On Sequence Overlaps Minimizing Post-normalized Edit Distance

Petr RYŠAVÝ[1]

[1]Dept. of Computer Science, Czech Technical University, Technická 2, 166 27 Praha, Czech Republic

petr.rysavy@fel.cvut.cz

Abstract. *Computational biology deals with processing of genetic sequences. Next-generation sequencing machines produce sets of short substrings of a DNA sequence, called reads, which are further assembled to contigs. Recently there has been proposed a method how to measure the distance between two sequences only from an assembly at the level of contigs [8]. The original paper used a heuristic for finding a suitable overlap of two contigs. The heuristic was, however, not evaluated on its own in the original paper. Here, we address this issue, and our results show, that the relative error of the heuristic is small and that the heuristic provides the exact result in more than two-thirds of the cases.*

Keywords

edit distance, sequence overlap, genetic sequences

1. Introduction

The bioinformatics field deals with processing of genomic sequences. One of the most popular tasks is to estimate similarity of two organisms from their DNA sequences. The similarity can be calculated using a conventional distance measures as the *Levenshtein distance* [5] (sometimes called the *edit distance*), which counts the minimum number of changes that are needed to transform one sequence to the other.

The Levenshtein distance can be used to build *phylogenetic trees*. A phylogenetic tree is a rooted tree, where the root represents a hypothetical common ancestor of several organisms, which are located in leaves of the tree. The inner nodes of the tree show the evolutionary events that occurred in the past and they correspond to a species splitting into two new species. The tree can be built from a distance matrix between the sequences using, for example, the neighbor-joining algorithm [9] or UPGMA [11].

The problem with the current sequencing technologies is that they are not able to read the whole sequence at once. Instead, only short fragments, called *reads*, are sequenced. Their length is usually from tens to hundreds of symbols. The group of methods that rely on short reads and highly parallel sequencing is called the *next-generation sequencing* (NGS) [1].

The rest of the work needs to be done in silico. The reads are assembled by an assembly algorithm (for example [10, 2, 13]). In an ideal case, the assembly algorithm produces a single putative sequence that matches the true genome of the sequenced organism.

In the real world, this is usually not the case. The assembly algorithms often fail to produce a single sequence, but several shorter *contigs* are produced guided by prefix-suffix overlaps of reads. Without any additional wet-lab sequencing, contigs cannot be further ordered or joined based on short reads. The assembly process often fails due to the nature of genetic data. First, there may be a region in the original sequence, which is not contained in any reads. Assembly algorithm then cannot decide the missing part, nor the order of contigs. Second, there are often repeating blocks in DNA. If those repeats are longer than the read length, the algorithm does not know how to order contigs that are between repeats. Third, the sequence assembly problem is known to be NP-hard, and therefore, heuristic approaches are commonly used.

In our recent publication [8], we have published a method how to build a phylogenetic tree from contig data, i.e., only from partially assembled sequences. Part of this publication deals with finding an overlap of two contigs, which minimizes so-called *post-normalized edit distance*. We based our algorithm in [8] on a quadratic heuristic; however, the original paper did not contain any evaluation or guarantees on the quality of the overlap. In this paper, we present an exact algorithm, which runs in cubic time and provides an optimal solution. We use this algorithm to show that on contigs produced by an assembly algorithm, the heuristic from [8] finds the optimum in more than two-thirds of cases.

2. Related Work

The *Levenshtein distance* [5] counts the minimum number of changes that are needed to transform one sequence to the other. The formal definition is as follows.

Definition 2.1 (Levenshtein distance) *Let A and B be two sequences. Then the **Levenshtein distance** of A and B (denoted $\mathrm{dist}(A, B)$) is the minimum number of single symbol edit operations* insert*,* delete *and* substitute *needed to transform A to B.*

Often we require a distance measure to be normalized to the interval $[0, 1]$. There are several ways how to normalize the Levenshtein distance. The normalization should be done so that the final measure is still a metric, as proposed in [6]. In that paper, the Levenshtein distance is defined as a minimum of $W(A, B)/L(A, B)$, where $W(A, B)$ is the cost for an alignment of A and B, and $L(A, B)$ is the length of this alignment. However, for our purposes, we deal only with the *post-normalized Levenshtein distance*, which is not a metric and is defined as follows.

