Approximating the Longest Approximate Common Subsequence Problem

Wen-Chen Hu, Gerhard X. Ritter, Mark S. Schmalz
Center for Computer Vision and Visualization
Department of Computer and Information Sciences and Engineering
University of Florida, Gainesville, Florida 32611-6120
{wenchen, ritter, mssz}@cise.ufl.edu

Abstract — *Finding a longest common subsequence of two strings is a well-known problem. We generalize this problem to a* longest approximate common subsequence *problem that produces a maximum-gain approximate common subsequence of two strings. An approximate subsequence of a string X is a string edited from a subsequence of X. String Z is an approximate common subsequence of two strings X and Y if Z is an approximate subsequence of both X and Y. The gain function, g assigns a nonnegative real number to each subsequence. The problem is divided into smaller segments in order to lessen its complexity with some of these segments having been proven to be NP-hard. A heuristic approximation algorithm and an optimization neural network are constructed to find a near-optimal solution for the problem, where a ratio bound of the approximation algorithm is given, and a technique of interception is used to determine the values of the network weights. Some experimental results and the comparative performance of the two methods also are discussed.*

1 Introduction

Finding a longest common subsequence of two strings occurs in a number of computing and data-processing applications. A classical *longest common subsequence* problem [2] (abbreviated LCS) is, given two strings X and Y, to find a maximum length common subsequence of X and Y. A subsequence of a given string is just the given string with some symbols (possibly none) left out. String Z is a common subsequence of X and Y if Z is a subsequence of both X and Y. Finding an LCS is mainly used to measure the discrepancies between two strings. An LCS, however, does not always reveal the degree of difference between two strings that some problems require. For example, if $s_0 = \langle a, b\rangle$, $s_1 = \langle b, b\rangle$ and $s_2 = \langle b, a\rangle$, an LCS $\langle b\rangle$ of s_0 and s_1 is the same as an LCS of s_0 and s_2. From the viewpoint of LCS, the resemblance of s_1 and s_0 is the same as the resemblance of s_2 and s_0. However, s_2 has more symbols in common with s_0 than s_1 does, although not in the same order. Approximating an LCS may better characterize the discrepancies between two strings.

This paper addresses the *longest approximate common subsequence* problem (abbreviated LACS) that produces a maximum-gain approximate common subsequence of two strings. An approximate subsequence of a string X is a string edited from a subsequence of X. The only editing operation allowed here is an adjacent symbol interchange. String Z is an approximate common subsequence of two strings X and Y if Z is an approximate subsequence of both X and Y. The gain function g, which will be described later, assigns a nonnegative real number to each subsequence. Formally, the LACS problem is defined as follows: Given two strings X and Y, a weight $W_m>0$ for a symbol in an approximate common subsequence, and a weight $W_s\leq 0$ for an adjacent symbol interchange operation, a string Z is a longest approximate common subsequence of X and Y if Z satisfies the following two conditions:

1) Z is an approximate common subsequence of X and Y, and
2) the gain $g(X,Y,Z,W_m,W_s) = |Z|W_m+\delta(X,Z)W_s+\delta(Y,Z)W_s$ is maximum among all approximate common subsequences of X and Y, where $\delta(X,Z)$ is the minimum edit distance from a subsequence of X to Z, so is $\delta(Y,Z)$ to Y and Z.

A string Z is said to be of edit distance k to a string Z' if Z can be transformed to be equal to Z' with a minimum sequence of k adjacent symbol interchanges. The following is an LACS example. Let $X = \langle B, A, C, E, A, B\rangle$, $Y = \langle A, C, D, B, B, A\rangle$, $W_m = 3$, and $W_s = -1$. A longest approximate

common subsequence of X and Y is $Z = \langle A, B, C, B, A \rangle$ with the gain $g(X,Y,Z,W_m,W_s) = |Z|W_m+\delta(X,Z)W_s+\delta(Y,Z)W_s = 5\times3+2\times(-1)+1\times(-1) = 12$.

This paper is organized as follows. Analysis of the computational complexity of the LACS problem is provided in Section 2. Section 3 introduces a heuristic approximation algorithm for the LACS problem, including a ratio bound. An optimization neural network is designed in Section 4. Section 5 gives experimental results and a comparative performance of the above two methods. The final section examines further research directions and develops conclusions.

