Comparing Sequences with Segment Rearrangements*

Funda Ergun[1], S. Muthukrishnan[2], and S. Cenk Sahinalp[3]

[1] Department of EECS, CWRU
afe@eecs.cwru.edu
[2] Rutgers Univ. and AT&T Research
muthu@cs.rutgers.edu
[3] Depts of EECS, Genetics and
Center for Comp. Genomics, CWRU.
cenk@cwru.edu

Abstract. Computational genomics involves comparing sequences based on "similarity" for detecting evolutionary and functional relationships. Until very recently, available portions of the human genome sequence (and that of other species) were fairly short and sparse. Most sequencing effort was focused on genes and other short units; similarity between such sequences was measured based on character level differences. However with the advent of whole genome sequencing technology there is emerging consensus that the measure of similarity between long genome sequences must capture the rearrangements of large segments found in abundance in the human genome.
In this paper, we abstract the general problem of computing sequence similarity in the presence of segment rearrangements. This problem is closely related to computing the smallest grammar for a string or the block edit distance between two strings. Our problem, like these other problems, is NP hard. Our main result here is a simple $O(1)$ factor approximation algorithm for this problem. In contrast, best known approximations for the related problems are factor $\Omega(\log n)$ off from the optimal. Our algorithm works in linear time, and in one pass. In proving our result, we relate sequence similarity measures based on different segment rearrangements, to each other, tight up to constant factors.

1 Introduction

Similarity comparison between biomolecular sequences play an important role in computational genomics due to the premise that sequence similarity usually indicates evolutionary and functional similarity. For example, popular computational tools for both multiple sequence alignment and evolutionary tree construction are based on iteratively measuring the similarity between pairs of available sequences.

* Funda Ergun's research supported in part by NSF grant CCR 0311548. Muthukrishnan's research supported in part by NSF EIA 0087022, NSF ITR 0220280 and NSF EIA 02-05116. Sahinalp's research supported in part by NSF Career Award, Charles B. Wang foundation and a DIMACS visit.

P.K. Pandya and J. Radhakrishnan (Eds.): FSTTCS 2003, LNCS 2914, pp. 183–194, 2003.
© Springer-Verlag Berlin Heidelberg 2003

With recent advancements in genome sequencing technologies we now have access to complete genome sequences of humans and many other species [37, 23]. As a result, the notion of similarity between genome sequences has changed dramatically in the last three years. In particular, due to the availability of short fragments only, previous notions of similarity between genome sequences focused on edit operations mimicking "point mutations", i.e., single character replacements, insertions and deletions only. Until recently precious little was known about large range/scale evolutionary mechanisms for genome modification. With a much improved understanding of the human genome sequence composition, genomicists are now studying genome-wide structural modifications. (See [12] for some of the recent topics of interest.) There is emerging consensus that in order to capture evolutionary mechanisms underlying genome evolution one must consider a richer set of sequence modifications than single character edits. This has lead to a new breed of sequence comparison methods that capture similarity based on not merely individual nucleotides (i.e. characters) but involving longer "segments" of the genome.

This paper formalizes the sequence similarity problem in the presence of segment rearrangements, based on insights from half a dozen biological studies over the past three years (e.g. [25,4,26,5]). This problem is related to the study of string grammars and certain "block" edit distances, but it is still quite distinct from these and other string comparison problems in the literature. Known methods for computing sequence similarity in the presence of segment rearrangements rely on greedy heuristics, but provide no provable guarantees. Our main result here is a simple and efficient greedy algorithm that provably approximates the measure of similarity defined, up to an $O(1)$ factor. In contrast, the best known approximation ratios on related sequence comparison problems is $\Omega(\log n)$. Our proof method connects different models for transforming a sequence into another via segment rearrangements and shows them to be equivalent modulo constant factor approximations. Furthermore, our algorithm works in one pass, with sublinear work space – which is desirable for handling massive sequences such as whole genomes – trading off space for accuracy. See [13] for background in genomics; here, we present our formal problem and our results.

1.1 Algorithmics: The Formal Problem

The focus of this paper is computing the distance between two strings in terms of edit operations that transform one string into the other. The edit operations of interest are

- *character edits*: character insertions, deletions and replacements, and
- *segment edits*: substring relocations, deletions, and duplications.

