1985

Remark on Hsu-Du New Algorithm for the LCS Problem

A. Apostolico

Report Number:
85-544
REMARK ON HSU-DU NEW ALGORITHM FOR THE LCS PROBLEM

A. Apostolico

Department of Computer Sciences

Purdue University

West Lafayette, Indiana 47907

ABSTRACT

One of the time bounds claimed by Hsu and Du for the modification of Hirschberg's strategy set up by them is not correct. This is of some consequence, notably, it voids the claim, made elsewhere in the same paper, that the proposed algorithm performs better than the Hunt-Szymanski's strategy in cases of sparse matches. In fact it is pointed out here that just the opposite is true. In addition, there are other cases in which the new algorithm fails to achieve the superior performance that the authors claim, namely, all cases where the number of matches is large compared to the length of the shorter input string.
In their recent paper on the Longest Common Subsequence (LCS) Problem [HD], the authors present a modification of Hirschberg's strategy that requires $O(n \cdot \log s)$ preprocessing and $O(p \cdot m \cdot \log (n/m) + p \cdot m)$ processing time to find an LCS of two input strings $A$ and $B$. Here $n$ and $m \leq n$ are the lengths of $B$ and $A$, respectively; $p$ is the length of the LCS $C$, and $s$ is the cardinality of the alphabet. (We will conform to the terminology and notation in [HD], throughout.)

In formulating this result (Theorem 2 in [HD]), however, the authors also claim for their Algorithm 1 a bound of $O(K \cdot \log (pn/K) + K)$, in terms of the number $K$ of keys ( = the minimal $k$-candidates in [HI]) in the matrix $L[i,j]$ associated with the input strings. (Actually, $p$ is missing from the statement of Theorem 2, which might be attributable to a mistake in printing. At any rate, reasoning in terms of the bound which appears in the statement of the theorem would have the only effect of strengthening our point.) In a variety of applications, $K$ may be much smaller than the number $P$ of points (pairs of matching positions) of $A$ and $B$, which would lead to the conclusion that Algorithm 1 always outperforms asymptotically the $O(P \cdot \log n)$ Hunt-Szymanski strategy [HS], a conclusion that is indeed reported in [HD] at the beginning of Section IV. Moreover, according to the claim, pairs of input strings characterized with approximately $m \cdot n$ matches but with a $K$ very close to $p$ (in the extreme case, say, the two identical strings $A = B = a^n$) would be processed by Algorithm 1 in time proportional to $p \cdot \log n \sim m \cdot \log n \ (< n$, for $n >> m$) versus the $m \cdot n$ required by [HI] and the $m \cdot n \cdot \log n$ required by [HS].

It is easy to check that the $O(K \cdot \log (pn/K) + K)$ bound does not follow from, and in fact does not accurately reflect, the analysis which is offered to substantiate it. The only bound that can be consistently drawn from that analysis for the processing
time taken by Algorithm 1 is $O(m \cdot p + K \cdot \log(pn/K) + K)$. This rests on the immediate observation that Algorithm 1 consists of two nested loops, the outer one going through $p$ iterations, the inner one through exactly $m$ iterations. For the reader’s convenience, Algorithm 1 of Hsu-Du is reproduced below:

Algorithm 1

** $A[1..m], B[1..n]$ are the input strings; $Low[i]$ points to the smallest $j$-value to be considered at row $i$; with $\theta = A[i]$, $High[\theta]$ points to the largest $j$-value to be considered at row $i$; $Thresh$ is the current threshold; $PB[\theta,1] \cdots PB[\theta,N[\theta]]$ is the ordered list of $j$-values for which $B[j]$ is an occurrence of the symbol $\theta$ in $B$ **

0 $Low[i] = 1$ for $i = 1$ to $m$
1 $K = 0$
2 Repeat
3 $High[A[i]] = N[A[i]]$ for $i = 1$ to $m$
4 $K = K + 1$
5 $Key[K] = \phi$
6 $Thresh = \infty$
7 for $i = 1$ to $m$ do
8 \hspace{1em} $\theta = A[i]$
9 \hspace{1em} if $Low[i] \leq N[\theta]$
10 \hspace{1.5em} begin
11 \hspace{2em} $j = PB[\theta, Low[i]]$
12 \hspace{2em} $Case 1. j > Thresh$
13 \hspace{2.5em} $High[\theta] = Low[i]$
14 \hspace{1em} $Case 2. j = Thresh$
15 \hspace{2.5em} $High[\theta] = Low[i]$
16 \hspace{2.5em} $Low[i] = Low[i] + 1$
17 \hspace{1em} $Case 3. j < Thresh$
18 \hspace{2.5em} $Key(K) = (i,j)$
19 \hspace{2.5em} $Find (t, i, Low, High, Thresh)$
20 \hspace{2.5em} $Thresh = j$
21 \hspace{2.5em} $High[\theta] = Low[i]$
22 \hspace{2.5em} $Low[i] = t + 1$
23 \hspace{1em} end {if}
24 end {for}
25 until $Key[K] = \phi$
26 $p = K - 1$

It is interesting that, for each of its $p \cdot m$ overall iterations, Algorithm 1 performs at least one operation not inherent to the control of those iterations, namely, the test
It is also noteworthy that the cases that the authors report among the best for Algorithm 1 turn out in actuality to rank among the worst ones.

Indeed, in the "dense" case, reported above, of two identical strings consisting of \( n \) repetitions of just one symbol of the alphabet, Algorithm 1 spends the \( k \)-th iteration of the outer loop running through \( k-1 \) instances where test (6.2) is not passed, one instance of Case 3 and \( n-k \) instances of Case 2, in succession. Another extreme occurs when one considers the very "sparse" case of two identical \( n \)-symbol strings, both representing some permutation of the integers \( 1, 2, \ldots, n \). In this case the \( k \)-th iteration of the outer loop in Algorithm 1 consists of \( k-1 \) failures of test (6.2), one instance of Case 3 and \( n-k \) instances of Case 1. In general, given any input of size \( m+n \), it is not difficult to transform it into another input of size \( \Theta(n+m) \) that forces Algorithm 1 to execute at least \( \Theta(n \cdot m) \) non-control instructions. For instance, given \( A = a_1a_2\ldots a_m \), one may construct \( A' = a_1sa_2sa_3\ldots sa_mS \), with \( S \) an extra symbol not appearing in \( A \) and \( B \), and then run Algorithm 1 with input \( A' \) and \( BS \).

References

