tilde{P}_t)$. From Lemmas 5 and 6, the vote contributed by a partition is proportional to its length. Hence we conclude that $|\vec{P}_s| < |\tilde{P}_t|$.

Now assume, without loss of generality, that N is an integer and for all partitions P on $\tilde{Q}$, $|P| = N$. Since vote_by$_{\vec{T}}(\vec{P}_v) \leq$ vote_by$_{\vec{T}}(\vec{P}_s)$ for all $v \neq s$ (i.e., $|\vec{P}_v| \leq |\vec{P}_s|$), $|\vec{Q}| = \sum_{u=1}^{d+1} |\vec{P}_u| + d \leq \sum_{u=1}^{d+1} |\vec{P}_s| + d < \sum_{u=1}^{d+1} |\tilde{P}_t| + d = \sum_{u=1}^{d+1} N + d = |Q|$, contradicting the fact that $|\vec{Q}| = |Q|$. □

Therefore

$$\begin{aligned} \text{vote}(E) &\geq \text{vote}(S_E) && \text{(Lemma 1)} \\ &\geq \text{vote}(S_E, T, Q) && \text{(Lemma 2)} \\ &\geq \text{vote}(\tilde{S}, \tilde{T}, \tilde{Q}) && \text{(Theorem 1 and the above claim).} \end{aligned}$$

Now to calculate vote$(\tilde{S}, \tilde{T}, \tilde{Q})$, we look at the vote contributed by the longest partition on $\tilde{Q}$. In showing (i), if N is an integer, then all the partitions on $\tilde{Q}$ have the same length, namely N. By substituting N for m in Lemma 6, we obtain the asserted formula. If N is not an integer, then the longest partition on $\tilde{Q}$ must have length $\lfloor N \rfloor + 1$. By substituting $\lfloor N \rfloor + 1$ for m in Lemma 6, we obtain the asserted formula. Similarly, by substituting N or $\lfloor N \rfloor + 1$ for m in Lemma 5, we obtain the formulae asserted in (ii).

This completes the proof. □

6) Example 6: Consider again the trace T in Fig. 6(a). Fig. 8 shows a minimum vote dual $\tilde{T}$ of T. Table III shows the vote contributed by each partition on $\tilde{Q}$ and the position in $\tilde{S}$ to which the vote is added. In this example, $|Q| = |\tilde{Q}| = 25$; $d = 4$. $N = (25 - 4)/(4+1) = 4.2$; $\lfloor N \rfloor = 4$. Suppose the segment length n is 6. From Theorem 2(ii), we know vote$(E) \geq$ vote$(\tilde{S}, \tilde{T}, \tilde{Q}) = (4)(4+1)/2 = 10$. Referring to Fig. 8 and Table III, we can see that this lower bound value 10 is obtained from vote_by$_{\tilde{T}}(\tilde{Q}[1,5])$ (which is added to the position 1 in $\tilde{S}$).

V. Performance Analysis

We carried out a series of experiments to evaluate the performance of the proposed index structure and the lower bound. The programs were written in C and run on a Sun SPARC workstation under the SUN operating system version 4.1.2. The database contained 10 000 randomly generated messages, with lengths ranging from 100 to 3000. The length of segments was fixed at 6. The hash functions used were the minimal perfect hash functions described in [12].

Theorem 2 (Main Theorem):

Let

$$N = \frac{|Q| - d}{d+1}.$$

(i) If $N \geq n$, then

$$\text{vote}(E) \geq \begin{cases} (N - n + 1) \times (n-2) + \frac{(n-1)(n-2)}{2} & \text{if } N \text{ is an integer} \\ (\lfloor N \rfloor - n + 2) \times (n-2) + \frac{(n-1)(n-2)}{2} & \text{otherwise} \end{cases}$$

(ii) If $N < n$, then

$$\text{vote}(E) \geq \begin{cases} \frac{N(N-1)}{2} & \text{if } N \text{ is an integer} \\ \frac{(\lfloor N \rfloor)(\lfloor N \rfloor + 1)}{2} & \text{otherwise} \end{cases}$$

TABLE III
VOTES CONTRIBUTED BY THE PARTITIONS ON $\tilde{Q}$ AND THE DISTINCT POSITIONS IN $\tilde{S}$ TO WHICH THE VOTES ARE ADDED

vote	contributed by partition	position in S to which the vote is added
10	$Q[1,5]$	1 - 1 + 1 = 1
6	$\tilde{Q}[7,10]$	6 - 7 + 1 = 0
6	$\tilde{Q}[12,15]$	10 - 12 + 1 = -1
6	$\tilde{Q}[17,20]$	14 - 17 + 1 = -2
6	$\tilde{Q}[22,25]$	18 - 22 + 1 = -3

