Finding the Longest Similar Subsequence of Thumbprints for Intrusion Detection

Ming D. Wan, Shou-Hsuan Stephen Huang, and Jianhua Yang
Department of Computer Science, University of Houston
Houston, Texas, 77204, USA
Email: {mingwan, shuang, jhyang}@cs.uh.edu
http://www.cs.uh.edu

Technical Report Number UH-CS-05-26

December 13, 2005

Keywords: Network Security, Intrusion Detection, Thumbprint, Similarity, Dynamic Programming.

Abstract

One way to detect intruders on the Internet is to compare the similarity of two thumbprints. A thumbprint is a summary of a connection that characterizes the connection. The packet gap thumbprint consists of sequences of non-negative real number representing the time gaps between "send" packets. This paper formalized definitions of similarity between two non-negative real number sequences, by introducing ε-similarity, partial sum and longest ε-similar subsequence (LSS). Length of LSS is a measurement of similarity between two sequences. The Longest ε-Similar Subsequence (LSS) problem is a generalization of the well known Longest Common Subsequence (LCS) problem. The goal of this paper is to find an optimal solution to the LSS problem. We analyzed the property of partial sums and proposed to focus on the minimum matched partial sum which leads to an optimal solution to LSS while reduce the problem space. As the LSS problem has optimal structure, we proposed an Algorithm based on dynamic programming technique. Time complexity of this algorithm is $O(m^2n^2)$. By using a property of the partial sums, we reduced the time complexity to $O(mn(m+n))$.

Finding the Longest Similar Subsequence of Thumbprints for Intrusion Detection

Ming D. Wan, Shou-Hsuan Stephen Huang, and Jianhua Yang
Department of Computer Science, University of Houston
Houston, Texas, 77204, USA
Email: {mingwan, shuang, jhyang}@cs.uh.edu

Abstract

One way to detect intruders on the Internet is to compare the similarity of two thumbprints. A thumbprint is a summary of a connection that characterizes the connection. The packet gap thumbprint consists of sequences of non-negative real number representing the time gaps between "send" packets. This paper formalized definitions of similarity between two non-negative real number sequences, by introducing ε-similarity, partial sum and longest ε-similar subsequence (LSS). Length of LSS is a measurement of similarity between two sequences. The Longest ε-Similar Subsequence (LSS) problem is a generalization of the well known Longest Common Subsequence (LCS) problem. The goal of this paper is to find an optimal solution to the LSS problem. We analyzed the property of partial sums and proposed to focus on the minimum matched partial sum which leads to an optimal solution to LSS while reduce the problem space. As the LSS problem has optimal structure, we proposed an Algorithm based on dynamic programming technique. Time complexity of this algorithm is $O(m^2n^2)$. By using a property of the partial sums, we reduced the time complexity to $O(mn(m+n))$.

Index Terms

Network Security, Intrusion Detection, Thumbprint, Similarity, Dynamic Programming.

1. Introduction

One way of detecting stepping-stone is by monitoring a site's incoming and outgoing traffic. In general we have the problem of determining whether two connections belong to the same connection chain [ZP00]. In a recent paper, we proposed to use time gaps between packets as a temporal thumbprint to identify a connection [YH05]. The method requires us to compare two such thumbprints to see if they are "similar." Even though there are efficient heuristic algorithms to compare two gap sequences, the issue has never been studied formally. We propose a formal definition of the problem by introducing ε-similarity, partial sum and longest ε-similar subsequence (LSS) and cast it as a generalization of the well known Longest Common Subsequence (LCS) problem. It is not a trivial task to define what is similar in two sequences of numbers which may differ in length. The definition and solution we derive here can be used in comparing many different thumbprints.

The LSS problem is much more complicated than LCS problem due to partial sums involved. We analyze the property of partial sums, proposed to focus on the minimum matched partial sum (MMPS). The Longest ε-Similar Subsequence (LSS) problem has similar optimal structure like LCS problem.

