Smooth q-Gram, and Its Applications to Detection of Overlaps among Long, Error-Prone Sequencing Reads*

Haoyu Zhang
Indiana University Bloomington
Bloomington, IN, USA
hz30@umail.iu.edu

Qin Zhang
Indiana University Bloomington
Bloomington, IN, USA
qzhangcs@indiana.edu

Haixu Tang
Indiana University Bloomington
Bloomington, IN, USA
hatang@indiana.edu

ABSTRACT

We propose *smooth q-gram*, the first variant of q-gram that captures q-gram pair within a small edit distance. We apply smooth q-gram to the problem of detecting overlapping pairs of error-prone reads produced by single molecule real time sequencing (SMRT), which is the first and most critical step of the *de novo* fragment assembly of SMRT reads. We have implemented and tested our algorithm on a set of real world benchmarks. Our empirical results demonstrated the significant superiority of our algorithm over the existing q-gram based algorithms in accuracy.

ACM Reference Format:
Haoyu Zhang, Qin Zhang, and Haixu Tang. 2018. Smooth q-Gram, and Its Applications to Detection of Overlaps among Long, Error-Prone Sequencing Reads. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271688

1 INTRODUCTION

Q-gram, also called n-gram, k-mer/shingle, has been used extensively in the areas of bioinformatics [1, 3, 17, 23, 24], databases [25, 28, 29], natural language processing [19], etc. In particular, q-gram was used to construct the *de Bruijn graph* [12, 24], a data structure commonly exploited for fragment assembly in genome sequencing, especially for short reads obtained using next-generation sequencing (NGS) technologies [21]. Another important application of q-gram in bioinformatics is in *sequence alignment*, which aims to detect highly similar regions between long strings (e.g., genomic sequences). Following the *seed-extension* approach, many sequence alignment algorithms (including the popular BLAST [1] and more recent algorithms [5, 16, 27]) first search for q-gram matches (i.e., *seeds*) between each pair of input strings, and then extend these matches into full-length alignment by using dynamic programming algorithms. Recently, this approach was adopted for detecting *overlaps* between *long, error-prone reads* [3, 17, 23] generated by single molecule (also called the third generation) sequencing technologies, including the *single molecule real time sequencing (SMRT)* [26] and the *MinION sequencers* [20]. Comparing with the NGS reads, the single molecule technologies generate *reads* much longer and more error-prone. As a result, two overlapping reads contain highly similar but not identical substrings (with a relatively small edit distance[1] due to sequencing errors), which should be addressed by an overlap detection algorithm.

A straightforward application of the *seed-extension* approach to overlap detection may be hurdled by an inherent limitation: two strings sharing a highly similar substring may share only a small number of, or even zero, matched q-gram pairs (*seeds*), due to the pattern of sequencing errors within the shared substring. Consequently, a seed-extension algorithm may fail to detect such overlaps because of the lack of seeds between the reads. Let us illustrate this point by an example. Consider the following two input strings:

00000 00000 00000 00000 00000 0000, and
00000 00001 00000 00001 00000 0000,

Their edit distance is 2, however, they share no matched 10-gram pairs (seeds).

To address this issue, in this paper, we propose a variant of q-gram called the *smooth q-gram*, using which we can identify not only those exactly matched q-gram pairs (with certainty), but also those q-gram pairs that have small edit distances (each with a high probability). Our smooth q-gram construction is based on a recent advance in metric embedding [10] that maps a string from the edit distance space to the Hamming distance space while (approximately) preserving the distance; we will illustrate the details of this embedding in Section 2.1. For the example mentioned above, our smooth q-gram based approach can, with a very high probability, find most pairs of q-grams of the two input strings whose edit distances are at most 1.

Application in SMRT data. We applied the smooth q-gram to the overlap detection among sequencing reads produced by SMRT, which is the first and most critical step of the *de novo* fragment assembly of SMRT reads. Notably, SMRT sequencers generate reads of 1,000-100,000 bps long with 12-18% sequencing errors (including most insertions/deletions and some substitutions); in comparison, Illumina sequencers

*H. Zhang and Q. Zhang were supported in part by NSF CCF-1525024 and IIS-1633215. H. Tang was supported by NIH 1R01AI108888.

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271688

[1] The edit distance between two strings x and y is defined to be the minimum number of letter insertions, deletions and substitutions needed to transfer x to y.

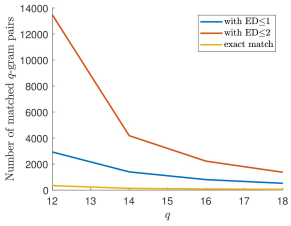


Figure 1: Average number of matched q-gram pairs for 50 overlapping read pairs in the `E.coli` dataset.

(a common NGS platform) generate reads of 100-300 bps long with $< 1\%$ errors . We have evaluated our approach using real-world SMRT datasets.

We formalize the *overlap detection problem* as follows. Given a collection of strings $\mathcal{X} = \{x_1, \ldots, x_n\}$, the goal is to output all overlapping string pairs $\{(x, y) \mid x, y \in \mathcal{X}\}$ and their shared substrings x_{sub} and y_{sub}, such that the lengths of the substrings are above a threshold Γ, and their edit distance is below a threshold θ.

For long, error-prone reads produced by SMRT, finding a good number of exactly matched q-grams between reads could be difficult (or just impossible). Thus the overlap detection problem becomes challenging for conventional "seed-extension" approaches. For example, in Figure 1 we plotted the average number of matched q-grams between 50 overlapping SMRT reads sampled from a real word dataset `E.coli` under different matching thresholds (edit distance threshold being 0, 1, and 2, respectively). We can see that when $q = 12$, the number of q-gram pairs with edit distance (ED) no more than 2 is 39.2 times of that of exactly matched q-gram pairs. Obviously, for a pair of reads, with more matched q-grams (seeds) detected, the sensitivity increases for detecting putative overlaps between error-prone reads. Therefore, the smooth q-gram approach proposed in this paper can outperform the existing q-gram based "seed-extension" approaches. Indeed, our evaluation showed that, for the overlap detection in the real-world datasets that we have tested, the smooth q-gram based algorithm *always* achieved F_1 scores (i.e., the harmonic average of the precision and recall) above 0.9, while the F_1 score achieved by the best q-gram based algorithm can be as low as 0.77.

Our Contribution. We summarize our contribution below.

(1) We proposed smooth q-gram, the first variant of q-gram that captures q-gram pair within a small edit distance.
(2) We applied smooth q-gram to the problem of detecting overlapping pairs of error-prone reads produced by single molecule sequencing technologies, such as SMRT.
(3) We implemented our smooth q-gram based algorithm and tested it on a set of real world benchmarks. Our empirical results demonstrated the significant superiority of our algorithm over the existing q-gram based algorithms in precision, recall and F_1 scores.

