A New Practical Linear Space Algorithm for the Longest Common Subsequence Problem[*]

H. Goeman, M. Clausen

Institut für Informatik V
Universität Bonn
Römerstraße 164
D–53117 Bonn
Germany

e-mail: {goeman,clausen}@cs.uni-bonn.de

Abstract. This paper deals with a new practical method for solving the longest common subsequence (LCS) problem. Given two strings of lengths m and n, $n \geq m$, on an alphabet of size s, we first present an algorithm which determines the length p of an LCS in $O(ns + \min\{mp, p(n-p)\})$ time and $O(ns)$ space. This result has been achieved before [Ric94, Ric95], but our algorithm is significantly faster than previous methods. We also provide a second algorithm which generates an LCS in $O(ns + \min\{mp, m \log m + p(n-p)\})$ time while preserving the linear space bound, thus solving the problem posed in [Ric94, Ric95]. Experimental results confirm the efficiency of our method.

Key words: Design and analysis of algorithms, edit distance, longest common subsequence.

1 Introduction

Let $x = x_1 \dots x_m$ and $y = y_1 \dots y_n$, $n \geq m$, be two strings over an alphabet $\Sigma = \{\sigma_1, \dots, \sigma_s\}$ of size s. A *subsequence* of x is a sequence of symbols obtained by deleting zero or more characters from x. The *Longest Common Subsequence* (LCS) *Problem* is to find a common subsequence of x and y which is of greatest possible length.

It will be convenient to describe the problem in another way. An ordered pair (k, ℓ), $1 \leq k \leq m$, $1 \leq \ell \leq n$, is called a *match* if $x_k = y_\ell$. The set M of all matches can be identified with a *matching matrix* of size $m \times n$ in which each match is marked with a dot. For example, if $x = abacbcba$ and $y = cbabbacac$, then M is as shown in Fig. 1 (a). Define a partial order $\ll$ on $\mathbb{N} \times \mathbb{N}$ by establishing $(k, \ell) \ll (k', \ell')$ iff both $k < k'$ and $\ell < \ell'$. A *chain* $C \subseteq M$ is a set of points which are pairwise comparable, i.e., for any two distinct $p_1, p_2 \in C$, either $p_1 \ll p_2$ or $p_1 \gg p_2$, where $p_1 \gg p_2$ means $p_2 \ll p_1$. Then the LCS problem can be viewed as finding a chain of maximal cardinality in M. One such chain is indicated as a path in Fig. 1 (b).

Finding an LCS is closely related with the computation of string edit distances [LW75, MP80, Wag75, WC76] and shortest common supersequences [GMS80]. It was

[*]Research supported by Deutsche Forschungsgemeinschaft, Grant CL 64/3-1

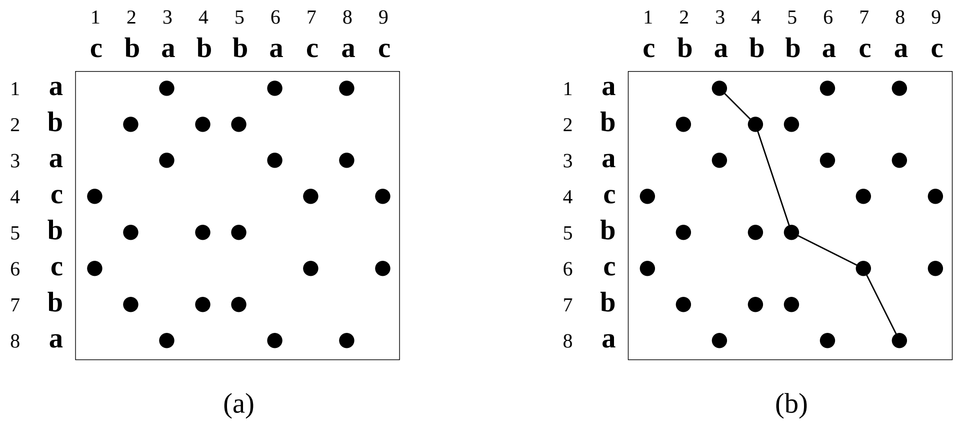

Figure 1: (a) matching matrix, (b) path representing an LCS.

first used by molecular biologists to study similar amino acids [Day65, Day69, NW70, SC73]. Other applications are in data compression [AHU76, GMS80, Mai78] and pattern recognition [FB73, LF78].

The LCS problem can be solved in $O(mn)$ time by a dynamic programming approach [SK83, WF74], while the asymptotically fastest general solution uses the "four russians" trick and takes $O(nm/\log n)$ time [MP80]. A lot of other algorithms have also been developed which are sensitive to other problem parameters, e.g., the length p of an LCS. They usually perform much better than the latter algorithms, although they all have a worst case time complexity at least of $\Omega(mn)$. For example, Hunt and Szymanski [HS77] have presented an $O((r+n)\log n)$ algorithm, where $r := |M|$. Thus their approch is fast when r is small, e.g., $r = O(n)$, but its worst–case time complexity is $O(n^2 \log n)$. Later, this has been improved to $O(mn)$ [Apo86]. There are also several routines which run in $O(n(n+1-p))$ or $O(n(m+1-p))$ time, and thus are efficient when an LCS is expected to be long [Mye86, NKY82, Ukk85, WMM90]. Other algorithms have running times $O(n(p+1))$ or $O(m(p+1))$ and should be used for short LCS [Apo87, AG87, Hir77, HD84]. However, it might be very difficult to *a priori* select a good strategy because in general the length p cannot be easily estimated. Also, when having a small alphabet, we can expect p to be of intermediate size, e.g., for $s = 4$, the average length of an LCS is bounded between $0.54 \cdot m \leq p \leq 0.71 \cdot m$ [CS75, DP94, Dek79, PD94, SK83]. Then none of the above methods performs well. Therefore recent research has been concentrated on more flexible algorithms which are efficient for short, intermediate, and long LCS, such as the method proposed by Chin/Poon [CP94]. Another approach from Rick [Ric94, Ric95] with running time $O(ns + \min\{mp, p(n-p)\})$ has been widely accepted as the fastest algorithm for the general LCS problem.

In this paper, we shall develop a new algorithm which is based on a kind of dualization of Rick's method. A detailed description of the theoretical background will be given in Sect. 2 and 3. Our idea does not improve the $O(ns + \min\{mp, p(n-p)\})$ time bound, but two important advantages are obtained. First, the number of matches processed while computing the length of an LCS is significantly decreased, resulting in a faster execution speed. The corresponding algorithm will be presented in Sect. 4. Second, when generating an LCS, we can achieve linear space through a divide–and–conquer scheme similar to that of other (but slower) algorithms [ABG92, Hir75, KR87]. This will be explained in Sect. 5. The methods mentioned before all need at least $\Omega(nm/\log n)$ space in their worst cases (see [PD94] for a survey), and most of them, including Rick's approach, cannot be combined with the divide–and–conquer technique. The open problem of a linear space implementation of Rick's algorithm [Ric95] is hereby solved. Experimental results presented in Sect. 6 confirm the efficiency of our method.

2 A New Approach to the LCS Problem

As already mentioned in the introduction, the LCS problem is equivalent to finding a chain of maximum cardinality in M. Dilworth's fundamental theorem [Dil50] states that this cardinality equals the minimum number of disjoint *antichains* into which M can be decomposed (an antichain of M consists of matches which are pairwise incomparable). In our example, this number (called the *Sperner number* of M) equals five. A suitable decomposition is shown in Fig. 2 (f). To find such a minimum decomposition, we first split $[1:m] \times [1:n]$ into subsets denoted by T^i, L^i, B^i, and R^i, where

$$\begin{aligned} T^i &:= \{i\} \times [i : n+1-i] \\ L^i &:= [i+1 : m+1-i] \times \{i\} \\ B^i &:= \{m+1-i\} \times [i+1 : n+1-i] \\ R^i &:= [i+1 : m-i] \times \{n+1-i\} \end{aligned}$$

and $1 \leq i \leq \lceil m/2 \rceil$ (see Fig. 2 (a) for an illustration). Additionally, let

$$T^{\leq i} := \bigcup\nolimits_{j \leq i} T^j, \quad L^{\leq i} := \bigcup\nolimits_{j \leq i} L^j, \quad B^{\leq i} := \bigcup\nolimits_{j \leq i} B^j, \quad R^{\leq i} := \bigcup\nolimits_{j \leq i} R^j \ .$$

Now for $i = 1, 2, \ldots, \lceil m/2 \rceil$, we construct four sets of antichains $A^{T,i}$, $A^{L,i}$, $A^{B,i}$, and $A^{R,i}$ which decompose (a suitable subset of) $T^{\leq i}$, $L^{\leq i}$, $B^{\leq i}$, and $R^{\leq i}$, respectively. The decompositions are generated by updating the previous sets, using the matches found in T^i, L^i, B^i, and R^i (details are given below). We use $A^{T,i}_u$ to denote an antichain in $A^{T,i}$, where u is an index between 1 and the size $e^{T,i} := |A^{T,i}|$ of $A^{T,i}$. Therefore $e^{T,i}$ is also called the *end index* of $A^{T,i}$. For $A^{L,i}$, $A^{B,i}$, and $A^{R,i}$, we introduce analogous notations. Furthermore, there are two *start indices* $s^{TL,i}$ and $s^{BR,i}$. The first one is used to split both $A^{T,i}$ and $A^{L,i}$ into two parts. One part contains all antichains with indices less than $s^{TL,i}$, and the other part consists of the rest. Only the latter part will be used for the updating process, whereas the former one will be copied to $A^{T,i+1}$ resp. $A^{L,i+1}$ without change. $s^{BR,i}$ similarly splits $A^{B,i}$ and $A^{R,i}$.

Fig. 2 (b), (c), (d), and (e) give a preview of the construction in the sample matching matrix after step $i = 1$, 2, 3, and 4, respectively. The centered grey box represents the remaining part of M which has not been processed so far. By our construction, with each step, it shrinks by two rows and columns.