With dynamic programming technique, we have an O(m2n2) algorithm to find the optimal solution to the LSS problem. Based on the property of partial sums as defined by this paper, we come up with a more efficient optimal algorithm with time complexity of $O(mn(m+n))$. Practically, matches of very big sized minimum matched partial sum (MMPS), which summed up a large number of elements together, are likely false match-ups in thumbprint application. By limiting the size of MMPS to a constant number s, we reduced the complexity a suboptimal solution to $O(smn)$.

The rest of this paper is organized as following: In Section 2, we proposed formal definitions. In Section 3, we define a particular partial sum, which sums up consecutive elements before one element. In Section 4, property of LSS is studied. Dynamic programming technique is used to solve the LSS problem. In Section 5, a more efficient optimal algorithm and a heuristic algorithm are proposed to reduce time complexity to *O(mn(m+n))* and *O(smn)* respectively.

2. Definitions

The first major difficulty of our problem is the issue of similarity. The following definitions define the ε-similarity between two sequences.

ε-similarity: Given a ratio ε between 0 and 1 inclusively, and two non-negative real number *a* and *b*, we define *a* and *b* to be ε-similar to each other if $|(a-b)/(a+b)| \leqslant \varepsilon$. In other words, the two numbers are within ε of each other proportionally (difference divided by the sum). We then generalize the similarity to two sequences of the same length. Two sequences *a*[1..n] and *b*[1..n] are ε-similar if *a*[i] and *b*[i] are ε-similar for all i =1,…, n.

Partial Sum: Given a sequence *a*[1], …, *a*[m], a partial sum of the sequence is the sum of one or more consecutive elements of the sequence, *a*[i]+*a*[i+1]+…+*a*[j] for some i < j.

Partial Sum Subsequence: A partial sum subsequence of *a*[1], …, *a*[m] is a sequence of partial sum *a'*[1], …, *a'*[m'] such that $a'[j] = \sum_{k=s[j]}^{t[i]} a[k]$ where $1 \leqslant s[j] \leqslant t[j] \leqslant s[j+1] \leqslant t[j+1] \leqslant m$ for all j = 1, …, *m'-1*. The length of the (partial sum) subsequence is defined as *m'*.

Longest ε-Similar Subsequence (LSS): Given two sequences of non-negative real numbers *a*[1], …, *a*[m] and *b*[1], …, *b*[n], if (i) there exists a (partial sum) subsequence *a'*[1…p] of *a* and a subsequence *b'*[1..p] of *b*, such that *a'* and *b'* are ε-similar and (ii) there are no other such sequences with a longer length, we define *a'* and *b'* as the longest ε-similar subsequence (LSS) of *a* and *b*. The length of the longest ε-similar can then be used to measure the similarity of the two sequences. The similarity ratio of the two subsequences is defined as *p/min(a,b)*.

An example of an ε-similar subsequence (LSS) of length 4 is shown in Figure 1 below.

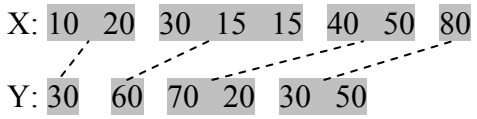

Figure 1: LSS example with *ε=0*

If we choose *ε=0* and *s[j]=t[j]* for all j in the definition of partial sum subsequence (i. e., no non-trivial partial sums allowed), the LSS becomes LCS. Therefore, the LSS problem is a generalized LCS problem. See Figure 2 for an LSS example that is also an LCS.

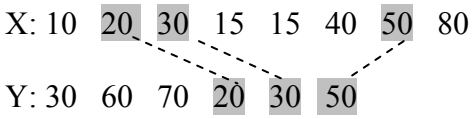

Figure 2: *LCS example*

To solve the LSS problem, a brute-force approach is to enumerate all subsequences of *a[1..m]* and check each partial sum subsequences to see if it is also a partial sum subsequence of *b[1..n]*, keeping track of the longest subsequence found. For a sequences of *a[1..m]*, there are many partial sum subsequences, making it impractical for long sequences. The well known LCS problem has an *O(mn)* algorithm [CL01], where *m* and *n* are the lengths of the two sequences, based on Dynamic Programming technique to find a solution to the LCS problem [KC72, MP80].

