Extending the Four Russian Algorithm to Compute the Edit Script in Linear Space

Vamsi Kundeti and Sanguthevar Rajasekaran

Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269, USA
{vamsik,rajasek}@engr.uconn.edu

Abstract. Computing the *edit distance* between two strings is one of the most fundamental problems in computer science. The standard dynamic programming based algorithm computes the *edit distance* and *edit script* in $O(n^2)$ time and space. Often the *edit script* is of more importance than the value of the *edit distance*. The Four Russian Algorithm [1] computes the *edit distance* in $O(n^2/\log n)$ time but does not address how to compute *edit script* within that runtime. Hirschberg [2] gave an algorithm to compute *edit script* in linear space but the runtime remained $O(n^2)$. In this paper we present algorithms that compute both the *edit script* and *edit distance* in $O(\frac{n^2}{\log n})$ time using $O(n)$ space.

Keywords: edit distance, edit script, linear space, four russian algorithm, hirschberg's algorithm.

1 Introduction

The *edit distance* between strings $S_1 = [a_1, a_2, a_3 \dots a_n]$ and $S_2 = [b_1, b_2, b_3 \dots b_n]$ is defined as the minimal cost of transforming S_1 into S_2 using the three operations *Insert*, *Delete*, and *Change(C)* (see e.g., [3]). The first application(global alignment) of the *edit distance* algorithm for protein sequences was studied by Needleman [4]. Later algorithms for several variations (such as local alignment, affine gap costs, etc.) of the problem were developed (for example) in [5], [6], and [7]. The first major improvement in the asymptotic runtime for computing the value of the edit distance was achieved in [1]. This algorithm is widely known as the Four Russian Algorithm and it improves the running time by a factor of $O(\log n)$ (with a run time of $O(n^2/\log n)$) to compute just the value of the edit distance. It does not address the problem of computing the actual *edit script*, which is of wider interest rather than just the value. Hirschberg [2] has given an algorithm that computes the actual script in $O(n^2)$ time and $O(n)$ space. The space saving idea from [2] was applied to biological problems in [8] and [9]. However the asymptotic complexity of the core algorithm in each of these remained $O(n^2)$. Also, parallel algorithms for the *edit distance* problem and its application to sequence alignment of biological sequences were studied extensively (for example) in [10] and [11]. In paper [12] linear space parallel algorithms for the sequence alignment problem were given, however they assume that $O(n^2)$ is the optimal asymptotic complexity of the sequential algorithm. Please refer to [13] for an excellent survey on all these algorithms. A special case is one where each of these operations is of unit cost. *Edit Script* is the actual sequence of operations that converts S_1 into S_2. In particular, the edit script is a sequence $E_{script} = \{X_1, X_2, X_3 \ldots X_n\}, X_i \in I, D, C$. Standard dynamic programming based algorithms solve both the distance version and the script version in $O(n^2)$ time and $O(n^2)$ space. The main result of this paper is an algorithm for computing the edit distance and edit script in $O\left(\frac{n^2}{\log n}\right)$ time and $O(n)$ space.

The rest of the paper is organized as follows. In Sec. 2 we provide a summary of the four Russian algorithm [1]. In Sec. 3 we discuss the $O(n^2)$ time algorithm that consumes $O(n)$ space and finally in Sec. 4 we show how to compute the edit distance and script using $O(\frac{n^2}{\log n})$ time and $O(n)$ space.

2 Four Russian Algorithm

In this section we summarize the Four Russian Algorithm. Let D be the dynamic programming table that is filled during the edit distance algorithm. The standard edit distance algorithm fills this table D row by row after initialization of the first row and the first column. Without loss of generality, throughout this paper we assume that all the edit operations cost unit time each.

The basic idea behind the Four Russian Algorithm is to partition the dynamic programming table D into small blocks each of width and height equal to t where t is a parameter to be fixed in the analysis. Each such block is called a *t-block*. The dynamic programming table is divided into *t-blocks* such that any two adjacent *t-blocks* overlap by either a row or column of width (or height) equal to t. See Fig. 1 for more details on how the dynamic programming table D is partitioned. After this partitioning is done The Four Russian algorithm fills up the table D block by block. Algorithm 1 has more details.

