An $O(n^4)$ algorithm for alignment with non-overlapping inversions

Alair Pereira do Lago[1], Ilya Muchnik[2], and Casimir Kulikowski[2]

[1] Universidade de São Paulo
alair@ime.usp.br
[2] Rutgers University
muchnik@dimacs.rutgers.edu,kulikows@cs.rutgers.edu

Abstract. Alignment of sequences is widely used for biological sequence comparisons, and only biological events like mutations, insertions and deletions are considered. Other biological events like inversions are not automatically detected by the usual alignment algorithms, thus some alternative approaches have been tried in order to include inversions or other kind of rearrangements.
Despite many important results in the last decade, the complexity of the problem of alignment with inversions is still unknown. In 1992, Schöniger and Waterman proposed the simplification hypothesis that the inversions do not overlap. They also presented an $O(n^6)$ exact solution for the alignment with non-overlapping inversions problem, and introduced a heuristic for it that brings the running-time complexity down. In this paper, n denotes the maximal length of the two aligned sequences.
The present paper gives an exact algorithm for the simplified problem. We give a quite simple dynamic program with $O(n^4)$-time and $O(n^2)$-space complexity for alignments with non-overlapping inversions.

1 Introduction

In evolution history, some biological events introduce changes in the DNA sequences. Some typical biological events include *mutations*, in which a nucleotide is substituted by another one, *deletions* and *insertions* of nucleotides. Hence, any sequence comparison must take into consideration the possibility of these events if it is expected to identify high similarity between two sequences. Typical alignment procedures try to identify which parts do not change and where these biological events are, after exhibiting one best alignment according to some optimization criteria.

Alignments can be associated with a set of edit operations that transform one sequence to the other. Usually, the only edit operations

that are considered are the *substitution* (mutation) of one symbol by another one, the *insertion* of one symbol and *deletion* of one symbol. If costs are associated with each operation, there is a classic $O(n^2)$ dynamic program that computes a set of edit operations with minimal total cost and exhibit the associated alignment, which has good quality and high likelihood for realistic costs.

Consider three new possible edit operations:

- the 2-*reversion*, which reverses the order of *two consecutive symbols*;
- the *reversion* operation, which reverses the order of *any segment* of symbols instead of a segment of length 2;
- the *inversion* operation, which substitutes any segment by its *reverse complement* sequence. The inversion operation is the operation that is interesting in molecular biology.

Associated with any of these three operations, we can define new alignment problems. For instance, given two sequences and fixed costs for each kind of edit operation, the *alignment with inversions* problem is an optimization problem that queries the minimal total cost of an edit operations set that transforms one sequence to the other. Moreover, one may also be interested in the exhibition of its correspondent alignment and/or edit operations. Similarly, we can define the *alignment with* 2-*reversions* and the *alignment with reversions* problems.

In 1975, Wagner [1] studied the alignment with 2-reversions problem and proved that it admits a polynomial solution if the cost of 2-reversion is null. On the other hand, he also proved that the obtainment of an optimal solution is *NP-hard*, if any operation has a constant positive cost.

To the best of our knowledge, the computational complexities of alignment with reversions and alignment with inversions problems are unknown.

In order to deal with alignments with inversions, three main approaches have been considered through the years:

- non-overlapping inversions [2];
- sorting unsigned permutations by reversals [3,4] and;
- sorting signed permutations by reversals [5,6,7].

In 1992, Schöniger and Waterman [2] introduced a *simplification hypothesis*: *all regions involving the inversions do not overlap*. This led to the *alignment with non-overlapping inversions* problem. They presented a $O(n^6)$ solution for this problem and also introduced a *heuristic* for it that reduced the running-time. This heuristic uses the algorithm by Waterman and Eggert [8] that reports the K best non mutual intersecting local alignments in order to reduce the running time to something between $O(n^2)$ and $O(n^4)$, depending on the data.