We need the following terminology for the description of the construction process. For two antichains $C, D \subseteq M$ the set

$$IP(C,D) := \{p_1 \in C \mid \forall\, p_2 \in D\colon \neg(p_1 \ll p_2 \vee p_1 \gg p_2)\}$$

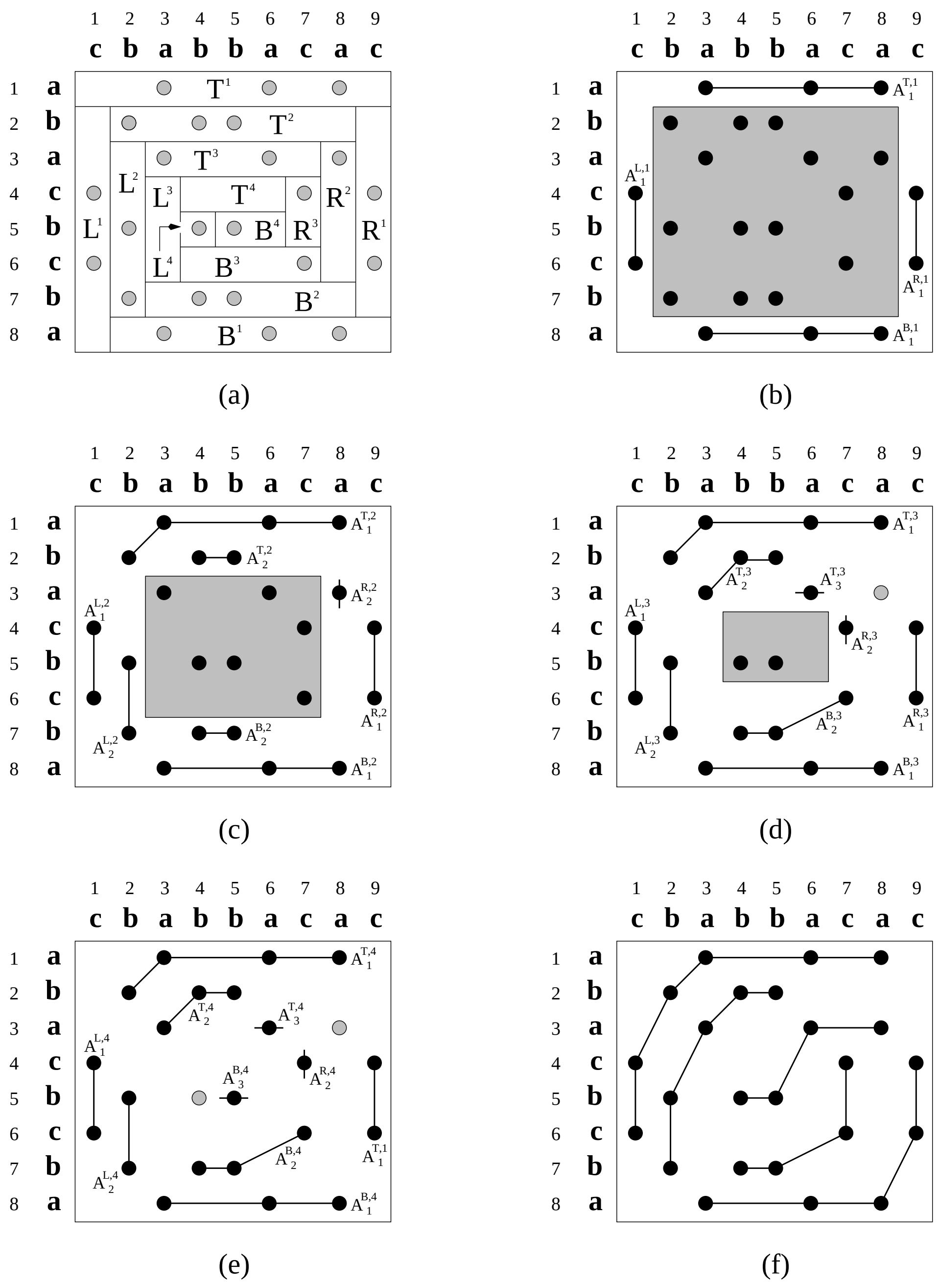

Figure 2: (a) splitting of M, (b)–(e) construction of antichains, (f) final decomposition.

is called the *incomparable part* of C relative to D. Clearly, $IP(C, D) \cup D$ is the greatest antichain above D contained in $C \cup D$. We say C is *incomparable* to D if $IP(C, D) = C$, and a single match $p_1 \in M$ is *incomparable* to D if $IP(\{p_1\}, D) = \{p_1\}$.

We are now prepared to discuss the generation of the antichains in more detail. Initially, there are no antichains, i.e., we have $A^{T,0} = A^{L,0} = A^{B,0} = A^{R,0} = \emptyset$ by initializing each start and end index to 1 and 0, respectively. Then, for each step $i = 1, \ldots, \lceil m/2 \rceil$, we start with T^i to determine $A^{T,i}$ from $A^{T,i-1}$. Let $s := s^{TL,i-1}$ and $e := e^{T,i-1}$. The first $s - 1$ antichains remain unchanged and are simply copied from $A^{T,i-1}$ to $A^{T,i}$. Now define $A^{T,i}_s$ as $A^{T,i-1}_s \cup IP(T^i \cap M, A^{T,i-1}_s)$. For example, when processing T^2 in Fig. 2 (b), $IP(T^2 \cap M, A^{T,1}_1) = \{(2, 2)\}$, and thus the match $(2, 2)$

combined with $A_1^{T,1}$ makes up $A_1^{T,2}$ as shown in Fig. 2 (c). Next, for $u = s+1, \ldots, e$, the antichain $A_u^{T,i-1}$ is handled in the same way to set up $A_u^{T,i}$, but only those matches in T^i not belonging to $A_s^{T,i}, \ldots, A_{u-1}^{T,i}$ are considered. Finally, we establish $s^{TL,i} := s$ and, if there are no matches left, $e^{T,i} := e$. Otherwise, we set $e^{T,i}$ to $e+1$ and collect all remaining matches in a new antichain $A_{e+1}^{T,i}$. Also, if $A^{R,i-1} \neq \emptyset$, we check whether its last antichain $A_{\tilde{e}}^{R,i-1}$, $\tilde{e} := e^{R,i-1}$, is incomparable to $A_{e+1}^{T,i}$. In this case we say $A_{\tilde{e}}^{R,i-1}$ is *inactivated* by $A_{e+1}^{T,i}$, and we remove $A_{\tilde{e}}^{R,i-1}$ from $A^{R,i}$ by setting $e^{R,i} := e^{R,i-1}$. Continuing our example with T^2 in Fig. 2 (b), we see there are two matches $(2,4)$ and $(2,5)$ left after processing $A_1^{T,2}$. Therefore a new antichain $A_2^{T,2}$ is created, but $A_1^{R,1}$ remains unchanged because, for example, $(2,4) \ll (4,9)$. The final set $A^{T,2}$ is shown in Fig. 2 (c) (the modifications to the other antichains are described below). Now let us consider the work involved with T^3. The match $(3,3)$ cannot be put into $A_1^{T,3}$, but into $A_2^{T,3}$, and the other match $(3,6)$ makes up the new antichain $A_3^{T,3}$. This time $(3,6)$ inactivates $(3,8)$, and thus $A_2^{R,2}$ is removed. The result is illustrated in Fig. 2 (d) (all matches located in deleted antichains are indicated by grey dots).

```
$S := T^i \cap M$;     (* Determine $A^{T,i}$ *)
For $u := s^{TL,i-1}$ To $e^{T,i-1}$ Do {
   $A_u^{T,i} := A_u^{T,i-1} \cup IP(S, A_u^{T,i-1})$;
   $S := S \setminus IP(S, A_u^{T,i-1})$;
};
If $S \neq \emptyset$ Then {
   $e^{T,i} := e^{T,i-1} + 1$; $e := e^{T,i}$; $A_e^{T,i} := S$;
   $e^{R,i} := e^{R,i-1}$; $\tilde{e} := e^{R,i}$;
   If $s^{BR,i-1} \leq e^{R,i-1}$ Then {
      If $IP(A_{\tilde{e}}^{R,i-1}, A_e^{T,i}) = A_{\tilde{e}}^{R,i-1}$ Then {
         $D^{TR} := D^{TR} \cup A_{\tilde{e}}^{R,i-1}$;
         $e^{R,i} := \tilde{e} - 1$;
      };
   };
} Else { $e^{T,i} := e^{T,i-1}$; $e^{R,i} := e^{R,i-1}$ };
For $u := 1$ To $s^{TL,i-1} - 1$ Do $A_u^{T,i} := A_u^{T,i-1}$;

$S := L^i \cap M$;     (* Determine $A^{L,i}$ *)
For $u := s^{TL,i-1}$ To $e^{L,i-1}$ Do {
   $A_u^{L,i} := A_u^{L,i-1} \cup IP(S, A_u^{L,i-1})$;
   $S := S \setminus IP(S, A_u^{L,i-1})$;
};
If $S \neq \emptyset$ Then {
   $e^{L,i} := e^{L,i-1} + 1$; $e := e^{L,i}$; $A_e^{L,i} := S$;
   $e^{B,i} := e^{B,i-1}$; $\tilde{e} := e^{B,i}$;
   If $s^{BR,i-1} \leq e^{B,i-1}$ Then {
      If $IP(A_{\tilde{e}}^{B,i-1}, A_e^{L,i}) = A_{\tilde{e}}^{B,i-1}$ Then {
         $D^{BL} := D^{BL} \cup A_{\tilde{e}}^{B,i-1}$;
         $e^{B,i} := \tilde{e} - 1$;
      };
   };
} Else { $e^{L,i} := e^{L,i-1}$; $e^{B,i} := e^{B,i-1}$ };
For $u := 1$ To $s^{TL,i-1} - 1$ Do $A_u^{L,i} := A_u^{L,i-1}$;
$s^{TL,i} := s^{TL,i-1}$;
```

(a)

```
$S := B^i \cap M$;     (* Determine $A^{B,i}$ *)
For $u := s^{BR,i-1}$ To $e^{B,i}$ Do {
   $A_u^{B,i} := A_u^{B,i-1} \cup IP(S, A_u^{B,i-1})$;
   $S := S \setminus IP(S, A_u^{B,i-1})$;
};
If $S \neq \emptyset$ Then {
   $e^{B,i} := e^{B,i} + 1$; $e := e^{B,i}$; $A_e^{B,i} := S$;
   If $s^{TL,i} \leq e^{L,i}$ Then {
      $\tilde{e} := e^{L,i}$;
      If $IP(A_{\tilde{e}}^{L,i}, A_e^{B,i}) = A_{\tilde{e}}^{L,i}$ Then {
         $D^{BL} := D^{BL} \cup A_{\tilde{e}}^{L,i}$;
         $e^{L,i} := \tilde{e} - 1$;
      };
   };
};
For $u := 1$ To $s^{BR,i-1} - 1$ Do $A_u^{B,i} := A_u^{B,i-1}$;

$S := R^i \cap M$;     (* Determine $A^{R,i}$ *)
For $u := s^{BR,i-1}$ To $e^{R,i}$ Do {
   $A_u^{R,i} := A_u^{R,i-1} \cup IP(S, A_u^{R,i-1})$;
   $S := S \setminus IP(S, A_u^{R,i-1})$;
};
If $S \neq \emptyset$ Then {
   $e^{R,i} := e^{R,i} + 1$; $e := e^{R,i}$; $A_e^{R,i} := S$;
   If $s^{TL,i} \leq e^{T,i}$ Then {
      $\tilde{e} := e^{T,i}$;
      If $IP(A_{\tilde{e}}^{T,i}, A_e^{R,i}) = A_{\tilde{e}}^{T,i}$ Then {
         $D^{TR} := D^{TR} \cup A_{\tilde{e}}^{T,i}$;
         $e^{T,i} := \tilde{e} - 1$;
      };
   };
};
For $u := 1$ To $s^{BR,i-1} - 1$ Do $A_u^{R,i} := A_u^{R,i-1}$;
$s^{BR,i} := s^{BR,i-1}$;
```

(b)

Figure 3: The algorithms for generating $A^{T,i}$ & $A^{L,i}$ (a), and $A^{B,i}$ & $A^{R,i}$ (b).