Definition 2.2 (Post-normalized Levenshtein distance) *Let A and B be two sequences. The **post-normalized Levenshtein distance** of A and B is*

$$\overline{\mathrm{dist}}(A, B) = \frac{\mathrm{dist}(A, B)}{\max\{|A|, |B|\}}. \tag{1}$$

The normalization in (1) is based on the following property of the Levenshtein distance

$$\mathrm{dist}(A, B) \leq \max\{|A|, |B|\}. \tag{2}$$

If we consider overlap of two sequences, we would like their distance to be as small as possible, and at the same time, those two overlapping subsequences should be as long as possible. Those two requirements go one against the other and can be formalized as minimization of (1). An overlap of two sequences A and B can be formalized as a search for one of the following:

- a prefix of A (or B respectively), which is a suffix of B (A), or
- a substring of A (or B), which matches B (A).

In the first case, the sequences overlap at their ends, B preceding A (or A preceding B). In the second case, one sequence is a subsequence of the other. We would like to find from all possible overlaps the one minimizing (1). Formally as follows.

Definition 2.3 *Let $S(A, B)$ be the set of all possible overlaps of A and B as stated above. Find*

$$\mathsf{overlap}(A, B) = \operatorname*{arg\,min}_{(a,b)\in S(A,B)} \overline{\mathrm{dist}}(a, b). \tag{3}$$

In paper [8], we proposed a heuristic that approximates (3) in quadratic time. The paper however missed any evaluation how well this heuristic performs, which we will evaluate in the following sections. The heuristic is presented in Alg. 1. The heuristic is based on a straightforward modification of the standard Wagner-Fischer dynamic programming algorithm [12]. Besides array `dist`, which registers the Levenshtein distance of the prefixes $\mathrm{dist}(i, j)$, we store two arrays `lenA` and `lenB` that register lengths of those prefixes. An overlap corresponds one of the following alignments in the table:

- from the top to the bottom or from the left to the right, which corresponds to the option when a sequence is a substring of the other;
- from the top to the right or from the left to the bottom, which is the case of a suffix matching a prefix.

Those two options mean that the top row and the left column need to be initialized to zero. The minimum can be found either in the bottom row or the right column. To avoid short random overlaps (the size of the alphabet is 4), a threshold of 20 is applied to the length of the overlaps. The heuristic runs in quadratic time, same as the Wagner-Fischer algorithm, which it is based on.

3. Exact Algorithm

In this section, we provide an exact algorithm to the heuristic in Alg. 1. This algorithm runs in cubic time and is guaranteed to provide the optimal solution to (3). This algorithm was not part of [8]. Due to the high polynomial, it cannot be used in real-world applications; however, it can show that for real-world data, Alg. 1 is a good heuristic.

A naive implementation would run in $\mathcal{O}(|A|^2|B|^2)$. We would simply take all prefixes of one sequence, calculate their distance to all suffixes of the second sequence, and then repeat the whole procedure with the opposite roles of the sequences. This means trying $|A|$ prefixes and $|B|$ suffixes, where each possibility means evaluating a table of size up to $|A| \cdot |B|$.

We can notice that the algorithm does much redundant work. Consider prefix $b_1 b_2 \cdots b_k$ matching a suffix of A. If we calculate the distance of prefix $b_1 b_2 \cdots b_k b_{k+1}$ to the same suffix of A, the table looks the same but for the last row. This inefficiency may help us to reduce the amount of work as we can save a linear factor by reusing these results. In other words, we enforce a prefix of the second sequence to overlap with the whole suffix of the first sequence. Then we do the same procedure with the reversed role of the first and the second sequence. By this way, we avoid the $\mathcal{O}(|A|^2|B|^2)$ complexity and obtain a cubic algorithm. Pseudocode for the algorithm is given in Alg. 2.