We define the *distance* from a given string R into another S, denoted $d(R \to S)$, as the *minimum* number of permitted edit operations to transform R into S. The edit operations may overlap arbitrarily. Note that this distance is not symmetric

since arbitrary deletions are allowed: simply consider the case where R is the empty string and S is of length at least 2.[1]

The *sequence comparison problem* is to compute (or estimate) $d(R \to S)$. If only character edits (inserts, deletes and changes) are allowed, then $d(R \to S) = d(S \to R)$ becomes the well known Levenshtein edit distance [22]. Levenshtein edit distance [22] or its weighted/normalized versions have been used for decades for measuring the functional and evolutionary similarity of genome sequences [31, 1]. Our focus is on the general sequence comparison problem in presence of both character as well as segment edits.

1.2 Previous Work

Previously, distances permitting segmental rearrangements have been investigated in a limited context. The bulk of the work is on distances that allow relocations or reversals only [2,3,15,16,8,9]. In this context each "character" represents a long segment of the genome, most commonly a whole *gene* and thus occurs only once in the sequence, effectively making the sequence a *permutation* of such characters. Thus these problems are reduced into special sorting problems.

Distances permitting a larger spectrum of edit operations have been investigated only recently: Examples include the transformation distance of Varre et. al. [38] and its closely related companion, the compression distance of Li et. al. [25,4,26]. These works received recent attention from the general scientific community [5] because of their demonstrated success in capturing the evolutionary relationships between world languages demonstrated by the comparison of "declaration of human rights" in various languages [4], DNA and RNA sequences, in particular, the known mitochondrial DNA sequences of various species [25] and families of retrotransposons [38], as well as the identification of authorship for works of literature [4]. These papers upper bound sequence "similarity" by a number of heuristics inspired by data compression methods [38,25,4,26]; however, the fundamental drawback of these works is that they do not provide any analysis of how well these heuristics measure the similarity with segment rearrangements.

Our sequence comparison problem is related to two problems that have attracted recent attention.

The first one is the *smallest grammar problem* which is to find the smallest context-free grammar that exactly generates the given string σ. This problem can be thought of as a sequence comparison problem between ϕ the empty string and σ, where the transformations are defined by the production rules of the grammar. However, our problem is more general than what can be captured by a set of production rules in context-free languages. This is primarily because we allow deletions.

[1] It is possible to obtain a symmetric version of a given distance via any commutative operation on $d(R \to S)$ and $d(S \to R)$; e.g., one can take their mean, minimum or maximum.

For example $\sigma = aca, aba, aca$ can be obtained from ϕ by at most 6 edit operations: $\phi \rightarrow b \rightarrow ba \rightarrow aba \rightarrow aba, aba \rightarrow aca, aba \rightarrow \sigma$. However the smallest grammar has cost 9: $\sigma \rightarrow ABA\ B \rightarrow aba$, $C \rightarrow aca$. The cost of a grammar, as is standard, is the total number of symbols in the right hand side of all the production rules.

The best known approximation for the smallest grammar problem is within factor $O(\log n)$ [7]. Any significant improvement to this approximation would require progress on a longstanding algebraic problem [24].

Second, there is a body of algorithms for estimating the *block edit distance* between two sequences which is quite similar to the segment rearrangement distance. The key difference is in the segment rearrangement operations that are allowed: in block edit distances, segment deletes are either not allowed at all [11] or are allowed only in a restricted way [10,29]. In [10,29], more specifically, any deletion can be performed *only* on a segment that has another copy elsewhere in the reminder of the sequence: the intuition is that such a delete operation is an inverse of a segment copy edit. Under this definition, the block edit distance between a long sequence σ and the empty sequence ϕ may be quite large. On the other hand the segment rearrangement distance between any σ and ϕ is at most one. Other block edit distances such as the ones in [30,11,34] do not allow segment copies. There does not seem to be any biological justification for assuming such restrictions in segment rearrangements: both copies and arbitrary deletes have important roles. [38,25,4,26]. Therefore, we focus on the unrestricted segment rearrangement problem here.

Block edit distances can be approximated by embedding strings to vector spaces and thereby reducing the segment rearrangement distances to vector nroms: those methods provably do not work for the segment rearrangement problem. All known block edit distances are hard to compute exactly, and the best known algorithms yield an $\Omega(\log n \log^* n)$ approximation at best.