Having determined $A^{T,i}$, we continue with the necessary calculations for $A^{L,i}$ which are very similar. The first $s-1$ antichains are copied and then, for $u = s, \ldots, e^{L,i-1}$, $A_u^{L,i}$ is defined as the union of $A_u^{L,i-1}$ and the incomparable part of L^i relative to $A_u^{L,i-1}$, where only those matches are considered which have not already been used. Remaining matches form a new antichain and, if they are incomparable to the last antichain in $A^{B,i-1}$, we decrease $e^{B,i}$ by one. The corresponding algorithm in Fig. 3 (a) also introduces two additional sets D^{TR} and D^{BL} which contain all deleted matches. Details will be given in the next section.

Before processing $A^{B,i-1}$ and $A^{R,i-1}$ in an analogous way, we first check whether the first antichain in $A^{T,i}$ or $A^{L,i}$ is *TL–complete*, i.e., whether one of them contains a match (k,ℓ) such that $1 \leq k,\ell \leq i$. For example, in the configuration shown in Fig. 2 (c), $A_1^{T,2}$ is TL–complete due to the match $(2,2)$. As soon as $A_s^{T,i}$ is detected to be TL–complete, $s^{TL,i}$ is increased by one, thus the first antichains in both corresponding sets which are checked for additional matches remain unchanged from now on. If there is no such antichain in $A^{L,i}$ (i.e. $s > e^{L,i}$), but $s^{BR,i-1} \leq e^{B,i}$, then we additionally test whether $A_s^{T,i}$ is incomparable to the last antichain in $A^{B,i-1}$ and, should this situation arise, delete this antichain from $A^{B,i}$ by decreasing $e^{B,i}$.

Now assume $A_s^{L,i}$ is TL–complete. Then, as shown in Fig. 4 (a), we also increase $s^{TL,i}$, and similarly, if $s > e^{T,i}$ and $s^{BR,i-1} \leq e^{R,i}$, we decrease $e^{R,i}$ if $A_s^{L,i}$ inactivates the last antichain in $A^{R,i}$.

```
(* Check $A^{T,i}$ for TL–completeness *)
If $s^{TL,i} \leq e^{T,i}$ Then {
  $s := s^{TL,i}$;
  If $\exists\,(k,\ell) \in A_s^{T,i}:\ k,\ell \leq i$ Then {
    If $s > e^{L,i}$ Then {
      If $s^{BR,i-1} \leq e^{B,i}$ Then {
        $\tilde{e} := e^{B,i}$;
        If $IP(A_{\tilde{e}}^{B,i-1}, A_s^{T,i}) = A_{\tilde{e}}^{B,i-1}$ Then {
          $D^{BL,i} := D^{BL,i} \cup A_{\tilde{e}}^{B,i-1}$;
          $e^{B,i} := \tilde{e} - 1$;
        };
      };
      $e^{L,i} := s$; $A_s^{L,i} := \emptyset$;
    };
    $s^{TL,i} := s + 1$;
  };
};
(* Check $A^{L,i}$ for TL–completeness *)
If $s^{TL,i} \leq e^{L,i}$ Then {
  $s := s^{TL,i}$;
  If $\exists\,(k,\ell) \in A_s^{L,i}:\ 1 \leq k,\ell \leq i$ Then {
    If $s > e^{T,i}$ Then {
      If $s^{BR,i-1} \leq e^{R,i}$ Then {
        $\tilde{e} := e^{R,i}$;
        If $IP(A_{\tilde{e}}^{R,i-1}, A_s^{L,i}) = A_{\tilde{e}}^{R,i-1}$ Then {
          $D^{TR,i} := D^{TR,i} \cup A_{\tilde{e}}^{R,i-1}$;
          $e^{R,i} := \tilde{e} - 1$;
        };
      };
      $e^{T,i} := s$; $A_s^{T,i} := \emptyset$;
    };
    $s^{TL,i} := s + 1$;
  };
};
```

(a)

```
(* Check $A^{B,i}$ for BR–completeness *)
If $s^{BR,i} \leq e^{B,i}$ Then {
  $s := s^{BR,i}$;
  If $\exists\,(k,\ell) \in A_s^{B,i}:\ k > m-i \wedge \ell > n-i$ Then {
    If $s > e^{R,i}$ Then {
      If $s^{TL,i} \leq e^{T,i}$ Then {
        $\tilde{e} := e^{T,i}$;
        If $IP(A_{\tilde{e}}^{T,i}, A_e^{R,i}) = A_{\tilde{e}}^{T,i}$ Then {
          $D^{TR,i} := D^{TR,i} \cup A_{\tilde{e}}^{T,i}$;
          $e^{T,i} := \tilde{e} - 1$;
        };
      };
      $e^{R,i} := s$; $A_s^{R,i} := \emptyset$;
    };
    $s^{BR,i} := s + 1$;
  };
};
(* Check $A^{R,i}$ for BR–completeness *)
If $s^{BR,i} \leq e^{R,i}$ Then {
  $s := s^{BR,i}$;
  If $\exists\,(k,\ell) \in A_s^{R,i}:\ k > m-i \wedge \ell > n-i$ Then {
    If $s > e^{B,i}$ Then {
      If $s^{TL,i} \leq e^{L,i}$ Then {
        $\tilde{e} := e^{L,i}$;
        If $IP(A_{\tilde{e}}^{L,i}, A_e^{B,i}) = A_{\tilde{e}}^{L,i}$ Then {
          $D^{BL,i} := D^{BL,i} \cup A_{\tilde{e}}^{L,i}$;
          $e^{L,i} := \tilde{e} - 1$;
        };
      };
      $e^{B,i} := s$; $A_s^{B,i} := \emptyset$;
    };
    $s^{BR,i} := s + 1$;
  };
};
```

(b)

Figure 4: The algorithms for handling complete antichains in $A^{T,i}$ & $A^{L,i}$ (a), and in $A^{B,i}$ & $A^{R,i}$ (b).

The remaining work in step i concerns with the analogous construction of $A^{B,i}$ and $A^{R,i}$. (The analogue of TL–completeness is called *BR–completeness*. An antichain is BR–complete if it contains a match (k,ℓ) with $m-i < k \leq m$ and $n-i < \ell \leq n$.) Details are available from the algorithms shown in Fig. 3 (b) and Fig. 4 (b).

The main program shown in Fig. 5 is straightforward. Our next task is to elaborate the connection between the generated antichains and a minimal decomposition of M. This is done in the next section.

```
$i := 1$;      (* Initialization *)
$s^{T,0} := 1$; $s^{L,0} := 1$; $s^{B,0} := 1$; $s^{R,0} := 1$;
$e^{T,0} := 0$; $e^{L,0} := 0$; $e^{B,0} := 0$; $e^{R,0} := 0$;
**For** $i := 0$ **To** $\lceil m/2 \rceil$ **Do** $D^{TL,i} := \emptyset$;
**For** $i := 0$ **To** $\lfloor m/2 \rfloor$ **Do** $D^{BR,i} := \emptyset$;
**While** $i \leq \lfloor m/2 \rfloor$ **Do** {      (* Main loop *)
  Determine $A^{T,i}$ and $A^{L,i}$; (* see Fig. 3 (a) *)
  Look for TL-complete antichains in $A^{T,i}$ and $A^{L,i}$; (* see Fig. 4 (a) *)
  Determine $A^{B,i}$ and $A^{R,i}$; (* see Fig. 3 (b) *)
  Look for BR-complete antichains in $A^{B,i}$ and $A^{R,i}$; (* see Fig. 4 (b) *)
  $i := i + 1$;
};
**If** Odd($m$) **Then** {
  Determine $A^{T,\lceil m/2 \rceil}$ and $A^{L,\lceil m/2 \rceil}$; (* see Fig. 3 (a) *)
  Look for TL-complete antichains in $A^{T,\lceil m/2 \rceil}$ and $A^{L,\lceil m/2 \rceil}$; (* see Fig. 4 (a) *)
};
```

Figure 5: The main program for decomposing M

3 Analysis of the Construction

In this section, we study how to combine the antichains into larger ones such that a minimal decomposition of M is obtained. We further establish some results which later help us to construct an LCS in linear space.

Let us assume m is odd, and let $i = \lceil m/2 \rceil$. For technical reasons, we then put $A_u^{B,i} := A_u^{B,i-1}$ and $A_u^{R,i} := A_u^{R,i-1}$ for all $1 \leq u \leq e^{B,i-1}$ and $1 \leq u \leq e^{R,i-1}$. We also set $s^{BR,i} := s^{BR,i-1}$, $e^{B,i} := e^{B,i-1}$, and $e^{R,i} := e^{R,i-1}$. Furthermore, for $0 \leq i \leq \lceil m/2 \rceil$, we define $A_u^{T,i} := \emptyset$, $A_u^{L,i} := \emptyset$, $A_u^{B,i} := \emptyset$, and $A_u^{R,i} := \emptyset$ for $u > e^{T,i}$, $u > e^{L,i}$, $u > e^{B,i}$, and $u > e^{R,i}$, respectively.

Lemma 3.1 *Let $1 \leq i \leq \lceil m/2 \rceil$. Then the following holds:*

a) $\forall\, s^{TL,i-1} \leq u < v \leq e^{T,i}\ \forall p_1 \in A_v^{T,i}\ \exists p_2 \in A_u^{T,i}\colon\ p_1 \gg p_2$.

b) $\forall\, s^{TL,i-1} \leq u < v \leq e^{L,i}\ \forall p_1 \in A_v^{L,i}\ \exists p_2 \in A_u^{L,i}\colon\ p_1 \gg p_2$.

c) $\forall\, s^{BR,i-1} \leq u < v \leq e^{B,i}\ \forall p_1 \in A_v^{B,i}\ \exists p_2 \in A_u^{B,i}\colon\ p_1 \ll p_2$.

d) $\forall\, s^{BR,i-1} \leq u < v \leq e^{R,i}\ \forall p_1 \in A_v^{R,i}\ \exists p_2 \in A_u^{R,i}\colon\ p_1 \ll p_2$.