In the first experiment, we compared the running times of two searching methods. Method I used agrep [38] to compare a given query string with each message in the database. Method II, on the other hand, first used the index structure to prune unlikely messages and then applied agrep to the remaining ones to locate the qualifying messages. (A message was pruned if its vote was less than the lower bound, Theorem 2). Fig. 9 shows the results for varying query string lengths and distances allowed. Each point of the graphs represents the average value over thirty query strings. From the figure, we see that the index structure accelerates searching considerably. The longer a query string or the smaller the distance allowed, the more effective the index is. This happens because longer queries with smaller distances allowed yield a tighter lower bound (cf. Theorem 2). As a result, many nonqualifying messages can be pruned.

In the second experiment, we further evaluated the quality of our lower bound. The metric used is the false drop probability [11], denoted F_d, where

$$F_d = \frac{N_f}{|\mathcal{D}| - N_a} \times 100\%$$

N_f is the number of false drops. (A false drop is a message whose vote is greater than the lower bound, though the message doesn't contain the query string within the allowed distance.) N_a is the total number of qualifying messages. Fig. 10 shows the result. Each point of the graphs represents the average value over thirty query strings. It can be seen that the F_d is small in general. The longer a query string, the smaller the F_d (i.e., the tighter the lower bound). For the same length of queries, the F_d drops as the distance allowed decreases. This agrees with the results of the running times shown in Fig. 9.

It was also observed that the F_d heavily depends on the tested data. When many messages share common substrings with a query,[8] the F_d rises up. The reason is that a common substring in a message and the query may yield votes higher than the lower bound. As a result we are unable to prune the message, though the message as a whole doesn't qualify to be a solution. We also examined the effect of the database size and segment length. It was found that the F_d is insensitive to both of the two parameter values.

To learn more about the performance of our index structure in real applications, we have tested it on 2000 e-mail messages obtained from the "mbox" file of a faculty member. The lengths of the messages ranged from 110 to 5613. Their contents included greetings, conversations, meetings, seminars, call-for-papers and call-for-participation for various conferences. We stripped off blank lines and spaces in the text part of a message so that every two words are separated by a single space only. Also, we changed all upper case letters to lower case letters and removed some special characters such as the period, comma, "&", ":", etc. The query strings used in searches included both words (e.g., "presentation," "classification," etc.) and phrases (e.g., "molecular biology," "artificial intelligence," etc.).

[8] For example, the words "presentation" and "visualization" share a common substring "ation."

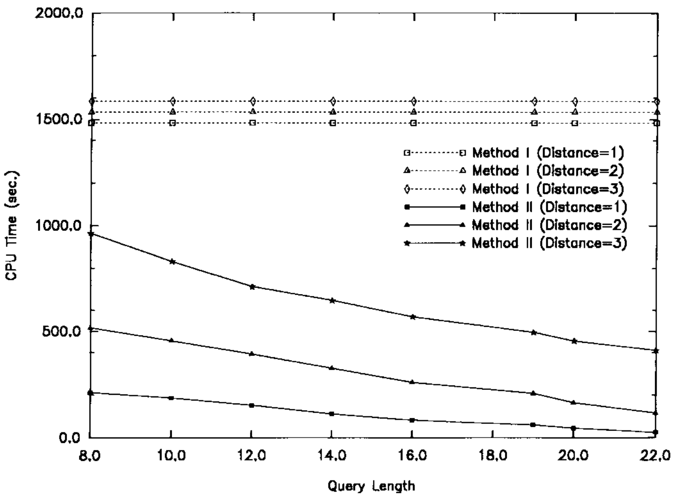

Fig. 9. Comparison of running times ($|\mathcal{D}| = 10\,000$, $|E| \in [100, 3000]$, $n = 6$, $|Q| \in [8, 22]$).

