Practical block sequence alignment with moves

Julien Bourdaillet and Jean-Gabriel Ganascia

Pierre and Marie Curie University
LIP6 - Computer Science Departement
8 rue du Capitaine Scott - 75015 Paris - France
{julien.bourdaillet,jean-gabriel.ganascia}@lip6.fr

Abstract. In this paper we study a sequence alignment problem motivated by *textual genetic criticism*, a humanities discipline where the notion of edit distance with moves has been rediscovered by philologists. We present a formulation of the problem and show that the usual notion of edit distance with moves does not address it correctly because it is harder. We present a heuristic algorithm for this problem and compare it with a greedy algorithm which computes the edit distance with moves. We show that our algorithm is superior for this task of block sequence alignment with moves.

1 Introduction

In this paper we study a sequence alignment problem motivated by an application in humanities (presented in Section 5.2). The problem is defined as follows. Let $A = a_1, a_2, ..., a_m = [a_i]_{1 \leq i \leq m}$ and $B = b_1, b_2, ..., b_m = [b_j]_{1 \leq j \leq n}$ be two natural language texts, i.e. two character sequences, defined over an alphabet Σ. The usual edit distance finds the minimal number of edit operations, i.e. character insertions, deletions or replacements, to transform A into B [1]. The side-effect of this computation is to produce an alignment between A and B where each character of one of the sequences is aligned with a character of the other sequence, or with the null character ϵ for inserted and deleted characters.

For natural language sequences, character-level alignment is not adapted because there exist higher-level structures, i.e. words, sentences or paragraphs, which make this granularity level too low. The notion of character block, i.e. a substring, is introduced and has to be handled by a natural language alignment algorithm. Nevertheless character-based modifications have still to be handled because they occur frequently.

Further, we wish to handle the block move detection between A and B because it brings a significant information on changes between the texts. The introduction of block moves makes the problem NP-complete under certain conditions (see Section 2.1).

This work is motivated by a practical application implying that the produced algorithm need practical complexities.

Problem Formulation

Definition 1. *Let $\mathcal{A}(A, B) = (INV, DEL, INS, REP, MOV)$ denote an alignment of two sequences A and B defined over a finite alphabet Σ where INV, DEL, INS, REP and MOV are the sets of invariant, deleted, inserted, replaced and moved blocks respectively.*

Definition 2. *A block is defined as* (p, l_A, q, l_B) *with* $-1 \leq p \leq |A| = m, 0 \leq l_A \leq m$ *and* $-1 \leq q \leq |B| = n, 0 \leq l_B \leq n$*. This specifies that the substring* $A[p..p + l_A - 1]$ *in the first sequence is related to the substring* $B[q..q + l_B - 1]$ *in the second sequence. The type of relation between both substrings is defined by the set they belong to in* $\mathcal{A}(A, B)$*. A block having either* p *or* q *equal to* -1 *represents an insertion or a deletion respectively; in this case,* l_A *or* l_B *are set to* 0 *respectively.*

These definitions do not specify the optimality criterion of an alignment. This can be done by reformulate the problem as a multiobjective optimization problem whose goal is to find an alignment $\mathcal{A}(A, B) = (INV, DEL, INS, REP, MOV)$ with the 12 following criteria, where equations 2 and 3 are parametric:

– Maximize the sum of the size of invariant blocks:

$$\max x_1 = \sum_{(p,l_A,q,l_B) \in INV} l_A + l_B \tag{1}$$

– Minimize the sum of the size of other blocks:

$$\min x_S = \sum_{(p,l_A,q,l_B) \in S} l_A + l_B \quad \text{with } S = (DEL, INS, REP, MOV) \tag{2}$$

– Maximize the average size of blocks:

$$\max y_S = \frac{\sum_{(p,l_A,q,l_B) \in S} l_A + l_B}{|S|} \quad \begin{array}{l}\text{with } S = (INV, DEL, INS, REP, MOV) \\ \text{and } |S| \text{ the number of blocks in } S\end{array} \tag{3}$$

– Maximize the ratio of moved blocks among non-invariant blocks:

$$\max z_1 = \frac{\sum_{(p,l_A,q,l_B) \in MOV} l_A + l_B}{\sum_{(p,l_A,q,l_B) \in (DEL,INS,REP,MOV)} l_A + l_B} \tag{4}$$