Like the LCS problem, the LSS problem has an optimal-substructure property, so it is possible to use Dynamic Programming technique to solve the problem. But, the LSS problem has partial sums involved, the solution of the LSS problem is more complex than the LCS problem, the solution to LSS problem requires more time comparing to LCS problem.

3. Partial Sum Selection

A partial sum may contain one or many consecutive elements of a sequence. For one element of a sequence, there are many ways to define its partial sum starting and ending elements. Thus extra complexity will be introduced. To reduce the problem space, we focus on one particular way to define partial sum starting and ending elements. If there is any optimal solution to the LSS problem, there shall exist an optimal solution consisting of partial sum selected by the defined method.

We start with defining of the partial sum selection and followed with the proof of its optimal property.

Given sequences *X[1..m]* and *Y[1..n]* and the maximum length of partial sum *s*, we choose to calculate partial sums backward starting from x_i and y_j, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$. There are *s* number of combinations of partial sums starting from x_i backward with maximum length of *s*, such as $\{(x_i), (x_{i-1}+x_i), \ldots, (x_{i-s+1}+x_{i-s}+..+x_i)\}$. There are the same number of partial sums for y_j . The set of partial sums are defined as Partial Sums ending at x_i and Partial Sums ending at y_j respectively. Figure 3 shows Partial Sums ending at x_i and y_j.

$$X: x_1 \ \ldots \ x_{i-s} \ \boxed{x_{i-s+1} \ \ldots \ x_{i-1} \ \ x_i} \ \ldots \ x_m$$

$$Y: y_1 \ \ldots \ y_{j-s} \ \boxed{y_{j-s+1} \ \ldots \ y_{j-1} \ \ y_j} \ \ldots \ y_n$$

Figure 3: Partial Sums ending at x_i and y_j

To check if there is any ε-similarity between Partial Sums ending at x_i and y_j, there are s^2 match evaluations. It is possible that there are several partial sums being ε-similar at the same time. To reduce the problem space, we don't need to consider all cases of those matched partial sums as long as we can find the one which can produce the Longest ε-similar Subsequence. The Minimum Matched Partial Sums can produce the longest ε-similar subsequence, thus we will only focus on this form of partial sum.

For convenience, we denote "≈" as "is ε-similar with" and "≠" as "not ε-similar with".

Minimum Matched Partial Sum (MMPS): Given sequences $X=\{x_1, x_2, \ldots, x_n\}$ and $Y=\{y_1, y_2, \ldots, y_n\}$ and, x'_i and x''_i are any Partial Sum ending at x_i respectively and, y'_j and y''_j are any Partial Sum ending at y_j respectively. Let

$$x'_i = \sum_{k=i-u}^{i} x_k \ ,$$

$$x''_i = \sum_{k=i-u'}^{i} x_k \ ,$$

$$y'_j = \sum_{k=j-v}^{j} y_k \ , \text{ and}$$

$$y''_j = \sum_{k=j-v'}^{j} y_k \ ,$$

where $1 \leq i \leq m$, $1 \leq j \leq n$, $0 \leq (u,u') < i$, $0 \leq (v,v') < j$. If $x'_i \approx y'_j$, $x''_i \approx y''_j$, $x'_i < x''_i$ and $y'_j < y''_j$, then x'_i and y'_j are a pair of Minimum Matched Partial Sum (MMPS).