2 Computational Complexity

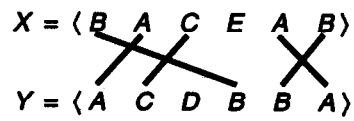
This section breaks up the complication of the LACS by sorting the problem into $LACS_i$, $0 \le i \le \min\{|X|,|Y|\}-1$ categories, where X and Y are input strings. Subsequently, we prove that $LACS_{\min\{|X|,|Y|\}-1}$ problem is NP-hard and conjecture that the problems from $LACS_1$ to $LACS_{\min\{|X|,|Y|\}-2}$ are at least as hard as the NP-complete problems.

2.1 LACS Categories

The LACS problem fits into categories according to the relation between the weights $W_m > 0$ and $W_s \le 0$:

- $LACS_0$—when $0 < W_m \le -W_s$, $LACS_0$ is reduced to an LCS problem since no adjacent symbol interchanges are allowed for any symbol in $LACS_0$;
- $LACS_1$—when $-W_s < W_m \le -2W_s$, any symbol in $LACS_1$ makes no more than 1 adjacent symbol interchange;
- $LACS_2$—when $-2W_s < W_m \le -3W_s$, any symbol in $LACS_2$ makes no more than 2 adjacent symbol interchanges;
- $LACS_i$—when $-iW_s < W_m \le -(i+1)W_s$, any symbol in $LACS_i$ makes no more than i adjacent symbol interchanges; and
- $LACS_{\min\{|X|,|Y|\}-1}$—when $-(\min\{|X|,|Y|\}-1)W_s < W_m$, X and Y are input strings, any symbol in $LACS_{\min\{|X|,|Y|\}-1}$ makes no more than $\min\{|X|,|Y|\}-1$ adjacent symbol interchanges, which is the maximum number of interchanges a symbol is allowed to make.

Another useful abbreviation is that an $LACS_i(X,Y)$ equals an $LACS_i$ of X and Y. The $LACS_i$ problem can be interpreted in another way called a *trace* [6]. Diagrammatically aligning the input strings X and Y and drawing lines from symbols in X to their matches in Y provides the trace of X and Y. Figure 1 illustrates the example in Section 1 through trace. In an $LACS_i$ trace, each line is allowed to have a maximum of i line-crossings, i.e. the symbol touched by the line may make no more than i adjacent symbol interchanges. The total number of line-crossings in a trace is $\delta(X,Z) + \delta(Y,Z)$.



$LACS_2(X,Y) = Z = \langle A\ B\ C\ B\ A \rangle$

$g(X,Y,Z,3,-1) = 5\times3+3\times(-1) = 12$

Figure 1: An $LACS_2$ Illustrated Through Trace

2.2 The $LACS_{\min\{|X|,|Y|\}-1}$ Problem is NP-Hard

In Theorem I, we show that any instance of extended string-to-string correction problem, which was proven to be an NP-complete problem by Wagner in 1975 [5], can be reduced in polynomial time to an instance of $LACS_{\min\{|X|,|Y|\}-1}$. The extended string-to-string correction problem (ESSCP)—given finite alphabet Σ, two strings X and $Y \in \Sigma^*$, and a positive integer k—determines whether there is a way to derive the string Y from the string X by a sequence of k or fewer operations of single symbol deletion or adjacent symbol interchange.

Theorem I ($LACS_{\min\{|X|,|Y|\}-1}$ is NP-hard)
If X and Y are input strings, then the $LACS_{\min\{|X|,|Y|\}-1}$ problem is NP-hard.

Proof We first show that $LACS_{\min\{|X|,|Y|\}-1}$ does not belong to NP. Given an instance of the problem, we use as a certificate an $LACS_{\min\{|X|,|Y|\}-1}$ Z of X and Y. A verification algorithm checks if the gain $g(X,Y,Z,W_m,W_s) = |Z|W_m+\delta(X,Z)W_s+\delta(Y,Z)W_s \ge k'$, a nonnegative real number. From the ESSCP, we know it is unlikely to find $\delta(X',Z)$ and $\delta(Y',Z)$ in polynomial time, where X' and Y' are subsequences of X and Y, respectively. If $\delta(X',Z)$ and $\delta(Y',Z)$ cannot be found in polynomial time, then $\delta(X,Z)$ and $\delta(Y,Z)$ definitely cannot be found in polynomial time. Therefore, we could say with certainty that a polynomial-time verification algorithm does not exist.