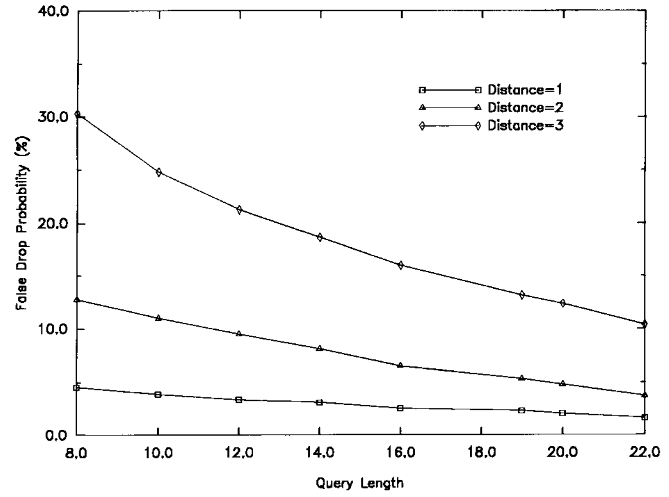

Fig. 10. Effect of the query length $|Q|$ and the allowed distance d on the false drop probability $F_d(|\mathcal{D}| = 10\,000$, $|E| \in [100, 3000]$, $n = 6$, $|Q| \in [8, 22])$.

The results obtained from these messages are consistent with those of generated data. With the real e-mail messages, we have also evaluated *recall* and *precision* [6], [27]. Let NR denote the number of retrieved relevant messages (i.e., those approximately containing the query string) and let NT denote the total number of relevant messages. Recall is defined as NR/NT. Table IV shows recall for varying distance values. Each recall value represents the average over thirty query

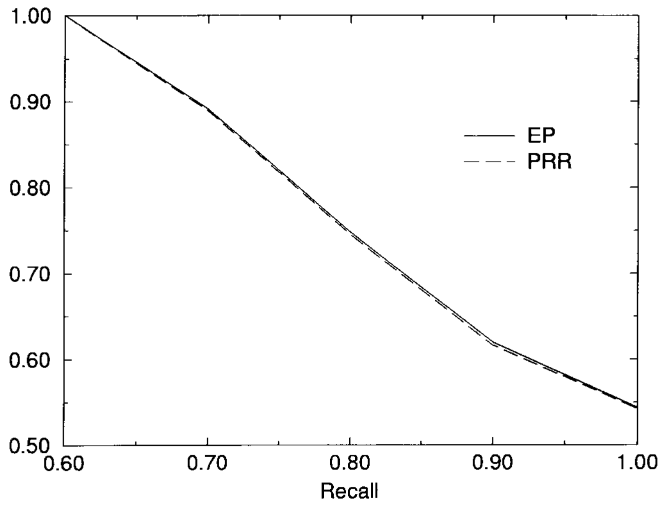

Fig. 11. PRR and EP as a function of recall ($|\mathcal{D}| = 2000$, $|E| \in [110, 5613]$, $n = 6$).

TABLE IV
EFFECT OF THE ALLOWED DISTANCE d ON RECALL
($|\mathcal{D}| = 2,000$, $|E| \in [110, 5613]$, $n = 6$)

	$d = 0$	$d = 1$	$d = 2$	$d = 3$
Recall	0.62	0.71	0.84	1.00

strings. We see that as the distance allowed becomes large, recall increases. This happens because with larger distances, more relevant messages can be retrieved.[9] In evaluating precision, we use two measures: PRR and EP [2], [25]. PRR is the probability that a retrieved message is relevant. EP is the expected precision (i.e., the expected value of the ratio of NR to NR plus the possible number of nonrelevant messages incurred as a result of retrieving NR relevant messages). They are two useful measures for presenting experimental results in a descriptive sense [2], [25]. Fig. 11 graphs PRR and EP as a function of recall. Each point of the graphs represents the average value over thirty query strings. The figure shows $\mathrm{EP} \geq \mathrm{PRR}$, consistent with the study reported in [2]. Note that both PRR and EP drop as recall increases (i.e., as the distance allowed increases). This is understandable given our previous analysis concerning the false drop probabilities.

VI. CONCLUSION

In this paper we presented an index structure, called fingerprint files, for retrieving electronic messages that contain mistyped words or spelling errors. Given a query string Q, a database of messages $\mathcal{D}$ and an integer d, the index is used to quickly locate the messages in $\mathcal{D}$ that approximately contain the query within the allowed distance d. To construct the index, we segment each message, generate fingerprints from those segments and hash the fingerprints into the index structure. When the query is given, we process it using the same hash functions. A histogram of votes on the messages is then constructed. We derive a lower bound, based on which one can prune nonqualifying messages. Our experimental results involving the real and generated data demonstrated the good performance of both the index and the lower bound.

The work reported here is part of a project for pattern matching and retrieval in scientific, program and document databases [28], [32], [36]. The implementation of the proposed index structure and other tools developed from the project [37] are being incorporated into several systems that we [14], [35] and others are building and are available from the authors.