Figure 4 shows two pairs of ε-similar partial sums where $x_i^{'} = \sum_{k=i-u}^{i} x_k$ is ε-similar to $y_j^{'} = \sum_{k=j-v}^{j} y_k$, *and* $x_i^{"} = \sum_{k=i-u'}^{i} x_k$ is ε-similar to $y_j^{"} = \sum_{k=j-v'}^{j} y_k$, x'_i, y'_j . Even though x''_i and y''_j are ε-similar, but only $x_i^{'}$ and $y_j^{'}$ forms a pair of Matched Partial Sum *with minimum value*, thus they are Minimum Matched Partial Sum (MMPS)

$X = x_1\ x_2 \ldots x_{i-u'} \ldots x_{i-u} \ldots x_{i-1}\ x_i\ x_{i+1} \ldots x_m$

$Y = y_1\ y_2 \ldots y_{j-v'} \ldots y_{j-v} \ldots y_{j-1}\ y_j\ y_{j+1} \ldots y_n$

Figure 4: Minimum Matched Partial Sums

Theorem 1: Given sequences $X=\{x_1, x_2, \ldots, x_m\}$ and $Y=\{y_1, y_2, \ldots, y_n\}$ and, x'_i is a Partial Sums ending at x_i and, y'_j is a Partial Sums ending at y_j. Let $x'_i = \sum_{k=i-u}^{i} x_k$, $y'_j = \sum_{k=j-v}^{j} y_k$, where $1 \le i \le m$, $1 \le j \le n$, $0 \le u < i$, $0 \le v < j$. x'_i and y'_j are pair of Minimum Matched Partial Sum (MMPS). MMPS has a property of optimal solution to LSS problem for sequences of X and Y.

Proof: MMPS have the longest prefix sequences $X[1 \ldots u-1]$ and $Y[1 \ldots v-1]$. Larger Matched Partial Sums have shorter prefix subsequences. MMPS's prefix sequences have the most elements appended to shorter prefix sequences $X[1 \ldots u'-1]$ and $Y[1 \ldots v'-1]$ of other larger Matched Partial Sums. Thus it will produce at least the same length of ε-similar subsequence as other shorter prefix sequences. If MMPS can't produce longer ε-similar subsequence, other matched partial sums can't do so either. ■

By only considering Minimum Matched Partial Sum ending at x_i and y_j respectively, we avoid further evaluation of all other larger candidate Partial Sums, thus computation time for those candidate Partial Sums is reduced, resulting in a better performance.

4. Property and Solution of LSS

4.1 Characterizing a Longest ε-Similar Subsequence

Like the well known LCS problem, the LSS problem has an optimal-substructure property. The natural classes of sub-problems correspond to pairs of "prefixes" of the two input sequences.

Theorem 2 (Optimal substructure of an LSS) Let $X=\{x_1, x_2, \ldots, x_m\}$ and $Y=\{y_1, y_2, \ldots, y_n\}$ be sequences, and $Z=\{z_1, z_2, \ldots, z_k\}$ by any LSS of X and Y, and x'_m is a minimum matched partial sum (MMPS) ending at x_m, $x'_m = \sum_{i=u}^{m} x_i$, $1 \le u \le m$ y'_n is a MMPS ending at y_n , $y'_n = \sum_{j=v}^{n} y_v$, $1 \le v \le n$. $x'_m = x_m$, $y'_n = y_n$ if x'_m, y'_n is not a MMPS.

Then there are 3 cases:

1. if $x'_m \approx y'_n$, then $z_k \approx x'_m$ and y'_n, and z_{k-1} is an LSS of X_{u-1} and Y_{v-1}.
2. if $x'_m \neq y'_n$, and $z_k \neq x'_m$, implies that Z is an LSS of X_{m-1} and Y.
3. if $x'_m \neq y'_n$, and $z_k \neq y'_n$, implies that Z is an LSS of X and Y_{n-1}.

Proof: (1) If $z_k \neq x'_m$, then we could append $x'_m \approx y'_n$ to Z to obtain a ε-similar subsequence of X and Y of length $k+1$, contradicting the supposition that Z is a Longest ε-similar subsequence of X and Y. Thus, we must have $z_k \approx x'_m \approx y'_n$. Now, the prefix Z_{k-1} is length of $(k-1)$ ε-similar subsequence of x_{m-u} and y_{n-v}. We wish to show it is an LSS. Suppose for purpose of contradiction that there is a ε-similar subsequence W of x_{u-1} and y_{v-1} with length greater than $k-1$. Then, appending $x'_m \approx y'_n$ to W produces a ε-similar subsequence of X and Y whose length is greater than k, which is a contradiction.