Proof. We only show the first claim, the other proofs are similar. Let $p_1 = (k, \ell)$. Since $A_v^{T,i} \subseteq T^{\leq \lceil m/2 \rceil}$, p_1 has been added to $A_v^{T,k}$ while processing T^k in step k, and $k \leq i$. Clearly, from the way S is handled in lines 1–5 of Fig. 3 (a), $p_1 \notin IP(T^k \cap M, A_j^{T,k-1})$, for $s^{TL,k-1} \leq j < v$. Hence, since $s^{TL,k-1} \leq s^{TL,i-1} \leq u < v$, there is some $p_2 \in A_u^{T,k-1}$ such that $p_1 \gg p_2$ or $p_1 \ll p_2$. But the second case would imply $p_2 \in T^{k'}$ for some $k' > k$ which is impossible during the first k steps of our construction. Finally observe that the algorithm never removes matches while updating an antichain, thus p_2 is still present in $A_u^{T,i}$. □

Lemma 3.2 *The following holds:*

a) $\forall\, 1 \le i \le \lceil m/2 \rceil \; \forall\, v\colon\; v < s^{TL,i} \iff A^{T,i}_v$ *or* $A^{L,i}_v$ *is TL–complete* .

b) $\forall\, 1 \le i \le \lceil m/2 \rceil \; \forall\, v\colon\; v < s^{BR,i} \iff A^{B,i}_v$ *or* $A^{R,i}_v$ *is BR–complete* .

Proof. We only prove the first claim, the other one is similar.
If. By contradiction, let i be the first step such that $A^{T,i}_v$ or $A^{L,i}_v$ is TL–complete, but $v \ge s^{TL,i}$. Clearly $v \ne s^{TL,i-1}$, otherwise the TL–completeness would have been detected by the algorithm shown in Fig. 4 (a), and thus, contradicting the property of v, we would have $v < s^{TL,i} = s^{TL,i-1}+1$. Hence $v > s^{TL,i-1}$. By the TL–completeness, there is some match $(k,\ell) \in A^{T,i}_v \cup A^{L,i}_v$ such that $1 \le k, \ell \le i$. Furthermore, by Lemma 3.1, there exists some match $(k',\ell') \in A^{T,i}_{v-1} \cup A^{L,i}_{v-1}$ such that $(k',\ell') \ll (k,l)$. But then $1 \le k', \ell' < i$, and therefore either $A^{T,i}_{v-1}$ or $A^{L,i}_{v-1}$ would be TL-complete after step $i-1$, a contradiction to the choice of i.
Only if. Obvious from the management of the start indices. □

Lemma 3.3 *For all* i, u *define* $A^{TL,i}_u := A^{T,i}_u \cup A^{L,i}_u$ *and* $A^{BR,i}_u := A^{B,i}_u \cup A^{R,i}_u$. *Then*

a) $\forall\, 0 \le i \le \lceil m/2 \rceil \; \forall\, 1 \le u \le \min\{e^{T,i}, e^{L,i}\}\colon\; A^{TL,i}_u$ *is an antichain* .

b) $\forall\, 0 \le i \le \lceil m/2 \rceil \; \forall\, 1 \le u \le \min\{e^{B,i}, e^{R,i}\}\colon\; A^{BR,i}_u$ *is an antichain* .

Proof. We prove the first claim by induction on i. The base $i = 0$ it trivial because $A^{T,0} = A^{L,0} = \emptyset$. For the induction step $i-1 \to i$, we consider three different cases.
Case a: $1 \le u < s^{TL,i-1}$. Then $A^{T,i}_u = A^{T,i-1}_u$ and $A^{L,i}_u = A^{L,i-1}_u$ (see lines 15 and 30 in Fig. 3 (a), respectively). Thus, by the induction hypothesis, $A^{TL,i}_u$ is an antichain.
Case b: $s^{TL,i-1} \le u \le \min\{e^{T,i-1}, e^{L,i-1}\}$. By definition the set $T := IP(S, A^{T,i-1}_u)$ added to $A^{T,i}_u$ in line 3 (Fig. 3 (a)) is incomparable to $A^{T,i-1}_u$, but it is also incomparable to $A^{L,i}_u$ as we now demonstrate. Let $(k,\ell) \in IP(S, A^{T,i-1}_u)$ and $(k',\ell') \in A^{L,i}_u$. Observe $k = i$ and $\ell \ge i$. Also note that $k' > \ell'$ and $\ell' \le i$ because $A^{L,i}_u \subseteq L^{\le i}$. Thus $(k,\ell) \ll (k',\ell')$ would contradict $\ell \ge i \ge \ell'$. Furthermore, $(k',\ell') \ll (k,\ell)$ would imply $\ell' < k' < k = i$, i.e., $A^{L,i-1}_u$ would be TL-complete, a contradiction to Lemma 3.2 and the choice of u. Similar arguments can be used for the set $L := IP(S, A^{L,i-1}_u)$ added to $A^{L,i}_u$ in line 19. Finally note that $T \subseteq T^i$ and $L \subseteq L^i$ are also incomparable.
Case c: $\min\{e^{T,i-1}, e^{L,i-1}\} < u \le \min\{e^{T,i}, e^{L,i}\}$. Clearly, this case is only possible if $u = e^{T,i} = e^{T,i-1}+1$ or $u = e^{L,i} = e^{L,i-1}+1$. If both conditions hold, then $A^{T,i}_u \subseteq T^i \cap M$ (lines 1 and 7) and $A^{L,i}_u \subseteq L^i \cap M$ (lines 17 and 23), thus their union obviously makes up an antichain. Otherwise, only one new antichain is generated whereas the other one is updated, and we can argument as in the second case to show that both antichains are incomparable.

The proof of the second claim is similar. □

Lemma 3.4 *Let* $1 \le i \le \lceil m/2 \rceil$. *Then the following holds:*

a) $\forall\, j \le \max\{e^{T,i}, e^{L,i}\} \; \forall\, p_j \in A^{TL,i}_j \; \exists\, p_1 \in A^{TL,i}_1, \ldots, p_{j-1} \in A^{TL,i}_{j-1}\colon$
$p_1 \ll \ldots \ll p_j$.

b) $\forall\, j \le \max\{e^{B,i}, e^{R,i}\}\ \forall\, p_j \in A_j^{BR,i}\ \exists\, p_1 \in A_1^{BR,i}, \ldots, p_{j-1} \in A_{j-1}^{BR,i}$:
$p_1 \gg \ldots \gg p_j$.

Proof. We prove the first claim by choosing p_v for $v = j-1, \ldots, 1$.

Consider step $j' \le i$ when p_{v+1} was added to $A_{v+1}^{TL,j'} \subseteq A_{v+1}^{TL,i}$. Then Lemma 3.1 implies the existence of p_v if $v \ge s^{TL,j'-1}$. Otherwise, by Lemma 3.2, $A_v^{T,j'-1}$ or $A_v^{L,j'-1}$ has been detected to be TL–complete before step j', i.e., $A_v^{TL,j'-1}$ contains a match (k', ℓ') such that $k', \ell' < j'$. But p_{v+1} is of the form (k, ℓ) with $k, \ell \ge j'$, thus we can choose $p_v := (k', \ell')$.

Similar arguments can be used for the second claim. □

Lemma 3.5 *For $0 \le i \le \lceil m/2 \rceil$, there are two chains*

$$C^{TR,i}, C^{BL,i} \subseteq T^{\le i} \cup L^{\le i} \cup B^{\le i} \cup R^{\le i}$$

of length $e^{T,i} + e^{R,i}$ and $e^{B,i} + e^{L,i}$, respectively.

Proof. We prove the existence of the first chain $C^{TR,i}$ by induction on i. The base $i = 0$ is trivial. For the induction step $(i-1) \to i$, we have to analyse the situations which cause $e^{T,i} + e^{R,i}$ to be greater than $e^{T,i-1} + e^{R,i-1}$. One such situation is given in lines 7–14 of Fig. 3 (a) if the condition in line 10 is not satisfied because then $e := e^{T,i} = e^{T,i-1} + 1$ and $\tilde{e} := e^{R,i} = e^{R,i-1}$. But since $IP(A_{\tilde{e}}^{R,i-1}, A_e^{T,i}) \ne A_{\tilde{e}}^{R,i-1}$ there exist two comparable matches $c^T \in A_e^{T,i}$ and $c^R \in A_{\tilde{e}}^{R,i-1}$. More precisely, since $c^T \in T^i$ and $c^R \in R^{\le i-1}$, we must have $(k, \ell) \ll (k', \ell')$. Thus, by Lemma 3.4, we can construct a chain

$$p_1 \ll \ldots \ll p_{e-1} \ll c^T \ll c^R \ll p'_{\tilde{e}-1} \ll \ldots \ll p'_1$$

of length $e + \tilde{e}$.

Similar arguments can be used for the remaining situations and for the other chain. □

Our next task is to reveal the structure in D^{TR} and D^{BL}. We shall show that for each deleted match there always is some antichain which is incomparable to this match. In order to prove this property, we keep track of each deleted match by *assigning* it to some antichain during the construction process. More precisely, whenever an antichain A is removed due to the existence of some other antichain B which inactivates it, all matches in A are assigned to B, e.g., considering the situation in Fig. 2 (d), the match $(3, 8)$ is assigned to $A_3^{T,3}$. Furthermore, all previously deleted matches assigned to A now also belong to B. The assigned matches are inherited when an antichain is updated, e.g., in Fig. 2 (e), $(3, 8)$ also belongs to $A_3^{T,4}$. These rules guarantee that after step i, each deleted match is assigned to exactly one antichain in $A^{T,i} \cup A^{L,i} \cup A^{B,i} \cup A^{R,i}$. We write $D(A)$ to denote the set of matches assigned to an antichain A.

Lemma 3.6 *Let $1 \le i \le \lceil m/2 \rceil$, and assume $(k, \ell) \in D(A)$ for some antichain A in $A^{T,i}$, $A^{L,i}$, $A^{B,i}$, or $A^{R,i}$. Then*

a) $(k, \ell) \in D^{TR} \implies \forall\, (k', \ell') \in A\colon\ k \le k' \wedge \ell \ge \ell'$.

b) $(k,\ell) \in D^{BL} \implies \forall\,(k',\ell') \in A\colon\ k \geq k' \wedge \ell \leq \ell'$.