Related Work. Since it is proposed recently, the problem of detecting overlaps among long, error-prone reads from SMRT has drawn a significant attention in bioinformatics [3, 17, 23]. All existing overlap detection algorithms follow the "seed-extension" approach, in which the seeds are defined based on q-grams.

The only line of work, as far as we have concerned, that has a similar spirit as ours is the *gapped q-gram* [6–8] (also referred to as the *spaced seeds* in bioinformatics applications [13, 18]). The idea of gapped q-gram is to take substrings of each string of a specific pattern. For example, the gapped 3-grams of the string "ACGTACGT" with pattern "XX-X" are {ACT, CGA, GTC, TAG, ACT}. That is, instead of taking the contiguous substrings as that in the traditional q-gram approach, the gapped q-gram breaks the adjacency dependencies between the characters. Now if we are allowed to choose multiple gapped q-gram patterns, then one will need more edits to make all gapped q-grams between two strings mismatched. However, the optimal pattern of gapped q-gram is difficult to find: it needs an exhaustive search on all possible patterns, and the running time for the search has an exponential dependency on length of the pattern [13]. This might be the reason why there is no previous work applying gapped q-gram to solve the overlap detection problem for SMRT data. In contrast, our smooth q-grams are systematically generated, and always have the same theoretical guarantees on all datasets.

2 SMOOTH q-GRAM

As mentioned, the major innovation of this paper is to replace the standard q-gram based approach for overlap detection with the smooth q-gram based approach. The advantage of smooth q-gram is that it tolerates a small edit distance between matched q-grams and is thus able to identify similar strings at higher sensitivity. In this section, we discuss the details of smooth q-gram construction and discuss its properties.

We will use m to denote the length of a smooth q-gram, and κ to denote the length of a q-gram after CGK-embedding.

2.1 The CGK-Embedding

The key tool that we will use in our construction of smooth q-gram is the CGK-embedding, which convert a string $s \in \Sigma^q$ to $s' = \Sigma^\kappa$ for a value κ using a random string R_1, where Σ is the alphabet (for nucleotides, $\Sigma = \{A, C, G, T\}$).

More precisely, let $j = 1, 2, \ldots, \kappa$ denote the time steps of the embedding. We also maintain a pointer i to the string s, initialized to be $i = 1$. At each step j, we first copy $s[i]$ to $s'[j]$, and set $j \leftarrow j + 1$. We then determine whether we should increment i or not. We sort characters in Σ in an arbitrary but fixed order. For a character $\sigma \in \Sigma$, let $\mathrm{Index}(\sigma)$ denote the index of σ in this order. We set

$$i \leftarrow i + R_1[j \cdot |\Sigma| - \mathrm{Index}(s[i]) + 1].$$

When i reaches $q+1$ while $j < \kappa$, we simply pad $\kappa - j$ copies of '$\perp$' to s' to make its length equal to κ, where $\perp \notin \Sigma$ is an arbitrary character.

Denote the CGK-embedding as a function $CGK(\cdot, R_1)$ for a fixed string (sampled randomly from $\{0,1\}^{\kappa|\Sigma|}$). Given

Algorithm 1 Generate-Smooth-q-Gram(s, R_1, R_2)

Input: s: q-gram $s \in \Sigma^q$;
R_1: random string from $\{0,1\}^{\kappa|\Sigma|}$;
R_2: random string from $\{0,1\}^{\kappa}$ under the constraint that there are m 1-bit;
Output: $\bar{s}$: smooth q-gram of s of size m
1: $s' \leftarrow \text{CGK}(s, R_1)$
2: $\bar{s}$ is generated by removing coordinates i in s' s.t. $R_2[i] = 0$
3: **return** $\bar{s}$

$s, t \in \Sigma^q$, let $s' = CGK(s, R_1)$ and $t' = CGK(t, R_1)$. It has been shown in [10] that for any $\kappa \geq 2q + c\sqrt{q}$ for some large enough constant c, we have with probability 0.999 that

$$\text{ED}(s,t) \leq \text{HAM}(s',t') \leq O\left((\text{ED}(s,t))^2\right),$$

where $\text{ED}(\cdot,\cdot)$ and $\text{HAM}(\cdot,\cdot)$ denote the edit distance and the Hamming distance respectively.

It is easy to see that after the CGK-embedding, q-grams with small edit distance will likely have small Hamming distance, and those with large edit distance will likely have large Hamming distance. In particular, if $s = t$, then we have $s' = t'$ with certainty.

The CGK-embedding has recently been used for sketching edit distance [2] and performing edit similarity joins [30].

2.2 From q-Gram to Smooth q-Gram

We show how to construct a smooth q-gram from a standard q-gram using random string R_2. For convenience we will write "smooth q-gram" instead of "smooth m-gram" although the resulting smooth q-gram will have length m. Our algorithm is very simple. Given a q-gram s, we first perform the CGK-embedding on s to get a string s' of length κ, and then construct a substring $\bar{s}$ of length m by picking the coordinates i in s' where $R_2[i] = 1$. The algorithm is depicted in Algorithm 1.

The motivation of introducing smooth q-gram is that we hope that the corresponding smooth q-grams of two q-grams s and t for which $\text{ED}(s,t)$ is small, can be identical with a good probability. More precisely, let $k = \text{ED}(s,t)$, and let $s' = CGK(s, R_1)$ and $t' = CGK(t, R_1)$. By the property of the CGK-embedding, we know that $\text{HAM}(s',t') \leq k^2$. Let $d = \text{HAM}(s',t')$. If we randomly sample without replacement m bits from two κ-bit strings s' and t' at the same indices, the probability that all the sampled bits are the same is

$$\begin{aligned} & \frac{\kappa - d}{\kappa} \times \frac{\kappa - d - 1}{\kappa - 1} \times \cdots \times \frac{\kappa - d - (m-1)}{\kappa - (m-1)} \\ = \quad & \frac{(\kappa - m) \times \cdots \times (\kappa - (d-1) - m)}{\kappa \times (\kappa - 1) \times \cdots (\kappa - (d-1))}. \end{aligned} \tag{1}$$

In our experiments we typically choose $m = \kappa/c$ for a constant c, and we are only interested in d being at most 4. In this case we can approximate (1) as $((c-1)/c)^d$ for some constant c. In other words, for a non-trivial fraction of pairs of q-gram, their corresponding smooth q-gram will be matched. Finally, we note that when $s = t$, with fixed R_1 and R_2 we must have $\bar{s} = \bar{t}$ with certainty.

We note that our construction of smooth q-gram is very different from just a subsampling of the original q-grams. Indeed, given two q-grams s and t where t is obtained by a cyclic shift of s by one coordinate (that is, we move the first coordinate of s to the end of s, and $\text{ED}(s,t) = 2$), if we just sample say a constant fraction of coordinates from s and t using common randomness, getting $\bar{s}$ and $\bar{t}$, then $\bar{s}$ and $\bar{t}$ will be different with very high probability.