(2) If $x'_m \neq y'_n$, then Z is a ε-similar subsequence of X_{m-1} and Y. If there were a ε-similar subsequence W of X_{m-1} and Y with length greater than k, then W would also be a ε-similar subsequence X_m and Y, contradicting to the assumption that Z is an LSS of X and Y.

(3) The proof is symmetric to (2).

The theorem shows that an LSS of two sequences contains an LSS of prefixes of the two sequences. Thus, the LSS problem has an optimal-substructure property. ■

4.2 A Recursive Solution

Since the LSS problem has an optimal-substructure property, we could apply dynamic programming technique to solve this problem. To find LSS of X and Y, we may need to find the LSS's of X and Y_{n-1} and of X_{m-1} and Y. But each of these sub-problems has the same sub-sub-problem.

To summarize it all, we enumerate the sub-problems:

1. if $x'_m \approx y'_n$, then the sub-problem is the LSS of X_{u-1} and Y_{v-1};
2. if $x'_m \neq y'_n$, then the sub-problem is the LSS of X_m and Y_{n-1} or of X_{m-1} and Y_n.

Let $c[i,j]$ to be the length of an LSS of the sequences X_i and Y_j. If either $i=0$ or $j=0$, one of the sequences has length 0, so the LSS has length 0. The optimal substructure of LSS problem gives the recursive formula:

$c[i,j]=max\{(c[u-1,v-1]+\delta), c[i-1,j], c[i,j-1]\}$ *where*

$\delta=1$, *if* $x'_i \approx y'_j$, $x'_i = \sum_{k=u}^{i} x_k$,

$y'_j = \sum_{k=v}^{j} y_k$ *and* $1 \le i \le m$, $1 \le u < i$, $1 \le j \le n$, $1 \le v < j$.

$\delta=0$, *otherwise*

As stated in the above formula, a condition in the problem restricts which sub-problem we may consider. When $x'_i \approx y'_j$, we need to find the LSS of X_{u-1} and Y_{v-1}. Otherwise, we need to find LSS of X_i and Y_{j-1} or of X_{i-1} and Y_j.

4.3 Algorithm of Computing LSS

Procedure LSS_Length takes two sequences $X=\{x_1, x_2, ..., x_m\}$ and $Y=\{y_1, y_2, ..., y_n\}$ as inputs. It stores $c[i, j]$ values in a table $c[0..m, 0..n]$ whose entries are computed in row-major order. The first row and column are initiated to 0. When a ε-similar matched Partial Sum (MMPS) is found,

- $b[1..m, 1..n]$: stores a pointer to the optimal structure.
- $s[1..m, 1..n]$: $s[i,j]=0$ if it is not a entry element of MMPS, otherwise 1.

The procedure returns c, b, s tables. Table $c[m,n]$ contains the length of an LSS of X and Y. The subroutine MMPS1() was called for each x_i and y_j to check if there exists a ε-similar minimum matched Partial Sum (MMPS) ending at x_i and y_j.

```
LSS_LENGTH (X, Y)
  m = length[X];
  n = length[Y];
  c = 0;
  for i = 1 to m
  for j =1 to n
    if (MMPS1(i,j) is true) and
       (c[u-1,v-1]+1 >=
          max(c[i-1,j],c[i,j-1])) {
      c[i,j]=c[u-1,v-1]+1;
      b[i,j] = (u-1, v-1);
      s[i,j] =1;
    } else {
      s[i,j] =0;
```

```
        if c[i,j-1]>c[i-1,j] {
          c[i,j] = c[i,j-1];
          b[i,j] = (i, j-1);
        } else {
          c[i,j] = c[i-1,j];
          b[i,j] = c[i-1,j];
        }
      }
return (c,b,s)
```