The other two approaches apply well for alignment of *sequences of genes* and have been very used with mitochondrial genomes. It does not apply for sequences of nucleotides nor for sequences of aminoacids because *no repetitions* of symbols are allowed. (Repeated genes and paralogs are not allowed.) Moreover, *no insertion* and *no deletion* are considered and the only admitted operation is the reversal, where a *reversal* is defined to transform a sequence like $(1, 2, 3, 4, 5)$ into $(1,\ 4, 3, 2,\ 5)$ ($(1,\ -4, -3, -2,\ 5)$ in the signed version).

This paper gives an algorithm for the alignment with non-overlapping inversions problem, which is the first approach. Algorithm 1 is a $O(n^4)$ solution that uses $O(n^2)$ space.

2 Basic Definitions

Let A be any *alphabet*, a set of *letters*. Any finite sequence on A is also called a *word on* A or simply a *word* if the alphabet is clear. Let A^* be the set of all words on A, including the *empty word* denoted by 1. We identify words of length 1 to the letters they contain. The concatenation $\cdot$ of words is an associative operation defined over A^* and it will be often omitted. Let $w = w_1 w_2 \dots w_k$ be a word. We denote by $|w|$ the *length* k of w. We will also denote w_i, the i-th letter of w, by $w[i]$. Let $x, y, z \in A^*$. We denote by xA^* the set $\{xy \mid y \in A^*\}$, we denote by A^*x the set $\{yx \mid y \in A^*\}$ and we denote by A^*xA^* the set $\{yxz \mid y, z \in A^*\}$. We say that x is a *prefix of* w if $w \in xA^*$, we say that x is a *suffix of* w if $y \in A^*x$ and we say that x is a *factor of* w if $w \in A^*xA^*$. For $1 \leq i \leq j \leq k$, the factor $w_i w_{i+1} \cdots w_j$ of w is also represented by $w[i\,..\,j]$.

Let $-$, also called *inversion*, be any operation on A^* that satisfies the following properties:

1. $\overline{a} \in A, \forall a \in A$
2. $\overline{x \cdot y} = \overline{y} \cdot \overline{x}, \forall x, y \in A^*$

Notice that the inversion operation on A^* is defined by its values on the letters of A. For instance, let $A = \{a, c, t, g\}$ and let $s \in A^*$ be any DNA sequence. If the inversion is defined by $\overline{a} = a$, $\overline{t} = t$, $\overline{c} = c$ and $\overline{g} = g$, it maps s to its *reverse sequence*. On the other hand, if the inversion is defined by $\overline{a} = t$, $\overline{t} = a$, $\overline{c} = g$ and $\overline{g} = c$, it maps s to its *reverse complement sequence*. Last case is the interesting case for DNA sequences in molecular biology.

Let $\omega : A \times A \longrightarrow \mathbb{R} \cup \{-\infty\}$ be any *weight* function. We say that $(a, b) \in A \times A$ is a *match* if $\omega(a, b) \neq -\infty$.

Let $s = s_1 s_2 \cdots s_k$ and $t = t_1 t_2 \cdots t_{k'}$ be two words. We define the *matching graph of s and t* as the weighted and colored bipartite graph $G = G(s, t, \omega, -) = (V, E)$. The vertex set is a set of symbols $V = \{s_1, \ldots, s_k, t_1, \ldots, t_{k'}\}$ and has size $|s| + |t|$. (We do not identify two symbols s_i and t_j even in the case the letters s_i and t_j are the same letter in A.) The edge set E is a double copy of $K_{|s|,|t|}$: for any pair of vertices s_i and t_j we link them with one *blue/dark-gray edge of weight* $\omega(s_i, t_j)$ and one *red/light-gray edge of weight* $\omega(s_i, \overline{t_j})$. (In fact, edges with weight $-\infty$ will not be used for our purposes and could be deleted.) An edge with weight that is not $-\infty$ is called a *direct match* if it is blue and it is called *inverted match* if it is red. In Figure 1 we have an example of a matching graph. In this figure, as in others, we do not draw edges with weight $-\infty$.