[9]One issue here is exactly how large the distance should be in practice. Statistically, if at most k edits are needed in general to correct a misspelled word or phrase in a message, then the distance d used in searching should be k [3], [9]. In our experiments, we found that letting d be 3 is suitable—when $d = 3$, recall reaches 1 (cf. Table IV).

ACKNOWLEDGMENT

The authors thank J. McHugh, P. Ng, D. Shasha, and M. Winslett for useful conversations concerning this work. They also thank an anonymous reviewer, whose comments helped to improve the paper.

REFERENCES

[1] C. R. Blair, "A program for correcting spelling errors," *Inform. Contr.*, vol. 3, pp. 60–67, 1960.
[2] P. Bollmann, V. V. Raghavan, G. S. Jung, and L. C. Shu, "On probabilistic notions of precision as a function of recall," *Inf. Process. Manage.*, vol. 28, no. 3, pp. 291–315, 1992.
[3] C. P. Bourne, "Frequency and impact of spelling errors in bibliographic data bases," *Inf. Process. Manage.*, vol. 13, no. 1, pp. 1–12, 1977.
[4] A. Califano and I. Rigoutsos, "FLASH: A fast look-up algorithm for string homology," in *Proc. 1st Int. Conf. Intelligent Systems Molecular Biology*, Bethesda, MD, July 1993.
[5] S. K. Chang and L. Leung, "A knowledge-based message management system," *ACM Trans. Office Inf. Syst.*, vol. 5, pp. 213–236, July 1987.
[6] H. Chen and K. J. Lynch, "Automatic construction of networks of concepts characterizing document databases," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 5, pp. 885–902, 1992.
[7] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, pp. 171–176, Mar. 1964.
[8] L. Davidson, "Retrieval of misspelled names in an airlines passenger record system," *Commun. ACM*, vol. 5, pp. 169–171, Mar. 1962.
[9] M. W. Du and S. C. Chang, "A model and a fast algorithm for multiple errors spelling correction," *Acta Informatica*, vol. 29, pp. 281–302, 1992.
[10] C. Faloutsos, "Access methods for text," *ACM Comput. Surveys*, vol. 17, pp. 49–74, Mar. 1985.
[11] C. Faloutsos and S. Christodoulakis, "Signature files: An access method for documents and its analytical performance evaluation," *ACM Trans. Off. Inf. Syst.*, vol. 2, pp. 267–288, Oct. 1984.
[12] E. A. Fox, L. S. Heath, Q. F. Chen, and A. M. Daoud, "Practical minimal perfect hash functions for large databases," *Commun. ACM*, vol. 35, pp. 105–121, Jan. 1992.
[13] P. A. V. Hall and G. R. Dowling, "Approximate string matching," *ACM Comput. Surveys*, vol. 12, pp. 381–402, Dec. 1980.
[14] X. Hao, J. T. L. Wang, M. P. Bieber, and P. A. Ng, "Heuristic classification of office documents," *Int. J. Artif. Intell. Tools*, vol. 3, no. 2, pp. 233–265, 1994.
[15] D. Harman, E. Fox, R. A. Baeza-Yates, and W. Lee, "Inverted files," in *Information Retrieval, Data Structures and Algorithms*, W. B. Frakes and R. A. Baeza-Yates Eds. Englewood Cliffs, NJ: Prentice-Hall, 1992, pp. 28–43.
[16] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.
[17] C.-W. R. Leng and D. L. Lee, "Optimal weight assignment for signature generation," *ACM Trans. Database Syst.*, vol. 17, pp. 346–373, June 1992.
[18] C. R. Litecky and G. B. Davis, "A study of errors, error-proneness, and error diagnosis in COBOL," *Commun. ACM*, vol. 19, pp. 33–37, Jan. 1976.
[19] R. Lowrance and R. A. Wagner, "The extended string-to-string correction problem," *J. ACM*, vol. 22, pp. 177–183, Apr. 1975.
[20] T. W. Malone, K. R. Grant, K. Y. Lai, R. Rao, and D. Rosenblitt, "Semistructured messages are surprisingly useful for computer-supported coordination," *ACM Trans. Off. Inf. Syst.*, vol. 5, pp. 115–131, Apr. 1987.