1.3 Our Results

Our main result is a linear time, greedy algorithm that we prove approximates the segment rearrangement distance between R and S to within a constant factor.

This is the first known, provably approximate result for this problem. In contrast to our result, approximation factors for related problems such as the smallest grammar problem and block edit distance problems are at best $\Omega(\log n)$. Our approximation algorithm is quite simple to understand and implement, and is likely to find use in practice.

We modify our result so that it can work in one-pass, utilizing small workspace; in particular, we show that our algorithm can be made to work in $O(k + n/k)$ workspace with an $O(k)$ factor increase in the approximation factor.

This is in the spirit of a number of recent results for processing massive data sets in the streaming model where the goal is to perform computations in one pass, with as small a workspace as possible (see the survey in [28]). Our result falls in this category. In computational genomics, it has long been the case that workspace proves to be a bottleneck in comparing sequences. For example, in

a classical result, Hirschberg [14] showed that the longest common subsequence problem can be implemented using only linear workspace; in contrast, other sequence comparison algorithms including the ones for standard edit distance computation use larger, even quadratic space, which is prohibitive. In the massive genome context we study here, workspace becomes even more of a critical resource, and hence, our sublinear space results may prove not only useful in practice, but also necessary.

Finally, our approach to solving this problem gives improved bounds for related problems such as computing the block edit distances.

For example, for a block edit distance problem, i.e., in the presence of non-overlapping character edits and block copies, relocations and restricted deletions on segments that have copies elsewhere, the best known previous algorithm had an approximation ratio $O(\log n \log^* n)$ [10,29]. We can get an $O(1)$ factor approximation for this problem as a byproduct of our main result.

1.4 Technical Overview

There are three different technical approaches to solving edit distance problems in general. Dynamic programming is useful for character edits, but typically does not apply for segment rearrangement distances. In [7] on the smallest grammar problem, Lempel-Ziv-77 compression method is used to parse a given sequence to construct a grammar in a tree-like manner. Our method here will also be a greedy parsing, but subtly different from the Lempel-Ziv-77 compression. Finally, in [10,29], sequences are parsed using a sophisticated algorithm to embed them into a high dimensional vector space. This approach provably does not work for estimating the segment rearrangement problem. In fact, our $O(1)$ factor approximation algorithm applies to sequence comparison, but not for embedding sequences in general.

The technical crux of our work is the analysis of our greedy algorithm. The segment rearrangement distance from R to S restricts one to work on R and transform it to S via the smallest number of available segment and character edit operations. One can define two versions of this problem based on the "source" and the "destination" of segment copy operations allowed: the more restrictive, *external* version and the general, *internal* version. These are formalizations of the models implicitly assumed by [38] and [25,4,26] respectively. Enroute to our main result, we prove fairly tight relationships between such versions of segment rearrangement problems.

In Section 2, we define the problem formally in detail (including the "external" and "internal" versions). In Section 3, we present our results for the external version of the problem which are quite simple, but crucial for our main results in Section 4 on the "internal" segment rearrangement problem which is the general problem. Our small workspace version of the algorithm can be found in [13].

2 Approximating the Segment Rearrangement Distance

Recall that the segment rearrangement distance $d(R \to S)$ is the minimum number of character and segment edit operations needed to transform sequence R into S. Since we allow unrestricted segment edits, we can drop character edits from special consideration henceforth since any character edit is also a segment edit. Our problem is to compute or estimate this distance.

Any transformation that uses edit operations can be trivially emulated by another transformation which constructs S from an initially empty string S' while using R only as a repository of substrings which can be copied into S'.[2] Henceforth, we will focus on such transformations only. Such a transformation is called *internal* if the repository also includes S' itself, i.e. segment copies (and thus segment relocations) within S' are allowed. A transformation is called *external* if the repository is set to be R throughout, i.e., any segment inserted anywhere in S' must be copied from R. In particular, in the external model, derived segments of S' that are not found in R can not be reused in a segment edit.

Definition 1. *We denote by $b_{int}(R \to S)$ the minimum number of segment edit operations (relocations, copies, and deletions) needed to transform R into S in the internal model. Similarly we denote by $b_{ext}(R \to S)$ the minimum number of segment edit operations needed to transform R in to S in the external model.*

Both versions of the segment rearrangement distance above are NP-hard to compute as they reduce to the optimal dictionary selection problem [35].

