Estimating Sequence Similarity from Contig Sets

Petr Ryšavý and Filip Železný

Dept. of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague, Czech Republic
{petr.rysavy,zelezny}@fel.cvut.cz

Abstract. A key task in computational biology is to determine mutual similarity of two genomic sequences. Current bio-technologies are usually not able to determine the full sequential content of a genome from biological material, and rather produce a set of large substrings (*contigs*) whose order and relative mutual positions within the genome are unknown. Here we design a function estimating the sequential similarity (in terms of the inverse Levenshtein distance) of two genomes, given their respective contig-sets. Our approach consists of two steps, based respectively on an adaptation of the tractable Smith-Waterman local alignment algorithm, and a problem reduction to the weighted interval scheduling problem soluble efficiently with dynamic programming. In hierarchical-clustering experiments with Influenza and Hepatitis genomes, our approach outperforms the standard baseline where only the longest contigs are compared. For high-coverage settings, it also outperforms estimates produced by the recent method [8] that avoids contig construction completely.

1 Introduction

A key task in computational biology is to determine mutual similarity of two genomic sequences. This is important for purposes such as database retrieval of similar genomes or for construction of phylogenetic trees by means of hierarchical genome clustering.

Current laboratory devices that identify the sequential content of genomes from biological material, so called *sequencers*, are not able to read sequences in their entirety (e.g. billions of nucleotides). Rather, they read small (e.g. tens or hundreds of nucleotides) subsequences, called *reads*, at random positions of the target sequence. The number of such reads is set high enough so that most of the nucleotides in the sequence are covered by multiple reads. Thus most of the reads overlap with some other reads.

Mutual read overlaps then allow guiding the in-silico reconstruction of the target sequence. For example, a possible assembly of reads $\{\mathsf{AGGC}, \mathsf{TGGA}, \mathsf{GCT}\}$ is $\mathsf{AGGCTGGA}$, and another possible assembly of the same reads is $\mathsf{GCTGGAGGC}$. These two solutions correspond to two different Hamiltonian paths in the *overlap graph* where vertices and directed edges represent reads and non-empty suffix-prefix overlaps, respectively. By an Occam-razor heuristic, the former (shorter)

assembly would be considered a more likely estimate of the target sequence. This motivates the generally accepted formulation of the assembly task: given a bag (multiset) of reads such that their overlap graph is connected, find the *shortest* superstring of all the reads.

A straightforward way to measure similarity of two sequences represented by their read bags would be to assemble each bag first and then compute the sequential similarity of the results. The problem, however, is that the shortest superstring problem is NP-hard. In previous work [8] we tackled this problem by proposing a similarity function computable directly on the input read sets thus avoiding the intractable assembly task.

Here we address a different problem, namely that popular assembly algorithms such as [11,2,7,13,14] are typically not able to produce a single putative sequence but rather yield a set of so called *contigs*. Contigs are maximal, mutually non-overlapping sequences which can be assembled from reads. Their ordering and mutual distance within the target sequence cannot be determined from the read bags alone. This has statistical reasons (the original sequence may not be completely covered by the reads, and the overlap graph may be disconnected) as well as more principal reasons. The latter are best illustrated by considering single-letter repeats longer than any read. Such repeat regions simply cannot be reconstructed, and their left and right neighborhoods need to be treated separately. There are however other sequential patterns more complex than single-letter repeats that also incur a break-down into contigs.

With additional requirements on laboratory equipment, the ordering of contigs and pairwise distances of neighboring contigs can be determined through a process called *scaffolding* (cf. Fig. 1). We consider this process unnecessary if the objective is just to estimate genome similarity, and enter the study with the hypothesis that the latter can be estimated directly from the contig sets.

On a superficial level, the problem considered here seems technically similar to the one we studied in [8]. Indeed, in both cases, we have two sets of substrings of the respective two sequences and want to determine the similarity of the latter two. However, contig sets do not satisfy some important properties of read sets (constant and short read length, roughly uniform distribution on the target sequence) and call for different methods. Briefly, the similarity (or, more precisely *dissimilarity* measure) in [8] was based on a one-to-one matching of reads between the respective read bags and an adaptation of the Monge-Elkan similarity principle. On the contrary, in the present study, we need to assume M-to-N matching between various-size contigs. Our present strategy consists of a tractable heuristic algorithm adapting the Smithe-Waterman local alignment algorithm, and a subsequent problem reduction to the weighted interval scheduling problem which is solved efficiently through dynamic programming.

