Bioinformatics, 36(19), 2020, 4838–4845
doi: 10.1093/bioinformatics/btaa252
Advance Access Publication Date: 20 April 2020
Original Paper

Sequence analysis

Overlap detection on long, error-prone sequencing reads via smooth *q*-gram

Yan Song, Haixu Tang, Haoyu Zhang and Qin Zhang*

Department of Computer Science, Indiana University Bloomington, Bloomington, IN 47405, USA

*To whom correspondence should be addressed.

Associate Editor: Yann Ponty

Received on November 6, 2019; revised on March 27, 2020; editorial decision on April 9, 2020; accepted on April 13, 2020

Abstract

Motivation: Third generation sequencing techniques, such as the Single Molecule Real Time technique from PacBio and the MinION technique from Oxford Nanopore, can generate long, error-prone sequencing reads which pose new challenges for fragment assembly algorithms. In this paper, we study the overlap detection problem for error-prone reads, which is the first and most critical step in the *de novo* fragment assembly. We observe that all the state-of-the-art methods cannot achieve an ideal accuracy for overlap detection (in terms of relatively low precision and recall) due to the high sequencing error rates, especially when the overlap lengths between reads are relatively short (e.g. <2000 bases). This limitation appears inherent to these algorithms due to their usage of *q*-gram-based seeds under the *seed-extension* framework.

Results: We propose *smooth q-gram*, a variant of *q*-gram that captures *q*-gram pairs within small edit distances and design a novel algorithm for detecting overlapping reads using smooth *q*-gram-based seeds. We implemented the algorithm and tested it on both PacBio and Nanopore sequencing datasets. Our benchmarking results demonstrated that our algorithm outperforms the existing *q*-gram-based overlap detection algorithms, especially for reads with relatively short overlapping lengths.

Availability and implementation: The source code of our implementation in C++ is available at https://github.com/FIGOGO/smoothq.

Contact: qzhangcs@indiana.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Q-gram, also called *n*-gram, *k*-mer/shingle, has been used extensively in the areas of bioinformatics (Altschul *et al.*, 1990; Berlin *et al.*, 2015; Li, 2016; Myers, 2014; Pevzner *et al.*, 2001), databases (Qin *et al.*, 2011; Wang *et al.*, 2012; Xiao *et al.*, 2008), natural language processing (Manning and Schütze, 1999), etc. In particular, *q*-gram was used to construct the *de Bruijn graph* (Compeau *et al.*, 2011; Pevzner *et al.*, 2001), a data structure commonly exploited for fragment assembly in genome sequencing, especially for short reads obtained using next-generation sequencing (NGS) technologies (Miller *et al.*, 2010). Another important application of *q*-gram in bioinformatics is for *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 (Altschul *et al.*, 1990) and more recent algorithms (Brudno *et al.*, 2003; Kurtz *et al.*, 2004; Schwartz *et al.*, 2003), first search for *q*-gram matches (i.e. *seeds*) between each pair of input strings and then extend these matches into full-length alignment using dynamic programming algorithms.

Recently, the seed-extension approach was adopted for detecting overlaps between long, error-prone reads (Berlin *et al.*, 2015; Chaisson and Tesler, 2012; Li, 2016, 2018; Myers, 2014) generated by *single molecule sequencing technologies*, including the *PacBio Single Molecule Real Time (SMRT)* technologies (Roberts *et al.*, 2013) and *Oxford MinION* technologies (Mikheyev and Tin, 2014). Compared to the NGS reads, the single molecule sequencing technologies generate *reads* that are much longer but more error-prone. Notably, SMRT sequencers generate reads 1000–100 000 bps long with up to 12–18% sequencing errors (including most insertions/deletions and some substitutions); in comparison, Illumina sequencers generate reads of 100–300 bps long with $<1\%$ errors. As a result, two overlapping reads contain highly similar but not identical substrings (with a relatively small edit distance (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.) 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 some overlaps because of the lack of seeds between the reads.

To address this issue, we propose a variant of q-gram named the *smooth q-gram*, which we can use to 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 (Chakraborty *et al.*, 2016) 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 method in Section 2.1. Our smooth q-gram-based approach can, with a high probability, find most pairs of q-grams of the two input strings whose edit distances are small. We introduce experiments to further show the advantages of smooth q-gram in Supplementary Material S1.1.