Because of this computational hurdle, most biological work has resorted to using a simpler and computationally easier approach based on the Lempel-Ziv-77 data compression method. The Lempel-Ziv-77 method greedily partitions a string S into "phrases" from left to right such that each phrase $S[i:j]$ for $j > i$ must occur previously in S.

Definition 2. *Denote by $LZ(S)$ the number of Lempel-Ziv-77 phrases obtained in S. We define $c_{int}(R \to S)$, the* internal compression distance *from R to S, as the number of phrases in a parsing of S which can occur not only earlier in S but also anywhere in R. We define $c_{ext}(R \to S)$, the* external compression distance *from R to S analogously as the number of greedy phrases in S each of which occurs as a substring in R only.*

If $R||S$ denotes the concatenation of R and S separated by delimiter $||$ which is not a part of the alphabet of S or R, then $c_{int}(R \to S) = LZ(R||S) - LZ(R)$.

The $c_{int}(R \to S)$ (resp. $c_{ext}(R \to S)$) distance intuitively captures the following internal (resp. external) transformation from R to S: the only edit operations used are insertion of the longest possible segment from R, S' (resp. R only), or a single character to the right end of (initially empty) S' such that S' continues to be a prefix of S.

[2] To maintain the invariant that R is never changed, the emulation, if necessary, can start off by copying R as a whole into S'.

It is important to note that both $c_{ext}(R \to S)$ and $c_{int}(R \to S)$ are computable in $O(|R| + |S|)$ time via the use of (on-line) suffix tree construction algorithms [32]. Thus they provide easy to compute upper bounds to $b_{ext}(R \to S)$ and $b_{int}(R \to S)$ respectively. The main result of our paper is that these upper bounds are quite tight, i.e., they also provide lower bounds up to constant factors.

2.1 External Transformations

We first focus on the external segment rearrangement distance $b_{ext}(R \to S)$.

Theorem 1. *The external compression distance provides a* 4 *approximation to the external segment rearrangement distance, i.e.,* $b_{ext}(R \to S) \leq c_{ext}(R \to S) \leq 4 \cdot b_{ext}(R \to S)$. *Therefore, there exists a linear time,* 4 *approximation algorithm for external segment rearrangement distance computation.*

The proof is achieved through the following three claims.

Claim. Let $e_{ext}(R \to S)$ be the distance based on the external transformation that allows only character insertions and segment copies. Then $e_{ext}(R \to S)$ approximates $b_{ext}(R \to S)$ within a factor of 2, i.e. $b_{ext}(R \to S) \leq e_{ext}(R \to S) \leq 2 \cdot b_{ext}(R \to S)$.

To see why the above claim holds, consider the deletion of a segment B in S. Segment B must have been created in S' through one or more segment copies and possibly other operations. These operations as well as the deletion of B can be emulated by only two segment copy operations: one involving the segment on the right boundary of B and the other involving the segment on the left boundary of B, excluding their characters that intersect with B. Thus the number of operations is preserved for this case.

Now consider replacement of a character c in S by another character. This can be emulated by a character insertion followed by a character deletion (which in turn can be emulated without changing the number of operations as shown above). The claim follows as segment relocations are not allowed in the external model.

Claim. Let $g_{ext}(R \to S)$ be the distance which is defined in terms of the external transformation that allows segment copy and character insert operations only at the right and left boundaries of S. Then $c_{ext}(R \to S) = g_{ext}(R \to S)$.

The correctness of the claim can be shown as follows. Let $E = e(1), e(2), \ldots, e(k)$ be the smallest sequence of segment copy and character insert operations in the external model applied to both boundaries of S', i.e. $g_{ext}(R \to S) = k$. Among these edit operations let $E_L = e_L(1), e_L(2), \ldots, e_L(k_L)$ be those which are applied on the left boundary of S', maintaining the original order in E. Similarly, let $E_R = e_R(1), e_R(2) \ldots, e_R(k_R)$ be those operations which are applied to the right boundary of S', maintaining the original order in E. Obviously $k = k_R + k_L$. Since the two sequences of operations E_R and E_L do not interfere with each other any interleaving of the