A quick qualitative analysis of the algorithm is as follows. After the partitioning of the dynamic programming table D into *t-blocks* we have $\frac{n^2}{t^2}$ blocks and if processing of each of the block takes $O(t)$ time then the running time is $O(\frac{n^2}{t})$. In the case of standard dynamic programming, entries are filled one at a time (rather than one block at a time). Each entry can be filled in $O(1)$ time and hence the total run time is $O(n^2)$. In the Four Russian algorithm, there are $\frac{n^2}{t^2}$ blocks. In order to be able to fill each block in $O(t)$ time, some preprocessing is done. Theorem 1 is the basis of the preprocessing.

Theorem 1. *If D is the edit distance table then $|D[i,j] - D[i+1,j]| \leq 1$, and $|D[i,j] - D[i,j+1]| \leq 1 \forall (0 \leq i, j \leq n)$.*

Proof. Note that $D[i,j]$ is defined as the minimum cost of converting $S_1[1:i]$ into $S_2[1:j]$. Every element of the table $D[i,j]$ is filled based on the values from

$D[i-1,j-1]$,$D[i-1,j]$ or $D[i,j-1]$. $D[i,j] \geq D[i-1,j-1]$(characters at $S_1[i]$ and $S_2[j]$ may be same or different), $D[i,j] \leq D[i,j-1]+1$ (cost of insert is unity),$D[i,j-1] \leq D[i-1,j-1]+1$(same inequality as the previous one rewritten for element $D[i,j-1]$). The following inequalities can be derived from the previous inequalities.

$$\begin{aligned}
-D[i,j] &\leq -D[i-1,j-1] \\
D[i,j-1] &\leq D[i-1,j-1]+1 \\
-D[i,j]+D[i,j-1] &\leq 1 \\
D[i,j-1]-D[i,j] &\leq 1 \\
D[i,j] &\leq D[i,j-1]+1 \text{ \{Started with this\}} \\
-1 &\geq D[i,j-1]-D[i,j] \\
|D[i,j-1]-D[i,j]| &\leq 1
\end{aligned}$$

Along the same lines we can also prove that $|D[i-1,j]-D[i,j]| \leq 1$ and $D[i-1,j-1] \leq D[i,j]$.

Theorem 1 essentially states that the value of the edit distance in the dynamic programming table D will either increase by 1 or decrease by 1 or remain the same compared to the previous element in any row or a column of D. Theorem 1 helps us in encoding any row or column of D with a vector of $0, 1, -$. For example a row in the edit distance table $D[i,*] = [k, k+1, k, k, k-1, k-2, k-1]$ can be encoded with a vector $v_i = [0, 1, -1, 0, -1, -1, 1]$. To characterize any row or column we just need the vector v_i and k corresponding to that particular row or column. For example, if $D[i,*] = [1, 2, 3, 4, \ldots, n]$, then $k = 1$ for this row and $v_i = [0, 1, 1, 1, 1, 1, 1, \ldots, 1]$. For the computation of the edit distance table D the leftmost column and the topmost row must be filled (or initialized) before the start of the algorithm. Similarly in this algorithm we need the topmost row (A) and leftmost column (B) to compute the edit distance within the t-block see Fig. 1. Also see Algorithm 2. It is essential that we compute the edit distance within any t-block in constant time.

In the Four Russian algorithm the computation of each t-block depends on the variables A, B, K, C, E (see Fig. 1). The variable A represents the top row of the t-block and B represents the the left column of the t-block. C and E represent the corresponding substrings in the strings S_1 and S_2. K is the intersection of A and B. If the value of the variable K is k then from Theorem 1 we can represent A and B as vectors of {0,1,-1} rather than with exact values along the row and column.