In the next section, we develop the novel dissimilarity measure. In Sect. 3 we test it empirically in hierarchical clustering tasks with 12 influenza virus genomes and 81 hepatitis strains, in comparison to a range of alternative approaches. Section 4 concludes the paper.

2 Dissimilarity Measure

Let A be a string over an alphabet Σ. Here we consider $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. Then $\mathsf{dist}(A, B)$ denotes the *Levenshtein distance* [5] between strings A and B. The Levenshtein distance is one of the most common string distance measures that is used not only in bioinformatics. It measures the number of insertions, deletions and substitutions that are needed to convert one string into the other. It can be viewed as a simplified model that shows how many mutations had to be introduced during evolution. It holds that

$$\mathsf{dist}(A, B) \leq \max\{|A|, |B|\}, \tag{1}$$

where $|\cdot|$ denotes the length of A when A is a string. In our approach, we will use the post-normalized Levenshtein distance defined as

$$\overline{\mathsf{dist}}(A, B) = \frac{\mathsf{dist}(A, B)}{\max\{|A|, |B|\}}. \tag{2}$$

There are other approaches [6] to normalize the Levenshtein distance dealing with the fact that $\overline{\mathsf{dist}}$ does not satisfy the triangle inequality. However, here we adhere to (2).

A *read bag* R_A is a multiset of $|R_A|$ substrings of length $l \ll |A|$ sampled with replacement from a distribution on all $|A| - l + 1$ substrings of length l of A. *Coverage* α of R_A is defined as

$$\alpha = l\frac{|R_A|}{|A|}. \tag{3}$$

Informally, the coverage α indicates the average number of reads covering a particular position in A. In other words, coverage is a ratio between the amount of data produced by the sequencing machine and the true DNA content. We assume that all genomes are sequenced with the same coverage.

Assembly is the process of reconstruction of string A from R_A. For reasons explained earlier, the complete reconstruction is usually impossible, and a set of *contigs* C_A is obtained instead. Contigs are mutually non-overlapping approximate substrings of A that are longer than l. The adjective *approximate* means that contigs usually contain errors incurred by the assembly process. The relationships between the genome carrier DNA, reads, contigs and scaffolds is illustrated in Fig. 1.

Our goal is to propose a dissimilarity measure $\mathsf{Dist}(C_A, C_B)$ that approximates $\mathsf{dist}(A, B)$. The plan is briefly as follows. First, given any pair ($a \in C_A, b \in C_B$) we estimate their most likely overlap in the unknown optimal alignment of A and B, assuming they do overlap. Note that here the term 'overlap' is with respect to the mutual positioning of a and b in the *alignment* of A and B, so the parts of a and b deemed to overlap may, in general, include insertions and mismatches. Second, the $|C_A||C_B|$ estimates resulting from the latter for all pairs of contigs are filtered using further (heuristic) constraints imposed on M-to-N

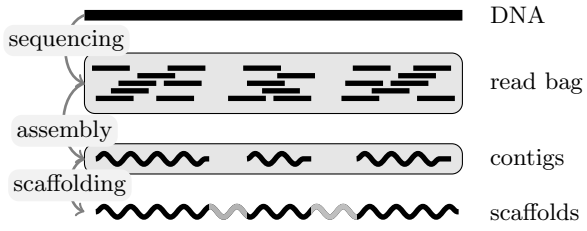

Fig. 1. Types of strings that are connected with the sequencing and assembly process.

contig matching; here we mainly want to filter out those pairs of contigs which likely *do not* overlap in the optimal alignment of A and B. Lastly, the resulting set of hypothesized contig overlaps is used to estimate the distance between A and B. These steps are described in the following three subsections, respectively.