– Maximize the ratio of replaced blocks among deleted and inserted blocks:

$$\max z_2 = \frac{\sum_{(p,l_A,q,l_B) \in REP} l_A + l_B}{\sum_{(p,l_A,q,l_B) \in (DEL,INS,REP)} l_A + l_B} \tag{5}$$

Equations 1 and 2 are similar to the classical global alignment problem [2], introducing a term for moves. Equation 3 specifies that we wish to obtain character blocks of maximal size in order to avoid the fragmentation of the alignment in small blocks. For molecular sequence alignment, this is traditionally achieved by the introduction of *gap penalties*. Equations 4 and 5 allow the ordering between preferred non-invariant blocks: moved blocks are preferred over other non-invariant blocks because they capture a significant information between A and B; and replacements are preferred over deletions

and insertions because they enable the pairing of two blocks of A and B whereas deletions and insertions do not.

This formulation enables a clear definition of the problem, unfortunately it does not provide a way to solve it. A solution $\mathcal{A}(A, B)$ to the problem is a set of points over the two sequences A and B which partition them in blocks; if the number of such points were known, the positions of the points and the type of blocks they define would be the decision variables of the multiobjective optimization problem. But this number is unknown and the space of possibilities is clearly exponential because it is the set of possible partitions of A and B. Finally this formulation does not provide a way to find them, it only allows to characterize and compare solutions. Nevertheless equations 1-5 are used to define a single objective function in Section 3.

Further, it must be noted that this formulation is only a model of the problem, which is to pair text blocks where these pairings have to be *semantically valid*. That is, we wish to discover relations between text blocks which are valid and relevant at the natural language semantic level. Hence, the model is one possible formulation of the problem, others are possible but we claim that this one is valuable.

Section 2 presents related works on textual alignment where moves are considered. In Section 3 we demonstrate the NP-completeness of the problem by reducing it to the block edit model of Lopresti and Tomkins [3]. We present our algorithm in Section 4 and evaluate it in Section 5. Finally we conclude in Section 6.

2 Background

The usual notion of edit distance is not general enough to handle this problem. Further, because the longest common subsequence problem is a direct instantiation of this notion, it can not be used directly to model it [4].

We present below several models directly related to our problem.

2.1 Edit Distance with Moves

Tichy was the first to introduce the edit distance with block moves [5]. Shapira and Storer proved the NP-completeness of this problem and gave an approximation algorithm called GREEDY [6].

Character deletions and insertions and block moves are allowed for a cost of 1; this cost function enables a block move for a small cost. The goal is to minimize the number of operations to transform one string into another. They proved that the computation of the optimal alignment is NP-complete and gave an approximation algorithm called GREEDY which approximates the edit distance with moves up to a constant logarithmic factor. If n is the number of optimal blocks and l the length of the longest optimal block, then GREEDY identifies $O(n \log l)$ blocks.

This modelization of the alignment by GREEDY involves the creation of character blocks with the block move operator. But the two other operators do not involve the creation of blocks, hence there are no constraint for the formation of deleted and inserted blocks and this results in “fragmented” alignments (see Section 5.1).

Further, GREEDY has time complexity in $O(mn)$ which becomes very quickly impractical even for small sequences.

2.2 Block Edit Distance

Lopresti and Tomkins introduced block edit distance [3]. Let A and B be two strings defined over an alphabet Σ. A t-block substring family of A, $A|_t = \left\{A^{(1)}, ..., A^{(t)}\right\}$, is defined as a multiset containing t substrings of A. In the same way, $B|_t$ denotes a t-block substring family of B. Because $A|_t$ and $B|_t$ are multisets, substrings can be repeated several times.

If the substrings in $A|_t$ do not overlap, the family is said to be *disjoint*. If each character of A is contained in some substring, the family represents a *cover* of A. The notation C is used if one family must be a cover and $\overline{\text{C}}$ otherwise. And D is used if a family must be disjoint, $\overline{\text{D}}$ otherwise. For example, if the first substring family must be a disjoint cover and the second only a cover, the model is called CD-$\overline{\text{CD}}$.