As an example, consider the first t-block which is the intersection of the first t rows and the first t columns of D. For this t-block the variables $\{A, B, K, C, E\}$ have the following values: $K = D[0,0]$, $A = D[0,*] = [0, 1, 1, 1, \ldots, 1]$, $B = D[*,0] = [0, 1, 1, 1, \ldots, 1]$, $C = S_2[0, 1, \ldots, t]$, and $E = S_1[0, 1, \ldots, t]$. For any t-block we have to compute $\{A', B', K'\}$ as a function of $\{A, K, B, C, E\}$ in $O(1)$ time. In this example plugging in $\{A, B, K, C, E\}$ for the first t-block gives $K' = D[t,t]$, $A' = [D[0,t], \ldots, D[t,t]]$,$B' = [D[t,0], \ldots, D[t,t]]$. To accomplish the task of computing the edit distance in a t-block in $O(1)$ time, we precompute

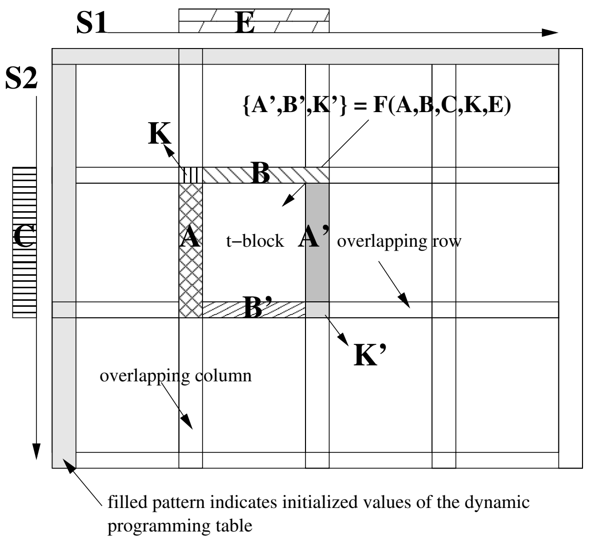
S1
E
S2
{A',B',K'} = F(A,B,C,K,E)
K
B
C
A
t−block
A'
overlapping row
B'
K'
overlapping column
filled pattern indicates initialized values of the dynamic programming table

Fig. 1. Using preprocessed lookup table $\{A', B', K'\} = F(A, B, C, K, E)$

all the possible inputs in terms of variables $\{A, B, 0, C, E\}$. We don't have to consider all possible values of K since if K'_1 is the value of K' we get with input variables $\{A, B, 0, C, E\}$ then the value of K' for inputs $\{A, B, K, C, E\}$ would be $K'_1 + K$. Thus this encoding(and some preprocessing) helps us in the computation of the edit distance of the t-block in $O(1)$ time. The algorithm is divided into two parts *pre-processing step* and *actual computation.*

Algorithm 1. Four Russian Algorithm, t is a parameter to be fixed.

```
INPUT   : Strings $S_1$ and $S_2$, $\Sigma$, $t$
OUTPUT: Optimal Edit distance
/*Pre-processing step*/
$F$ = PreProcess($\Sigma$, $t$) ;
for $i = 0; i < n; i+ = t$ do
    for $j = 0; j < n; j+ = t$ do
        $\{A', B', D'\} = LookUpF(i, j, t)$ ;
        $[D[i + t, j] \ldots D[i + t, j + t] = A'$ ;
        $[D[i, j + t] \ldots D[i + t, j + t] = B'$ ;
    end
end
```

2.1 Pre Processing Step

As we can see from the previous description, at any stage of the Algorithm 1 we need to do a lookup for the edit distance of any t-block and as a result get the row and column for the adjacent t-blocks. From Theorem 1 its evident

Algorithm 2. LookUp routine used by Algorithm 1.

```
INPUT    : $i$,$j$,$t$
OUTPUT: $A', B', D'$
$A = [D[i,j] \ldots D[i,j+t]]$;
$B = [D[i,j] \ldots D[i+t,j]]$;
$C = [S_2[j] \ldots S_2[j+t]]$;
$E = [S_1[j] \ldots S_1[j+t]]$;
$K = D[i,j]$;
/*Encode A,B*/
for $k = 1$;$k < t$;$k++$ do
    $A[k] = A[k] - A[k-1]$;
    $B[k] = B[k] - B[k-1]$;
end
/*Although $K$ is not used in building lookup table $F$ we maintain the
consistency with Fig. 1 */
return $\{A', B', D'\} = F(A, B, C, K, E)$ ;
```

that any input $\{A, B, K, C, E\}$ (see Fig. 1) to the t-block can be transformed into vectors of $\{-1, 0, 1\}$. In the preprocessing stage we try out all possible inputs to the t-block and compute the corresponding output row and column ($\{A', B', K'\}$ (see Fig. 1). More formally, the row (A') and column(B') that need to be for any t-block can be repesented as a function F (lookup table) with inputs $\{A, B, K, C, E\}$, such that $\{A', B', K'\} = F(A, B, K, C, E)$. This function can be precomputed since we have only limited possibilities. For any given t, we can have 3^t vectors corresponding to A and B.