two will yield the same string S. Then applying the interleaving sequence $E' = e_L(k_L), e_L(k_L - 1), \ldots, e_L(1), e_R(1), e_R(2), \ldots, e_R(k_R)$ all to the right boundary of (the initially empty) S' constructs S as well. Such a transformation simply allows segment copies and character inserts only to the right boundary of S'. To minimize the number of such edit operations, the greedy choice for the next segment to be copied (to the right boundary of S') is clearly optimal: Suppose that the optimal strategy constructs a longer prefix of S than the greedy strategy after i steps but not after $i - 1 \geq 0$ steps. Then in the i^{th} step the greedy strategy must have copied a segment which is a proper substring of the segment B copied by the optimal strategy. This contradicts the greediness of step i as all suffixes of B are substrings of R as well; this completes the proof of the claim. Now we can complete the proof of the theorem by showing:

Claim. $e_{ext}(R \to S) \leq g_{ext}(R \to S) \leq 2 \cdot e_{ext}(R \to S)$.

The claim follows from the fact that segment copy and character insert operations to arbitrary locations in S' can be reordered from left to right in a way that each segment copy or character insert operation which splits an earlier copied segment B into two portions B_1 and B_2, can be emulated by at most two operations in the transformation that yields $g_{ext}(R \to S)$: the original segment copy or the character insertion itself and another segment copy for B_2.

Note that the approximation factor of 2 is tight: Let $R = x, y$ and $S = x^i, y^i$. It is easy to verify that $c_{ext}(R \to S) = g_{ext}(R \to S) = 2i$ and $e_{ext}(R \to S) = i$ (consecutively insert R in the middle of S'). This completes the proof of Theorem 1.

2.2 Internal Transformations

Now we focus on generalizing $b_{ext}(R \to S)$ and $c_{ext}(R \to S)$ to $b_{int}(R \to S)$ and $c_{int}(R \to S)$ respectively and proving that $c_{int}(R \to S)$ approximates $b_{int}(R \to S)$ up to a constant factor.

Theorem 2. *Distance $c_{int}(R \to S)$ provides a 12 factor approximation to $b_{int}(R \to S)$, i.e., $b_{int}(R \to S) \leq c_{int}(R \to S) \leq 12 \cdot b_{int}(R \to S)$. Therefore, there is a linear time algorithm for approximating $b_{int}(R \to S)$ within a constant factor.*

As an illustration, consider the example:$R = v, w, x, y, z$,$Q = z, y, x, w, v$ and $S = w, v,\ x, w,\ y, x,\ z, y,\ y, x, w,\ x, w, v,\ z, y, x,\ z, y, x, w,\ y, x, w, v,\ z, y, x, w, v$. The reader can verify that $c_{int}(R \to Q) = 5$ and $c_{int}(Q \to S) = 10$. Then, $b_{int}(R \to S) \leq 5 + 10 + 1 = 16$ because one can transform R into Q, Q into QS and finally, QS into S with a single deletion. In contrast, $c_{int}(R \to S) = 20$.

One may think that as the length of S and R grow, "workspace" Q may help the ratio $(c_{int}(R \to S))/(\min_Q(c_{int}(R \to Q) + c_{int}(Q \to S) + 1))$ grow in an unbounded manner. We prove below that there is an upper bound on this ratio, and that ui used in the proof of the theorem above. The proof is achieved through the following steps.

Claim. Let $g_{int}(R \to S)$ be the internal transformation distance that allows only segment copy and character insert operations and only at the right boundary of S'. Then $c_{int}(R \to S) = g_{int}(R \to S)$.

The proof of this claim simply follows from the proof of Claim 2.1.

Claim. Let $e_{int}(R \to S)$ be the internal transformation distance that permits only segment copy and character insertion operations to the right boundary of S' as well as one final prefix deletion operation. Then, $e_{int}(R \to S) \leq c_{int}(R \to S) \leq 3 \cdot e_{int}(R \to S)$.

The transformation giving rise to $e_{int}(R \to S)$ first constructs an intermediate string $S' = P$ then constructs $S' = PS$. We will partition the construction of P into substeps $i = 1, 2, \ldots$. Each substep i adds a new substring P_i to the right end of S' in such a way that P_i is constructed externally by copying substrings of $R, P_1 \ldots P_{i-1}$ to its right end. For any given $k \geq 0$, one can now define $e_{int}(R \to S)_k$ in terms of a transformation which is similar to the one that defines $e_{int}(R \to S)$ with the extra constraint that the maximum number of partitions P_i is upper bounded by k. It is easy to see that $e_{int}(R \to S)_0 = c_{ext}(R \to S)$.