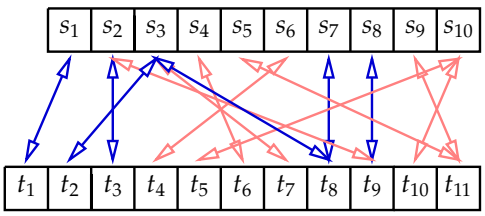

Fig. 1. Example of a matching graph

Given $u \in V^*$ a nonempty factor of $s_1 s_2 \cdots s_k$ and $v \in V^*$ a nonempty factor of $t_1 t_2 \cdots t_{k'}$, we call $B = (u, v)$ a *block*. Given a block $B = (u, v)$, there exist integers $1 \leq i' \leq i \leq |s|$ and $1 \leq$

$j' \leq j \leq |t|$ such that $u = s[i' \,..\, i]$ and $t = t[j' \,..\, j]$. Hence, we can associate $([i' \,..\, i] \times [j' \,..\, j])$, the *rectangle associated with the block* B.

Consider an edge that links s_i to t_j and an edge that links $s_{i'}$ to $t_{j'}$. We say that they *cross* each other if $(i - i')(j - j') < 0$, we say that they *touch* each other if $(i - i')(j - j') = 0$ and we say that they are *parallel* if $(i - i')(j - j') > 0$. Let $M \subseteq E$ be any set of edges in a matching graph. Recall that M is called a *matching* if any two edges of M do not touch each other. Moreover, M is called a *direct matching* if it has only direct matches and any two of them are parallel. Furthermore, M is called an *inverted matching* if it has only inverted matches and any two of them cross each other. The *restriction of* M *to a block* $B = (s[i' \,..\, i], t[j' \,..\, j])$ is the submatching of all edges of M with vertices in $s[i' \,..\, i]$ and $t[j' \,..\, j]$). Finally, M is called a *blockwise inverted matching*, or simply a *bimatching*, if, considering words in V^*,

- there is $l \geq 1$;
- there are l blocks $B_i = (u_i, v_i)$ such that $s = s_1 s_2 \dots s_k = u_1 u_2 \dots u_l$ and $t = t_1 t_2 \dots t_{k'} = v_1 v_2 \dots v_l$;
- for any $i \in \{1, \dots, l\}$, the restriction M_i of M to the block B_i is either a direct matching or an inverted matching;
- $M = \cup_{i=1}^{l} M_i$.

Given a bimatching M, the number of blocks l is not unique since any direct submatching M_i can also be split as a union of smaller direct submatchings. However, the number of blocks such that the corresponding restriction M_i is an inverted matching does not change. This is the *number of inversions of* M, $\iota(M)$. Given a bimatching M, the smallest possible value for l is called the *number of blocks of* M.

One can notice that in Figure 2 we have a bimatching M with four blocks. There are $\iota(M) = 2$ maximal inverted submatchings.

3 The Algorithm for Optimal Bimatching

Given a bimatching M, one can easily deduce an alignment with non-overlapping inversions associated with it. One could naturally define the weight of a matching in a matching graph to be the sum of the weights of its edges. However, it is quite common for alignments

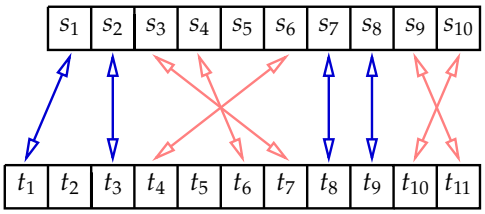

Fig. 2. Example of a bimatching with four blocks

the establishment of penalties for biological events like mutations, insertions or deletions. In order to be more general, we attribute a *inversion penalty* $I \geq 0$ for every inverted submatching M_i. Hence, we define the *weight of a bimatching* M as

$$\omega(M) = \sum_{e \in M} \omega(e) - \iota(M)I.$$

Hence, the following problem is the optimization problem we are interested in solving.

Problem 1. Given a matching graph $G = G(s, t, \omega, -)$, we want to compute the maximal weight $\omega(M)$ for all possible bimatchings M. As usual, we may also be interested in a bimatching of maximal weight.

Such a bimatching M^* of maximal weight is called an *optimal bimatching* of s and t and its weight $\omega(M^*)$ is denoted by $\mathrm{BIM}(s, t)$. The weight of an *optimal direct matching* is denoted by $\mathrm{DM}(s, t)$.