For a given alphabet of size Σ we have Σ^t possible inputs corresponding to C and E. K will not have any effect since we just have to add K to $A'[t]$ or $B'[t]$ at the end to compute K'. The time to preprocess is thus $O((3\Sigma)^{2t}t^2)$ and the space for the lookup table F would be $O((3\Sigma)^{2t}t)$. Since $t^2 \leq (3\Sigma)^t$, if we pick $t = \frac{\log n}{3\log(3\Sigma)}$, the preprocessing time as well as the space for the lookup table will be $O(n)$. Here we make use of the fact that the word length of the computer is $\Theta(\log n)$. This in particular means that a vector of length t can be thought of as one word.

2.2 Computation Step

Once the preprocessing is completed in $O(n)$ time, the main computation step proceedes scanning the t-blocks row by row and filling up the dynamic programming table(D). Algorithm 1 calls Algorithm 2 in the inner most *for* loop. Algorithm 2 takes $O(t)$ time to endcode the actual values in D and calls the function F which takes $O(1)$ time and returns the row (A') and column (B') which are used as input for other t-blocks. The runtime of the entire algorithm is $O(\frac{n}{t}\frac{n}{t}t) = O(\frac{n^2}{t})$. Since $t = \Theta(\log n)$ the run time of the Four Russian Algorithm is $O(\frac{n^2}{\log n})$.

3 Hirschberg's Algorithm to Compute the Edit Script

In this section we briefly describe Hirschberg's [2] algorithm that computes the edit script in $O(n^2)$ time using $O(n)$ space. The key idea behind this algorithm is an appropriate formulation of the dynamic programming paradigm. We make some definitions before giving details on the algorithm.

- Let S_1 and S_2 be strings with $|S_1| = m$ and $|S_2| = n$. A substring from index i to j in a string S is denoted as $S[i \dots j]$.
- If S is a string then S^r denotes the reverse of the string.
- Let $D(i, j)$ stand for the optimal edit distance between $S_1[1 \dots i]$ and $S_2[1 \dots j]$.
- Let $D^r(i, j)$ be the optimal edit distance between $S_1^r[1 \dots i]$ and $S_2^r[1 \dots j]$.

Lemma 1. *$D(m, n) = min_{0 \le k \le m}\{D[\frac{n}{2}, k] + D^r[\frac{n}{2}, m-k]\}$.*

The Lemma 1 essentially says that finding the optimal value of the edit distance between strings S_1 and S_2 can be done as follows: Split S_1 into two parts (p_{11} and p_{12}) and S_2 into two parts (p_{21} and p_{22}); Find the edit distance (e_1) between p_{11} and p_{21}; Find the edit distance (e_2) between p_{12} and p_{22}; Finally add both the distances to get the final edit distance ($e_1 + e_2$); Since we are looking for the minimum edit distance we have to find a breaking point (k) that minimizes the value of ($e_1 + e_2$).

We would not miss this minimum even if we break one of the strings deterministically and find the corresponding breaking point in the other string. As a result of this we keep the place where we break in one of the strings fixed. (Say we always break one of the strings in the middle). Then we find a breaking point in the other string that will give us minimum value of ($e_1 + e_2$).

The k in Lemma 1 can be found in $O(mn)$ time and $O(m)$ space for the following reasons. To find the k at any stage we need two rows($D[\frac{n}{2}, *]$ and $D^r[\frac{n}{2}, *]$) from forward and reverse dynamic programming tables. Since the values in any row of the dynamic programming table just depend on the previous row, we just have to keep track of the previous row while computing the table D and D^r. Once we find k we can also determine the path from the previous row ($\frac{n}{2} - 1$) to row ($\frac{n}{2}$) in both the dynamic programming tables D and D^r (see Fig. 2). Once we find these subpaths we can continue to do the same for the two subproblems (see Fig. 2) and continue recursively. The run time of the algorithm can be computed by the following reccurence relation.

$$T(n, m) = T(\tfrac{n}{2}, k) + T(\tfrac{n}{2}, m-k) + mn$$
$$T(\tfrac{n}{2}, k) + T(\tfrac{n}{2}, m-k) = \tfrac{mn}{2} + \tfrac{mn}{4} + \dots = O(mn)$$

In each stage we use only $O(m)$ space and hence the space complexity is linear.