[21] H. L. Morgan, "Spelling correction in systems programs," *Commun. ACM*, vol. 13, pp. 90–94, Feb. 1970.
[22] F. E. Muth and A. L. Tharp, "Correcting human error in alphanumeric terminal input," *Inf. Process. Manage.*, vol. 13, no. 6, pp. 329–337, 1977.
[23] J. L. Peterson, "Computer program for detecting and correcting spelling errors," *Commun. ACM*, vol. 23, pp. 676–687, Dec. 1980.
[24] J. J. Pollock and A. Zamora, "Automatic spelling correction in scientific and scholarly text," *Commun. ACM*, vol. 27, pp. 358–368, Apr. 1984.
[25] V. V. Raghavan, P. Bollmann, and G. S. Jung, "A critical investigation of recall and precision as measures of retrieval system performance," *ACM Trans. Inf. Syst.*, vol. 7, no. 3, pp. 205–229, 1989.
[26] H. J. Rogers and P. Willett, "Searching for historical word forms in text databases using spelling correction methods: Reverse error and phonetic coding methods," *J. Documentation*, vol. 47, pp. 333–353, 1991.
[27] G. Salton, *Automatic Text Processing*. Reading, MA: Addison-Wesley, 1989.
[28] D. Shasha, J. T. L. Wang, K. Zhang, and F. Y. Shih, "Exact and approximate algorithms for unordered tree matching," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 668–678, Apr. 1994.
[29] D. Tsichritzis *et al.*, "A system for managing structured messages," *IEEE Trans. Commun.*, vol. COM-30, pp. 66–73, Jan. 1982.
[30] J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, vol. II. Rockville, MD: Computer Science, 1989.
[31] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, pp. 168–173, Jan. 1974.
[32] J. T. L. Wang, G.-W. Chirn, T. G. Marr, B. A. Shapiro, D. Shasha, and K. Zhang, "Combinatorial pattern discovery for scientific data: Some preliminary results," in *Proc. 1994 ACM SIGMOD Int. Conf. Management Data*, Minneapolis, MN, May 1994, pp. 115–125.
[33] J. T. L. Wang, T. G. Marr, D. Shasha, B. A. Shapiro, and G.-W. Chirn, "Discovering active motifs in sets of related protein sequences and using them for classification," *Nucleic Acids Res.*, vol. 22, no. 14, pp. 2769–2775, 1994.
[34] J. T. L. Wang, T. G. Marr, D. Shasha, B. A. Shapiro, G.-W. Chirn, and T. Y. Lee, "Complementary classification approaches for protein sequences," *Protein Eng.*, vol. 9, no. 5, pp. 381–386, 1996.
[35] J. T. L. Wang and P. A. Ng, "TEXPROS: An intelligent document processing system," *Int. J. Software Eng. Knowledge Eng.*, vol. 2, pp. 171–196, June 1992.
[36] J. T. L. Wang, K. Zhang, and G.-W. Chirn, "Algorithms for approximate graph matching," *Inf. Sci.*, vol. 82, pp. 45–74, 1995.
[37] J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha, "A system for approximate tree matching," *IEEE Trans. Knowledge Data Eng.*, vol. 6, pp. 559–571, Aug. 1994.
[38] S. Wu and U. Manber, "Fast text searching allowing errors," *Commun. ACM*, vol. 35, pp. 83–91, Oct. 1992.
[39] E. M. Zamora, J. J. Pollock, and A. Zamora, "The use of trigram analysis for spelling error detection," *Inf. Process. Manage.*, vol. 17, pp. 305–316, 1981.

Jason Tsong-Li Wang (S'88–M'90) received the B.S. degree in mathematics from National Taiwan University, Taipei, Taiwan, R.O.C., and the Ph.D. degree in computer science from the Courant Institute of Mathematical Sciences, New York University, in 1991.

He is currently Associate Professor in the Department of Computer and Information Science at New Jersey Institute of Technology, Newark. He is also Director of the University's Data and Knowledge Engineering Laboratory, and a Member of the Ph.D. in Management Faculty at the Graduate School of Rutgers University, Newark. His research interests include data and knowledge management systems, multimedia information retrieval, software development, pattern discovery, and computational biology. He has published 50 papers in conference proceedings, books, and journals, including IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, *ACM Transactions on Database Systems*, *ACM Transactions on Information Systems*, and *Nucleic Acids Research*.

Dr. Wang serves on the Editorial Advisory Board of *Information Systems*, is listed in *Who's Who among Asian Americans* (1994), and a member of ACM, AAAI, and SIAM.

Chia-Yo Chang received the B.S. degree in mathematics from National Tsing Hua University, Taiwan, R.O.C., and the M.S. degree in computer science from the New Jersey Institute of Technology, Newark, where he is currently pursuing the Ph.D. degree.

His research interests include data management systems, multimedia information retrieval, pattern classification, and computational biology.