To prove that $LACS_{\min\{|X|,|Y|\}-1}$ is NP-hard, we show that ESSCP $\le_p$ $LACS_{\min\{|X|,|Y|\}-1}$. In other words, any instance of ESSCP can be reduced in polynomial time to an instance of $LACS_{\min\{|X|,|Y|\}-1}$. Let the two input strings be the same at both problems. We now show that string X needs k operations of deletion or interchange to derive string Y if and only if the corresponding $LACS_{\min\{|X|,|Y|\}-1}(X,Y)$ problem has an LACS Z with a gain $g(X,Y,Z,W_m,W_s) \ge |Y|W_m+(k-|X|+|Y|)W_s$. Suppose that string X needs k operations of single symbol deletion or adjacent symbol interchange to derive string Y. The number of deletions

has to be $|X|-|Y|$, making the number of interchanges $k-|X|+|Y|$. Thus, for an $LACS_{\min\{|X|,|Y|\}-1}(X,Y)$ trace, there are $|Y|=|Z|$ lines (matchings) and $k-|X|+|Y|$ line-crossings (interchanges). Therefore, the gain $g(X,Y,Z,W_m,W_s)$ is $|Y|W_m+(k-|X|+|Y|)W_s$. Conversely, suppose that $Z = LACS_{\min\{|X|,|Y|\}-1}(X,Y)$ has a gain $g(X,Y,Z,W_m,W_s) = |Y|W_m+(k-|X|+|Y|)W_s$. Because X has $|Y|$ symbols in common with Y, Z could equal Y. X then needs $|X|-|Y|$ deletions and $k-|X|+|Y|$ interchanges to derive Z, i.e., Y. Thus, for the ESSCP, string X needs $|X|-|Y|$ deletions and $k-|Y|+|X|$ interchanges to derive string Y. The total number of operations required to derive Y from X, therefore, is $k = (|X|-|Y|)+(k-|Y|+|X|)$. □

3 A Heuristic Approximation Algorithm

Despite the unlikelihood of finding a polynomial-time algorithm for solving the LACS problem exactly, near-optimal solutions in polynomial time may still be possible.

3.1 ALACS Algorithm

An approximation algorithm finds an approximate LACS (abbreviated ALACS) of two strings. The procedure APPROX_LACS repeatedly calls an LCS-routine $\lceil W_m/W_s \rceil$ times, beginning with allowing zero number of edit operations for each selected symbol. Each round, the allowable number of edit operations is increased by one. The symbols selected are marked off from the input strings to prevent consideration in the next round. Suppose X and Y are input strings, and the trace T is empty and $i = 0$ at first. It executes the following steps.

1. Find an LCS of X and Y.
2. Select symbols from the LCS such that each symbol makes no more than i line-crossings in trace T.
3. Eliminate the selected symbols from X and Y, and add them to T.
4. $i = i + 1$.
5. Repeat the above steps until $i \geq \lceil W_m/W_s \rceil$.

The procedure uses two arrays $m[0..|X|]$ and $n[0..|Y|]$ to flag which symbols in X and Y are selected. If a symbol is selected, it stores the index of matching symbol of the other string. If the symbol is not selected, it stores 0. An LCS function *LCS_LENGTH* is borrowed from [1], and is modified to include checking whether symbols in X and Y are selected.

```
APPROX_LACS (X, Y, W_m, W_s)
      /* X and Y: input strings */
      /* W_m: weight for a symbol in an approximate
common subsequence */
      /* W_s: weight for an adjacent common symbol
interchange operation */
1   for h ← 0 to ⌈W_m/W_s⌉ - 1
2       do for i ← 1 to |X|
3             do m[i] ← 0
4          for j ← 0 to |Y|
5             do n[j] ← 0
6          LCS_LENGTH (X, Y, m, n, b)
7          SELECT_SYM (X, Y, |X|, |Y|, m, n, b, h)
```