2.1 Estimating overlaps for contig pairs

Consider two contigs $a \in C_A$ and $b \in C_B$. In the optimal alignment of A and B, there are several options how a and b can be positioned with respect to each other, assuming they have an overlap. Let $\mathsf{P}(a)$ ($\mathsf{S}(a)$) be the set of all prefixes (suffixes) longer[1] than 20 of string a, and let $c \in a$ denote that c is a substring of a. The first option is that a suffix of one contig overlaps with a prefix of the other contig, which leads to the following set of overlap candidates: $U(a,b) = \{\, (c,d) \mid c \in \mathsf{S}(a), d \in \mathsf{P}(b) \,\}$. The second option is that one contig matches a substring of the other contig, which gives $V(a,b) = \{\, (a,d) \mid d \in b \,\}$. The set of candidate pairs is then the symmetric closure of the union of both options: $S(a,b) = U(a,b) \cup U(b,a) \cup V(a,b) \cup V(b,a)$. The following function then yields our estimate of the most likely mutual overlap of a and b, assuming that the latter overlap at all.

$$\mathsf{overlap}(a,b) = \underset{(c,d) \in S(a,b)}{\mathsf{arg\,min}} \; \overline{\mathsf{dist}}(c,d). \tag{4}$$

To calculate $\mathsf{overlap}(a,b)$ we modify the Smith-Waterman local alignment algorithm [12]. For this purpose we maintain an array `dist` storing the Levenshtein distance of suffixes of prefixes of a and b. Two other arrays `lenA` and `lenB` store lengths of matching substrings. When filling each cell we decide the value based on (2). The complete algorithm that we use is described in Alg. 1. It is a heuristic algorithm; we have not been able to design an exact algorithm with time-complexity of $\mathcal{O}(|a||b|)$ solving (4).[2] An execution of the algorithm is exemplified in Tab. 1.

[1] The threshold of 20 nucleotides is set to prevent very short random overlaps.

[2] While the problem is polynomial, the time complexity of the brute-force solution incurs a polynomial of order 4 rendering it unusable even on small datasets.

Algorithm 1 Pseudocode for the heuristic used for finding $\mathsf{overlap}(a, b)$

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 option that leads to 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

A technical remark is in order. Prior to executing the above algorithm, we need to pre-process the contig sets for reasons irrelevant to the algorithmic principles described. In particular, because contigs represent DNA, we do not know which strand they come from. The two strands of DNA contain the same sequence; however, pairs of symbols A and T and C and G are switched. The complementary strands are being read from the opposite ends, and this direction is fixed due to the chemical structure of DNA molecule. Therefore if we calculate overlap of a and b, we do not know whether to match a and b or a with reversed complement[3] of b. To deal with this, we simply expand one (but not both, for obvious reasons) of the two contig sets by the reversed complements of all its elements.

2.2 Estimating overlaps for contig sets

The procedure from the previous section can be used to yield the most likely overlap for each possible pair of contigs $a \in C_A$, $b \in C_B$. Of course, not all such $|C_A||C_B|$ pairs actually overlap in the unknown optimal alignment of A and B. To filter the overlap candidates towards a smaller, more plausible set, we adhere to the following rules. (1) For a given $a \in C_A$, we should only pick elements from $\{\ \mathsf{overlap}(a, b) \mid b \in C_B\ \}$ which do not overlap between themselves, (2) the resulting overlap pairs should minimize the sum of the $\overline{\mathsf{dist}}$ values.

[3] For example for string ACCGGATT its reversed complement is AATCCGGT.

dist matrix

		A	A	G	C
	0	1	2	3	4
0	0	0	0	0	0
C 1	0	1	1	1	0
A 2	0	0	1	2	1
T 3	0	1	1	2	2
G 4	0	2	2	**1**	2

lenA matrix

		A	A	G	C
	0	1	2	3	4
0	0	0	0	0	0
C 1	0	1	1	1	1
A 2	0	1	1	1	2
T 3	0	2	2	2	3
G 4	0	3	3	**3**	3

lenB matrix

		A	A	G	C
	0	1	2	3	4
0	0	0	0	0	0
C 1	0	0	0	0	1
A 2	0	1	2	3	1
T 3	0	1	2	3	1
G 4	0	1	2	**3**	4

Table 1. An illustration of Alg. 1 showing the three involved data matrices for inputs $a = \mathsf{AAGC}$, $b = \mathsf{CATG}$. The entries yielding the minimal value of the criterion are marked in boldface, producing $(\mathsf{ATG}, \mathsf{AAG})$ as the detected overlap.