Using smooth q-grams as seeds, we designed a novel algorithm for the overlap detection problem. Our algorithm works for both PacBio and Nanopore sequencing data and only requires a single set of parameters on all datasets. Our experiments show that for all real-world datasets that we have tested, the smooth q-gram-based algorithm achieves a F_1 score about 3.8% higher than the best existing algorithm on average. [The F_1 score is the harmonic average of the precision and recall (see Section 4).] The advantage of our algorithm is even greater on pairs with short overlaps (e.g. with lengths 500–2000 bps), which are the relatively more challenging cases. For example, the recall of our algorithm for pairs with overlap lengths < 2000 bps is about 7.0% higher than the best existing algorithm on average.

1.1 Related work

The only line of work, as far as we have concerned, that has a similar spirit as our smooth q-gram is the *gapped q-gram* (Burkhardt and Kärkkäinen, 2001, 2002, 2003), which is also referred to as the *spaced seeds* in bioinformatics applications (Keich *et al.*, 2004; Ma *et al.*, 2002). The idea of gapped q-gram is to make 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 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 (Keich *et al.*, 2004). In contrast, our smooth q-grams are systematically generated, and always have the same theoretical guarantees on all datasets.

The problem of detecting overlaps among long, error-prone reads has drawn significant attention in bioinformatics (Berlin *et al.*, 2015; Chaisson and Tesler, 2012; Li, 2016, 2018; Myers, 2014; Sović *et al.*, 2016). Existing overlap detection algorithms follow the 'seed-extension' approach and use conventional q-grams as seeds. We choose the most sensitive existing algorithms MHAP, Minimap2 and DALIGNER in our comparisons and introduce them in more detail in Section 4. Note that we find the GraphMap (Sović *et al.*, 2016) is only optimized for Nanopore data and has worse performance compared with other algorithms. Thus, we did not include it in our comparison.

2 Smooth q-gram

As mentioned above, the innovation of this paper is to extend the conventional q-gram-based approach to the smooth q-gram-based approach for overlap detection. The main advantage of smooth q-gram is that it tolerates a small edit distance between matched q-grams, and is thus able to identify similar substrings at higher sensitivity. In this section, we present the details of smooth q-gram construction and 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 the 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, $\Sigma = \{A, C, G, T\}$) and R_1 is a random string from $\{0,1\}^{\kappa|\Sigma|}$.

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 Index(σ) be the index of σ in this order. We set

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

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 $s, t \in \Sigma^q$, let $s' = CGK(s, R_1)$ and $t' = CGK(t, R_1)$. It has been shown in Chakraborty *et al.* (2016) that for any $\kappa \geq 2q + c\sqrt{q}$, for some large enough constant c, we have with probability 0.999 that

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

where ED$(\cdot,\cdot)$ and 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, whereas 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 by Belazzougui and Zhang (2016) and performing edit similarity joins by Zhang and Zhang (2017).

2.2 From q-gram to smooth q-gram

We show below how to construct a smooth q-gram from a conventional 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 easy to describe (Algorithm 1): 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 $R_2[i] = 1$. We include running examples of Algorithm 1 in Supplementary Material S1.2.The motivation of introducing the smooth q-gram is that we expect the corresponding smooth q-grams of two q-grams s and t for which ED(s,t) is small, can be identical with a good probability. More precisely, let $k = \text{ED}(s,t)$, and let

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** $\bar{s}$

$s' = CGK(s, R_1)$ and $t' = CGK(t, R_1)$. According to the property of the CGK-embedding, we know that $\mathrm{HAM}(s', t') \leq k^2$. Let $d = \mathrm{HAM}(s', t')$. 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

$$\frac{\kappa - d}{\kappa} \times \frac{\kappa - d - 1}{\kappa - 1} \times \cdots \times \frac{\kappa - d - (m-1)}{\kappa - (m-1)} = \frac{(\kappa - m) \times \cdots \times (\kappa - (d-1) - m)}{\kappa \times (\kappa - 1) \times \cdots (\kappa - (d-1))}. \quad (1)$$

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, a significant fraction q-gram pairs with distance of d or below will have their corresponding smooth q-grams to be identical. Finally, we note that for identical q-gram pairs, e.g. $s = t$, with fixed R_1 and R_2 we always have $\bar{s} = \bar{t}$.