Procedure SELECT_SYM finds an LCS and selects symbols from it. It makes sure the selected symbols do not make more than the maximum allowable number of edit operations.

```
SELECT_SYM (X, Y, i, j, m, n, b, h)
1   if i = 0 or j = 0
2      then return
3   if b[i,j] = ↖
4     then if ¬OVER_CROSSING(X, Y, i, j, m, n, h)
5             then m[i] ← j
6                  n[j] ← i
7          SELECT_SYM(X, Y, i-1, j-1, m, n, b, h)
8   elseif b[i,j] = '↑'
9     then SELECT_SYM(X, Y, i-1, j, m, n, b, h)
10  else SELECT_SYM(X, Y, i, j-1, m, n, b, h)
```

Procedure OVER_CROSSING checks a line from the ith symbol of X to the jth symbol of Y does not cross more than h other lines in trace.

```
OVER_CROSSING (X, Y, i, j, m, n, h)
1   h1 = 0
2   for i1 ← 1 to i-1
3       do if m[i1] > j
4             then h1 = h1 + 1
5   for j1 ← 1 to j-1
6       do if n[j1] > i
7             then h1 = h1 + 1
8   if h1 > h
9      then return TRUE
10     else return FALSE
```

For this approximation algorithm, the running time is $O(\lceil W_m/W_s \rceil (|X|+|Y|)^2)$ and the space needs is $O(|X||Y|)$ for the *LCS_LENGTH* function. Figure 2 illustrates the progress of the approximation algorithm on an instance.

We name the trace T after executing the ith LCS-routine

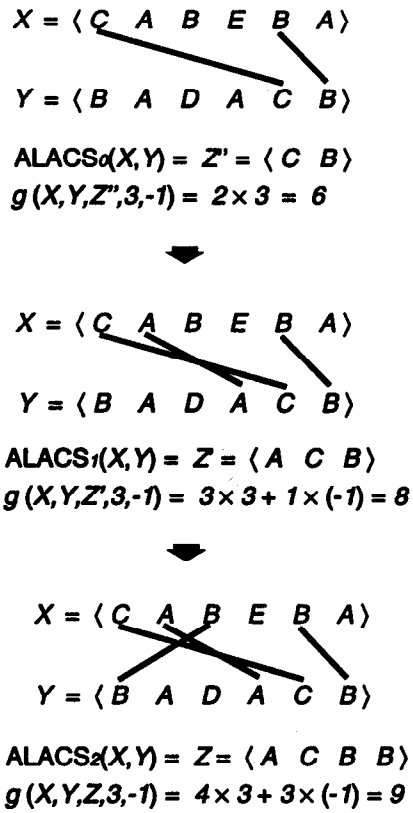


Figure 2: The Sequence of $ALACS_{0..2}$ Produced by APPROX_LACS on an Instance of LACS Problem

as an $ALACS_{i-1}$, $1 \le i \le \min\{|X|,|Y|\}$. The $ALACS_i$ means a symbol in $ALACS_i$ can make no more than i adjacent symbol interchanges. Figure 3 is an instance of an $LACS_2$ problem. The difference between the optimal gain and approximate gain is *1*.

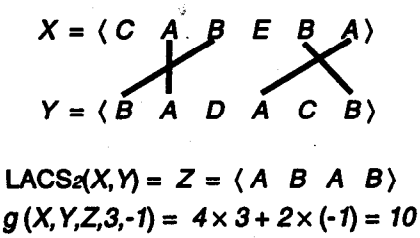


Figure 3: An $LACS_2$

3.2 The Ratio Bound

For a maximization problem, we say that an approximation algorithm for the problem has a *ratio bound* $\rho(n)$ if for any input of size n, the gain G of the solution produced by the approximation algorithm is within a factor of $\rho(n)$ of the gain G^* of an optimal solution, namely, $G^*/G \le \rho(n)$. For an $ALACS_i(X,Y)$, $0 \le i \le \min\{|X|,|Y|\}-1$, problem, the worst ratio bound is:

$$\rho_i(n) = \frac{max\ gain\ of\ LACS_i}{min\ gain\ of\ ALACS_i}. \quad (1)$$

The next two lemmas show how to compute the minimum gain of $ALACS_i$ and the maximum gain of $LACS_i$, respectively.