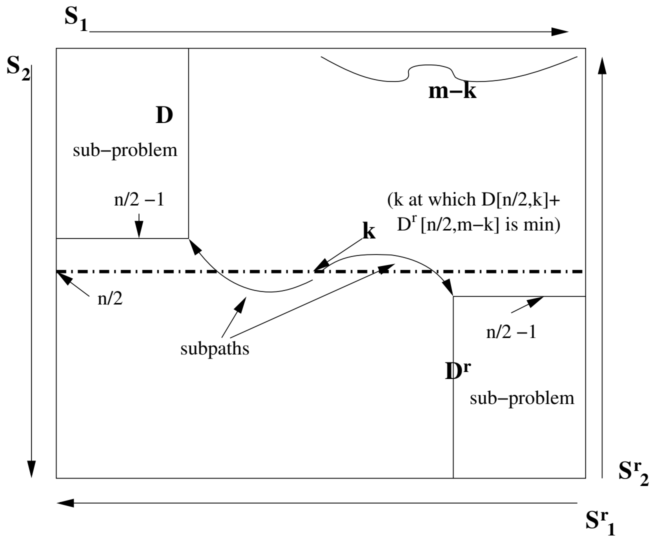

Fig. 2. Illustration of Hirschberg's recursive algorithm

4 Our Algorithm

Our algorithm combines the frameworks of the Four Russian algorithm and that of Hirschberg's Algorithm. Our algorithms finds the edit script in $O\left(\frac{n^2}{\log n}\right)$ time using linear space. We extend the Four Russian algorithm to accommodate Lemma 1 and to compute the *edit script* in $O(n)$ space.

At the top-level of our algorithm we use a dynamic programming formulation similar to that of Hirschberg. Our algorithm is recursive and in each stage of the algorithm we compute k and also find the sub-path as follows.

$$D(m,n) = min_{0\leq k\leq m}\{D(\frac{n}{2},k) + D^r(\frac{n}{2},m-k)\}$$

The key question here is how to use the Four Russian framework in the computation of $D(\frac{n}{2},k)$ and $D^r(\frac{n}{2},m-k)$ for any k in time better than $O(n^2)$? . Hirschberg's algorithm needs the rows $D(\frac{n}{2},*)$ and $D^r(\frac{n}{2},*)$ at any stage of the recursion. In Hirschberg's algorithm at recursive stage $(R(m,n))$, $D(\frac{n}{2},k)$ and $D^r(\frac{n}{2},m-k)$ are computed in $O(mn)$ time. We cannot use the same approach since the run time will be $\Omega(n^2)$. We have to find a way to compute the rows $D(\frac{n}{n},*)$ and $D^r(\frac{n}{2},*)$ with a run time of $O(\frac{n^2}{\log n})$.

The top-level outline of our algorithm is illustrated by the pseudo-code in TopLevel (see Algorithm 3). The algorithm starts with input strings S_1 and S_2 of length m and n, respectively. At this level the algorithm applies Lemma 1 and finds k. Since the algorithm requires $D(\frac{n}{2},*)$ and $D^r(\frac{n}{2},*)$ at this level it calls the algorithm FourCompute to compute the rows $D(\frac{n}{2},*)$, $D(\frac{n}{2}-1,*)$, $D^r(\frac{n}{2},*)$ and

$D^r(\frac{n}{2}-1,*)$. Note the fact that although for finding k we require rows $D(\frac{n}{2},*)$ and $D^r(\frac{n}{2},*)$, to compute the actual *edit script* we require rows $D(\frac{n}{2}-1,*)$ and $D^r(\frac{n}{2}-1,*)$. Also note that these are passed to algorithm FindEditScript to report the *edit script* around index k.