Algorithm 1 Pseudocode for the heuristic used for finding $\mathsf{overlap}(A, B)$ [8].

function overlap(A,B)
  $dist, lenA, lenB \leftarrow$ 2D arrays of zeros of size $(|A|+1) \times (|B|+1)$
  for $i \in \{1, 2, \ldots, |A|\}$ **do** ▷ for each row
    for $j \in \{1, 2, \ldots, |B|\}$ **do** ▷ for each column
      choose the option that leads to the lowest $\frac{d}{\max\{lA, lB\}}$:
      gap in A**:** $d \leftarrow dist[i, j-1] + 1; lA \leftarrow lenA[i, j-1]; lB \leftarrow lenB[i, j-1] + 1$
      gap in B**:** $d \leftarrow dist[i-1, j] + 1; lA \leftarrow lenA[i-1, j] + 1; lB \leftarrow lenB[i-1, j]$
      (mis)match: $d \leftarrow dist[i-1, j-1] + [\![A[i-1] \neq B[j-1]]\!]$;
        $lA \leftarrow lenA[i-1, j-1] + 1; lB \leftarrow lenB[i-1, j-1] + 1$
      $dist[i, j] \leftarrow d, lenA[i, j] \leftarrow lA, lenB[i, j] \leftarrow lB$
    end for
    CHECKOPTIMUM($i, |B|$)
  end for
  for $j \in \{1, 2, \ldots, |B|-1\}$ **do** CHECKOPTIMUM($|A|, j$) **end for**
end function

function CHECKOPTIMUM(i, j)
  if $\frac{dist[i,j]}{\max\{lenA[i,j], lenB[i,j]\}}$ is the smallest & $\min\{lenA[i, j], lenB[i, j]\} \geq 20$ **then**
    the new optimum is located at (i, j), store it
  end if
end function

Algorithm 2 Exact algorithm for the heuristic in Alg. 1.

function overlap(A, B)
  for $i \in \{0, 1, \ldots, |B|-20\}$ **do** overlap(A, B, i) **end for** ▷ for each suffix of A
  for $i \in \{0, 1, \ldots, |B|-20\}$ **do** overlap(B, A, i) **end for** ▷ for each suffix of B
end function

function overlap($A, B,$ *aoffset*)
  $dist \leftarrow$ 2D array of zeros
  for $j \in \{1, 2, \ldots, |B|\}$ **do** $dist[$*aoffset*$][j] = j$ **end for** ▷ initialize the first row
  for $i \in \{$*aoffset*$,$ *aoffset* $+ 1, \ldots, |A|-1\}$ **do** ▷ fill the table as in Wagner-Fischer
    $dist[i+1][0] = dist[i][0]$
    for $j \in \{0, 1, \ldots, |B|\}$ **do**
      $dist[i+1][j+1] = \min\{dist[i+1][j] + 1, dist[i][j+1] + 1, dist[i][j] + [\![A[i] \neq B[j]]\!]\}$
    end for
    if $i -$ *aoffset* $+ 1 \geq 20$ and $\frac{dist[i+1][|B|]}{\max\{i-aoffset+1, |B|\}}$ is the smallest **then**
      the new optimum is at $i+1, |B|$ ▷ check the last column for an optimum
    end if
  end for
  for $j \in \{20, 21, \ldots, |B|-1\}$ **do**
    if $\frac{dist[|A|][j]}{\max\{|A|-aOffset, j\}}$ is the smallest **then**
      the new optimum is at $|A|, j$ ▷ check the last row for an optimum
    end if
  end for
end function