Let $P = argmin_Q(c_{ext}(R \to Q) + c_{int}(R, Q \to S)))$. Then clearly $e_{int}(R \to S)_1 \leq c_{ext}(R \to Q) + c_{int}(R, Q \to S)$. Any Q can be written as a concatenation of substrings $Q_1, Q_2, \ldots$, each of which is the result of a single segment copy from R. Let S be the concatenation of substrings $S_1, S_2, \ldots$, each of which is the result of a segment copy from Q, of the form $Q'_{i_1} Q_{i_2} \ldots Q_{i_{j-1}} Q'_{i_j}$ for some j, where Q'_{i_1} (resp. Q'_{i_j}) is a suffix of Q_{i_1} (resp. prefix of Q_{i_j}). The transformation defining $c_{ext}(R \to S)$ can emulate the transformation defining $e_{int}(R \to S)_1$ by processing each S_i via subdividing it into smaller segments $S_{i_1}, S_{i_2}, \ldots$ and copying them either from R or from the already constructed prefix of S.

To bound the segment copies in the emulation, we use a charging scheme that creates three tokens for each copy operation related to $e_{int}(R \to S)_1$ and then uses them to pay for the copy operations related to $c_{int}(R \to S)$. We create l, r, and c tokens, for "left", "right", and "center", as follows. Each Q_i in Q gets a c token for being copied from R. Each segment S_j, for being copied from Q into S, gets a c, an r, and an l token. As we will show below this creates $c_{ext}(R \to Q) + 3 \cdot c_{int}(R, Q \to S)$ tokens, which will imply that $c_{int}(R \to S) \leq c_{ext}(R \to Q) + 3 \cdot c_{int}(R, Q \to S)$.

Recall that each S_i the transformation defining $e_{int}(R \to S)_1$ is of the form $Q'_{i_1} Q_{i_2} \ldots Q_{i_{j-1}} Q'_{i_j}$ for some j. In the transformation defining $c_{int}(R \to S)$, multiple copies will be made to emulate the copying of S_i. Here is how the charging works. Every time a segment $S = S'_i(j)$ is either copied from R, covering a Q_i entirely, or copied from S', covering an S_k entirely, we use up the c token for that Q_i or S_k (if multiple are covered, we use an arbitrary one). While simulating the construction of an S_i, if one needs to copy a prefix (resp. suffix) of some Q_j, the r (resp. l) token of S_i is used.

The intuition behind this charging is that the r and l tokens will be used for (few) characters at the two ends of a segment corresponding to an S_i in the

compression transformation and a c token will be used for copying whole Q_i or S_i segments. We observe that there will always be a segment for what needs to be copied in the compression transformation that has an unused c token. This is due to the fact that a c token is only used up in simulating an S_i that entirely covers the segment that "owns" that token. However, the construction of S_i in $e_{int}(R \to S)_1$ creates another c token itself that provably no other copy operations will request.

One can induce on k to show by a similar argument that for any string $P = P_1 \dots P_k$, $c_{int}(R \to S) \leq c_{int}(R \to P_1, \dots, P_{k-1}) + 3 \cdot c_{ext}(R, P_1 \dots P_{k-1} \to P_k) + 3 \cdot c_{int}(R, P_1, \dots P_k \to S)$.

Because $e_{int}(R \to S)_k = \min_{\forall P = P_1, \dots, P_k}[(\sum_{i=1}^{k} c_{ext}(R, P_1 \dots P_{i-1} \to P_i)) + c_{int}(R, P \to S)]$, for any k, this implies that $c_{int}(R \to S) \leq 3 \cdot e_{int}(R \to S)$ which completes the proof of Claim 2.2.

We finish off the proof of the theorem by the following claim.

Claim. $e_{int}(R \to S) \leq 4 \cdot b_{int}(R \to S)$.

Consider the transformation defining $b_{int}(R \to S)$. Let $R_0 = R$ and let the result of the i^{th} segment edit operation give the string R_i. Thus if $b_{int}(R \to S) = k$, then $R_k = S$. We emulate each operation performed on R during the transformation via at most 4 operations permitted in the transformation defining $e_{int}(R \to S)$.