We note that the construction algorithm for smooth q-gram is different from a simple subsampling of the original q-grams. Indeed, given two q-grams s and t with small edit distance i.e. $\mathrm{ED}(s, t) = 2$, if we just sample a constant fraction of symbols from s and t using common randomness, the resulting strings $\bar{s}$ and $\bar{t}$ will be different with high probability.

As mentioned in Section 1, if we are able to identify pairs of near-identical q-grams (under edit distance), then we are able to catch similar substrings between two strings that will otherwise be missed by the conventional q-gram-based approaches. Therefore, we can significantly improve the sensitivity of the overlap detection algorithm. Of course, by allowing approximate q-gram matching, we may also increase the number of false positives, i.e. dissimilar pairs of substrings may share some identical smooth q-grams, and will thus be considered as seeds. We need to check the actual edit distance between pairs of q-grams that share the same smooth q-grams, in the seed identification step for overlap detection.

3 Materials and methods

In this section, we show how to use smooth q-gram for overlap detection among long, error-prone sequence reads. In Table 1, we list a set of global parameters/notations that will be used in our algorithms. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$.

Our main algorithm (Algorithm 2) consists of three stages. The first stage (Line 1–11) is *signature generation*: for each input sequence x_i and each of its q-gram, we generate a smooth q-gram-based signature (seed). The second stage (Line 12–15) is *subsampling and filtering*: we sample a portion of signatures for each x_i and only use them to detect candidate overlaps. The last stage (Line 16–31) is *detection and verification*: we detect candidate overlaps, verify and report true overlaps. Below, we will describe each stage in more detail.

Table 1. List of global parameters

Parameters	Explanation
m	Length of a smooth q-gram
κ	Length of the q-gram after CGK-embedding
α	Fraction of positions selected as smooth q-grams
η	Frequency filtering threshold
K	Edit distance threshold
C	Threshold for #matched signatures between two overlapping reads
L	Overlap length
ϵ	Error rate tolerance
Π	$\Pi : \Sigma^m \to (0, 1)$ a random hash function
R_1	A random string from $\{0,1\}^{\kappa\lvert\Sigma\rvert}$
R_2	A random string from $\{x \in \{0,1\}^{\kappa} \mid \lVert x\rVert_1 = m\}$

3.1 Signature generation

In the signature generation stage (Line 1–11 of Algorithm 2), we use the following data structure to store useful information of a q-gram as a signature.

Definition 1 (q-gram signature) *Let $\delta(s, t, r, i, p)$ be the signature of a q-gram; the parameters are interpreted as follows:*

- *s: the q-gram;*
- *t: the smooth q-gram of s;*
- *r: the hash rank of t;*
- *i, p: q-gram s is taken from the i-th input string x_i from the position p,* i.e. $s \leftarrow x_i[p, p + q - 1]$.

Obviously, t and r are fully determined by s, given the randomness R_1, R_2 and Π, but for the convenience of presentation, we still include t, r as parameters in the definition of the signature. We omit the signature generation for the reverse complement of sequences for the sake of convenience. In practice, for each input sequence, we consider both the sequence itself and its reverse complement, and generate signatures for both of them separately.

3.2 Subsampling and filtering

In the subsampling and filtering stage (Line 12–15 of Algorithm 2), we first perform a subsampling of an α-fraction of smooth q-grams for each sequence using a random hash function Π (Line 13). We then only focus on these sampled smooth q-grams to identify similar substrings. The subsampling step could improve the efficiency of our algorithm by reducing the number of smooth q-gram matches to consider.On the remaining subsampled signatures, we filter out those smooth q-grams whose frequency is above a certain threshold (Line 15). This is a common practice to reduce the number of false positives in overlap detection algorithms, such as MHAP (Berlin *et al.*, 2015), Minimap2 (Li, 2016, 2018) and DALIGNER (Myers, 2014). The motivation behind the filtering step is that the highly frequent smooth q-grams often correspond to highly frequent q-grams, which do not carry many important features about the sequence (similar to the frequent words such as 'a' and 'the' in English sentences). On the other hand, these common smooth q-grams will contribute to false positives and consequently increase the running time of subsequent steps. As a result, with a properly chosen filtering threshold η, we could reduce the running time without much loss on the sensitivity of detecting true similar pairs.

3.3 Detection and verification

In the detection and verification stage (Line 16–31 of Algorithm 2), we use the smooth q-gram signatures to detect candidate overlaps, verify each of them and output the true overlaps. Below, we describe the detection and verification steps separately.