Once the algorithm finds the appropriate k for which the edit distance would be minimum at this stage, it divides the problem into two sub problems (see Fig. 2) $(S_1[1\dots k_1-1], S_2[1\dots \frac{n}{2}-1])$ and $(S_1[m-k_2+1\dots m], S_2[\frac{n}{2}+1\dots n]$. Observe that k_1 and k_2 are returned by FindEditScript. FindEditScript is trying to find if the sub-path passes through the row $\frac{n}{2}$ (at the corresponding level of recursion) and updates k so that we can create sub-problems (please see arcs (sub-paths) in Fig. 2). Once the sub-problems are properly updated the algorithm solves each of these problems recursively.

We now describe algorithm FourCompute which finds the rows $D(\frac{n}{2},*)$ and $D^r(\frac{n}{2},*)$ (that are required at each recursive stage of TopLevel (Algorithm 3)) in time $O(\frac{nm}{t})$ where t is the size of blocks used in the Four Russian Algorithm. We do exactly the same pre-processing done by the Four Russian Algorithm and create the lookup table F. FourCompute is called for both forward (S_1,S_2) and reverse strings (S_1^r,S_2^r). The lookup table $F(A,B,K,C,E)$ has been created for all the strings from Σ of length t. We can use the same lookup table F for all the calls to FourCompute. A very important fact to remember is that in the Four Russian algorithm whenever a lookup call is made to F the outputs $\{A', B'\}$ are always aligned at the rows which are multiples of t, i.e., at any stage of the Four Russian algorithm we only require the values of the rows $D(i,*)$ such that $i \bmod t = 0$. In our case we cannot directly use the Four Russian Algorithm in algorithm FourCompute because the lengths of the strings which are passed to FourCompute from each recursive level of TopLevel is not necessarily a multiple of t. Suppose that in some stage of the FourCompute algorithm a row i is not a multiple of t. We apply the Four Russian Algorithm and compute till row $D(\lfloor\frac{i}{t}\rfloor,*)$, find the values in the row $D(\lfloor\frac{i}{t}\rfloor-t,*)$ and apply lookups for rows $\lfloor\frac{i}{t}\rfloor-t, \lfloor\frac{i}{t}\rfloor-t+1,\dots,$ and $\lfloor\frac{i}{t}\rfloor-t+i \bmod t$. Basically we need to slide the t-block from the row $\lfloor\frac{i}{t}\rfloor-t$ to $\lfloor\frac{i}{t}\rfloor-t+i \bmod t$.

Thus we can compute any row that is not a multiple of t in an extra $i \bmod t * \frac{m}{t}$ time (where m is the length of the string represented across the columns). We can also use the standard edit distance computation in rows $\lfloor\frac{i}{t}\rfloor, \lfloor\frac{i}{t}\rfloor+1,\dots$ $\lfloor\frac{i}{t}\rfloor+i \bmod t$ which also takes the same amount of extra time. Also consider the space used while we compute the required rows in the FourCompute algorithm. We used only $O(m+n)$ space to store arrays $D\prime[0,*]$ and $D\prime[*,0]$ and reused them. So the space complexity of algorithm FourCompute is linear. The run time is $O((\frac{n}{t})(\frac{m}{t})(t))$ to compute a row $D(n,*)$ or $D^r(n,*)$. We arrive at the following Lemma.

Lemma 2. *Algorithm* FourCompute *Computes rows $D^r(\frac{n}{2},*)$, $D(\frac{n}{2},*)$ required by Algorithm* TopLevel *at any stage in $O(\frac{mn}{t})$ time and $O(m+n)$ space.*

The run time of the complete algorithm is as follows. Here c is a constant.

$$T(n,m) = T(\tfrac{n}{2},k) + T(\tfrac{n}{2},m-k) + c\tfrac{mn}{2t}.$$
$$T(n,m) = c(\tfrac{mn}{2t} + \tfrac{mn}{4t} + \cdots) = O(\tfrac{mn}{t}).$$

Since $t = \Theta(\log n)$ the run time is $O(n^2/\log n)$.