As mentioned in the introduction, if we are able to match near-identical q-grams (under edit distance), then we are able to catch similar pairs of strings which will otherwise be missed by standard q-gram approaches. In this way we can significantly improve the recall of the algorithm. Of course, by allowing approximate matching we may also increase the number of false positives, that is, dissimilar pairs of strings may have many identical smooth q-grams, and will thus be considered as similar pairs. To maintain a good precision we may need to perform a verification step on the candidate pairs of similar strings, which will increase the running time. Therefore, for a particular application, one needs to select a good tradeoff between the accuracy improvement and the extra running time cost.

We also comment that we can further enhance the precision by performing multiple CGK-embeddings (say, d times), and/or multiple subsamplings (say, z times), so that for each q-gram we will create $d \times z$ smooth q-grams. However, these operations will increase the number of false positives as well, and consequently the running time. In our experiments in Section 4 we have computed the number of matching q-grams on various datasets when varying the number of CGK-embeddings and subsamplings. But for our application of detecting overlapping error-prone sequencing reads, we have noticed that a single run of CGK-embedding and subsampling already gives satisfactory accuracy.

3 APPLICATIONS TO OVERLAP DETECTION AMONG LONG, ERROR-PRONE SEQUENCING READS

In this section we show how to use smooth q-gram to solve the overlap detection problem for long, error-prone sequence reads. We approach the problem in two steps. In the first step (Section 3.1), we show how to use smooth q-gram to detect putative pairs of overlapping strings. And then for each of such pairs, we design an efficient verification procedure to reduce the number of false positives (Section 3.2).

In Table 1 we have listed a set of global parameters/notations that will be used in our algorithms. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$.

3.1 Detecting Putative Pairs of Overlapping Strings

Our algorithm for detecting overlapping pairs of strings is presented in Algorithm 2.

m	length of smooth q-gram
κ	length of q-gram after CGK-embedding
α	signature selection rate
η	frequency filtering threshold
K	edit distance threshold
C	threshold for #matched signatures
L	targeting overlap length
ϵ	error tolerance rate
Π	$\Pi : \Sigma^m \to (0,1)$ a random hash function
R_c	random string from $\{0,1\}^{\kappa\lvert\Sigma\rvert}$
R_s	random string from $\{x \in \{0,1\}^{\kappa} \mid \lVert x\rVert_1 = m\}$

Table 1: List of Global Parameters

We will use the following data structure to store useful information of a q-gram.

Definition 3.1 (q-gram signature). Let $\delta(s,t,r,i,p)$ be a signature for a q-gram; the parameters are interpreted as follows:

- s is the q-gram;
- $t \leftarrow$ Generate-Smooth-q-gram(s, R_c, R_s)
- $r \leftarrow \Pi(t)$, which can be seen as a hash rank of t;
- i, p denote that s is taken from the i-th input string x_i from the position p, that is, $s \leftarrow x_i[p, p+q-1]$.

It is easy to see that t and r are fully determined by s given the randomness R_s, R_c and Π, but for convenience we still include them as parameters in the definition of the signature.

We now describe Algorithm 2 in words. The algorithm can be divided into three stages. The first stage (Line 1 -11) is the initialization: for each input string x_i, and for each of its q-gram, we generate the corresponding q-gram signature. In the second stage (Line 12 - 22) we try to find a set of candidate overlapping pairs of input strings. We will explain how this works in the rest of this section. The last stage (Line 23-29) is a verification step and will be illustrated in Section 3.2.

In the second stage of the algorithm, the first step is to filter out those smooth q-grams whose frequency is above a certain threshold (Line 12). This is a common practice, and has been used in a number of previous algorithms, such as `MHAP`[3], `Minimap`[17], `DALIGNER`[23]. The motivation of this pruning step is that frequent smooth q-grams often correspond to frequent q-grams, which do not carry much important features/information about the sequence (similar to the frequent words like 'a', 'the' in English sentences). On the other hand, these common smooth q-grams will contribute to many false positives and consequently increase the running time of subsequent steps. It is inevitable that in this pruning procedure some true positives are also filtered out. However, we have observed that by appropriately choosing the filtering threshold η, we can significantly reduce the number of false positives at the cost of introducing a small number of false negatives. We will show this with experiments in Section 4.3.

After the filtering step we perform a subsampling of an α-fraction of q-grams using the random hash function Π (Line 14). We then only focus on these sampled q-grams when measuring the string similarity. The purpose of performing such a subsampling is to reduce the total running time of the verification step (Line 24) by producing a set of smaller matching lists $\mathcal{M}_{i,j}$ (Line 20). On the other hand, it will not affect the accuracy of the algorithm by much. This is because in the verification step we will consider a pair of input strings (x_i, x_j) who have at least C matched q-gram pairs, and subsampling q-grams by a ratio of α corresponds to subsampling the matched q-gram pairs by a ratio of α^2. Therefore we can scale the threshold C correspondingly to obtain a similar set of candidate string pairs.

We next try to find for each pair of input strings (x_i, x_j), their set of matching q-grams (Line 13-22). This is done by calling a subroutine Algorithm 3 to find for each q-gram, a list of its matching q-grams (with edit distances less than or equal to K). More precisely, in Algorithm 3 we try to find for a q-gram s an (incomplete) list of matching q-grams

Algorithm 2 Find-overlapping-Strings($\mathcal{X}$)

Input: $\mathcal{X} = \{x_1, \ldots, x_n\}$: set of input strings;
Output: $\mathcal{O} \leftarrow$ {overlapping pair (x_i, x_j) and their shared substrings $x_i[p_i^s, p_i^e]$ and $x_j[p_j^s, p_j^e]$}

1: Initialize an empty table D
2: **for each** $i \in [n]$ **do**
3: $\quad\Delta_i \leftarrow \emptyset$
4: $\quad$**for each** $p \in [|x_i| - q + 1]$ **do**
5: $\quad\quad s \leftarrow x_i[p, p+q-1]$
6: $\quad\quad t \leftarrow$ Generate-Smooth-q-Gram(s, R_c, R_s)
7: $\quad\quad r \leftarrow \Pi(t)$
8: $\quad\quad \delta \leftarrow (s,t,r,i,p)$
9: $\quad\quad \Delta_i \leftarrow \Delta_i \cup \delta$
10: $\quad$**end for**
11: **end for**
12: Count for all t the number c_t of signatures in the form of $(\cdot, t, \cdot, \cdot, \cdot)$ in $\bigcup_{i\in[n]} \Delta_i$, and remove all (s,t,r,i,p) in Δ_i with $c_t \geq \eta \sum_t c_t$ for all $i \in [n]$
13: **for each** $i \in [n]$ **do**
14: $\quad$Construct Δ_i' from Δ_i by keeping signatures in Δ_i with the smallest $\alpha|x_i|$ of hash ranks r.
15: $\quad\mathcal{L}_i \leftarrow \emptyset$
16: $\quad$**for each** δ in Δ_i' **do**
17: $\quad\quad\mathcal{L}_i \leftarrow \mathcal{L}_i \cup$ Search-Similar-q-Grams$(\delta,\ D)$
18: $\quad$**end for**
19: $\quad$**for each** $j < i$ **do**
20: $\quad\quad\mathcal{M}_{ij} \leftarrow \{(u,v) \mid (j,u,v) \in \mathcal{L}_i\}$
21: $\quad$**end for**
22: **end for**
23: **for each** $|\mathcal{M}_{ij}| \geq C$ **do**
24: $\quad(o, pos) \leftarrow$ Verify$(x_i,\ x_j,\ \mathcal{M}_{ij})$
25: $\quad$**if** $(o, pos) \neq null$ **then**
26: $\quad\quad(p_i^s, p_i^e, p_j^s, p_j^e) \leftarrow$ Find-Shared-Substrings$(x_i,\ x_j,\ o,\ pos,\ \Delta_i,\ \Delta_j)$
27: $\quad\quad\mathcal{O} \leftarrow \mathcal{O} \cup \big(x_i, x_j, [p_i^s, p_i^e], [p_j^s, p_j^e]\big)$
28: $\quad$**end if**
29: **end for**
30: **return** $\mathcal{O}$