Lemma I (Minimum gain of $ALACS_i$)
If Z is an LCS of strings X and Y, the minimum gain of an $ALACS_i(X,Y)$, $0 \le i \le \min\{|X|,|Y|\}-1$, is $|Z|W_m$ for any i.

Proof The procedure APPROX_LACS is implemented by calling the LCS-routine $\lceil W_m/W_s \rceil$ times. Each time, they select symbols from the LCS, add them to the ALACS, and remove them from input strings. Since it selects every symbols from the first LCS, the minimum gain of the approximation algorithm is $|Z|W_m$. □

Before discussing Lemma II, some terminologies need introduction. A set of lines is completely-crossing if each line crosses every other line in the set. If a set of lines is independent, then every line in the set does not cross any line in the other set.

Lemma II (Maximum gain of $LACS_i$)
If Z is an LCS of strings X and Y, the maximum gain of an $LACS_i(X,Y)$, $0 \le i \le \min\{|X|,|Y|\}-1$, is $(i+2)|Z|W_m/2$.

Proof We show that an $LACS_i$ has a maximum gain when there are $|Z|$ independent, completely-crossing and $i+1$-line sets in its trace. The requirement can be broken into four conditions which are examined separately.

1. $|Z|$ sets: Every independent set contributes at least one symbol to an LCS. If the number of independent sets is more than $|Z|$, then the length of the LCS of X and Y is longer than $|Z|$. It contradicts the assumption that Z is an LCS of X and Y. On the other hand, the gain is not maximum if the number of independent sets is less than $|Z|$ since each set contributes a fixed gain, which will be explained later. Therefore, the number of independent sets is $|Z|$ when $LACS_i$ has a maximum gain.
2. Completely-crossing: Suppose one independent set is not completely-crossing, then there must have some lines not crossing one another, i.e., the lines are parallel in the sense of LCS. Each of the parallel lines contributes one symbol to an LCS. Since the length of the LCS is fixed, an $LACS_i$ achieves the maximum total gain only by having the maximum gain from each line. For an $LACS_i$, the maximum gain one line can reach

is when the set of this line and other i lines is completely-crossing. Therefore, each set has to be completely-crossing in order to have the maximum gain of an $LACS_i$.

3. Independent: Every set is independent, otherwise it is not completely-crossing.
4. $i+1$-line: From the above discussion, we know each set is completely-crossing. For a line in the set, it is only allowed to have no more than i line-crossings because of the limitation of $LACS_i$. Therefore, the maximum lines (also the maximum gain) a set can own is $i+1$.

In short, there are $|Z|$ sets in the trace, where each set has $i+1$ lines and $1+2+\ldots+i = i(i+1)/2$ line-crossings, when an $LACS_i$ has a maximum gain. Consequently, the maximum gain of an $LACS_i$ is $|Z|[(i+1)W_m+i(i+1)W_s/2] \le (i+2)|Z|W_m/2$ since $-iW_s<W_m\le-(i+1)W_s$. □

Theorem II gives the worst ratio bound of $ALACS_i$ by applying the above two lemmas. Since we take a very conservative approach to find $\rho_i(n)$, the actual ratio bound is expected to be much less than $(i+2)/2$ when $i > i_0$, a positive constant.

Theorem II (The worst ratio bound of ALACS$_i$) The worst ratio bound $\rho_i(n)$ of an $ALACS_i(X,Y)$, $0\le i\le\min\{|X|,|Y|\}-1$, is $(i+2)/2$ for any input size n.

Proof From Equation 1 and Lemmas I and II, the worst ratio bound $\rho_i(n)$ of an $ALACS_i(X,Y)$, $0\le i\le\min\{|X|,|Y|\}-1$, is $[(i+2)|Z|W_m/2]/(|Z|W_m) = (i+2)/2$ no matter what the input size n is. □

4 An Optimization Neural Network

A modified Hopfield neural network is designed to solve the LACS problem. A technique of interception is used to determine the values of network weight.