A distance between two substrings of A and B is introduced such that:

$$dist : \{i, j | 1 \leq i \leq j \leq |A|\} \times \{k, l | 1 \leq k \leq l \leq |B|\} \rightarrow \Re \tag{6}$$

Finally we can define the block edit distance $\mathcal{B}$ as follows:

$$\mathcal{B} = \min_{t} \min_{A|_t, B|_t} \min_{\sigma \in S(t)} \left\{ t.c_{block} + \sum_{i=1}^{t} dist(A^{(i)}, B^{(\sigma(i))}) \right\} \tag{7}$$

This corresponds to the best way to choose two substring families of A and B and to pair each member of $A|_t$ with some member of $B|_t$ such that the cost induced by pairings be minimal. Pairing cost is based on $dist$ plus a per-block cost c_{block}.

This model is a meta-model which depends on the requirements for strings A and/or B to be disjoint and/or a cover. The generality of the model comes from the fact that the requirements for A and/or B to be disjoint and/or a cover are not specified by equation 7. Hence, depending on the application studied, the meta-model is instantiated in a model where requirements are specified by the application.

When at least one of the two substrings is unconstrained, i.e. neither disjoint nor a cover, or $\overline{\text{CD}}$, the problem is proved to have either cubic- or biquadratic-time algorithm in function of the size of the input. In other cases, Lopresti and Tomkins proved that the problem is NP-complete.

2.3 Other Models

A distance measuring the number of character which have to be deleted in A in order that all remaining substrings be also substrings of B is presented in [7]. Their greedy algorithm, with time complexity in $O(mn)$, is the following: find the longest prefix of A ending in i which matches a substring of B, then this is repeated with the remaining suffix of A (i.e. $A[i + 1..|m|]$) until no matching is possible. This algorithm does not identify invariant blocks from moved blocks and its complexity is quadratic.

An 1.5 approximation algorithm for sorting by transpositions (i.e. moves) with time complexity in $O(mn)$ is proposed in [8]. It is based on the assertion that B is a permutation of A: this assertion is very strong and can not be applied in our case.

This model is generalized with the rearrangement distance presented in [9].

3 Model

From Lopresti and Tomkins' proof that their block edit model is NP-complete for certain variants, we can state that:

Theorem 1. *Block sequence alignment with moves problem is NP-complete.*

Proof. The problem can be reduced to the following: identify and pair only invariant and moved blocks where blocks do not have to cover A and B but must be disjoint. Then deletions and insertions can be deducted as blocks not identified as invariant or moved (replacements can be omitted because they are equivalent to a deletion plus an insertion). This problem is to find a $\overline{\text{C}}$D-$\overline{\text{C}}$D alignment which has been proved to be NP-complete by Lopresti and Tomkins.

Because of the NP-completeness of the problem, two options follow to solve it: we can either find an approximation algorithm or a heuristic algorithm. Shapira and Storer derive a simple greedy approximation algorithm for edit distance with moves: this problem is a single-objective optimization problem and the deduction of greedy algorithm is natural. For a multiobjective optimization problem, there is no simple way to derive a greedy algorithm. Lopresti and Tomkins' block edit models are single-objective also. They propose polynomial-time algorithms for non NP-complete versions of the problem but do not consider NP-complete versions. These two single-objective optimization models are suitable for theoretical studies but do not handle the properties which are required by the problem.

As shown in Section 1, we have a multiobjective optimization problem but the set of decision variables is unknown, hence classical numerical methods can not be used to solve it. Nevertheless from these objective functions, a single-objective function can be derived in order to evaluate solutions in a mono-dimensional space: equations 1-5 can be combined in the following simple way.

Equations 1 and 2 can be combined and normalized as follows:

$$\max x = \frac{\left(1 + \frac{x_1 - \sum_S x_S}{|A|+|B|}\right)}{2} \text{ with } S = (DEL, INS, REP, MOV) \tag{8}$$

The 5 equations induced by equation 3 can be combined and normalized as follows:

$$\max y = \frac{\left(\frac{\sum_S y_S}{\max(S)}\right)}{5} \text{ with } S = (INV, DEL, INS, REP, MOV) \tag{9}$$

where $\max(S)$ returns the size of the largest block inside set S. Equations 8 and 9 are normalized between 0 and 1. Equations 4 and 5 are combined in an evident way:

$$\max z = \frac{z_1 + z_2}{2} \tag{10}$$

Finally a similarity function can be formulated as:

$$sim(A, B) = \max\left(c_1 x + c_2 y + c_3 z\right) \quad \text{with } 0 \leq c_3 \leq c_2 \leq c_1 \leq 1, c_1 + c_2 + c_3 = 1 \tag{11}$$