Algorithm 3 Search-Similar-q-Grams(δ, D)

Input: $\delta = (s, t, r, i, p)$: a signature for q-gram s (see Definition 3.1 for detailed explanation of the parameters);
D: a table with buckets indexed by t;
Output: $\mathcal{L} \leftarrow \{(i', p, p') \mid \exists \delta' = (s', t, r, i', p') \in D \text{ s.t. } \mathrm{ED}(s, s') \leq K\}$

1: $\mathcal{L} \leftarrow \emptyset$
2: **for each** $\delta' = (s', t, r, i', p')$ stored in $D(t)$ **do**
3: **if** $\mathrm{ED}(s, s') \leq K$ **then**
4: $\mathcal{L} \leftarrow \mathcal{L} \cup (i', p, p')$
5: **end if**
6: **end for**
7: Add δ to the $D(t)$
8: **return** $\mathcal{L}$

s' by considering all q-gram s' such that the corresponding smooth q-grams of s and s' fall into the same bucket in table D (Line 2). We then perform a brute-force edit distance computation (Line 3) to make sure that $\mathrm{ED}(s, s') \leq K$; if this holds then we record the pair and the positions of the match into $\mathcal{L}$. Finally at Line 7 we add the signature of s into table D to build the table D gradually while performing the search.

3.2 Verification

In this section we discuss how to verify whether a pair of input strings (x, y) overlap at a significant length given a list of their matching q-grams, and if it is the case, what are the shared substrings in the respective strings. For this purpose we employ two subroutines: Algorithm 4 performs a basic verification, and outputs a pair of positions on x and y inside the shared substrings if (x, y) is considered as an overlapping pair. We then use Algorithm 5 to recover the actual shared substrings.

We now describe Algorithm 4 and Algorithm 5 in words. Let $\mathcal{M}$ be the list of starting positions of the matching pairs of q-grams of input strings x_i and x_j. We construct bipartite graph $G_{i,j}$ with characters of x_i as nodes on the left side, and characters of x_j as nodes on the right side. For each matching pair (u, v), there is an edge connecting $x_i[u]$ and $x_j[v]$. For convenience, we slightly abuse the notation by using (u, v) to denote the edge between $x_i[u]$ and $x_j[v]$, and call $(u - v)$ the *shift* of the edge.

It is not hard to imagine that if x_i and x_j overlap, there must be a large cluster of edges of similar shifts in $G_{i,j}$. Algorithm 4 consists of two filtering steps. In the first step we try to identify a good reference shift o (Line 1-5), and remove all the edges whose shifts are far away from o (Line 6) (more precisely, those pairs (u, v) with $|(u - v) - o| > \frac{\epsilon}{2} \cdot L$). According to the previous literature, SMRT sequencing reads have accuracy $82\% - 88\%$ [14]. We thus set the error tolerance rate ϵ to be 0.2.

After finding a good reference shift, we try to find a dense area (or simply, a reference position *pos* in x_i) which contains many edges whose shifts are close to o (Line 7-11). We then remove all the edges that are not in this dense area (Line 12).

Algorithm 4 Verify(x_i, x_j, $\mathcal{M}$)

Input: x_i, x_j: two input strings;
$\mathcal{M} = \{(u, v)\}$: set of pairs of matched q-gram positions in x_i and x_j;
Output: o: reference offset
pos: reference position

1: $I \leftarrow \emptyset$
2: **for each** $(u, v) \in \mathcal{M}$ **do**
3: $I \leftarrow I \cup [u - v - \frac{\epsilon}{2} \cdot L, u - v + \frac{\epsilon}{2} \cdot L]$
4: **end for**
5: Find a value o s.t. $|\{[a, b] \mid o \in [a, b], [a, b] \in I\}|$ is maximized
6: Remove all pairs $(u, v) \in \mathcal{M}$ s.t. $u - v < o - \frac{\epsilon}{2} \cdot L$ or $u - v > o + \frac{\epsilon}{2} \cdot L$
7: $J \leftarrow \emptyset$
8: **for each** $(u, v) \in \mathcal{M}$ **do**
9: $J \leftarrow J \cup [u - \frac{L}{2}, u + \frac{L}{2}]$
10: **end for**
11: Find a value *pos* s.t. $|\{[a, b] \mid pos \in [a, b], [a, b] \in J\}|$ is maximized
12: Remove all pairs $(u, v) \in \mathcal{M}$ s.t. $u < pos - \frac{L}{2}$ or $u > pos + \frac{L}{2}$
13: **if** $|\mathcal{M}| < C$ **then**
14: **return** *null*
15: **else**
16: **return** (o, pos)
17: **end if**

Finally, we count the number of edges in the dense areas; if the number is at least C, then we consider (x_i, x_j) an overlapping pair and return the reference edge (determined by o and *pos*); otherwise we simply return *null* (Line 13-17).

We should note that all of these operations are performed on a subset $\mathcal{M}$ of matched q-gram pairs in x_i and x_j. By "subset" we mean that $\mathcal{M}_{i,j}$ is constructed after the subsampling step at Line 14 in Algorithm 2. As mentioned above, the purpose of the subsampling is to reduce the running time in the verification step. In contrast, when the actual shared substrings between x_i and x_j are found by Algorithm 5, we exploit the *complete* set of matched q-gram pairs, which will not significantly increase the overall running time because after verification, the number of input string pairs becomes much smaller.

Now, we turn to the details of the algorithm for determining the actual shared substrings between x_i and x_j (Algorithm 5). We again first construct the list $\mathcal{M}$ of matching q-grams. This can be done by a synchronized linear scan on the two sets Δ_i and Δ_j, after sorting the tuples by their r values. Next, starting from the reference edge determined by o and *pos*, we first locate the corresponding dense areas (Line 2-4). We then try to extend this dense area by adding one by one the matching edges outside this dense areas but still within a distance of L from the dense area, in the increasing order of the distances between these matching edges to the dense area (Line 5-13). Finally the algorithm returns the extended area as the shared substrings between x_i and x_j.