Proof. For the first claim, let us assume (k,ℓ) was assigned to A while executing line 11 in Fig. 3 (a) during step $j \leq i$ (the following arguments can analogously be applied to the other instructions which modify D^{TR}). Thus $A = A^{T,i}_{e}$, where $e = e^{T,j}$. Now we consider two cases concerning the status of (k,ℓ) before step j.
Case a: $(k,\ell) \in A^{R,j-1}_{\tilde{e}} \subseteq R^{\leq j-1}$, $\tilde{e} = e^{R,j-1}$. Then $\ell > n - j + 1$. From lines 1, 6, 7, and 10 we see that (k,ℓ) is incomparable to any match (k'',ℓ'') in $A^{T,j}_{e}$. But $A^{T,j}_{e} \subseteq T^{j}$, thus $k'' = j$ and $\ell'' \leq n-j+1$. Hence, the incomparability implies $k \leq j$. Now observe that $A^{T,j}_{e}$ is the first constructed part of $A^{T,i}_{e}$, later extensions are taken from $T^{j+1},\ldots,T^{i}$. Thus every match $(k',\ell') \in A^{T,i}_{e}$ fulfills $k' \geq j$ and $\ell' \leq n-j+1$, and the claim follows.
Case b: (k,ℓ) is assigned to $A^{R,j-1}_{\tilde{e}}$. We can inductively assume

$$\forall\,(k'',\ell'') \in A^{R,j-1}_{\tilde{e}}\colon\ k \leq k'' \wedge \ell \geq \ell''$$

Deleted matches are never assigned to empty antichains. Thus there is at least one match $(k'',\ell'') \in A^{R,j-1}_{\tilde{e}}$, and we can prove as in the first case that $k'' \leq k'$ and $\ell'' \geq \ell'$. Hence we have $k \leq k'$ and $\ell \geq \ell'$.

The proof of the second claim follows similar arguments and is therefore omitted. □

Lemma 3.7 *Let* $1 \leq i \leq \lceil m/2 \rceil$*. Then the following holds:*

a) $\forall\, 1 \leq u \leq e^{T,i}\colon\ D^{BL} \cap D(A^{T,i}_{u}) \neq \emptyset \implies A^{L,i}_{u} = \emptyset \wedge A^{T,i}_{u}$ *is TL–complete .*

b) $\forall\, 1 \leq u \leq e^{L,i}\colon\ D^{TR} \cap D(A^{L,i}_{u}) \neq \emptyset \implies A^{T,i}_{u} = \emptyset \wedge A^{L,i}_{u}$ *is TL–complete .*

c) $\forall\, 1 \leq u \leq e^{B,i}\colon\ D^{TR} \cap D(A^{B,i}_{u}) \neq \emptyset \implies A^{R,i}_{u} = \emptyset \wedge A^{B,i}_{u}$ *is BR–complete .*

d) $\forall\, 1 \leq u \leq e^{R,i}\colon\ D^{BL} \cap D(A^{R,i}_{u}) \neq \emptyset \implies A^{B,i}_{u} = \emptyset \wedge A^{R,i}_{u}$ *is BR–complete .*

Proof. We again only show the first claim. From lines 10 and 11 in Fig. 3 (a), we see that all matches assigned there to $A^{T,i}_{u}$ are either placed into D^{TR}, or they have been assigned before to some non–complete antichain in $A^{R,i-1}$. But concerning the latter case, we see from lines 26 and 27 in Fig. 3 (b) that any such match has been put into D^{TR} as well, or again belongs to some non–complete antichain in $A^{T,j}$, $j < i$. Repeating this argument, we conclude that all matches assigned to $A^{T,i}$ are contained in D^{TR}. The only exception is given by lines 8 and 9 in Fig. 4 (a), where deleted matches are assigned to $A^{T,i}_{u}$, but added to D^{BL}. But then, from lines 3, 4, and 13, the claim follows. □

Lemma 3.8 *All matches assigned to an antichain A are pairwise incomparable, thus by Lemma 3.6, they extend the antichain to a larger one.*

Proof. Whenever a match is deleted, the algorithm always removes a complete antichain. By induction, this antichain B together with its assigned matches forms a larger antichain C. If there already is a set of matches D assigned to A (which is only possible when A is detected to be complete), then, following the arguments given in the proof of Lemma 3.7, $C \subseteq D^{BL}$ and $D \subseteq D^{TR}$ or vice versa, and Lemma 3.6 immediately implies that B and D are pairwise incomparable. $\square$

We are now prepared to construct a minimal decomposition of M. We start by decomposing $M \setminus (D^{TR} \cup D^{BL})$, the deleted matches are later considered in Thm. 3.9 below. The construction is as follows. Using Lemma 3.3, we combine the first $e^{TL} := \min\{e^{T,\lceil m/2\rceil}, e^{L,\lceil m/2\rceil}\}$ antichains in $A^{T,\lceil m/2\rceil}$ and $A^{L,\lceil m/2\rceil}$ to larger ones. We also connect the first $e^{BR} := \min\{e^{B,\lceil m/2\rceil}, e^{R,\lceil m/2\rceil}\}$ antichains in $A^{B,\lceil m/2\rceil}$ to the corresponding ones in $A^{R,\lceil m/2\rceil}$. For example, in Fig. 2 (e), we have $e^{T,\lceil m/2\rceil} = e^{B,\lceil m/2\rceil} = 3$ and $e^{L,\lceil m/2\rceil} = e^{R,\lceil m/2\rceil} = 2$, thus this generates four combined antichains. Concerning the remaining antichains we consider four different cases.
Case a: $e^{T,\lceil m/2\rceil} \leq e^{L,\lceil m/2\rceil}$ and $e^{B,\lceil m/2\rceil} \geq e^{R,\lceil m/2\rceil}$. Then we leave the remaining antichains as they are and have $p := e^{L,\lceil m/2\rceil} + e^{B,\lceil m/2\rceil}$ antichains in total. But by Lemma 3.5, there also exists a chain of this length. Thus, by Dilworth's theorem, the decomposition is minimal.
Case b: $e^{T,\lceil m/2\rceil} > e^{L,\lceil m/2\rceil}$ and $e^{B,\lceil m/2\rceil} \leq e^{R,\lceil m/2\rceil}$. Similar to the first case we have $p := e^{T,\lceil m/2\rceil} + e^{R,\lceil m/2\rceil}$ antichains, and also a chain of this length.
Case c: $e^{T,\lceil m/2\rceil} \leq e^{L,\lceil m/2\rceil}$ and $e^{B,\lceil m/2\rceil} < e^{R,\lceil m/2\rceil}$. From the management of the start and end indices, we have $e^{T,\lceil m/2\rceil} \geq s^{TL,\lceil m/2\rceil} - 1$. Thus, by Lemma 3.2, $A_u^{L,\lceil m/2\rceil}$ is not TL–complete for $u > e^{T,\lceil m/2\rceil}$. This implies $k > \lceil m/2\rceil$ and $\ell \leq \lceil m/2\rceil$ for any match $(k,\ell) \in A_u^{L,\lceil m/2\rceil} \subseteq L^{\leq\lceil m/2\rceil}$. For all $v > e^{B,\lceil m/2\rceil}$ and $(k',\ell') \in A_v^{R,\lceil m/2\rceil}$ we similarly have $k' \leq \lceil m/2\rceil$ and $\ell' > n - \lfloor m/2\rfloor \geq \lceil m/2\rceil$. Thus $A_u^{L,\lceil m/2\rceil}$ and $A_v^{R,\lceil m/2\rceil}$ are incomparable. Now assume $e^{L,\lceil m/2\rceil} \geq e^{R,\lceil m/2\rceil}$. Then we can connect all remaining antichains in $A^{R,\lceil m/2\rceil}$ to corresponding ones in $A^{L,\lceil m/2\rceil}$ and obtain $p := e^{L,\lceil m/2\rceil} + e^{B,\lceil m/2\rceil}$ antichains in total, thus again a minimal decomposition. If $e^{L,\lceil m/2\rceil} < e^{R,\lceil m/2\rceil}$, then similarly $p := e^{T,\lceil m/2\rceil} + e^{R,\lceil m/2\rceil}$ is the optimal length of a chain in $M \setminus (D^{TR} \cup D^{BL})$.
Case d: $e^{T,\lceil m/2\rceil} > e^{L,\lceil m/2\rceil}$ and $e^{B,\lceil m/2\rceil} > e^{R,\lceil m/2\rceil}$. Finding a minimal decomposition is slightly more complicated in this case. Consider the following algorithm. Starting with $u := e^{T,\lceil m/2\rceil}$ and $v := e^{R,\lceil m/2\rceil} + 1$, we check whether $A_u^{T,\lceil m/2\rceil}$ and $A_v^{B,\lceil m/2\rceil}$ are incomparable. If they are not, then we backup u and v in $\tilde{u}$ and $\tilde{v}$, respectively, and increase v by one. Otherwise the antichains are connected, u is set to $u-1$, and v is set to $v+1$. We repeat this until all remaining antichains in either $A^{T,\lceil m/2\rceil}$ or $A^{B,\lceil m/2\rceil}$ have been used, i.e., $u = e^{L,\lceil m/2\rceil}$ or $v > e^{B,\lceil m/2\rceil}$. Then the total number of antichains is $p := u + e^{B,\lceil m/2\rceil}$. Thus, if $u = e^{L,\lceil m/2\rceil}$, we have $p = e^{L,\lceil m/2\rceil} + e^{B,\lceil m/2\rceil}$, and the decomposition is optimal. Now assume $u > e^{L,\lceil m/2\rceil}$. If $\tilde{u}$ and $\tilde{v}$ are unused, then all remaining antichains in $A^{B,\lceil m/2\rceil}$ have been connected to corresponding antichains in $A^{T,\lceil m/2\rceil}$, and we have $p = e^{T,\lceil m/2\rceil} + e^{R,\lceil m/2\rceil}$. Hence, in this case the decomposition is also a minimal one. Finally assume that $\tilde{u}$ and $\tilde{v}$ have been used for saving u and v at least once. Then for $j = \tilde{v}+1, \ldots, e^{B,\lceil m/2\rceil}$, $A_j^{B,\lceil m/2\rceil}$ has been connected to $A_{\tilde{u}+\tilde{v}-j}^{T,\lceil m/2\rceil}$, and we have $u = \tilde{u} - (e^{B,\lceil m/2\rceil} - \tilde{v})$. Thus $p = \tilde{u} - (e^{B,\lceil m/2\rceil} - \tilde{v}) + e^{B,\lceil m/2\rceil} = \tilde{u} + \tilde{v}$. But from the properties of $\tilde{u}$ and $\tilde{v}$, it can be shown (similar to the proof of Lemma 3.5) that there is a chain of length $\tilde{u} + \tilde{v}$ which contains two matches $p_1 \in A_{\tilde{u}}^{T,\lceil m/2\rceil}$ and $p_2 \in A_{\tilde{v}}^{B,\lceil m/2\rceil}$. Hence, the constructed decomposition is optimal.