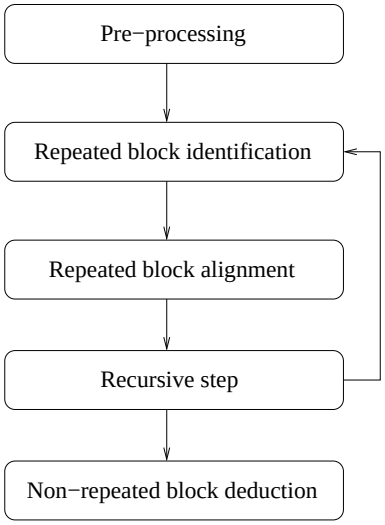

Fig. 1. Algorithm

Equation 11 is a simple normalized linear combination of equations 1-5. c constants have to be chosen in function of the application but superiority constraint between them clearly state priorities.

4 Algorithm

Our algorithm, named MEDITE, is closely related to fragment alignment commonly used in bioinformatics [10, 11]. Sequences are processed in four steps. The first step is a pre-processing step where character equivalence classes are set. The second step identifies repeated character blocks between A and B. The third step aligns these repeated blocks in order to determine which are invariant and which are moved. The fourth step is a recursive iteration of steps 2 and 3 between each pair of aligned blocks during step 2. The last step is the deduction of insertions, deletions and replacements. Figure 1 presents the algorithm.

4.1 Pre-processing

Sequences can optionally be pre-processed in the following way. In natural language sequences, there exist classes of characters which might be considered as equivalent: identical upper- and lower-case characters (i.e. “D” and “d”), accentuated or not characters (i.e. “é” and “e”) and separators (i.e. “?” and “!”). Upper-case letters are converted to their lower-case version, accentuated characters to non-accentuated and all separators are converted to the dot character (“.”). Hence sequences A and B can be pre-processed with these equivalence classes in linear time. This pre-processing step allows, in further steps, the matching of blocks with differences for a small computational cost.

To illustrate the algorithm, we use the following example: the alignment of the two sentences “This morning the cat observed little birds in the trees.” and “The cat was observing birds in the little trees this morning, it observed birds for two hours.”. After preprocessing, sequences become: “this.morning.the.cat.observed.little.birds.in.the.trees.” and “the.cat.was.observing.birds.in.the.little.trees.this.morning.it.observed.birds.for.two.hours.”.

4.2 Repeated Block Identification

The identification of repeated character blocks is done by building a generalized suffix tree between A and B [12]. This data structure enables to find all repeated character blocks between A and B. The size of this set of blocks is exponential and only a subset is interesting: the subset of super maximal exact matches (SMEMs) defined below [13].

Definition 3. *A block (p, l_A, q, l_B) is a SMEM if and only if:*

- $A[p..p + l_A - 1] = B[q..q + l_A - 1]$ *(exact matching);*
- $A[p - 1] \neq B[q - 1]$ *and* $A[p + l_A] \neq B[q + l_B]$ *(maximality);*
- *and neither $A[p..p + l_A - 1]$ nor $B[q..q + l_B - 1]$ are included in another maximal match (super-maximality).*

This definition does not prevent overlapping between SMEMs whereas SMEMs have to be non-overlapping in our application. Overlaps are resolved heuristically by cutting them on separators because it is better to have an inter-word cut than an intra-word cut in natural language sequences. This first step results in two lists A' and B' of non-overlapping SMEMs.

After the second step, the following SMEMs are identified: “this.morning. the.cat. observed. little. birds.in.the. trees.” and “the.cat. was.observing. birds.in.the. little. trees. this.morning. it. observed. birds.for.two.hours.”. The word “birds” repeated 3 times does not appear in the SMEM list because the first two occurrences are included in longer SMEMs. Had it been in the list, the super-maximality would not be respected.

4.3 Repeated Block Alignment

Each of these SMEMs can be either an invariant or a moved block. The pairwise alignment of SMEMs enables to make the decision: aligned SMEMs are considered as invariants and unaligned as moved. Because the space of possible alignments is combinatorial, we use an A* heuristic algorithm.