Note that the two selection rules are only a heuristic. Not even rule (1) is dictated strictly by biological principles. Indeed, it may, in fact, happen that two contigs $b, b' \in C_B$ map to the same contig (or its substring) $a \in C_A$ in a way making b and b' overlap.[4]

The application of the two selection rules reduces to the *weighted interval scheduling* problem defined in [4]. In this problem, we have n tasks, each of a value v_t (for $t = 1, 2, \ldots, n$) and a starting and finishing time. Our goal is to select a subset of non-overlapping tasks that maximizes the sum of the selected task values.

Weighted interval scheduling can be solved in $\mathcal{O}(n \log n)$ time by a simple dynamic programming algorithm. We pass the tasks ordered by the finishing time, and for each task we have two options — to include it or not. If S_t is an optimal solution for all tasks up to a task t (in the sorted order), then S_{t+1} is the maximum of S_t and $v_{t+1} + S_{t'}$, where t' is the task with the largest finishing time, which is smaller than the starting time of $t + 1$.

In our case the starting and finishing times represent location of c in a. The value we assign to a pair (c, d) is given by

$$v_{(c,d)} = \frac{1}{\overline{\mathsf{dist}}(c, d)} = \frac{\max\{|c|, |d|\}}{\mathsf{dist}(c, d)}. \tag{5}$$

Note that this value can be computed since Alg. 1 has maintained for each potential match (c, d) the values of $\mathsf{dist}(c, d)$, $|c|$, $|d|$.

To sum up, the procedure described accepts $a \in C_A$ and C_B, and produces a set we denote $o(a, C_B)$ which is selected from the initial overlap candidates, i.e.

$$o(a, C_B) \subseteq \{\, \mathsf{overlap}(a, b) \mid b \in C_B \,\}.$$

Further, overloading the $\mathsf{overlap}$ functor for contig sets, we denote

$$\mathsf{overlap}(C_A, C_B) = \bigcup_{a \in C_A} o(a, C_B).$$

[4] Often long substrings, called repeats, occur multiple times in a DNA sequence. Assembly algorithms may identify a repeat as a single contig or as two contigs based on the number of mutations.

Note that the above function is not symmetric and this fact will be dealt with in the next section.

2.3 Combining the Results

Having the filtered set of suspected overlaps, we first define the pre-distance $d(C_A, C_B)$ of contig sets C_A, C_B as the sum of the distances associated with the individual overlaps

$$d(C_A, C_B) = \sum_{(c,d)\in \mathsf{overlap}(C_A,C_B)} \mathsf{dist}(c, d).$$

Since $\mathsf{overlap}(C_A, C_B)$ is not symmetric, neither is $d(C_A, C_B)$ and the final measure $\mathsf{Dist}(C_A, C_B)$ averages $d(C_A, C_B)$ and $d(C_B, C_A)$. Furthermore, it normalizes the scale.

$$\mathsf{Dist}(C_A, C_B) = Z\frac{d(C_A, C_B) + d(C_B, C_A)}{2}, \tag{6}$$

where Z is a normalizing factor

$$Z = \frac{l \max\{|R_A|, |R_B|\}}{\alpha \sum_{(c,d)\in \mathsf{overlap}(C_A,C_B)} \max\{|c|, |d|\}}$$

dividing by the maximum distance that all matching substrings can have and multiplying by the maximum distance that A and B can have. For the latter, it estimates of sequence lengths $|A|, |B|$ from (3).

3 Experimental Evaluation

Here we test the dissimilarity measure (6) on data in comparative experiments where the distance is used to infer phylogenetic trees through hierarchical clustering.

We compare methods for estimating the Levenshtein distance $\mathsf{dist}(A, B)$ for DNA sequences. The methods include (i) the reference distance $\mathsf{dist}(A, B)$ (ground truth), (ii) our newly proposed dissimilarity measure (6), (iii) measure $\mathsf{Dist}_{\mathsf{MESSGq}}$ taken from [9,8], and applied directly on read-sets without contig assembly (iv) distance of two longest contigs, which represents the 'standard' state of the art option, (v) a trivial baseline method estimating $\mathsf{dist}(A, B)$ as $\max\{|R_A|, |R_B|\}$ based on (1). The methods were implemented in Java maximizing shared code. For assembly, we used current official implementations of five common assembly algorithms, namely ABySS [11], Edena [2], SSAKE [13], SPAdes [7] and Velvet [14].