Algorithm 3. TopLevel which calls FourCompute at each recursive level.

```
Input: Strings $S_1$,$S_2$,$|S_1| = m$,$|S_2| = n$
Output: Edit Distance and Edit Script
$D(\frac{n}{2},*) = \mathsf{FourCompute}(\frac{n}{2}, m, S_1, S_2, D(*,0), D(0,*))$;
$D^r(\frac{n}{2},*) = \mathsf{FourCompute}(\frac{n}{2}, m, S_1^r, S_2^r, D^r(*,0), D^r(0,*))$;
/*Find the $k$ which gives min Edit Distance at this level*/
$Minimum = (m+n)$ ;
for $i = 0$ to $n$ do
    if $(D(\frac{n}{2},i) + D^r(\frac{n}{2},m-i)) < Minimum$ then
        $k = i$ ;
        $Minimum = D(\frac{n}{2},i) + D^r(\frac{n}{2},m-i)$ ;
    end
end
/*Compute The EditScripts at this level */
$k_1 = \mathsf{FindEditScript}(D(\frac{n}{2},*), D(\frac{n}{2}-1,*), k, Forward)$ ;
$k_2 = \mathsf{FindEditScript}(D^r(\frac{n}{2},*), D^r(\frac{n}{2}-1,*), k, Backward)$ ;
/*Make a recursive call If necessary*/ ;
$\mathsf{TopLevel}(S_1[1 \ldots k_1 - 1], S_2[1 \ldots \frac{n}{2} - 1])$ ;
$\mathsf{TopLevel}(S_1[m - k_2 + 1 \ldots m], S_2[\frac{n}{2} + 1 \ldots n])$ ;
```

4.1 Space Complexity

The space complexity is the maximum space required at any stage of the algorithm. We have two major stages where we need to analyze the space complexity as follows. The first during the execution of the entire algorithm and the second during preprocessing and storing the lookup table.

4.2 Space during the Execution

The space for algorithm TopLevel is clearly linear since we need to store just 4 rows at any stage: Rows $D(\frac{n}{2},*)$, $D(\frac{n}{2}-1,*)$, $D^r(\frac{n}{2},*)$ and $D^r(\frac{n}{2}-1,*)$. From Lemma 2 the space required for FourCompute is also linear. So the space complexity of the algorithm during execution is linear.

4.3 Space for Storing Lookup Table *F*

We also need to consider the space for storing the lookup table F. The space required to store the lookup table F is also linear for an appropriate value of t (as has been shown in Sec. 2.1). The runtime of the algorithm is $O\left(\frac{n^2}{\log n}\right)$.

5 Conclusion

In this paper we have shown that we can compute both the edit distance and edit script in time $O(\frac{n^2}{\log n})$ using $O(n)$ space.

Acknowledgments. This research has been supported in part by the NSF Grant ITR-0326155 and a UTC endowment.

References

1. Arlazarov, V.L., Dinic, E.A., Kronrod, M.A., Faradzev, I.A.: On economic construction of the transitive closure of a directed graph. Dokl. Akad. Nauk SSSR 194, 487–488 (1970)
2. Hirschberg, D.S.: Linear space algorithm for computing maximal common subsequences. Communications of the ACM 18(6), 341–343 (1975)
3. Horowitz, E., Sahni, S., Rajasekaran, S.: Computer Algorithms. Silicon Press (2008)
4. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48(3), 443–453 (1970)
5. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
6. Gotoh, O.: Alignment of three biological sequences with an efficient traceback procedure. Journal of Theoretical Biology 121(3), 327–337 (1986)
7. Huang, X., Hardison, R.C., Miller, W.: A space-efficient algorithm for local similarities. Computer Applications in the Biosciences 6(4), 373–381 (1990)
8. Gotoh, O.: Pattern matching of biological sequences with limited storage. Computer Applications in the Biosciences 3(1), 17–20 (1987)
9. Myers, E.W., Miller, W.: Optimal alignments in linear space. Computer Applications in the Biosciences 4(1), 11–17 (1988)
10. Edmiston, E., Wagner, R.A.: Parallelization of the dynamic programming algorithm for comparison of sequences, pp. 78–80 (1987)
11. Ranka, S., Sahni, S.: String editing on an simd hypercube multicomputer. Journal of Parallel and Distributed Computing 9(4), 411–418 (1990)
12. Rajko, S., Aluru, S.: Space and time optimal parallel sequence alignments. IEEE Transactions on Parallel and Distributed Systems 15(12), 1070–1081 (2004)
13. Gusfield, D.: Algorithms of Strings Trees and Sequences. Cambridge (1997)