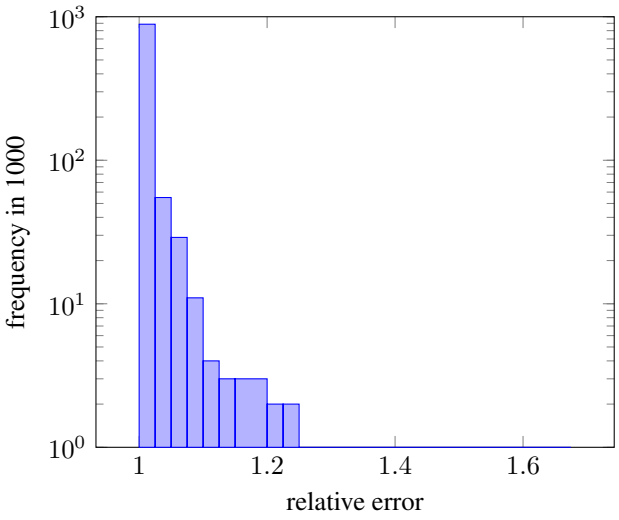

Fig. 1. A histogram of the relative error of Alg. 1 to its exact version in Alg. 2. The plot shows the histogram of the relative error on 1000 sequences.

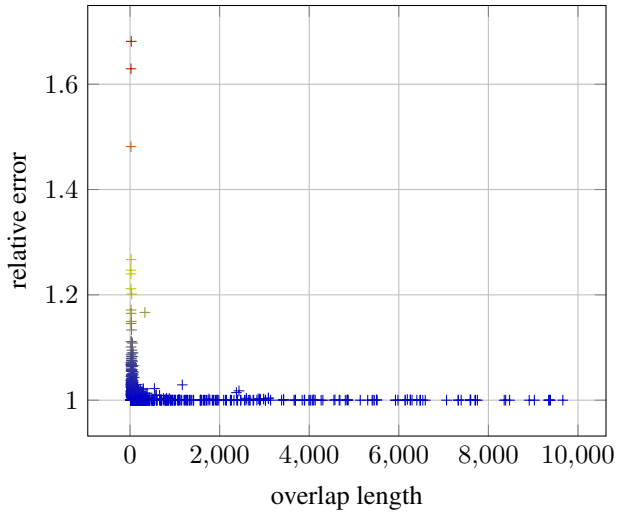

Fig. 2. Relative error of Alg. 1 to its exact version in Alg. 2 on the length of the overlap.

4. Experimental Evaluation

In Alg. 1, we presented a heuristic for finding the overlap of two contigs such that the post-normalized Levenshtein distance from (1) is minimized. We did not provide any guarantees for the quality of the result, we stated that the heuristic only approximates the optimum. In Alg. 2 we presented an algorithm that provides the exact result, however not fast enough.

In this section, we compare Algs. 1 and 2 to see that on the real data the heuristic provides results that are usable in the real-world on contigs produced by an assembly algorithm. The contigs are results of an assembly of 81 Hepatitis A segments. All of the sequences were downloaded from the ENA repository [4]. Then we used the ART [3] program to simulate sequencing with coverage $\alpha = 10$ (the average number how often each nucleotide is read) and read length $l = 70$. The reads were assembled using the SPADES [7] assembly algorithm to produce contigs. From those contigs, we selected randomly one thousand pairs and calculated the value of (1) for both algorithms. We calculated the ratio to see how big the relative error is. The histogram of the results can be seen in Fig. 1.

From the figure, we see that the large errors are relatively sparse. In 698 cases out of 1000, the heuristic in Alg. 1 provided the same value of the optimized criterion (3) as the exact algorithm in Alg. 2. The maximum relative error is 1.68, which means that the heuristic found an overlap which was 68 % from the optimum in the worst case. This error was reached on an overlap shorter than 40 nucleotides. The extreme values are, however, quite rare. The fifth largest relative error is less than 25 %. Figure 2 indicates that large errors are more common for short overlaps close to the overlap threshold of 20 and with longer sequences, their frequency decreases.