Subroutine MMPS1() checks if there is MMPS ending at x_i and y_j, backward from minimum partial sum to larger ones. The subroutine MMPS1() will return the indices of the partial sum if a Minimum Matched Partial Sum was found, and saves running time by skipping to check all other larger Partial Sums.

```
MMPS1(i, j)
  for u = i to 1
    sum1 = sum(x[u..i]);
    for v = j to 1{
      sum2 = sum(y[v..j]);
      if (two sums are ε-similar)
        return (true, u, v)
    }
  return (false,0,0)
```

Figure 7 shows the result of an example using by the algorithm. The cells in row *i* and column *j* contains *c[i,j], b[i,j],* and *s[i,j]*. We combine the three tables into one for easier display. We use angle bracket to represent the pointer *b*[i,j]. The number before it is the *c* value and the number after it is the *s* value. The *c* value of 4 on the lowest right corner is the length of the LSS. MMPS's are illustrated in shaded rectangle boxes. When *s[i,j]*=1, it is a valid entry for MMPS. The LSS computed in this example is {30, 60, 90, 80}.

Computing the tables takes *O(mn)* iterations, and for each table entry, we need to check MMPS which takes *O(mn)* time each. Thus the total running time of the algorithm is $O(m^2n^2)$.

4.4 Constructing an LSS

The *b* and *s* table returned by LSS_LENGTH() can be used to construct an LSS of $X=\{x_1, x_2, ..., x_m\}$ and $Y=\{y_1, y_2, ..., y_n\}$. For row *i* and column *j*, when *s[i,j]=1*, it is a pair of valid partial sum ending at *(i,j)*. Pinter *b[i,j]=(u-1, v-1)* points to next element in the optimal structure. The pair of Partial Sums is *X[u…i]* and *Y[v…j]*. PRINT_LSS_X() traces the path of optimal structure, and prints elements of partial sums of *X[u…i]*.

The following procedure PRINT_LSS_X() running from right-lower corner of *b* table prints an LSS in terms of *X* elements in proper order:

```
PRINT_LSS_X (X, s, i, j)
   if (i=0 or j=0)
      return PRINT_LSS_X(X,s,b[i,j])
   if (i,j) is the entry element
       of Partial Sum ending at xi
   print Partial Sum elements
end
```

For *p[i,j]=(u-1, v-1)*, then elements of partial sum ending at x_i is *X[u…i],* and the elements of partial sum ending at y_j is *Y[v…j]*.

		1	2	3	4	5	6
	X \ Y	**30**	**60**	**70**	**20**	**30**	**50**
1	**10**	0<1,0>0	0<1,0>0	0<1,2>0	0<1,3>0	0<1,4>0	0<1,5>0
2	**20**	1<0,0>1	1<2,1>0	1<2,2>0	1<1,3>1	1<0,4>1	1<2,5>0
3	**30**	1<2,0>1	1<0,1>1	1<3,2>0	1<3,3>0	2<2,4>1	2<3,5>0
4	**15**	1<3,1>0	1<4,1>0	1<4,2>0	1<4,3>0	2<3,5>0	2<4,5>0
5	**15**	1<3,0>1	2<2,1>1	2<5,2>0	2<5,3>0	2<3,4>1	2<5,5>0
6	**40**	1<5,1>0	2<5,2>0	2<3,2>1	2<6,3>0	2<6,4>0	2<2,3>1
7	**50**	1<6,1>0	2<6,2>0	2<7,2>0	3<5,2>1	3<6,3>1	3<6,5>1
8	**80**	1<7,1>0	2<7,2>0	2<6,1>1	3<7,4>0	3<8,4>0	4<7,4>1

Figure 7: The combined table computed by LSS_LENGTH on sample sequences *X={10, 20, 30, 15, 15, 40, 50, 80}* and *Y={30, 60, 70, 20, 30, 50}*.