We measured

(i) the Pearson's correlation coefficient showing the similarity of the distance matrices produced by the respective methods to the true distance matrix,

(ii) the Fowlkes-Mallows index [1] that measures the similarity between trees produced based on distance estimates and the reference tree defined by $\mathsf{dist}(A, B)$. The Fowlkes-Mallows index shows how much two hierarchical clusterings differ in structure. Both hierarchical trees are first cut into k clusters for $k = 2, 3, \ldots, n-1$. Then clusterings are compared based on the number of common objects among each pair of clusters. In this way, we obtain a set of values B_k that shows distances of the trees at various levels.
(iii) assembly time (if applicable) and the distance matrix calculation time,
(iv) how many times was distance calculation successful (i.e. assembly produced at least one contig).

For hierarchical clustering, we used the neighbor-joining algorithm [10].

The testing data contain two datasets. The *influenza* dataset[5] contains 12 influenza virus genome sequences plus an outgroup sequence. The *hepatitis* dataset contains 81 hepatitis C strains from the ENA repository.

To generate read sets from the genomes, we employ two strategies. One is 'idealized', based on error-free sampling and consecutive contig generation, denoted as opt_θ below. The other one simulates closely real-life erroneous sequencing conducted by the Illumina technology, and is facilitated by the ART [3] program.

For the influenza dataset, we use the idealized option. For each position we calculated coverage, and we produced nine simulated assembly results opt_θ for $\theta = 10, 20, \ldots, 90$. To generate opt_θ we chose the highest number β such that at least θ percent of nucleotides are covered in β or more reads. The longest substrings of such nucleotides formed individual contigs. We sampled the influenza dataset for choices with a high range of coverage and read length[6]. For sampling reads from hepatitis dataset we used ART [3] program to simulate Illumina sequencing for $(\alpha, l) \in \{10, 30, 50\} \times \{30, 70, 100\}$.

The main experimental results are shown in Table 2, which shows the average results on both datasets. For averaging on influenza dataset we exclude three most extreme values of coverage and read length. Table 2 contains results only for the best assembly algorithm and the worst one. The columns of the table show the Pearson's correlation coefficient, run time and the Fowlkes-Mallows index for selected levels. Figure 4 shows plot of Pearson's correlation coefficient on coverage for influenza dataset. Figures 2 and 3 show the Fowlkes-Mallows index. Figure 5 shows the dependency of correlation on percent of nucleotides in simulated assembly, i.e. on parameter θ of opt_θ. For clarity, the figures do not plot all the methodological combinations; they are all included in the supplementary material.[7]

[5] AF389115, AF389119, AY260942, AY260945, AY260949, AY260955, CY011131, CY011135, CY011143, HE584750, J02147, K00423 and outgroup AM050555. The genomes are available at `http://www.ebi.ac.uk/ena/data/view/<accession>`.

[6] $(\alpha, l) \in \{0.1, 0.3, 0.5, 0.7, 1, 1.5, 2, 2.5, 3, 4, 5, 7, 10, 15, 20, 30, 40, 50, 70, 100\} \times \{3, 5, 10, 15, 20, 25, 30, 40, 50, 70, 100, 150, 200, 500\}$

[7] A sample implementation and supplementary material are available on `https://github.com/petrrysavy/ida2017`.

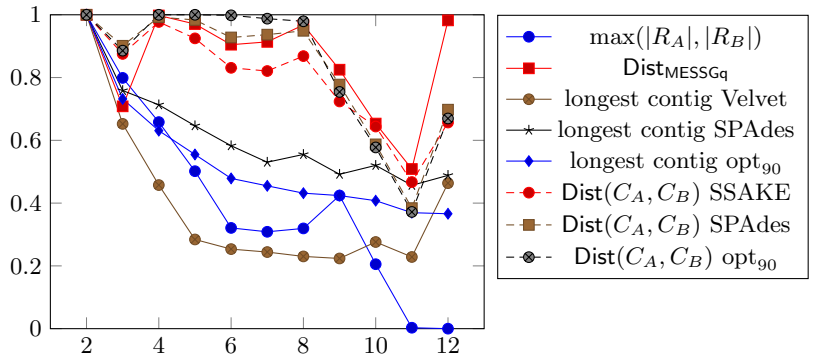

Fig. 2. Plot of average Fowlkes-Mallows index B_k versus k on influenza dataset. The index compares trees generated by the neighbor-joining algorithm. The tree is compared with the tree generated from the original sequences. If all values are equal to 1, the structures of the trees are the same.