Let us consider our example. Case d applies to the situation in Fig. 2 (e), and $A_3^{T,4}$ is compared with $A_3^{B,4}$. Since these antichains are incomparable, they are connected, and we obtain a decomposition consisting of 5 antichains in total.

Theorem 3.9 *The length of an LCS in M equals p as defined in the four cases above.*

Proof. Consider a combined antichain A of the decomposition. Assume an antichain $A_u^{T,\lceil m/2\rceil} \in A^{T,\lceil m/2\rceil}$ is one component of it (otherwise, we can handle the following construction in a similar way).
Case a: $A_u^{T,\lceil m/2\rceil}$ is the only component of A. Then we extend A with the set B of deleted matches assigned to $A_u^{T,\lceil m/2\rceil}$. Lemma 3.8 guarantees that the result is still an antichain.
Case b: $A_u^{T,\lceil m/2\rceil}$ has been combined with $A_u^{L,\lceil m/2\rceil}$. By Lemma 3.7, $B \subseteq D^{TR}$. Let $(k,\ell) \in A_u^{L,\lceil m/2\rceil}$ and $(k',\ell') \in A_u^{T,\lceil m/2\rceil}$. From $(k,\ell) \in L^{\lceil m/2\rceil}$, $(k',\ell') \in T^{\lceil m/2\rceil}$, and the incomparability of (k,ℓ) and (k',ℓ'), we have $k \geq k' \wedge \ell \leq \ell'$. Now consider a match $(k'',\ell'') \in B$. By Lemma 3.6, we have $k \geq k' \geq k''$ and $\ell \leq \ell' \leq \ell''$. Hence, $A_u^{L,\lceil m/2\rceil}$ is incomparable to B. We can use a similar way to show that the set C of deleted matches assigned to $A_u^{L,\lceil m/2\rceil}$ is a subset of D^{BL} and incomparable to $A_u^{T,\lceil m/2\rceil}$. Finally, B and C are clearly incomparable as well. Thus $A_u^{T,\lceil m/2\rceil} \cup A_u^{L,\lceil m/2\rceil} \cup B \cup C$ is still an antichain.
Case c: $A_u^{T,\lceil m/2\rceil}$ has been combined with some other antichain $D \in A^{B,i}$. Then, similar to the proof of the second case, we can show that the union of A and the two corresponding sets of assigned matches still make up an antichain.

By handling each combined antichain in this way, we can construct a decomposition of M without generating any additional antichains. The proof is complete. □

Fig. 2 (f) illustrates the corresponding decomposition for our example.

4 Implementation

We now describe an efficient implementation for the given algorithm and analyse its time and space complexity.

All new antichains created in step i are extensions from antichains generated during step $i-1$. Furthermore, the only antichains used for decomposing M are from the last step. Thus for the implementation it is sufficient to update the antichains of interest. The same is true for the start and end indices, and we thus sometimes drop the index i from now on. The necessary information for each actual antichain can be kept in one single number as follows. Let $1 \leq i \leq \lceil m/2\rceil$ and $1 \leq u \leq e^{T,i}$. We define *ThreshT*$[u]$ as the leftmost column used by some match in $A_u^{T,i}$, i.e.,

$$\mathit{ThreshT}[u] := \min\{\ell \mid \exists\, k\colon\ (k,\ell) \in A_u^{T,i}\} \ .$$

For example, in Fig. 2 (b), *ThreshT*$[1] = 3$, and in Fig. 2 (d), *Top-Thresh*$[1] = 2$, *ThreshT*$[2] = 3$, and *ThreshT*$[3] = 6$. To update this array in each step, we use an auxiliary array *LeftPos* on $\Sigma \times [1:n+1]$ given by

$$\mathit{LeftPos}[c,\ell] := \min(\{n+1\} \cup \{j \mid \ell \leq j \leq n \ \wedge\ y_\ell = c\}) \ ,$$

i.e., *LeftPos*$[a_i,\ell]$ equals the column number of the leftmost occurence of a match in row i located right to column ℓ, and equals $n+1$ if there is no such match. In our example ($y = cbabbacac$), we obtain the following values:

a	3	3	3	6	6	6	8	8	10	10
b	2	2	4	4	5	10	10	10	10	10
c	1	7	7	7	7	7	7	9	9	10

Now it is not difficult to see that the following routine correctly updates *ThreshT* when processing T^i, representing lines 1–7 in Fig. 3 (a). (Similar procedures are used in [AG87, Ric94, Ric95] to determine *contours* which correspond to the antichains used here.)

```
$k := LeftPos[a_i, i]$;
For $u := s^{TL}$ To $e^T$ Do {
  $j := ThreshT[u]$;
  If $k \le j$ And $k \le n - i + 1$ Then {
    $ThreshT[u] := k$; $k := LeftPos[a_i, j+1]$;
  };
};
If $k \le n - i + 1$ Then { $e^T := e^T + 1$; $ThreshT[e^T] := k$ };
```

For $A^{L,i}$, $A^{B,i}$, and $A^{R,i}$ we introduce additional arrays *ThreshL*, *ThreshB*, and *ThreshR* which similarly store the topmost rows, rightmost columns, and bottommost rows used by the corresponding antichains. To handle them analogously to *ThreshT*, we also need three more auxiliary arrays given by

$$\begin{array}{lll} TopPos[c,k] & := \min(\{m+1\} \cup \{j \mid k \le j \le m \wedge x_j = c\}) \ , & (1 \le k \le m+1) \ , \\ RightPos[c,\ell] & := \max(\{0\} \cup \{j \mid 1 \le j \le \ell \wedge y_\ell = c\}) \ , & (0 \le \ell \le n) \ , \\ BottomPos[c,k] & := \max(\{0\} \cup \{j \mid 1 \le j \le k \wedge x_j = c\}) \ , & (0 \le k \le m) \ . \end{array}$$

Note that in Fig. 3 and Fig. 4, each test for the incomparability of two antichains can be replaced by a rather simple conditional statement. For example, considering line 10 in Fig. 3 (a), we know that all matches in T^i are located to the left of any match in $R^{\le i-1}$. Thus, with $e := e^{T,i}$ and $\tilde{e} := e^{R,i}$, A^T_e and $A^R_{\tilde{e}}$ are incomparable if and only if $A^R_{\tilde{e}}$ is also completely contained in the first i rows, i.e., $ThreshR[\tilde{e}] \le i$. The algorithm presented in Fig. 6 shows how the other situations are handled. It also makes use of some special implementation details which cannot be discussed here, e.g., the construction starts with the bottommost row instead of the topmost one when m is even. In Fig. 6 some lines are marked with a dot (•) on their left sides. These lines are used for the construction of an LCS and should be ignored for the moment.

The complexity of the algorithm may be deduced as follows. The four auxiliary arrays can be easily preprocessed in $O(ns)$ time and space, where $s = |\Sigma|$. Clearly, during one of the $\lceil m/2 \rceil$ iterations of the main loop, none of the four inner **While**–loops takes more than $O(p)$ time, and when determining p, at most $\lceil m/2 \rceil$ pairs of antichains have to be compared. Thus the algorithm takes at most $O(ns + mp)$ time. Furthermore, observe that the j–th antichain in A^T (which is added to A^T during some step $i \ge j$) must contain a match (k, ℓ) with $\ell \le n - (p - j)$, otherwise it would be impossible to construct a chain of length p. But then this antichain is detected to be TL–complete after step $n - (p - j)$, therefore it is only considered for at most $n - (p - j) - i \le n - p$ times in the corresponding **While**–loop (lines 59–65). Similar arguments can be given for antichains in A^L, A^B, and A^R. Hence, we have shown the following theorem.

```
      Determine $TopPos$ and $LeftPos$;
      Determine $BottomPos$ and $RightPos$;
      **For** $u := 0$ **To** $\lceil m/2 \rceil$ **Do** {
         $ThreshT[u] := 0$; $ThreshL[u] := 0$;
 5    };
      **For** $u := 0$ **To** $\lfloor m/2 \rfloor$ **Do** {
         $ThreshB[u] := n + 1$; $ThreshR[u] := m + 1$;
      };
      $t := 1$; $\ell := 1$; $b := m$; $r := n$;
10    $s^{TL} := 1$; $e^T := 0$; $e^L := 0$;
      $s^{BR} := 1$; $e^B := 0$; $e^R := 0$;

      **If** $Odd(m)$ **Then Goto** Line 57;

      **While** $t \le b$ **Do** {      (* Main loop *)

         $k := RightPos[x_b, r]$;      (* Update $A^B$ *)
15       $u := s^{BR}$;
         **While** $u \le e^B$ **Do** {
            $j := ThreshB[u]$;
            **If** $k \ge j$ **Then** {
               $ThreshB[u] := k$; $k := RightPos[x_b, j - 1]$;
20          };
            $u := u + 1$;
         };
         **If** $k \ge \ell$ **Then** {
            $e^B := u$; $ThreshB[e^B] := k$;
25          **If** $ThreshL[e^L] \ge b$ **Then** $e^L := e^L - 1$
 •             **Else** Update $c^B$, $c^L$, $\ell^{BL}$;
         };

         $k := BottomPos[y_r, b - 1]$;      (* Update $A^R$ *)
         $u := s^{BR}$;
30       **While** $u \le e^R$ **Do** {
            $j := ThreshR[u]$;
            **If** $k \ge j$ **Then** {
               $ThreshR[u] := k$; $k := BottomPos[y_r, j - 1]$;
            };
35          $u := u + 1$;
         };
         **If** $k \ge t$ **Then** {
            $e^R := u$; $ThreshR[e^R] := k$;
            **If** $ThreshT[e^T] \ge r$ **Then** $e^T := e^T - 1$
 •             **Else** Update $c^T$, $c^R$, $\ell^{TR}$;
         };

         (* Check for BR–complete antichains *)
         **If** $ThreshB[s^{BR}] = r$ **Then** {
            **If** $s^{BR} > e^R$ **Then** {
45             **If** $ThreshT[e^T] \ge r$ **Then** $e^T := e^T - 1$
 •                **Else** Update $c^T$, $c^R$, $\ell^{TR}$;
            };
            $s^{BR} := s^{BR} + 1$;
         } **Else If** $ThreshR[s^{BR}] = b$ **Then** {
50          **If** $s^{BR} > e^B$ **Then** {
               **If** $ThreshL[e^L] \ge b$ **Then** $e^L := e^L - 1$
 •                **Else** Update $c^B$, $c^L$, $\ell^{BL}$;
            };
            $s^{BR} := s^{BR} + 1$;
55       };
         $t := t + 1$; $\ell := \ell + 1$;

         $k := LeftPos[x_t, \ell]$;      (* Update $A^T$ *)
         $u := s^{TL}$;
         **While** $u \le e^T$ **Do** {
60          $j := ThreshT[u]$;
            **If** $k \le j$ **Then** {
               $ThreshT[u] := k$; $k := LeftPos[x_t, j + 1]$;
            };
            $u := u + 1$;
65       };
         **If** $k \le r$ **Then** {
            $e^T := u$; $ThreshT[e^T] := k$;
            **If** $ThreshR[e^R] \le t$ **Then** $e^R := e^R - 1$
 •             **Else** Update $c^T$, $c^R$, $\ell^{TR}$;
70       };

         $k := TopPos[y_l, t]$;      (* Update $A^L$ *)
         $u := s^{TL}$;
         **While** $u \le e^L$ **Do** {
            $j := ThreshL[u]$;
75          **If** $k \le j$ **Then** {
               $ThreshL[u] := k$; $k := TopPos[y_l, j + 1]$;
            };
            $u := u + 1$;
         };
80       **If** $k \le b$ **Then** {
            $e^L := u$; $ThreshL[e^L] := k$;
            **If** $ThreshB[e^B] \le \ell$ **Then** $e^B := e^B - 1$
 •             **Else** Update $c^B$, $c^L$, $\ell^{BL}$;
         };

85       (* Check for TL–complete antichains *)
         **If** $ThreshT[s^{TL}] = \ell$ **Then** {
            **If** $s^{TL} > e^L$ **Then** {
               **If** $ThreshB[e^B] \le \ell$ **Then** $e^B := e^B - 1$
 •                **Else** Update $c^B$, $c^L$, $\ell^{BL}$;
90          };
            $s^{TL} := s^{TL} + 1$;
         } **Else If** $ThreshL[s^{TL}] = t$ **Then** {
            **If** $s^{TL} > e^T$ **Then** {
               **If** $ThreshR[e^R] \le t$ **Then** $e^R := e^R - 1$
 •                **Else** Update $c^T$, $c^R$, $\ell^{TR}$;
            };
            $s^{TL} := s^{TL} + 1$;
         };
         $b := b - 1$; $r := r - 1$;
100   };

      (* Determine length $p$ of an LCS *)
      **If** $e^T > e^L$ **And** $e^B > e^R$ **Then** {
         **If** $s^{TL} \le e^L$ **Then** $s^{TL} := e^L + 1$;
         **If** $s^{BR} \le e^R$ **Then** $s^{BR} := e^R + 1$;
105      $u := e^T$; $v := s^{BR}$;
         **While** $u \ge s^{TL}$ **And** $v \le e^B$ **Do** {
            **If** $ThreshT[u] \ge ThreshB[v]$
               **Then** $u := u - 1$
 •             **Else** { $\tilde{u} := u$; $\tilde{v} := v$ };
110         $v := v + 1$;
         };
         $p := u + e^B$;
113   } **Else** $p := \max\{e^L + e^B, e^T + e^R\}$;
```