3.3.1 Detection

In the detection step, we find, for each pair of input sequences (x_i, x_j), their set of matching q-grams (i.e. those with edit distances less than or equal to K), using Algorithm 3. More precisely, in Algorithm 3, we try to find for a q-gram s, an (incomplete) list of matching q-grams s' by considering all q-gram s' falls into the same bucket in hash table D (Line 2). We then check if their edit distance $\mathrm{ED}(s, s') \leq K$; if so, then we record the pair and their positions into $\mathcal{L}$. Finally, we add the signature of s into the table D to build the table gradually (Line 7).

We record all the matches between each pair of sequences (x_i, x_j) in $\mathcal{M}_{i,j}$, where a match (u, v) indicates the u-th q-gram of x_i matches the v-th q-gram of x_j. For each pair of sequences with at least C matched q-grams, we move on to the verification step to see if this pair indeed share an overlap of a significant length, and if it is the case, we also identify the shared region in the pair.

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 overlap region $x_i[p_i^s, p_i^e]$ and $x_j[p_j^s, p_j^e]$}
1: Initialize an empty table D
2: **for each** $i \in [n]$ **do**
3: $\quad \Delta_i \leftarrow \varnothing$
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_1, R_2)
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: **for each** $i \in [n]$ **do**
13: $\quad$ Construct Δ_i' from Δ_i by keeping signatures in Δ_i with the smallest $\alpha|x_i|$ of hash ranks r.
14: **end for**
15: Count for all t the number c_t of signatures in the form of $(\cdot, t, \cdot, \cdot, \cdot)$ in $\cup_{i\in[n]} \Delta_i'$, and remove all (s, t, r, i, p) in Δ_i' with c_t in the top η of all t for all $i \in [n]$
16: **for each** $i \in [n]$ **do**
17: $\quad \mathcal{L}_i \leftarrow \varnothing$
18: $\quad$ **for each** δ in Δ_i' **do**
19: $\quad\quad \mathcal{L}_i \leftarrow \mathcal{L}_i \cup$ Search-Similar-q-Grams(δ, D)
20: $\quad$ **end for**
21: $\quad$ **for each** $j < i$ **do**
22: $\quad\quad \mathcal{M}_{ij} \leftarrow \{(u, v) \mid (j, u, v) \in \mathcal{L}_i\}$
23: $\quad$ **end for**
24: **end for**
25: **for each** $|\mathcal{M}_{ij}| \geq C$ **do**
26: $\quad (o, L_e) \leftarrow$ Verify(x_i, x_j, $\mathcal{M}_{ij}$)
27: $\quad$ **if** $(o, L_e) \neq$ *null* **then**
28: $\quad\quad (p_i^s, p_i^e, p_j^s, p_j^e) \leftarrow$ Find-Shared-Substrings(x_i, x_j, o, L_e, Δ_i, Δ_j)
29: $\quad\quad \mathcal{O} \leftarrow \mathcal{O} \cup (x_i, x_j, [p_i^s, p_i^e], [p_j^s, p_j^e])$
30: $\quad$ **end if**
31: **end for**

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

Input: $\delta = (s, t, r, i, p)$: a signature for q-gram s (see Definition 1 for detailed explanation of the parameters); D: a table with buckets indexed by t;
Output: $\mathcal{L} \leftarrow \{(i', p, p') \mid \exists \delta' = (s', t, r, i', p') \in D \, s.t. \, \mathrm{ED}(s, s') \leq K\}$
1: $\mathcal{L} \leftarrow \varnothing$
2: **for each** $\delta' = (s', t, r, i', p')$ stored in $D(t)$ **do**
3: $\quad$ **if** $\mathrm{ED}(s, s') \leq K$ **then**
4: $\quad\quad \mathcal{L} \leftarrow \mathcal{L} \cup (i', p, p')$
5: $\quad$ **end if**
6: **end for**
7: Add δ to the $D(t)$
8: **return** $\mathcal{L}$

3.3.2 Verification

In the verification step, we employ two subroutines: Algorithm 4 to perform a basic verification that reports approximate overlap shift and length, and Algorithm 5 to compute the actual overlap regions. We describe Algorithms 4 and 5 in more detail.