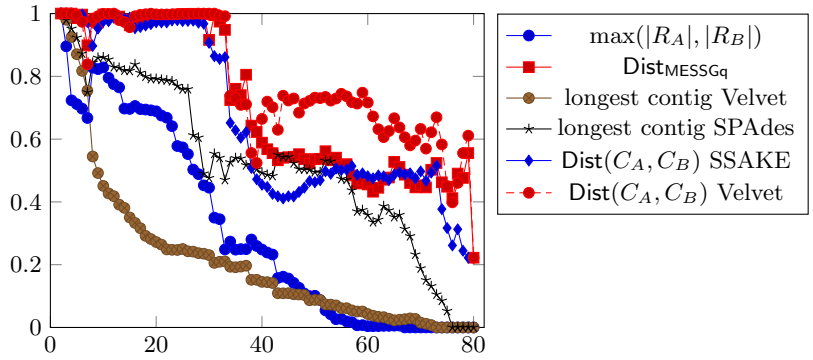

Fig. 3. Plot of average Fowlkes-Mallows index B_k versus k on hepatitis dataset. The index compares trees generated by the neighbor-joining algorithm.

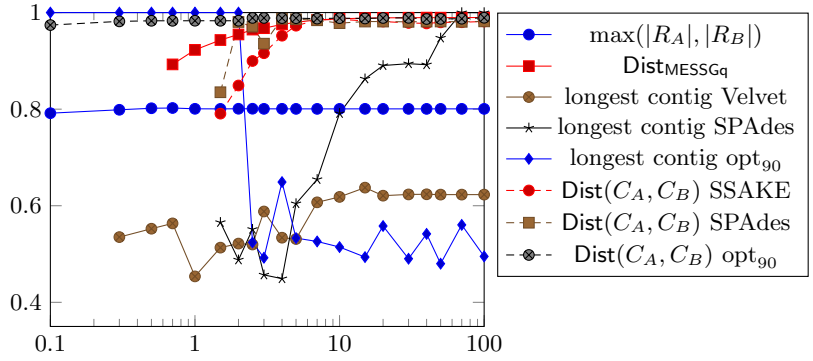

Fig. 4. Plot of average Pearson's correlation coefficient for several choices of coverage values.

Table 2. Average runtime, Pearson's correlation coefficient between distance matrices and Fowlkes-Mallows index for $k = 4$ and $k = 8$. The 'reference' method calculates distances from the original sequences. We show only assembly algorithms that gave the highest and the lowest correlation.

Dataset	method	finished	$\frac{\text{assem.}}{\text{ms}}$	$\frac{\text{distances}}{\text{ms}}$	corr.	B_4	B_8
Influenza	reference	112/112	0	2,518	1	1	1
	$\max(\|R_A\|, \|R_B\|)$	112/112	0	184	.801	.66	.32
	$\mathsf{Dist}_{\mathsf{MESSGq}}$	112/112	0	43,553	.966	1	.97
	longest contig Velvet	110/112	392	101	.569	.46	.23
	longest contig SPAdes	43/112	12,461	2,127	.751	.71	.56
	longest contig opt_{90}	112/112	0	1,208	.666	.63	.43
	$\mathsf{Dist}(C_A, C_B)$ SSAKE	67/112	2,115	17,483	.949	.98	.87
	$\mathsf{Dist}(C_A, C_B)$ SPAdes	43/112	12,461	20,968	.975	.99	.95
	$\mathsf{Dist}(C_A, C_B)$ opt_{90}	112/112	0	22,239	.987	1	.98
Hepatitis	reference	9/9	0	2,145,104	1	1	1
	$\max(\|R_A\|, \|R_B\|)$	9/9	0	7,738	.181	.72	.83
	$\mathsf{Dist}_{\mathsf{MESSGq}}$	9/9	0	701,726	.897	1	.98
	longest contig Velvet	9/9	22,860	3,447	.234	.93	.54
	longest contig SPAdes	9/9	103,683	1,872,233	.591	.95	.84
	$\mathsf{Dist}(C_A, C_B)$ SSAKE	9/9	96,446	29,465,436	.916	1	.9
	$\mathsf{Dist}(C_A, C_B)$ Velvet	9/9	22,860	28,186,784	.966	1	.98

From the results, we see that the proposed method $\mathsf{Dist}(C_A, C_B)$ gives results of quality comparable to $\mathsf{Dist}_{\mathsf{MESSGq}}$. For low coverage data $\mathsf{Dist}_{\mathsf{MESSGq}}$ is more successful because it does not need assembled contigs. On the opposite for high coverage data, $\mathsf{Dist}(C_A, C_B)$ produces results with higher correlation because it has more data available. Both methods are better than the baseline method $\max(|R_A|, |R_B|)$ and also than the simple approach that considers only the longest contig. Figure 5 indicates that the proposed method gives good estimates even if assembly identified only a fraction of the original sequence.