4.1 The Hopfield-Style Network

Hopfield [3] discovered a Liapunov function as the energy function of the network:

$$E = \left(-\frac{1}{2}\right)\sum_i\sum_j W_{ij}O_iO_j - \sum_j I_jO_j + \sum_j \theta_jO_j$$

In solving an LACS problem, this energy function is compared with another function built from LACS constraints in order to determine the network weights. Let strings $X = \langle x_1, x_2, \ldots, x_m\rangle$ and $Y = \langle y_1, y_2, \ldots, y_n\rangle$ be an instance of an LACS problem. The Hopfield net involves mn units represented as an $m\times n$ array. The energy function constructed from the LACS problem constraints is

$$E = E_1 + E_2 + E_3$$

where

$$E_1 = \frac{A}{2}\sum_x\left(\sum_y g_{xy}O_{xy} - h1_x\right)^2,$$
$$E_2 = \frac{B}{2}\sum_y\left(\sum_x g_{xy}O_{xy} - h2_y\right)^2, \text{ and}$$
$$E_3 = \frac{C}{2}\sum_x\sum_y\sum_{x'=x+1}^{|X|}\sum_{y'=1}^{y-1}O_{xy}O_{x'y'}.$$

A, B and C are constants. O_{ij}, with $1 \le i \le m$ and $1 \le j \le n$, indicates whether x_i matches y_j. Functions g_{ij}, $h1_i$ and $h2_j$ are

$$g_{ij} = \begin{cases} 0 & \text{if } x_i \ne y_j \\ 1 & \text{if } x_i = y_j \end{cases},$$
$$h1_i = |Y|_{x_i}/|X|_{x_i}, \text{ and } h2_j = |X|_{y_j}/|Y|_{y_j},$$

where $|X|_{x_i}$ is the number of symbol x_i in string X. E_1 and E_2 reflect the constraints that each row (X) or column (Y) contains a fraction $h1$ or $h2$ of a single 1. E_3 reflects the constraint that the minimum number of line-crossings is favored. By comparing this energy function with the Liapunov function, the weight and the external input are given by

$$W_{xy,x'y'} = -Ag_{xy}g_{xy'}\delta_{xx'} - Bg_{xy}g_{x'y}\delta_{yy'} - C\lambda_{xx'}\lambda_{y'y} \text{ and}$$
$$I_{xy} = Ag_{xy}h1_x + Bg_{xy}h2_y + \theta_{xy}$$

where

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \ne j \\ 1 & \text{if } i = j \end{cases} \text{ and } \lambda_{ij} = \begin{cases} 0 & \text{if } i \ge j \\ 1 & \text{if } i < j \end{cases}.$$

Some results are not valid when the input string is beyond a certain length, e.g. about 7. The validity can be improved by changing the coefficient values, but this may have the undesirable effect of sacrificing the gain. Therefore, the validity of results is checked after convergence, and two actions are taken to preserve the validity. When a line makes more than $\lceil W_m/W_s\rceil - 1$ line-crossings, it is canceled; and when a symbol is picked more than once, only the first pick is accounted for.

4.2 A Coefficient Value Determined by Interception

It has been observed that the convergence and the results of Hopfield net to the LACS problem is highly dependent upon the coefficients, and different input strings may have different optimal coefficient values. The values of coefficients A and B relative to the value of C affect the net, i.e. A/C (or B/C) affects the net. With sufficiently large values for A and B, the low-energy states will represent valid results and the maximum number of matchings, while a large value for C ensures a minimum number of line-crossings. The threshold value θ_j is fixed. Thus only the

value of coefficient A needs to be decided. Figure 4 shows a typical curve of gains and coefficient values. It resembles the shape of a trapezoid with the top slowing declining, and eventually becoming a constant. The value of coefficient A is determined by the interception of two lines, which are extrapolated from the two sides of the curve. From some random instance experiment, the peak of the curve occurs at about $A = 10$, and the curve becomes constant at $A \geq 200$. So $A = 1$ and $A = 5$ are picked for deciding the line on the left hand side of the curve, and $A = 100$ and $A = 200$ are picked for the other line. It turns out that the value of A is almost always selected correctly. This is because the curve shown in Figure 4 applies to most strings.