Possible alignments are evaluated with a cost function c; the goal is to find a minimal cost alignment. This is equivalent to a shortest path problem where the goal state is the minimal cost alignment. The initial state corresponds to the state where no pairing has been chosen yet. At each step of the algorithm one pairing must be chosen; this is done by estimating the alignment cost induced by each possible pairing with the function c. It is a greedy best first search algorithm, so the best pairing is chosen and then this process is iterated until the goal is achieved. In order to find the goal state, the heuristic

has to be admissible, i.e. to never overestimate the distance to the goal. In our case the heuristic must never overestimate the cost of an alignment; we detail below why c is admissible.

The evaluation of the alignment cost induced by the pairing of A'_i with B'_j is computed with $c(i,j)$ which breaks down into the cost of the alignment so far $g(i,j)$ and the heuristic cost of the next blocks to align $h(i,j)$, such that $c(i,j) = g(i,j) + h(i,j)$. These costs are computed using the following formulas:

- $U(i,j) = unaligned(A'[1..i-1], B'[1..j-1])$ is the set of unaligned blocks during previous steps, those that have not been paired and considered as moved.
- $g(i,j) = \Sigma_{b \in U(i,j)}|b|$ is the sum of previously unaligned blocks' size, that is the alignment cost is charged by moved blocks only.
- $SD(i,j) = A'[i+1..|A'|] \ominus B'[j+1..|B'|]$ is the symmetric difference of the next blocks to align during next steps. Blocks in $SD(i,j)$ are present either only in $A'[i+1..|A'|]$ or only in $B'[j+1..|B'|]$. It will never be possible to pair them during next steps and because of that they will be considered as moves and will charge the alignment cost.
- $h(i,j) = \Sigma_{b \in SD(i,j)}|b|$ is the sum of these blocks' size, i.e. $h(i,j)$ is the lower bound of the alignment cost of the next blocks. It never overestimates the alignment cost because the minimal alignment cost will be $h(i,j)$; this is why c is admissible and A* finds the optimum.

This computation is equivalent to finding an alignment that is optimal in the sense of the maximization of the sum of invariant block size and the minimization of the sum of moved block size.

In our example, after the third step, the following aligned blocks (in bold) are considered as invariant; the other squared blocks are moved: "this.morning. **the.cat.** observed. little. **birds.in.the.** **trees.**." and "**the.cat.** was.observing. **birds.in.the.** little. **trees.** this.morning. it. observed. birds.for.two.hours.".

4.4 Recursive Step

The fourth step consists in looping over the pairings resulting from step 3 and in considering the subsequences between each pair of aligned blocks. These subsequences are processed again with steps 2 and 3. It allows the pairing of new blocks which are then included in the main alignment. This recursive step enables the pairing of blocks which would otherwise have been left unaligned.

In the example, the subsequences "observed.little." and "was.observing." occur between the invariant blocks "the.cat." and "birds.in.the.". Recursive step 2 finds the SMEM "observ" and recursive step 3 aligns it. Hence the final alignment becomes: "this.morning. **the.cat.** **observ**ed. little. **birds.in.the.** **trees.**" and "**the.cat.** was. **observ**ing. **birds.in.the.** little. **trees.** this.morning. it.observed.birds.for.two.hours.". The moved block "observed." is lost but the invariant block "observ" is discovered. The algorithm favours local similarities rather than long-distance matching.

4.5 Other Block Deduction

Insertions, deletions and replacements can then be deduced. Deletions are non-repeated blocks in A and insertions are non-repeated blocks in B. Further, when there is a deleted block d in A and an inserted block i in B between two pairs of aligned blocks, and the ratio $|d|/|i|$ reaches a threshold t, then d and i are transformed in replacements r_1 and r_2, meaning that r_1 has been replaced by r_2. t is arbitrarily set to 0.5.

In the example, the not framed block of the first sequence is considered as a deletion and the three not framed blocks of the second sequence are considered as insertions.

Finally, a post-processing step enables to recover blocks in the original sequences (i.e. without all separators equivalent to "." for example).

5 Evaluation

5.1 Synthetic Data

The goal of this experiment is to evaluate the quality of our algorithm MEDITE versus GREEDY from Shapira and Storer [6] on synthetic data when a reference alignment exists. Given a text and a noise model, a second text is generated by altering the first one; the alignment between the texts is recorded during the alteration process. Then, it is possible to evaluate the quality of the aligner by comparing its results with the reference alignment.