We again start with $R'_0 = R$ and iteratively construct $R'_i = R'_{i-1}R_i = R_0, R_1 \dots, R_i$.
(1) If the i^{th} operation performed by the transformation is *deletion* of segment $R_i[k : l]$, then the transformation defining $e_{int}(R \to S)$ copies the segments $R'_i[1 : k-1]$ and $R'_i[l+1 : |R'_i|]$ from R'_i to its end so as to obtain $R'_i = R'_i, R_{i+1}$.
(2) Similarly, if the i^{th} operation *copies* segment $R'_i[l : h]$ in between $R'_i[k-1]$ and $R'_i[k]$, then the transformation defining $e_{int}(R \to S)$ copies the segments $R'_i[1 : k]$, $R'_i[l : h]$ and $R'_i[k+1 : |R'_i|]$.
(3) Finally, if the i^{th} operation *relocates* segment $R'_i[l : h]$ in between $R'_i[k-1]$ and $R'_i[k]$ (WLOG, assume that $k < l$) then the transformation defining $e_{int}(R \to S)$ copies the segments $R'_i[1 : k]$, $R'_i[l : h]$, $R'_i[k+1 : l-1]$ and $R'_i[h+1 : |R'_i|]$.

The above claim implies that $c_{int}(R \to S)$ approximates $b_{int}(R \to S)$ (and thus provides a lower bound to any distance in the internal model) within a factor of 12 completing the proof of Theorem 2. □

Using the proof of the previous theorem, it will follow that

Corollary 1. *Given any two distances $f(\cdot \to \cdot)$ and $g(\cdot \to \cdot)$ in the internal model where character insertions and segment copies are allowed, $f(R \to S) \leq \phi \cdot g(R \to S)$ for some constant ϕ for all strings R, S.*

This improves the $\Omega(\log n)$ approximation in [10,29] for this special case of segment rearrangement distance problem.

3 Concluding Remarks

We believe that the constants in our approximation factor can be improved by a more detailed analysis of the relationship between segment rearrangements. A recent work reports some progress on improving the constant factor in the approximation [34]. It is wide open how to prove lower bounds for our problem. For example, recently [17], a small constant lower bound was proved for embedding the Levenshtein edit distance into L_1 and L_2. We can prove a matching lower bound for string similarity based on segment rearrangements as well. However, that does not shed light on the general hardness of comparing sequences pairwise. Another open problem is to determine if there exists sequence *embedding* that gives $o(\log n)$ approximation of segment edit distances between any pair of sequences.

References

1. A. N. Arslan, O. Egecioglu, P. A. Pevzner *A new approach to sequence comparison: normalized sequence alignment*, *Proceedings of RECOMB 2001.*
2. V. Bafna, P. Pevzner. *Genome Rearrangements and Sorting by Reversals*, *Proc. IEEE FOCS* (1993) 148–157.
3. V. Bafna, P. Pevzner *Sorting Permutations by Transpositions*, *Proc. ACM-SIAM SODA*, (1995) 614–623.
4. D. Benedetto, E. Caglioti, V. Lorento. *Language Trees and Zipping*, *Physical Review Letters*, 88(4), Jan 2002.
5. P. Ball *Algorithm makes tongue tree*, *Nature*, Science update, Jan 22, 2002.
6. A. Borodin and R. Ostrovsky and Y. Rabani, *Lower Bounds for High Dimensional Nearest Neighbor Search and Related Problems*, *In Proc. of ACM STOC*, 1999.
7. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai, A. Shelat. *Approximating the smallest grammar: Kolmogorov complexity in natural models. Proc. ACM STOC* 2002: 792–801
8. A. Caprara. *Formulations and complexity of multiple sorting by reversals Proc. ACM RECOMB*, 1999.
9. D. Christie. *A 3/2 approximation algorithm for sorting by reversals Proc. ACM-SIAM SODA*, 1998
10. G. Cormode, M. Paterson, S. C. Sahinalp and U. Vishkin. *Communication Complexity of Document Exchange*, *Proc. ACM-SIAM Symp. on Discrete Algorithms*, 2000.
11. G. Cormode, S. Muthukrishnan. *The string edit distance matching problem with moves*, *Proc. ACM-SIAM Symp. on Discrete Algorithms*, 2001.
12. DIMACS Workshop on Whole Genome Comparison, 2001 *Durand, Nadeau, Salzberg, Sankof (eds).*
13. F. Ergun, S. Muthukrishnan and S.C. Sahinalp. Comparing sequences with segment rearrangements. `http://cs.rutgers.edu/ muthu/resrch_chrono.html`.
14. D. Hirschberg *A Linear Space Algorithm for Computing Maximal Common Subsequences. CACM* 18(6): 341–343 (1975)
15. S. Hannenhalli and P. Pevzner, *Transforming men into mice (polynomial algorithm for genomic distance problem)*, *Proc. IEEE FOCS*, 581–592, 1995.