Next lemma leads us to the dynamic programming algorithm given by Algorithm 1.

Lemma 1. *Let A be an alphabet, let $a, b \in A$ be any letters, let $u, v \in A^*$ be any words on A, let us give an inversion operation on $-$ on A^* and a weight function ω and let I be an inversion penalty. Hence, the following affirmatives are true:*

1. $\mathrm{BIM}(1, v) = \mathrm{BIM}(u, 1) = 0$;

2. $\mathrm{BIM}(ua, vb) = \max \left\{ \begin{array}{l} \mathrm{BIM}(u, v) + \omega(a, b) \\ \mathrm{BIM}(ua, v) \\ \mathrm{BIM}(u, vb) \\ B - I \end{array} \right\}$

where $B = \max\{\mathrm{BIM}(u_1, v_1) + \mathrm{DM}(u_2, \overline{v_2}) \mid ua = u_1u_2, vb = v_1v_2, u_2v_2 \neq 1\}$.

(Complete proofs are omitted in this paper due to space constraints. The three first cases in the computation of $\mathrm{BIM}(ua, vb)$ in last lemma correspond to the usual dynamic programming analysis where no inversions are considered. The case $B-I$ consider the cases where the restrictions of the bimatchings to the blocks (u_2, v_2) is an inverted matching.)

Algorithm 1 A $O(n^4)$-time and $O(n^2)$-space algorithm for BIM

```
BIM(s, t)
 1  ▷ Compute the table B[i, j] = BIM(s[1..i], t[1..j])
 2  Let B[i, j] be 0 for i = 0 or j = 0
 3  for i from 1 to |s| do
 4      for j from |t| downto 1 do
 5          B[i, j] ← −∞
 6      for j from 1 to |t| do
 7          B[i, j] ← max(B[i, j], B[i − 1, j], B[i, j − 1])
 8          B[i, j] ← max(B[i, j], B[i − 1, j − 1] + ω(s[i], t[j]))
 9          ▷ Compute L[i', j'] = DM(s[i'..i], t[j..j']) and set B[i, j']
10          Let L[i', j'] be 0 for i' = i + 1 or j' = j − 1
11          for j' from j to |t| do
12              for i' from i downto 1 do
13                  L[i', j'] ← max(L[i', j' − 1], L[i' + 1, j'])
14                  L[i', j'] ← max(L[i' + 1, j' − 1] + ω(s[i'], t[j']), L[i', j'])
15                  B[i, j'] ← max(B[i, j'], L[i', j'] + B[i' − 1, j − 1] − I)
16  return B
```

Algorithm 1 computes $\mathrm{BIM}(s, t)$ the maximal weight of a bimatching with time complexity $|s|^2|t|^2$ and space complexity $|s||t|$. As usual, by tracking any maximality choice, one can obtain also an optimal bimatching. The numbers present in positions (s_i, t_j) of the incidence matrix of Figure 3(a) show the corresponding values $B[i, j]$ computed for most important cells in Algorithm 1 if all matches have weight 1. Hence, one can obtain the bimatching of Figure 2 as the optimal bimatching for the matching graph of Figure 1.

Figure 3(b) shows an example of possible data that have been applied to the algorithm BIM. The DNA sequences correspond to two syntenic regions from two bacteria. These sequences were splitten in fragments of length 100 in order to form the alphabet. A match between two fragments is assumed if the alignment score is above an adequately chosen threshold.