The noise model allows the generation of a modified text from an original text in the following way. Ratios of insertions, deletions, replacements and moves on the original text size are set before processing. Then character blocks are repeatedly inserted in the modified text, deleted in the original text, replaced between both texts and shifted from one position in the original text to another one in the modified until ratios are reached. The positions where operations occur are chosen randomly (overlapping operations are not allowed). The operations deal with character blocks rather than single characters in order to simulate real operations on words; the size of character blocks is randomly chosen between 1 and 25 (i.e. single characters are allowed). During this process the positions in the original text where deletions occur, the positions in the modified text where insertions occur and the positions in both texts where replacements and moves occur are recorded. Finally, a reference alignment between the original and the modified text is produced.

For the experiment we have chosen 4 texts, referred as A, B, C and D, of size 2K, 6K, 18K and 40K characters for the original text. For each text, five modified documents were generated with the noise model. Then the documents were aligned with MEDITE and GREEDY, and similarities evaluated using equations 8-11. For this experiment the similarity measure sim (Equation 11) has been set to $sim = 0.5x + 0.35y + 0.15z$. Two series of tests with different modification ratios were conducted: in the first one there are 5 % of insertions, 5 % of deletions, 5 % of replacements and 5% of moves, which means that there is a 20 % difference between original and modified texts; in the second series, the ratio is set to 10 %, meaning that there is a 40 % difference. For each of the five kinds of character blocks (insertions, deletions, replacements, moves and invariants) the accuracy is defined as the number of correctly aligned characters divided

by the total number of characters; then the average accuracy is calculated. The average runtimes of alignments are also calculated. Table 1 and 2 present the average results for each series of texts and modification ratio with GREEDY and MEDITE respectively.

Table 1. Results of the synthetic data alignment with GREEDY

Modification ratio	5 %				10 %			
Text	A	B	C	D	A	B	C	D
Average x	0.7592	0.7968	0.8053	0.8051	0.6507	0.6720	0.6689	0.6721
Average y	0.1939	0.0980	0.0751	0.0435	0.1379	0.1112	0.0669	0.0360
Average z	0.2288	0.2439	0.2927	0.2692	0.3139	0.2693	0.3475	0.3153
Average sim	0.4818	0.4693	0.4728	0.4581	0.4207	0.4154	0.4100	0.3959
Average accuracy	0.4845	0.5202	0.4335	0.4695	0.4528	0.4882	0.4064	0.4448
Average runtime	0mn 3s	0mn 50s	6mn 50s	31mn 28s	0mn 7s	1mn 57s	22mn 55s	5h 3mn

Table 2. Results of the synthetic data alignment with MEDITE

Modification ratio	5 %				10 %			
Text	A	B	C	D	A	B	C	D
Average x	0.7509	0.7410	0.7391	0.7422	0.5144	0.5133	0.4982	0.5040
Average y	0.4847	0.3360	0.2705	0.2583	0.4031	0.3075	0.2581	0.2319
Average z	0.3783	0.3251	0.3488	0.3409	0.3615	0.3447	0.3460	0.3413
Average sim	0.6017	0.5369	0.5166	0.5127	0.4525	0.4160	0.3914	0.3843
Average accuracy	0.7417	0.7532	0.7548	0.7686	0.6393	0.6509	0.6452	0.6590
Average runtime	0mn 1s	0mn 1s	0mn 5s	0mn 16s	0mn 1s	0mn 1s	0mn 6s	0mn 21s

Results show that GREEDY and MEDITE present average sim of around 0.5 for the 5 % modification ratio and of around 0.4 for the 10 % modification ratio; but whereas their average z (ratio of moves and replacements among other modifications) is quite similar, average x and y are very different: for the 10 % modification ratio, average x is of around 0.65 with GREEDY and 0.5 with MEDITE, and average y is of around 0.1 with GREEDY and 0.3 with MEDITE. This means that GREEDY aligns more invariant blocks among other blocks than MEDITE (x) but these blocks are very small (y) and the resulting alignment is very "fragmented" which is very hard to understand for a human reader. Nevertheless the average accuracy is of around 0.45 with GREEDY and 0.65 with MEDITE: this shows that alignments produced by GREEDY are less good than those by MEDITE. Further, average runtimes are very different between the two algorithms.

The experiment was realized on a Pentium 4, 2.4 GHz with 1 GB of RAM. MEDITE is implemented in Python, a high-level language good for prototyping but slow in execution. A C language implementation would allow to win an order of magnitude in speed. Nevertheless, the speed bottleneck of our algorithm is the calculation of symmetric differences between lists of SMEMs (see Section 4.3) which is quadratic in the length of the lists. GREEDY is quadratic in the length of input sequences, that is why it is very slow.