5. **More Efficient Algorithms**

As mentioned in Section 4.3, LENGTH_LSS() takes $O(m^2n^2)$ time. The algorithm is not practical if the lengths of the sequences are large. By examining the property of the partial sums, there is room to improve the efficiency of the algorithm.

5.1 A More Efficient Optimal Algorithm:

As we described in Section 3, for Sequences *X[1…m]* and *Y[1…n]*, Partial Sums ending at x_i are $\{x_i\}$, $\{x_i+x_{i-1}\}$, $\{x_i+x_{i-1}+x_{i-2}\}$, …, Partial Sums ending at y_j are $\{y_j\}$, $\{y_i+y_{j-1}\}$, $\{y_j+y_{j-1}+y_{j-2}\}$, …. They are in monotonous increasing order. Using this property, we can come up with a more efficient procedure of MMPS1(). The first step of MMPS2() puts partial sums ending at x_i and y_j into arrays *pSum_x[]* and *pSum_y[]*, then compares them starting from the smallest one until a ε-similar match found. If a match was found, returns true with the position of starting elements of the pair of matched partial sums, otherwise returns false. Since the two partial sum arrays are monotonously increasing lists, we can use the Merge procedure [HS97] to compare elements between the two ordered lists. Merging two sorted lists of sizes *m* and *n* takes *O(m+n)* time.

```
MMPS2(i, j)
    pSum_x = pSum_y = 0;
    for u=i to 1
        pSum_x[u] = pSum_x[u-1]+X[u];
    for v=j to 1
        pSum_y[v] = pSum_y[v-1]+Y[v];
    u = i; v = j;
    while(u>0 and v>0)
        if pSum_x[u] ε-similar to pSum_y[v]
            return (true, u, v)
        else if pSum_x[u] > pSum_y[v]
            do u--
        else if pSum_x[u] < pSum_y[v]
            do v--
    return (false, 0, 0)
```

The time complexity for MMPS2() is *O(m+n),* thus total time complexity for procedure LENGTH_LSS()is reduced to *O(mn(m+n))*.

5.2 A Heuristic Algorithm

In comparing similarity of thumbprints, it is highly unlikely a true similarity for a pair of matched partial sums consisting of a large number of elements. We have experienced that a match of a pair of partial sums with 50 and 101 elements respectively. By limiting the size of partial sum to *s*, we can come up with a heuristic algorithm MMPS3(). The time complexity for MMPS3() is *O(s)*, thus the total time complexity for LSS_LENGTH() is *O(smn)*. Due to limitation of the space and its similarity to MMPS2(), the algorithm is not listed here.

6. Experimental Result

We test our algorithm on two thumbprints with sequences of 224 and 176 positive real numbers. Values of the sequence's element range between 50,000 to 4,000,000 (microseconds in terms of packet gap). The program runs on a PC with Intel's Pentium III CPU. Different ε values are used to determine performance of the algorithm. The result is shown in Table 1. Average run time is about 0.8 seconds.

The result of experiment 1 shows that LSS length decreases when ε decreases. When ε is very small, the LSS elements consist of large sized partials sums. For example, when **ε** =0, the partial sum consists 50 elements of *X*, and 101 elements of *Y*. In thumbprint application, too large size of partial sums may not present the true similarity of two thumbprints. The experiment also suggested an ε around 0.1 as an ideal similarity ratio. The maximum partial sum size of 3 to 4 agrees with our intuition understanding of the thumbprints. If we limit the size of partial sums to a small constant number *s*, the time complexity will be decreased to *O(smn)*.

ε	LSS Length	Max. Partial Sum Size		Total elements of matched partial sums	
		X	***Y***	***X***	***Y***
0.200	134	3	4	168	165
0.150	127	4	3	164	146
0.100	114	4	4	155	153
0.050	95	8	5	149	146
0.010	65	11	6	125	125
0.001	34	7	7	101	121
0.000	1	50	101	50	101

Table 1: No Size Limitation on Partial Sums.
(X length = 254, Y length = 176)

With limiting partial sum size to *s=5*, we have a suboptimal solution as shown in the Table 2. Average run time is about 0.06 second. For large enough ε (10% or more), the solutions are actually optimal. For smaller ε, the solutions are not as good as the optimal solutions.