16. S. Hannenhalli, P. Pevzner *Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals*, *Proc. ACM STOC* 178–189, 1995.
17. A. Andoni, M. Deza, A. Gupta, P. Indyk and S. Raskhodnikova. *Lower Bounds for Embedding of Edit Distance into Normed Spaces. To appear in 14th Symposium on Discrete Algorithms (SODA)*, 2003.
18. Y. Ji, E. E. Eichler, S. Schwartz, R. D. Nicholls, *Structure of Chromosomal Duplications and their Role in Mediating Human Genomic Disorders*, *Genome Research* 10, 2000.
19. H. Kaplan and R. Shamir and R. Tarjan, *A faster and simpler algorithm for sorting signed permutations by reversals*, *SIAM Journal on Computing*, 2000
20. R. Karp, M. Rabin, *Efficient randomized pattern-matching algorithms*, *IBM J. of Res. and Dev.* 31 (1987) 249–260.
21. E. Kushilevitz, R. Ostrovsky and Y. Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, *Proc. ACM STOC*, (1998) 614–623.
22. V. I. Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals*, *Cybernetics and Control Theory*, 10(8):707–710, 1966.
23. E. S. Lander et al., *Initial sequencing and analysis of the human genome*, *Nature*, 15:409, Feb 2001.
24. E. Lehman, A. Shelat. *Approximation algorithms for grammar-based compression. Proc. ACM-SIAM SODA* 2002: 205–212.
25. M. Li, J. H. Badger, C. Xin, S. Kwong, P. Kearney, H. Zhang, *An information based sequence distance and its application to whole mitochondrial genome phylogeny*, *Bioinformatics*, 17, 2001
26. M. Li, X. Chen, X. Li, B. Ma and P. Vitanyi, *The Similarity Metric*, *Proceedings of ACM-SIAM SODA*, Baltimore MD, 2003.
27. D. Lopresti and A. Tomkins. *Block edit models for approximate string matching*, *Theoretical Computer Science*, 1996.
28. S. Muthukrishnan. *Data streams: Algorithms and applications.* `http://athos.rutgers.edu/ muthu/stream-1-1.ps`. 2003
29. S. Muthukrishnan and S. C. Sahinalp, *Approximate nearest neighbors and sequence comparison with block operations Proc. ACM STOC*, 2000.
30. S. Muthukrishnan and S. C. Sahinalp, *Improved algorithm for sequence comparison with block reversals Proc. LATIN*, 2002.
31. Needleman SB, Wunsch CD, *A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol.* 1970 Mar;48(3):443–53.
32. M. Rodeh, V. Pratt and S. Even, *Linear Algorithm for Data Compression via String Matching. JACM* 28(1): 16–24 (1981).
33. P. Sellers, *The Theory and Computation of Evolutionary Distances: Pattern Recognition*, *Journal of Algorithms*, 1, (1980):359–373.
34. D. Shapira and J. Storer *In-place Differential File Compression Proc. DCC*, 2003, 263–272.
35. J.A. Storer, *Data compression: methods and theory*, *Computer Science Press*, 1988.
36. W. F. Tichy, *The string-to-string correction problem with block moves*, *ACM Trans. on Computer Systems*, 2(4): 309–321, 1984.
37. C. Venter et. al., *The sequence of the human genome*, *Science*, 16:291, Feb 2001.
38. J. S. Varre, J. P. Delahaye, E. Rivals, *The Transformation Distance: A Dissimilarity Measure Based on Movements of Segments*, *Bioinformatics*, 15:3, 194–202, 1999.