5.2 Application to *Textual Genetic Criticism*

The textual alignment problem we addressed is issued from a humanities discipline, called *textual genetic criticism*, which is interested on writers' drafts [14]. These drafts come from the writing process, where the author modifies his text several times. Textual geneticians study these drafts by comparing them in order to understand the genesis of the text and by using these four operators: insertions, deletions, replacements and moves. It is interesting to note that they rediscovered the notion of edit distance with moves by themselves.

For this experiment, different pairs of real versions of texts A, B, C and D are compared instead of synthetic texts. There exists no reference alignment for these text versions and accuracy can not be evaluated. Table 3 presents results with GREEDY and MEDITE.

Table 3. Results of the alignment of real data with GREEDY and MEDITE

Modification ratio	GREEDY				MEDITE			
Text	A	B	C	D	A	B	C	D
x	0.3654	0.2657	0.4106	0.7835	0.4934	0.2697	0.4936	0.9223
y	0.1161	0.0793	0.0784	0.1397	0.3331	0.2488	0.1951	0.2318
z	0.1971	0.2340	0.4096	0.1653	0.2003	0.1676	0.2937	0.2587
sim	0.2529	0.1957	0.2942	0.4655	0.3933	0.2471	0.3591	0.5811
Runtime	0mn 18s	12mn 5s	1h 1mn	29mn 3s	0mn 1s	0mn 2s	0mn 6s	0mn 2s

Table 3 shows that MEDITE achieves significantly better results: for all the texts and criteria (but z) as well as for the aggregated measure sim results are better for MEDITE. Further the examination of alignments produced by GREEDY are very hard to understand for a human reader because blocks are very small, this makes its results impractical. Runtimes for texts C and D are very different from the previous experiment because both versions of text C are very different and both versions of text D are quite similar.

6 Conclusion

We presented the block sequence alignment with moves problem and showed that it is harder than the computation of edit distance with moves because this problem would be formulated as a multiobjective optimization problem if it would not be NP-complete. We presented a heuristic algorithm which achieves better results than a greedy algorithm that computes the edit distance with moves on synthetic data.

More generally, we think that the usual notion of edit distance with moves is not rich enough to handle block alignment with moves of natural language texts. Further work is needed to study this problem, specially to find a bounded exact algorithm and to refine our heuristic algorithm.

References

1. Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversal. Cybernetics and Control Theory **10**(8) (1966) 707–710
2. Needleman, S., Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology **48**(3) (1970) 443–453
3. Lopresti, D.P., Tomkins, A.: Block Edit Models for Approximate String Matching. Theoretical Computer Science **181**(1) (1997) 159–179
4. Bergroth, L., Hakonen, H., Raita, T.: A Survey of Longest Common Subsequence Algorithms. In: SPIRE '00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval. (2000)
5. Tichy, W.F.: The String-to-String Correction Problem with Block Moves. ACM Trans. Comput. Syst. **2**(4) (1984) 309–321
6. Shapira, D., Storer, J.A.: Edit Distance with Move Operations. In Apostolico, A., Takeda, M., eds.: CPM. Volume 2373 of Lecture Notes in Computer Science., Springer (2002) 85–98
7. Ukkonen, E.: Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science **92** (1992) 191–211
8. Bafna, V., Pevzner, P.: Sorting by transpositions. SIAM Journal of Discrete Mathematics **11** (1998) 224–240
9. Amir, A., Aumann, Y., Benson, G., Levy, A., Lipsky, O., Porat, E., Skiena, S., Vishne, U.: Pattern matching with address errors: rearrangement distances. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, ACM Press (2006) 1221–1229
10. Wilbur, W.J., Lipman, D.J.: The context dependent comparison of biological sequences. SIAM J. Applied Mathematics **44**(3) (1984) 557–567
11. Bray, N., Dubchak, I., Pachter, L.: AVID: A Global Alignment Program. Genome Res. **13**(1) (2003) 97–102
12. Ukkonen, E.: On-Line Construction of Suffix Trees. Algorithmica **14**(3) (1995) 249–260
13. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computer Biology. Cambridge University Press (1997)
14. Deppman, J., Ferrer, D., Groden, M., eds.: Genetic Criticism - Texts and Avant-textes. University of Pennsylvania Press (2004)