$\mathsf{Dist}_{\mathsf{MESSGq}}$ is faster than the proposed method. On the opposite $\mathsf{Dist}(C_A, C_B)$ is approximately $10\times$ slower than the reference method. $\mathsf{Dist}(C_A, C_B)$ and the reference method have to fill dynamic programming tables of approximately same sizes and therefore their run times differ only by a multiplicative constant, as the proposed method has to fill entries in three tables instead of one.

4 Conclusion and Future Work

We have proposed and evaluated a method for estimating Levenshtein distance between two sequences from partially assembled data. In experiments, our approach was better in terms of Pearson's correlation coefficient than the one of [9] for higher coverage data. The proposed method was significantly better than the trivial estimates and a straightforward solution where we use only the longest contig to represent the assembly.

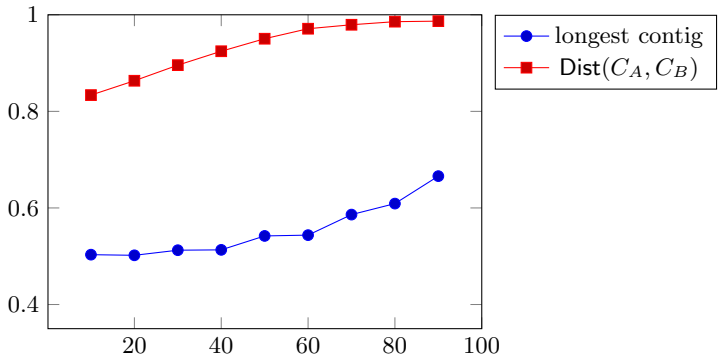

Fig. 5. Plot of average Pearson's correlation coefficient on θ parameter for simulated assembly opt_θ on influenza dataset.

The most promising goal for the follow-up work is to combine the method proposed in this paper with the one of [9] in order to get advantages of both. Method [9] is significantly faster and it produces better results for low coverage data. Instead of assembling whole contigs, we may assemble only a few neighboring reads in order to balance time needed for assembly together with distance estimation time and estimate accuracy.

References

1. Fowlkes, E.B., Mallows., C.L.: A method for comparing two hierarchical clusterings. Journal of the American statistical association 78(383), 553–569 (1983)
2. Hernandez, D., et al.: De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. Genome research 18(5), 802–809 (2008)
3. Huang, W., Li, L., Myers, J.R., Marth, G.T.: ART: a next-generation sequencing read simulator. Bioinformatics 28(4), 593–594 (2012)
4. Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2005)
5. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady 10(8) (1966)
6. Marzal, A., Vidal, E.: Computation of normalized edit distance and applications. IEEE transactions on pattern analysis and machine intelligence 15(9), 926–932 (1993)
7. Nurk, S., Bankevich, A., et al.: Assembling Genomes and Mini-metagenomes from Highly Chimeric Reads, pp. 158–170. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-37195-0_13
8. Ryšavý, P., Železný, F.: Estimating Sequence Similarity from Read Sets for Clustering Sequencing Data (IDA'2016 Frontier Prize Award), pp. 204–214. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-46349-0_18
9. Ryšavý, P., Železný, F.: Estimating Sequence Similarity from Read Sets for Clustering Next-Generation Sequencing data (2017), preprint, http://arxiv.org/abs/1705.06125

10. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution 4(4), 406–425 (1987)
11. Simpson, J.T., et al.: ABySS: a parallel assembler for short read sequence data. Genome research 9(6), 1117–1123 (2009)
12. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of molecular biology 147(1), 195–197 (1981)
13. Warren, R.L., et al.: Assembling millions of short DNA sequences using SSAKE. Bioinformatics 23(4), 500–501 (2007)
14. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome research 18(5), 821–829 (2008)