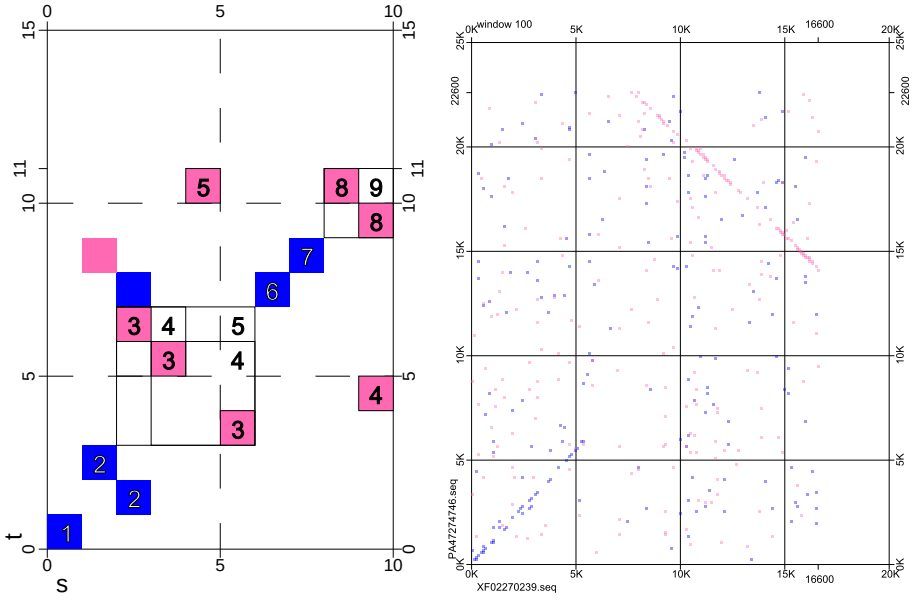

Fig. 3. (a) Running of Algorithm 1, (b) Real data for BIM

4 Conclusion

We gave a new algorithm for the alignment with non-overlapping inversions problem, improving the time complexity of an exact solution from $O(n^6)$ to $O(n^4)$ in Algorithm 1. We can also extend the results presented here. We can give a sparse dynamic programming implementation that gives the exact solution and can speed up even further if we have $o(n^2)$ matches. This is quite common for applications with large alphabets. We can also modify the algorithm in order

to deal with gap penalties or local alignments with non-overlapping inversions.

Let $\mathrm{LCS}(u, v)$ denote the length of the longest common subsequence of u and v. Motivated by these algorithms, one of the authors recently [10] proposed the following open problem: given two words s and t of length n, one can pre-process both words in such a way that any query $\mathrm{LCS}(u, v)$ can be answered in time $O(1)$, for u a factor of s and v a factor of t. This pre-processing can be done in $O(n^4)$. Can we do it in $O(n^2)$? In $O(n^3)$? Although a little stronger, one can think of a version with $\mathrm{DM}(u, v)$ queries instead of $\mathrm{LCS}(u, v)$ queries.

As far as we know, alignment with non-restricted inversions is open.

Acknowledgments

We would like to thank Joachim Messing and Marie-France Sagot for their comments. This work was partially supported by FAPESP # 2000/08039-3, Alfred P. Sloan Foundation # 99-10-8, NSF DBI 99-82983, NSF # 9975618, and Rutgers SROA #2-7472.

References

1. Wagner, R.: On the complexity of the extended string-to-string correction problem. In: Seventh ACM Symposium on the Theory of Computation, Association for Computing Machinery (1975)
2. Schöniger, M., Waterman, M.S.: A local algorithm for DNA sequence alignment with inversions. Bulletin of Mathematical Biology **54** (1992) 521–536
3. Kececioglu, J., Sankoff, D.: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica **13** (1995) 180–210
4. Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discrete Math. **12** (1999) 91–110 (electronic)
5. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J. ACM **46** (1999) 1–27
6. El-Mabrouk, N.: Sorting signed permutations by reversals and insertions/deletions of contiguous segments. J. Discrete Algorithms **1** (2000) 105–121
7. El-Mabrouk, N.: Reconstructing an ancestral genome using minimum segments duplications and reversals. J. Comput. System Sci. **65** (2002) 442–464 Special issue on computational biology 2002.
8. Waterman, Eggert: A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. Journal of Molecular Biology **197** (1987) 723–728

9. Muchnik, I., do Lago, A.P., Llaca, V., Linton, E., Kulikowski, C.A., Messing, J.: Assignment-like optimization on bipartite graphs with ordered nodes as an approach to the analysis of comparative genomic data. DIMACS Workshop on Whole Genome Comparison (2001) http://dimacs.rutgers.edu/Workshops/-WholeGenome/.
10. do Lago, A.P.: A sparse dynamic programming algorithm for alignment with inversions. Workshop on Combinatorics, Algorithms, and Applications (2003)