Algorithm 5 Find-Shared-Substrings(x_i, x_j, o, pos, Δ_i, Δ_j)

Input: x_i, x_j: two input strings;
o: reference offset;
pos: reference position;
Δ_i, Δ_j: sets of q-gram signatures of x_i and x_j
Output: $(p_i^s, p_i^e, p_j^s, p_j^e)$: $x_i[p_i^s, p_i^e]$ and $x_j[p_j^s, p_j^e]$ are shared substrings in x_i and x_j

$\mathcal{M} \leftarrow \{(p, p') \mid (s, t, r, i, p) \in \Delta_i, (s', t, r, j, p') \in \Delta_j, ED(s, s') \leq K\}$
$\mathcal{Q} \leftarrow \{(p, p') \in \mathcal{M} \mid p \in [pos - \frac{L}{2}, pos + \frac{L}{2}], (p - p') \in [o - \frac{\epsilon}{2} \cdot L, o + \frac{\epsilon}{2} \cdot L]\}$
$(p_i^s, p_j^s) = \arg\min_{(p,p') \in Q} p$, $(p_i^e, p_j^e) = \arg\max_{(p,p') \in Q} p$
Remove $(p, p') \in \mathcal{M}$ s.t. $p \in [pos - \frac{L}{2}, pos + \frac{L}{2}]$ from $\mathcal{M}$
Sort matches $(p, p') \in \mathcal{M}$ using $\max(p - p_i^e, p_i^s - p)$ in the increasing order
for each $(p, p') \in \mathcal{M}$ **do**
    if $0 < p - p_i^e < L \wedge |(p - p') - (p_i^e - p_j^e)| < \epsilon \cdot (p - p_i^e)$ **then**
        $(p_i^e, p_j^e) \leftarrow (p, \max(p', p_j^e))$
    end if
    if $0 < p_i^s - p < L \wedge |(p - p') - (p_i^s - p_j^s)| < \epsilon \cdot (p_i^s - p)$ **then**
        $(p_i^s, p_j^s) \leftarrow (p, \min(p', p_j^s))$
    end if
end for

4 EXPERIMENTS

In this section we present experimental studies of smooth q-gram and its application to detect overlaps among SMRT sequencing reads.

4.1 Tested Algorithms

To facilitate the investigation of properties of smooth q-grams, we introduce an additional algorithm named *Find-Similar-q-Gram-Pairs*, which uses the smooth q-gram technique to find pairs of input q-grams whose edit distances are at most K for a given distance threshold K. The algorithm is depicted in Algorithm 6. Let us describe it in words briefly. Essentially, *Find-Similar-q-Gram-Pairs* can be seen as running *Search-Similar-q-Grams* (Algorithm 3) for each input q-gram. Of course in this investigation we do not need to carry the data structure $\delta(s, \cdot, \cdot, \cdot, \cdot)$ for each q-gram s that we used in Algorithm 3 (for the application of overlap detection). Moreover, as mentioned at the end of Section 2, we can choose to repeat the CGK-embedding and the subsampling for d and z times respectively, so that for each q-gram s we create $d \cdot z$ smooth q-grams. By doing this we can generate more similar q-gram pairs which can be used to potentially boost the accuracy of our application. We will test Algorithm 6 for various d and z values. While in our applications in Section 4.4 we only perform the embedding and the subsampling *once*, which is enough for obtaining good accuracy.

In Section 4.4 we compared Algorithm 2 with existing overlap detection algorithms. For convenience, we call our algorithm `SmoothQGram`. We briefly describe each of the competitors below.

Algorithm 6 Find-Similar-q-Gram-Pairs($\mathcal{S}$, d, z)

Input: $\mathcal{S} = \{s_1, \ldots, s_n\}$: set of q-grams;
d: number of CGK-embeddings;
z: number of subsamplings;
Output: $\mathcal{O} \leftarrow \{(s_i, s_j) \mid s_i, s_j \in \mathcal{S}, i \neq j, \mathrm{ED}(s_i, s_j) \leq K\}$

$\mathcal{C} \leftarrow \emptyset$
for each $j \in [d]$ **do**
    Pick a random string R_c^j from $\{0,1\}^{\kappa|\Sigma|}$,
    for each $k \in [z]$ **do**
        Pick a random string R_s^k from $\{0,1\}^{\kappa}$ under the constraint that it contains m 1-bit
        Initialize a new table D^{jk}
        for each $i \in [n]$ **do**
            $t_i^{jk} \leftarrow$ Generate-Smooth-q-Gram (s_i, R_c^j, R_s^k)
        end for
        Count for each distinct smooth q-gram its frequency
        for each $i \in [n]$ **do**
            if frequency of t_i^{jk} is less than $\eta \cdot n$ **then**
                for each q-gram s stored in the $D^{jk}(t_i^{jk})$ **do**
                    $\mathcal{C} \leftarrow \mathcal{C} \cup (s, s_i)$
                end for
                Store s_i in $D^{jk}(t_i^{jk})$
            end if
        end for
    end for
end for
Remove duplicate pairs in $\mathcal{C}$
for each $(x, y) \in \mathcal{C}$ **do**
    if $\mathrm{ED}(x, y) \leq K$ **then**
        $\mathcal{O} \leftarrow \mathcal{O} \cup (x, y)$
    end if
end for

`MHAP`[3][2]: this algorithm generates q-grams of all sequences and then filters out those with frequencies greater than 0.00001 times the total number of q-grams. Next, it uses multiple *Minhash* [4] functions to find matching q-grams between sequences, and then select pairs of sequences that have at least 3 matching q-grams as candidate pairs. For each candidate pair, it uses a modified sort-merge algorithm to find more accurate q-gram matches, and then computes the boundary of the overlap region using a uniformly minimum-variance unbiased (UMVU) estimator [11].

`Minimap`[17][3]: this algorithm generates q-grams of all sequences and then filters out the top 0.001 fraction of the most frequent ones. Next, it hashes each q-gram to a value in Σ^q, and selects q-grams with the smallest hash values in every 5 consecutive q-grams as signatures of the input sequence. It then find all matching signatures between input sequences; pairs of sequences that have at least one shared signature are identified as candidate pairs. `Minimap` then calculates a

[2]Implementation obtained from https://github.com/marbl/MHAP
[3]Implementation obtained from https://github.com/lh3/minimap

cluster of q-gram matches for each candidate pair, and then finds a maximum colinear subset of matches by solving a longest increasing sequence problem. If the size of the subset is larger than 4, then `Minimap` computes and outputs the overlap region using the subset of matches.

`DALIGNER`[9][4]: this algorithm generates q-grams of all sequences and then filters out those that occur more than 100 times. It then considers all the remaining q-grams directly and computes all the matching q-grams between pairs of sequences. Pairs of sequences with at least one shared q-gram are identified as a candidate. Next, for each candidate, it uses a linear time difference algorithm [22] to compute a local alignment between the two sequences, and outputs the pair if the alignment length is greater than the given threshold.