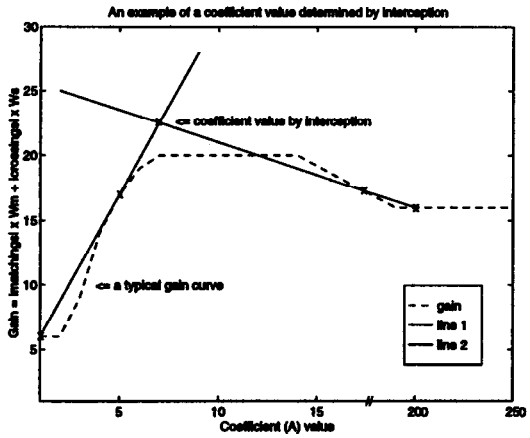


Figure 4: An Example of a Coefficient Value Determined by Interception

5 Experimental Results of the Two Approximation Methods

Figure 5 draws a relation of gains and running times for $W_m=6$ and $W_s=-1$. The output gains of ALACS and

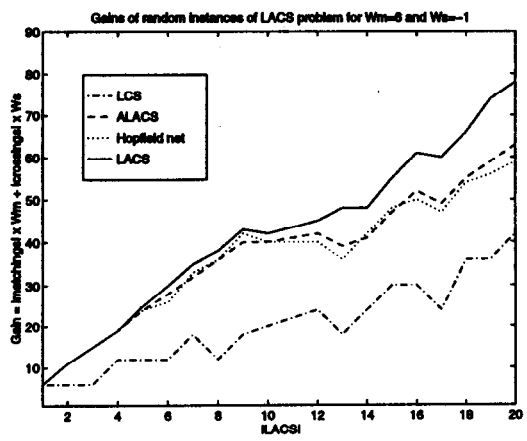


Figure 5: Gains and Running Time of Random Instances of LACS Problem for $W_m=6$ and $W_s=-1$

Hopfield net are pretty much the same, though the Hopfield net is a little bit poorer. The Hopfield net is much slower, but it requires less memory. The relation distance between LACS and ALACS (or Hopfield net), and also the distance between ALACS (or Hopfield net) and LCS, increases with increasing $W_m/-W_s$ or |LACS|. Table 1 lists the gains and running times of some random instances of LCS, ALACS, Hopfield net and LACS for $W_m=3$ and $W_s=-1$. The experimental results also show the ratio bound $\rho_i(n) = (i+2)/2$ of an $ALACS_i$ is exaggerated.

|LACS|	Gains and running times for LACS with $W_m=3$ and $W_s=-1$					
	LCS	ALACS		Hopfield net		LACS
	3|LCS|	gain	time	gain	time	gain
1	3	3	0.16	3	0.23	3
2	3	5	0.16	5	0.19	5
3	6	8	0.17	8	0.25	8
4	9	11	0.15	11	0.26	11
5	9	10	0.17	12	0.30	13
6	12	15	0.16	14	0.38	15
7	12	16	0.17	11	0.38	16
8	18	20	0.18	20	0.63	20
9	12	13	0.18	13	0.65	22
10	24	25	0.16	25	0.89	25

Table 1 Gains and Running Time of Some Random Instances of LACS Problem for $W_m=3$ and $W_s=-1$

6 Discussions and Conclusions

This paper is a preliminary look at the LCS approximation problem, but several open questions need answers before it becomes definitive. Thus far, only the last case of $LACS_i(X,Y)$, $1 \leq i \leq \min\{|X|,|Y|\}-1$, has been proven to be NP-hard. The other cases remain open, but it is likely that the vertex-cover problem [4] may be reduced in NP-hard proofs for these cases. For the heuristic approximation algorithm, an enduring difficulty is the worst ratio bound. Either it must prove to have a good average ratio bound, or another good ratio bound algorithm must be designed. Presently, only the worst ratio bound exists for the algorithm, which is $(i+2)/2$ for $ALACS_i$. It is believed that the actual ratio bound is a much smaller number. The performance of Hopfield-style net is slightly worse than the heuristic algorithm. This is due to the problem of local minima. Adding noise terms to the net input of each neuron is being investigated as a way to avoid this problem.

References

[1] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*, pages 314–319. The MIT Press, 1990.

[2] D.S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, October 1977.

[3] J.J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *In Proceedings of the National Academy of Science*, 81:3088–3092, 1984.

[4] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *In Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[5] R.A. Wagner. On the complexity of the extended string-to-string correction problem. *Proc. Seventh Annual ACM Symp. on Theory of Computing*, pages 218–223, 1975.

[6] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.