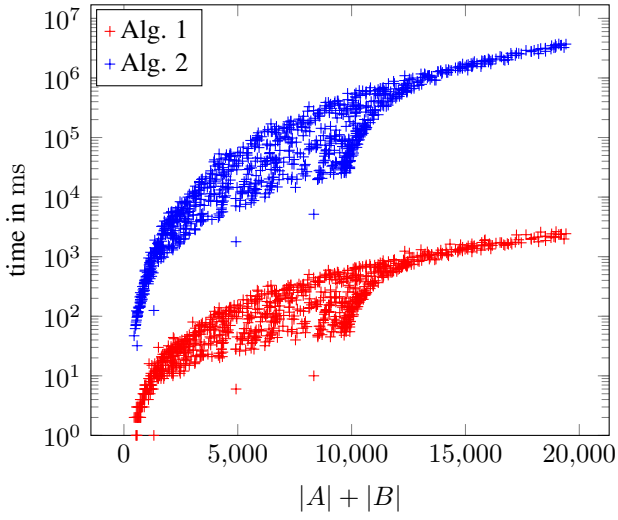

Fig. 3. Runtime of Algs. 1 and 2 on the sum of the sequence lengths.

Figure 3 shows a dependence of the runtime on the sum of sequence lengths and indicates that our theoretical evaluation in the previous section is correct. The logarithmic axis of the plot shows that both algorithms run in a polynomial time with different exponents, which are 2 in the case of Alg. 1 and 3 in the case of Alg. 2.

5. Conclusion

We have evaluated a method for finding an overlap of two sequences that minimizes the post-normalized Levenshtein distance. The heuristic approach was published in [8], and in this paper, we presented an exact algorithm for solving the problem. The original heuristic from [8] was not evaluated by experiments on its own, and this gap was filled in this paper.

There is however further work that needs to be done. The heuristic itself runs in a time comparable to the classical dynamic programming Wagner-Fischer algorithm, which is commonly used for evaluation of the Levenshtein distance. Despite its simplicity, sometimes the quadratic runtime may be a limiting factor, especially when bioinformaticians deal with long sequences. Therefore there is a need for a linear heuristic.

Acknowledgments

The research described in the paper was supervised by Prof. F. Železný, FEE CTU in Prague. This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS17/189/OHK3/3T/13. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

References

[1] Sara Goodwin, John Mcpherson and W. Richard Mccombie. Coming of age: Ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17: 333–351, 05 2016. doi: 10.1038/nrg.2016.49.

[2] David Hernandez, Patrice Franois, Laurent Farinelli, Magne sters and Jacques Schrenzel. De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*, 18(5):802–809, 2008. doi: 10.1101/gr.072033.107.

[3] Weichun Huang, Leping Li, Jason R. Myers and Gabor T. Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012. doi: 10.1093/bioinformatics/btr708.

[4] Rasko Leinonen, Ruth Akhtar, Ewan Birney, Lawrence Bower, Ana Cerdeno-Trraga et al. The European Nucleotide Archive. *Nucleic Acids Research*, 39(suppl_1): D28–D31, 2011. doi: 10.1093/nar/gkq967.

[5] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707, 1966.

[6] Andres Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE transactions on pattern analysis and machine intelligence*, 15 (9):926–932, Sep 1993. doi: 10.1109/34.232078.

[7] Sergey Nurk, Anton Bankevich et al. Assembling genomes and mini-metagenomes from highly chimeric reads. In Minghua Deng, Rui Jiang, Fengzhu Sun and Xuegong Zhang, editors, *Research in Computational Molecular Biology: 17th Annual International Conference, RECOMB 2013, Beijing, China, April 7-10, 2013. Proceedings*, pages 158–170. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-37195-0_13.

[8] Petr Ryšavý and Filip Železný. Estimating sequence similarity from contig sets. In Niall Adams, Allan Tucker and David Weston, editors, *Advances in Intelligent Data Analysis XVI: 16th International Symposium, IDA 2017, London, UK, October 26–28, 2017, Proceedings*, pages 272–283, Cham, 2017. Springer International Publishing. doi: 10.1007/978-3-319-68765-0_23.

[9] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4): 406–425, 1987. doi: 10.1093/oxfordjournals.molbev.a040454.

[10] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones and nan Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009. doi: 10.1101/gr.089532.108.

[11] Robert R. Sokal and Charles D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[12] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, January 1974. doi: 10.1145/321796.321811.

[13] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008. doi: 10.1101/gr.074492.107.