We note that all these algorithms are under the same "seed-extension" framework as ours: they first find all the matched q-grams between input sequence pairs, and then extend seeds to potential overlaps. The major difference between our `SmoothQGram` and the existing tools is that we have *relaxed* the strict q-gram matches to approximate q-gram matches (via smooth q-gram) to improve the accuracy of the output. Our specifics for overlap detection and determination of shared substrings are also different from the existing algorithms.

In our experiments we run `SmoothQGram` with parameters $q = 14, m = 21, \alpha = 0.15, K = 2, C = 3, L = 500, \epsilon = 0.2$, and $\kappa = 2q$ (except for Figure 2, where we have tested different κ values). We note that $q = 14$ is a common choice for SMRT data (`Minimap` and `DALIGNER` also set $q = 14$, and `MHAP` sets $q = 16$). We choose $\eta = 0.00003$ for `E.coli`, and $\eta = 0.0001$ for `Human` and `S.cerevisiae`, since `S.cerevisiae` and `Human` genome contain more repeats. Our η values are higher than the threshold 0.00001 in `MHAP`. This is due to the fact that smooth q-grams have higher frequencies than q-grams used in `MHAP` (multiple q-grams may share the same smooth q-gram).

We run `MHAP` with parameters "-num-hashes 1256", `Minimap` with parameters "-k 15 -Sw5 -L100 -m0 -t8", and `DALIGNER` with parameter "-H500". All other parameters were selected as default settings. Due to the complexity of the problem, all these algorithms have more than 5 parameters.

We note that the performance of `Minimap` is sensitive to the filter threshold ("-f") it uses. Thus, besides the default parameter, we also choose an alternative parameter "$-f$ 0.00000001" instead of 0.1% (as default) of the most frequent q-gram matches filtered (according to [15]'s recommendation), which essentially means that almost no q-gram will be filtered out. Intuitively, such a change will lead to better recall values (but possibly worse precision values) at the cost of greater memory usage and running time. We call the default version `Minimap-default` and the new version `Minimap-alternative`.

[4]Implementation obtained from https://github.com/thegenemyers/DALIGNER

Datasets	number of strings	Average Length
`E.coli-small`	100	4781
`E.coli`	46960	4221
`S.cerevisiae`	48279	3032
`Human`	47535	3100

Table 2: Statistics of Tested Datasets

4.2 The Setup

Datasets. We test algorithms using real world datasets from `PacBio` SMRT sequencing.[5] The statistics of these datasets are described in Table 2. In Section 4.3 we test Algorithm 6 using the dateset `E.coli-small`; the number of q-grams for ecoli-small is $100 \cdot 4781$. In Section 4.4 we compare different algorithms using much larger datasets `E.coli`, `S.cerevisiae` and `Human`.

Measurements. In Section 4.3 we report the number of matching q-gram pairs detected by the Algorithm 6. Each result is an average of 5 runs. In Section 4.4, we report four types of measurements in our experiments: *recall*, *precision*, *memory usage* and *running time*. We choose the evaluation program used by `MHAP` to calculate precision and recall; all parameters are selected as default except that the evaluation overlap length threshold Γ is set to be 500 or 2000. The evaluation program learns the ground truths from the reads to references mappings obtained by `Blasr` [9]. We note that the evaluation algorithm does not simply use edit distance as the criteria to compute precise and recall. Instead, it maps all the reads to the reference sequences, and computes for each pair of reads their overlap positions and lengths from the mapping results. This evaluation method is widely used in overlap detection because it considers ultimate use of overlaps between reads in final reads assembly.

We also present the F_1 score:

$$F_1 = 2 \times \frac{\text{precision } \times \text{recall}}{\text{precision } + \text{recall}},$$

which is an integrated metric evaluating both precision and recall.

All algorithms use multiple threads in execution; we thus measure the CPU time for comparison. The memory usage we report is the maximum memory usage of a program during its execution. We note that although all the tested algorithms are randomized, we use a fixed random seed for all of them to guarantee the consistency among outputs.

Computing Environment. All experiments are conducted on a Dell PowerEdge T630 server with 2 Intel Xeon E5-2667 v4 3.2GHz CPU with 8 cores each, and 256GB memory.

4.3 Finding q-gram Pairs with Small Edit Distance

In this section we present the performance of *Find-Similar-q-Gram-Pairs* (Algorithm 6). We set the default parameters

[5]The data was downloaded from `MHAP`'s supporting data website: http://www.cbcb.umd.edu/software/PBcR/mhap/index.html.

(parameters that are not shown in the plots) as: length of the smooth q-gram $m = 1.5 \times q$, number of CGK-embeddings $d = 1$, number of samplings $z = 1$, filter threshold $\eta = 1$ (i.e., no filter).

Matching Pairs of q-grams. We first study how different parameter values m (the length of the smooth q-gram), d (the number of CGK-embeddings), and z (the number of samplings) influence the number of almost matching q-gram pairs that we can find. Our results are presented in Figures 2, 3 and 4.

We observe that the number of matching q-grams increases when m decreases, and significantly increases when d and z increase. For example, fix q to be 14. For $m = 1.5 \times q$, we can detect 17.4 times q-gram matches with edit distance being at most 2 of that of exact matches, using only one subsampling and one embedding. With $d = 5$, we could detect 51.3 times (distinct) q-gram matches with edit distance being at most 2 of that of exact matches; and with $z = 5$ subsamplings, we could detect 40.8 times (distinct) q-gram matches.

In the rest of this section we will simply set $d = z = 1$, mainly for the sake of time/space saving. We found that by setting $d = z = 1$ we can already obtain very good accuracy, though greater d and z values can potentially further improve accuracy.

Number of Candidates. We next study how different parameters m, η influence the number of candidates. We call the pairs of q-grams whose edit distances are at most 2 *good pairs*, and those with edit distances larger than 2 *bad pairs*. Number of candidates is the sum of number of good and bad pairs.

Our results are presented in Figures 5 and 6.

We note that Figure 5 and Figure 2 come from the same set of experiments, but with different edit distance ranges and scales recorded. We observed that the ratio of number of candidate pairs and number of good pairs increases with η, and decreases sharply with m.

Figure 2 and Figure 5 also guide us on how to choose m to balance the number of good pairs and candidates. Under the condition that we get a good number of good pairs (i.e., q-gram pairs whose edit distance is at most 2), and we do not have too many candidates, it seems that $m = 1.5 \times q$ is a good choice and we set it as the default parameter.

When $q = 14$, $m = 1.5 \times q$ and $\eta = 1$ (i.e., no filter), we can detect 17.4 times good pairs while verify 209.7 times candidates (of the number of exact matches). By setting the filter threshold $\eta = 10^{-5}$, we can detect 12.3 times good pairs while only introduce 30.0 times candidates. This convinces us that removing frequent smooth q-grams have a greater impact on reducing candidates than good pairs, and is thus very useful for our purpose (i.e., to save the verification time at a minimal cost on the accuracy).