Figure 6: The $O(ns + \min\{mp, p(n-p)\})$ algorithm for determining the length p of an LCS.

Theorem 4.1 *The length p of an LCS can be computed in $O(ns + \min\{mp, p(n-p)\})$ time and $O(ns)$ space.*

This result has been achieved before by Rick [Ric94, Ric95], and in fact, the algorithm presented here is some kind of dualization of Rick's method, but our algorithm is significantly faster as we shall show in Sect. 6.

5 Construction of an LCS in Linear Space

This section deals with the generation of an LCS. The idea is to apply the divide–and–conquer scheme [ABG92, Hir75, KR87] which first identifies at least one point of an LCS such that this LCS is splitted into two parts of roughly the same size. Then the remainder is computed by recursive calls. The method presented here usually determines two LCS–neighbouring matches c^{TL} and c^{BR} which are located in $T^{\leq\lceil m/2\rceil} \cup L^{\leq\lceil m/2\rceil}$ and $B^{\leq\lceil m/2\rceil} \cup R^{\leq\lceil m/2\rceil}$, respectively. This is accomplished as follows.

In each step i of the construction described in Sect. 2, we subsequently update the following variables:

- p^{TL} is the match which caused $A_s^{T,i}$ or $A_s^{L,i}$ to become TL–complete, $s = s^{TL,i}-1$. For example, in Fig. 2 (c), $p^{TL} = (2,2)$, and in Fig. 2 (d) and (e), $p^{TL} = (3,3)$.
- p^{BR} has a corresponding meaning for the last BR–complete antichain in $A^{B,i}$ and $A^{R,i}$, e.g., in Fig. 2 (d), $p^{BR} = (6,7)$.
- c^T and c^R are the two matches introduced in the proof of Lemma 3.5. They both lie in $C^{TR,i}$ and are neighbours in this chain. Furthermore, c^T and c^R are always located in the first i topmost rows and i rightmost columns of M, respectively.
- c^B and c^L have analogous properties for $C^{BL,i}$.
- ℓ^{TR} and ℓ^{BL} is the position of c^T in $C^{TR,i}$ and of c^L in $C^{BL,i}$, respectively. Also, $\ell^{TR}+1$ and $\ell^{BL}+1$ is the position of c^R in $C^{TR,i}$ and of c^B in $C^{BL,i}$, respectively.

p^{TL} and p^{BR} can be easily updated. For example, consider lines 85–98 in Fig. 6 where new TL–complete antichains are handled. Let $p^{TL} = (u,v)$. If the condition in line 86 is satisfied, then we know p^{TL} has to be set to the bottommost match located in the first t rows and column ℓ. Therefore two additional statements can be inserted between lines 86 and 87 such that u is set to $BottomPos[y_\ell, t]$ and v is set to ℓ. Similar statements apply for the situation in lines 92–98, and this completes the description of the management for p^{TL}. p^{BR} can be handled in a similar way.

c^T, c^R, and ℓ^{TR} must be updated whenever the length of $C^{TR,i}$ increases. These situations are indicated in lines 40, 46, 69, and 95 in Fig. 6, and here we only sketch how to manage them. By arguments analogous to the ones given in the proof of Lemma 3.4, we have to distinguish two cases when updating c^T. If $s^{TL,i} > e^{T,i}$, then c^T is set to p^{TL}, otherwise c^T can be determined by some additional statements which are similar to the ones used for updating p^{TL}. In either case, we set ℓ^{TR} to $e^{T,i}$ because $e^{T,i}$ is the position of c^T in $C^{TR,i}$, as seen in the proof of Lemma 3.5. The management of c^B, c^L, and ℓ^{BL} is similar.

Now let us review the construction of the final decomposition given in the end of Sect. 3. If p is set to $e^{T,\lceil m/2\rceil} + e^{R,\lceil m/2\rceil}$, then we can use c^T and c^R as the appropriate matches for c^{TL} and c^{BR}. Similarly, if $p = e^{B,\lceil m/2\rceil} + e^{L,\lceil m/2\rceil}$, we establish $c^{TL} = c^L$ and $c^{BR} = c^B$. Finally, if a longest chain is determined by the algorithm described in case d of the construction (corresponding to lines 103–112 in Fig. 6), and p is not set to one of the above values, then we can use the backup values $\tilde{u}$ and $\tilde{v}$ to determine $c^{TL} := (\mathit{BottomPos}[y_{\hat{u}}, b], y_{\hat{u}})$ and $c^{BR} := (\mathit{TopPos}[y_{\hat{v}}, t], y_{\hat{v}})$, where $\hat{u} := \mathit{ThreshT}[\tilde{u}]$ and $\hat{v} := \mathit{ThreshB}[\tilde{v}]$.

Before recursively calling the algorithm for the remaining parts of the LCS, we see it is necessary for our routine to not only work on the complete matrix of size $[1:m] \times [1:n]$, but also on any subarea $[k_1 : k_2] \times [\ell_1 : \ell_2]$. The necessary changes are quite straightforward, and we do not provide any details here. Moreover, it might be impossible to locate both c^{TL} and c^{BR} (e.g., when $|M| = 1$), but then one recursive call can simply be skipped.

Theorem 5.1 *An LCS can be constructed in $O(ns + \min\{mp, m\log m + p(n-p)\})$ time and $O(ns)$ space.*

Proof. Clearly, for the top–level call, the additional overhead needed to keep track of the new variables is bounded by $O(m)$. Thus, not taking into account the time consumed by preprocessing or any recursive calls, we can assume the number of elementary operations to be bounded by $d(m+\min\{mp, p(n-p)\})$, for some appropriate constant d. We first examine the bound $d(m+mp)$. Let $c^{TL} = (k, \ell)$ and $c^{BR} = (k', \ell')$ (if only one match has been determined, the analysis is similar). Consider the two first–level recursive calls concerning the areas $M_1 := [1 : k-1] \times [1 : \ell - 1]$ and $M_2 := [k'+1 : m] \times [\ell'+1 : n]$. Let p_1 and p_2 denote the length of an LCS in M_1 and M_2, respectively, i.e., $p_1 + p_2 = p - 2$. Recall that c^{TL} is located in the first $\lceil m/2 \rceil$ rows and columns, i.e., the length of one side of M_1 is bounded by $\lceil m/2 \rceil - 1$. The same is true for M_2, and thus the number of operations taken for both first–level calls is bounded by

$$d(\lceil m/2 \rceil - 1)(p_1 + 1) + d(\lceil m/2 \rceil - 1)(p_2 + 1) \le dp\frac{m}{2}$$

Repeating this argument, we obtain a $dmp/2^i$ bound for the at most 2^i ith–level recursive calls. Since recursion ends at level $\lceil \log(m/2) \rceil$, this sums up to at most $2 \cdot dmp$ for the complete algorithm.

For the other bound $d(m + p(n-p))$, let $g := (\sqrt{5} - 1)/2 \approx 0.618$ and consider the following two cases.