ε	LSS Length	Max. Partial Sum Size		Total elements of matched partial sums	
		X	***Y***	***X***	***Y***
0.200	134	3	4	168	165
0.150	127	4	3	164	146
0.100	114	4	4	155	153
0.050	95	5	5	149	146
0.010	64	5	5	118	113
0.001	32	5	5	84	86
0.000	0	0	0	0	0

Table 2: Maximum Partial Sum size is limited to *s=5*.
(*X* length = 254, *Y* length = 176)

Figure 6 compares the optimal and the suboptimal results. For the suboptimal algorithm, with the size of partial sum limited to 5, we still can get an almost identical result with optimal solution. The benefit of heuristic algorithm is that without sacrificing too much precision, the running time is much faster than optimal solution (10 times faster) making it feasible for real-time application.

7. Conclusion

Longest ε-Similar Subsequence (LSS) problem is a generalization of the Longest Common Subsequence (LCS) problem. In intrusion detection application on the Internet, it may be necessary to compare two sequences of thumbprint to see if there are similar. In order to do so, we provided a definition of similarity in this context. Even though we use a specific thumbprint (packet gap) as an example, the algorithm will apply in most other cases since most thumbprints consist of a sequence of numbers.

By analyzed the property of partial sums, we limited our computation to the minimum matched partial sum (MMPS) which leads to our optimal solution of LSS while reduces problem space.

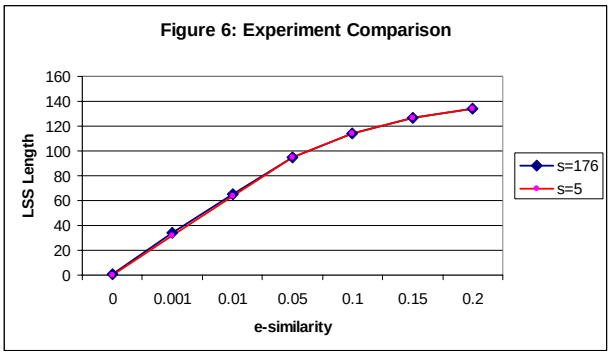

Figure 6 Comparison of Optimal and Suboptimal solutions.

With dynamic programming technique, an $O(m^2n^2)$ algorithm for the optimal solution to the LSS problem was described. Based on the property of partial sums, we derived a more efficient algorithm with time complexity of $O(mn(m+n))$. Practically, match-ups of very big sized minimum matched partial sum (MMPS), which summed up a large number of elements together, are likely to be false match-ups in thumbprint application. By limiting the size of MMPS to a small constant number s, we will have an algorithm with $O(smn)$ time complexity.

References:

[ZP00] Yin Zhang, Vern Paxson, "Detecting Stepping-Stone", Proceedings of the 9th USENIX Security Symposium, Denver, CO, August 2000, pp 67-81.

[YH05] Jianhua Yang, Shou-Hsuan Stephen Huang: "Matching TCP Packets and Its Application to the Detection of Long Connection Chains," Proceedings of IEEE International Conference on Advanced Information Networking and Applications, March 2005, Taipei, Taiwan, pp.1005-1010.

[CL01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms, Second Edition, pp. 350 – 355. The MIT Press, 2001.

[KC72] V. Chvatal, D. A. Klarner, and D.E. Knuth. Selected combinatorial research programs. Technical Report STAN-CS-72-292, Computer Science Department, Stanford University, 1972

[MP80] William J. Masek and Michael S. Paterson. "A faster algorithm computing string edit distances," Journal of Computer and System Science, 20(1):18-31, 1980.

[HS97] Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, Computer Algorithms, pp. 146-147. Computer Science Press, 1997.