4.4 Finding Overlapping Sequencing Reads

In this section we present the experimental results on detecting overlapping sequencing reads with Algorithm 2.

Accuracy. We study the precision, recall and F_1 scores of all tested algorithms. The results are presented in Table 3 and Table 4.

Based on our results, `SmoothQGram` has the best recall values at *all* times, the best precision values in most cases, and the best F_1 scores (the harmonic average of precision and recall) at *all* times. Its F_1 scores are always greater than 0.9. While the lower bound of the F_1 score of the best competitor is only 0.77 (`Minimap` on `S.cerevisiae`). The performance of `SmoothQGram` is also robust on data from different species and different overlap lengths Γ. We note that `Minimap` will filter out too many q-grams when the genome contains more repeats. As a result, `Minimap-default` fails on `S.cerevisiae` dataset with default parameter and we choose an alternative version `Minimap-alternative` for it.

We note again that we can further improve the accuracy of `SmoothQGram` by using multiple embeddings and subsamplings, at the cost of larger space and time.

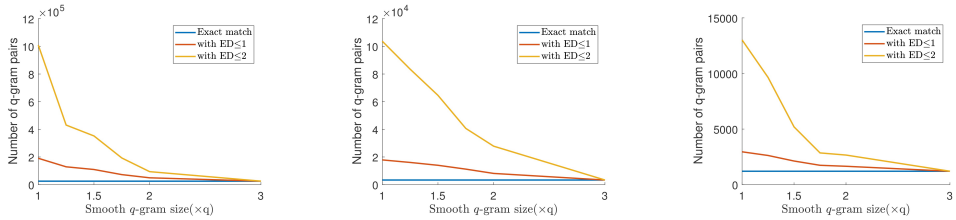
Comparing the results for the three species, we found that `E.coli` is generally easier to deal with than `S.cerevisiae` and `Human`, which may be due to the fact that `S.cerevisiae` and `Human` genome contain more repeats. For different overlap lengths Γ, we notice that all algorithms generally perform better on the greater length Γ than the smaller one, which is reasonable because longer overlaps are generally easier to be detected.

Time and Space. Finally, we study the running time and memory usage of tested algorithms. Our results are presented in Table 5. We observe that `Minimap` has the best time and memory performance among all algorithms. `DALIGNER` spends similar running time as `SmoothQGram`, but smaller amount of memory. `SmoothQGram` has the similar (slightly better) memory and time performance than `MHAP`. The reason why `SmoothQGram` uses relatively large time and memory is that `SmoothQGram` considers smooth q-gram instead of q-gram, which captures more matching information between sequences, and thus needs more time to verify candidate sequence pairs and uses more space. On the other hand, this is also why `SmoothQGram` significantly improved the accuracy for overlap detection.

We note that in our experimental studies, we mainly focused on accuracy which we think is the most important; our codes were not fully optimized for space and running time.

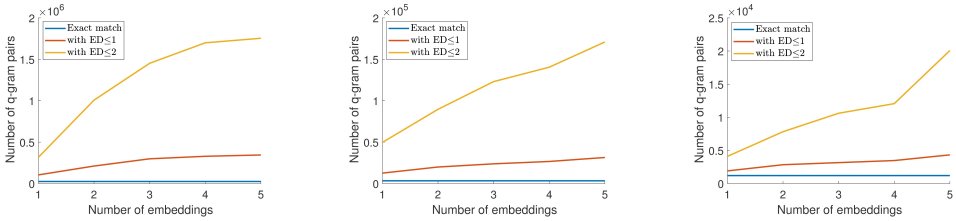
4.5 Summary

In this section, we have performed an extensive experimental study on smooth q-gram and its application to overlap detection. We observed that the smooth q-gram based approach achieved much better accuracy than the conventional q-gram based approaches for overlap detection, which is due to the fact that smooth q-gram is capable of capturing near-matches between subsequences. Employing smooth q-gram may introduce a larger number of false positives, but the number can be greatly reduced by applying a frequency-based filter. The performance of our algorithm is stable and robust on genome sequences from various species that we have tested, and using different overlap lengths Γ.



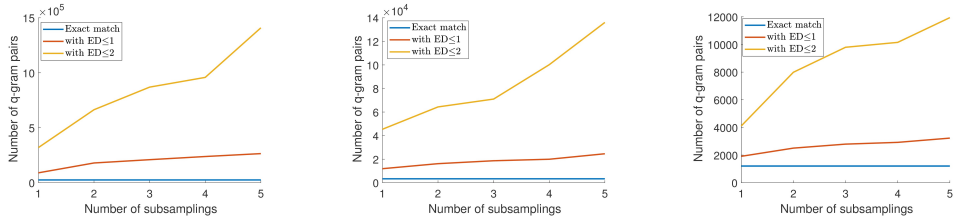
$q = 12$ $q = 14$ $q = 16$

Figure 2: Number of matching q-gram pairs vs smooth q-gram size m; on `E.coli-small`



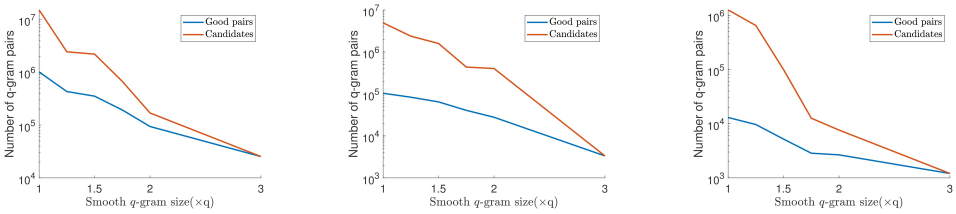
$q = 12$ $q = 14$ $q = 16$

Figure 3: Number of matching q-gram pairs vs number of embeddings d; on `E.coli-small`



$q = 12$ $q = 14$ $q = 16$

Figure 4: Number of matching q-gram pairs vs number of subsamplings z; on `E.coli-small`



$q = 12$ $q = 14$ $q = 16$

Figure 5: Number of good q-gram pairs and candidate pairs vs smooth q-gram size m; on `E.coli-small`

REFERENCES

[1] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. 1990. Basic local alignment search tool. *Journal of molecular biology* 215, 3 (1990), 403–410.

[2] Djamal Belazzougui and Qin Zhang. 2016. Edit Distance: Sketching, Streaming, and Document Exchange. In *FOCS*. 51–60.

[3] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. 2015. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology* 33, 6 (2015), 623–630.

[4] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 1998. Min-wise independent permutations. In *STOC*. ACM, 327–336.

[5] Michael Brudno, Chuong B Do, Gregory M Cooper, Michael F Kim, Eugene Davydov, Eric D Green, Arend Sidow, Serafim Batzoglou, NISC Comparative Sequencing Program, et al. 2003. LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome research* 13, 4 (2003), 721–731.