Case a: $p \le gm$. Then

$$2 \cdot dmp \le \frac{2}{1-g} d(1-g)mp = \frac{2}{1-g} d(m - gm)p \le \frac{2}{1-g} d(m-p)p \le \frac{2}{1-g} d(n-p)p$$

Case b: $p > gm$. Let $h := \max\{k-1, \ell-1\}$ and $h' := \max\{m-k', n-\ell'\}$. Clearly $h + h' \le n - 2$. Also note that $p_1, p_2 \le \lceil m/2 \rceil - 1$ because an LCS cannot exceed the length of any side of M_1 and M_2. But then the two first–level recursive calls use at most

$$\begin{aligned} & d(\lceil m/2 \rceil - 1 + p_1(h - p_1)) + d(\lceil m/2 \rceil - 1 + p_2(h' - p_2)) \\ & \le d(m + p_1(h-p_1) + p_2(h'-p_2)) \le d(m + (\lceil m/2 \rceil - 1)(h - p_1 + h' - p_2)) \\ & \le d(m + (\lceil m/2 \rceil - 1)(n-p)) \le d(m + \tfrac{1}{2g}p(n-p)) \end{aligned}$$

operations. Similarly, all ith–level recursive calls together use at most

$$d(m + p(n-p)/(2g)^i)$$

operations. This sums up to

$$d(m \log m + \frac{1}{1-1/(2g)} p(n-p)) = d(m \log m + \frac{2}{1-g} p(n-p)) \ .$$

Both cases imply that the algorithm takes at most $O(ns + \min\{mp, m \log m + p(n-p)\})$ time, and the worst case overhead factor can be expected to be $2/(1-g) < 5.25$. Furthermore, when comparing the divide–and–conquer routine with the algorithm which determines the length p of an LCS, we only need $O(\log m)$ additional stack space, and thus the $O(ns)$ space bound is still valid. $\square$

6 Experimental Results

We compared our routine with the algorithm proposed by Rick [Ric94, Ric95] which clearly outperforms any other method when constructing longest common subsequences of intermediate lengths. Rick's algorithm is also a flexible one, being very efficient for short and long LCS as well. It uses a strategy similar to the one presented here, but only constructs antichains (or *contours*) from the top and left side of M. While this substantially simplifies the implementation and also the preprocessing phase (i.e., we only have to compute *LeftPos* and *TopPos*), there are two severe drawbacks. First, in order to recover an LCS after determining its length, the so–called *dominant matches* must be saved during the construction of the contours, and this might take $\Omega(mn)$ space. Second, the number of checks of *Thresh*–values is significantly increased when decomposing M from only two sides. For an alphabet of size 8, Table 1 shows some sample results when determining p for different settings of m, n, and p.

Table 1: Frequency of checks of *Thresh*–values

m	n	p	Rick [Ric95]	New method
500	500	100	16864	14983
500	500	200	28962	23078
500	500	300	33276	23394
500	500	400	20384	13276

m	n	p	Rick [Ric95]	New method
1500	1500	300	145129	126796
1500	1500	600	265107	216845
1500	1500	900	280026	207000
1500	1500	1200	172846	121516

The corresponding running times are presented in Table 2. Both algorithms were programmed in a straightforward way, using no special optimizations, and were tested on an Intel Pentium II at 300 MHz. It can be seen that our algorithm only takes about 70% of the time needed by Rick's method when computing the length of an LCS which is of intermediate length. For very short or very long LCS our method slightly suffers from the additional overhead during the preprocessing phase, but is still very efficient.

Finally, we checked the running times and the consumed space when generating an LCS. Table 3 shows that in spite of the linear space restriction, our algorithm

Table 2: Running times in microseconds for determining the length p of an LCS.

m	n	p	Rick [Ric95]	New method
500	500	100	3352	3626
500	500	200	5659	4725
500	500	300	6978	4890
500	500	400	5000	3516

m	n	p	Rick [Ric95]	New method
1500	1500	300	24451	21868
1500	1500	600	46099	34835
1500	1500	900	54176	33791
1500	1500	1200	38791	22308

sometimes runs more than twice as fast as Rick's method. This is due to the significant overhead in Rick's routine which is caused by the additional statements responsible for saving the contours in memory. Furthermore, the worst case factor 5.25 calculated in the proof of Thm. 5.1 is much too pessimistic in practical situations. Instead, a comparison with Table 2 shows that it roughly equals 2.

Table 3: Running times in microseconds for constructing an LCS of length p.

m	n	p	Rick [Ric95]	New method
500	500	100	6319	6044
500	500	200	14341	9066
500	500	300	19505	9890
500	500	400	15769	7802

m	n	p	Rick [Ric95]	New method
750	750	250	23132	16374
750	750	400	39835	20495
750	750	550	38516	16758
750	750	700	16319	9945

Table 4: Allocated space in bytes for constructing an LCS of length p.

m	n	p	Rick [Ric95]	New method
500	500	100	64284	34072
500	500	200	143820	34072
500	500	300	199464	34072
500	500	400	176328	34072

m	n	p	Rick [Ric95]	New method
750	750	250	219244	51072
750	750	400	390172	51072
750	750	550	396136	51072
750	750	700	193780	51072

Conclusions

We have investigated a new algorithm for the Longest Common Subsequence Problem. In spite of the quite complicated technical details necessary for the construction and analysis, the final routines proved to be extremely practical. More precisely, we have shown three results. First, we have presented a new fast method for determining the length of an LCS. Second, we have developed a linear space algorithm for constructing an LCS in $O(ns + \min\{mp, m \log m + p(n-p)\})$ time, thus solving a previously open problem. And third, we have shown by some experimental results that this algorithm is by far the fastest one when dealing with usual applications.

Acknowledgement. We would like to thank Dr. F. Kurth for helpful comments.

References

[AHU76] Aho, A.V., Hirschberg, D.S., Ullman, J.D.: Bounds on the complexity of the longest common subsequence problem. J. ACM **23**(1), 1976, 1–12.

[Apo86] Apostolico, A.: Improving the worst–case performance of the Hunt–Szymanski strategy for the longest common subsequence of two strings. Inform. Process. Lett. **23**, 1986, 63–69.

[Apo87] Apostolico, A.: Remarks on the Hsu–Du new algorithm for the longest common subsequence problem. Inform. Process. Lett. **25**, 1987, 235–236.

[AG87] Apostolico, A., Guerra, C.: The longest common subsequence problem revisited. Algorithmica **2**, 1987, 315–336.

[ABG92] Apostolico, A., Browne, S., Guerra, C.: Fast linear–space computations of longest common subsequences. Theoret. Comput. Sci. **92**, 1992, 3–17.

[CP94] Chin, F.Y.L., Poon, C.K.: A fast algorithm for computing longest common subsequences of small alphabet size. J. Inform. Process. **13**(4), 1990, 463–469.

[CS75] Chvátal, V., Sankoff, D.: Longest common subsequences of two random strings. J. Appl. Prob. **12**, 1975, 306–315.

[DP94] Dančík, V., Paterson, M.: Upper bounds for the expected length of a longest common subsequence of two binary sequences. Proceedings, 11th Annual Symp. on Theoretical Aspects of Computer Science, LNCS **775**, 1994, 669–678.

[Day65] Dayhoff, M.O.: Computer aids to protein sequence determination. J. Theoret. Biol. **8**, 1965, 97–112.

[Day69] Dayhoff, M.O.: Computer analysis of protein evolution. Sci. Amer. **221**(1), 1969, 86–95.

[Dek79] Deken, J.G.: Some limit results for longest common subsequences. Discr. Math. **26**, 1979, 17–31.

[Dil50] Dilworth, R.P.: A decomposition theorem for partially ordered sets. Annals Math. **51**, 1950, 161–166.

[FB73] Fu, K.S., Bhargava, B.K.: Tree systems for syntactic pattern recognition. IEEE Trans. Comput. **C–22**(12), 1973, 1087–1099.

[GMS80] Gallant, J., Maier, D., Storer, J.A.: On finding minimal length superstrings. J. Comput. System Sci. **20**, 1980, 50–58.

[Hir75] Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Comm. ACM **18**(6), 1975, 341–343.

[Hir77] Hirschberg, D.S.: Algorithms for the longest common subsequence problem. J. ACM **24**(4), 1977, 664–675.

[HD84] Hsu, W.J., Du, M.W.: New algorithms for the LCS problem. J. Comput. System Sci. **29**, 1984, 133–152.

[HS77] Hunt, J.W., Szymanski, T.G.: A fast algorithm for computing longest common subsequences. Comm. ACM **20**(5), 1977, 350–353.

[KR87] Kumar, S.K., Rangan, C.P.: A linear space algorithm for the LCS problem. Acta Inform. **24**, 1987, 353–363.

[LW75] Lowrance, R., Wagner, R.A.: An extension of the string–to–string correction problem. J. ACM **22**(2), 1975, 177–183.

[LF78] Lu, S.Y., Fu, K.S.: A sentence–to–sentence clustering procedure for pattern analysis. IEEE Trans. Syst. Man. Cybernet. **SMC–8**(5), 1978, 381–389.

[Mai78] Maier, D.: The complexity of some problem on subsequences and supersequences. J. ACM **25**(2), 1978, 322–336.

[MP80] Masek, W.J., Paterson, M.S.: A faster algorithm for computing string edit distances. J. Comput. System Sci. **20**(1) 1980, 18–31.

[Mye86] Myers, E.W.: An $O(ND)$ difference algorithm and its variations. Algorithmica **1** 1986, 251–266.

[NKY82] Nakatsu, N., Kambayashi, Y., Yajima, S.: A longest common subsequence algorithm suitable for similar text strings. Acta Inform. **18**, 1982, 171–179.

[NW70] Needleman, S.B., Wunsch, C.S.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Molecular Biol. **48**, 1970, 443–453.

[PD94] Paterson, M., Dančík, V.: Longest common subsequences. Proceedings, 19th Intern. Symp. on Mathematical Foundations of Computer Science, LNCS **841**, 1994, 127–142.

[Ric94] Rick, C.: New algorithms for the longest common subsequence problem. Research report no. 85123–CS, Department of Computer Science, University of Bonn, Germany, 1994.

[Ric95] Rick, C.: A new flexible algorithm for the longest common subsequence problem. Proceedings, 6th Annual Symp. on Combinatorial Pattern Matching, LNCS **937**, 1995, 340–351.

[SC73] Sankoff, D., Cedergren, R.J.: A test for nucleotide sequence homology. J. Molecular Biol. **77**, 1973, 159–164.

[SK83] Sankoff, D., Kruskal, J.B.: Time Warps, String Edits, and Macromolecules: The Theory And Practice of Sequence Comparison. Addison–Wesley, Reading, MA, 1983.

[Ukk85] Ukkonen, E.: Algorithms for approximate string matching. Inform. and Control **64**, 1985, 100–118.

[Wag75] Wagner, R.A.: On the complexity of the extended string–to–string correction problem. Proceedings, 7th Ann. ACM Sympos. on Theory of Comput. 1975, 218–223.

[WF74] Wagner, R.A., Fischer, M.J.: The string–to–string correction problem. J. ACM **21**(1), 1974, 168–173.

[WC76] Wong, C.K., Chandra, A.K.: Bounds for the string editing problem. J. ACM **28**(1), 1976, 13–18.

[WMM90] Wu, S., Manber, U., Myers, G., Miller, W.: An $O(NP)$ sequence comparison algorithm. Inform. Process. Lett. **35**, 1990, 317–323.