[6] Stefan Burkhardt and Juha Kärkkäinen. 2001. Better filtering with gapped q-grams. In *CPM*. 73–85.

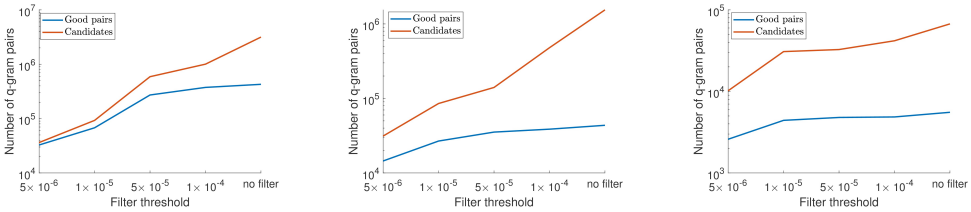
[7] Stefan Burkhardt and Juha Kärkkäinen. 2002. One-gapped q-gram filters for Levenshtein distance. In *CPM*. 225–234.

[8] Stefan Burkhardt and Juha Kärkkäinen. 2003. Better filtering with gapped q-grams. *Fundamenta informaticae* 56, 1-2 (2003), 51–70.

[9] Mark Chaisson and Glenn Tesler. 2012. Mapping single molecule sequencing reads using Basic Local Alignment with Successive Refinement (BLASR): Theory and Application. *BMC Bioinformatics* 13 (2012), 238.

[10] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. 2016. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *STOC*. 712–725.

[11] R. C. H. Cheng and N. A. K. Amin. 1983. Estimating Parameters in Continuous Univariate Distributions with a Shifted Origin.



$q = 12$ $q = 14$ $q = 16$

Figure 6: Number of good q-gram pairs and candidate pairs vs filter threshold η; on `E.coli-small`

	E.coli			S.cerevisiae			Human		
	Recall	*Precision*	F_1 *score*	*Recall*	*Precision*	F_1 *score*	*Recall*	*Precision*	F_1 *score*
`MHAP`	78.0%	99.8%	0.87	75.1%	92.5%	0.83	74.1%	83.7%	0.79
`Minimap – default`	92.4%	100.0%	0.96	15.4%	99.9%	0.26	68.9%	98.8%	0.81
`Minimap – alternative`	94.1%	99.8%	0.97	89.5%	93.1%	0.91	71.4%	40.4%	0.52
`DALIGNER`	86.1%	97.8%	0.92	82.8%	94.8%	0.88	80.6%	67.1%	0.73
`SmoothQGram`	95.1%	100.0%	0.97	90.9%	99.2%	0.95	84.7%	99.2%	0.91

Table 3: Accuracy for pairs with overlaps of lengths $\Gamma \geq 2000$

	E.coli			S.cerevisiae			Human		
	Recall	*Precision*	F_1 *score*	*Recall*	*Precision*	F_1 *score*	*Recall*	*Precision*	F_1 *score*
`MHAP`	66.3%	99.8%	0.80	65.8%	94.3%	0.77	77.1%	84.8%	0.81
`Minimap – default`	77.2%	99.9%	0.78	12.0%	99.8%	0.21	50.0%	99.2%	0.66
`Minimap – alternative`	79.8%	99.8%	0.89	72.4%	99.6%	0.84	58.2%	57.7%	0.58
`DALIGNER`	79.7%	94.3%	0.86	71.8%	90.9%	0.80	61.5%	63.7%	0.63
`SmoothQGram`	89.9%	100.0%	0.95	85.1%	98.6%	0.91	84.7%	95.5%	0.90

Table 4: Accuracy for pairs with overlaps of lengths $\Gamma \geq 500$

	E.coli		S.cerevisiae		Human	
	CPU Time(s)	*Memory*(Gb)	*CPU Time*(s)	*Memory*(Gb)	*CPU Time*(s)	*Memory*(Gb)
`MHAP`	9476	68.1	8025	75.2	7472	74.6
`Minimap – default`	103	6.4	61	4.8	56	4.7
`Minimap – alternative`	666	14.9	2836	15.5	2550	13.3
`DALIGNER`	2072	19.7	6376	17.4	8821	17.6
`SmoothQGram`	6734	85.5	7400	63.5	6736	63.1

Table 5: Running time and memory usage

Journal of the Royal Statistical Society 45, 3 (1983), 394–403.

[12] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. 2011. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology* 29, 11 (2011), 987–991.

[13] Uri Keich, Ming Li, Bin Ma, and John Tromp. 2004. On spaced seeds for similarity search. *Discrete Applied Mathematics* 138, 3 (2004), 253–263.

[14] Sergey Koren and Adam M Phillippy. 2015. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *Current opinion in microbiology* 23 (2015), 110–120.

[15] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. 2017. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research* 27, 5 (2017), 722–736.

[16] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. 2004. Versatile and open software for comparing large genomes. *Genome biology* 5, 2 (2004), R12.

[17] Heng Li. 2016. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* 32, 14 (2016), 2103–2110.

[18] Bin Ma, John Tromp, and Ming Li. 2002. PatternHunter: faster and more sensitive homology search. *Bioinformatics* 18, 3 (2002), 440–445.

[19] Christopher D Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

[20] Alexander S Mikheyev and Mandy MY Tin. 2014. A first look at the Oxford Nanopore MinION sequencer. *Molecular ecology resources* 14, 6 (2014), 1097–1102.

[21] Jason R Miller, Sergey Koren, and Granger Sutton. 2010. Assembly algorithms for next-generation sequencing data. *Genomics* 95, 6 (2010), 315–327.

[22] Eugene W. Myers. 1986. An O (ND) difference algorithm and its variations. *Algorithmica* 1, 1-4 (1986), 251–266.

[23] Gene Myers. 2014. Efficient Local Alignment Discovery amongst Noisy Long Reads. In *WABI*. 52–67.

[24] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. 2001. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences* 98, 17 (2001), 9748–9753.

[25] Jianbin Qin, Wei Wang, Yifei Lu, Chuan Xiao, and Xuemin Lin. 2011. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *SIGMOD*. 1033–1044.

[26] Richard J Roberts, Mauricio O Carneiro, and Michael C Schatz. 2013. The advantages of SMRT sequencing. *Genome biology* 14, 6 (2013), 405.

[27] Scott Schwartz, W James Kent, Arian Smit, Zheng Zhang, Robert Baertsch, Ross C Hardison, David Haussler, and Webb Miller. 2003. Human–mouse alignments with BLASTZ. *Genome research* 13, 1 (2003), 103–107.

[28] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*. 85–96.

[29] Chuan Xiao, Wei Wang, and Xuemin Lin. 2008. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB* 1, 1 (2008), 933–944.

[30] Haoyu Zhang and Qin Zhang. 2017. EmbedJoin: Efficient Edit Similarity Joins via Embeddings. In *SIGKDD*. 585–594.