Let $\mathcal{M}$ be the list of starting positions of the matching q-gram pairs in the input strings x_i and x_j. We construct a bipartite graph $G_{i,j}$ with characters of x_i as nodes on the left part and characters of x_j as nodes on the right part. For each matching pair (u, v), an edge is connected between the nodes $x_i[u]$ and $x_j[v]$, denoted as (u, v), where $(u - v)$ represents the *shift* corresponding to the edge. Intuitively, if x_i and x_j overlap, there must be a large cluster of edges with similar shifts in $G_{i,j}$.

Algorithm 4 consists of two filtering steps. In the first step, we aim to find a dense cluster of edges with similar shifts and remove the remaining edges (Line 1–2). This could be implemented by finding an interval I_o containing the maximum number of edges whose width is adjusted according to the maximum error rate and the minimum length of overlaps we want to detect. We set the default error rate tolerance ϵ to be 0.15, as PacBio and Nanopore reads have error rate around 15% (Berlin *et al.*, 2015; Jain *et al.*, 2015; Li, 2018). We then use the remaining edges to estimate the shift and the length of overlap (o, L_e) by finding a reference edge (u_m, v_m) with a median shift over all remaining edges (Line 3–5). In the second step, we try to find a dense region of edges with similar positions on x_i and remove all the edges remaining (Line 6–7). This could be implemented again by finding an interval I_{pos} containing the maximum number of edges. Finally, we count the number of unremoved edges: if the number is at least C, then we consider (x_i, x_j) to be an overlap candidate pair and return its approximate shift and overlap length; otherwise, we simply return *null* (Line 8–12).

Algorithm 5 computes the locations of shared substrings between x_i and x_j more precisely. In this step, we exploit the *complete* set of matching q-gram pairs in order to generate more accurate overlap. We first construct the list $\mathcal{M}$ of matching q-grams, by a synchronized linear scan on the two sets Δ_i and Δ_j after sorting the tuples based on their r values. Then, we remove edges with shifts far deviated from our estimate (Line 2). Next, we perform a two-stage merging process to identify shared substrings. We represent a cluster of edges after merging as a *window* where window $(p_i^s, p_i^e, p_j^s, p_j^e)$ means $x_i[p_i^s, p_i^e]$ and $x_j[p_j^s, p_j^e]$ are the shared substrings in x_i and x_j. We use *cur* to represent the current window to be considered. In the first merging process (Line 4–15), we determine if an edge (p, p') could be merged with *cur*. Considering the 'boundary' of *cur*: $cur[2], cur[4]$ (the ending positions of shared substrings on both sequences) and (p, p'), we compute their difference of shifts d and difference of locations *step* on both sequences. If d is no greater than $\epsilon \cdot step$ (Line 9), then we merge the edge with *cur*. Otherwise, we create a new window *cur* starting from the edge and store the previous window in $\mathcal{W}$. By the end of the first merge step, we have a collection of windows stored in $\mathcal{W}$. In the second merging process (Line 16–28), we merge adjacent windows if they satisfy similar criteria and store the new windows in $\mathcal{W}'$. This process is mainly designed for Nanopore datasets where overlap regions are often broken into smaller sub-regions due to continuous sequencing errors. Finally, we pick the largest window in $\mathcal{W}'$ as our output.

4 Results

We present our experimental studies in the section. We first introduce the setup of our experiments and then present our results.

4.1 Setup of experiments

4.1.1 Datasets

We test all algorithms on both simulated and real-world datasets (PacBio and Nanopore), on four genomes: *Escherichia coli*, *Saccharomyces cerevisiae*, *Drosophila melanogaster* and *Human*. The PacBio datasets are downloaded from MHAP's supporting data website (http://www.cbcb.umd.edu/software/PBcR/mhap). The Nanopore datasets are downloaded from the Sequence Read

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: estimated shift
L_e: estimated overlap length

$I_o \leftarrow [o_s, o_s + 2\epsilon \cdot L]$ s.t. $|\{(u,v) \mid (u,v) \in \mathcal{M}, (u-v) \in I_o\}|$ is maximized
Remove all pairs $(u, v) \in \mathcal{M}$ whose $(u - v) \notin I_o$
Find $(u_m, v_m) \in \mathcal{M}$ with median $(u - v)$ value among all $(u, v) \in \mathcal{M}$
$o \leftarrow u_m - v_m$
$L_e \leftarrow \min\{u_m, v_m\} + \min\{|x_i| - u_m, |x_j| - v_m\}$
$I_{pos} \leftarrow [o_s, o_s + L]$ s.t. $|\{(u,v) \mid (u,v) \in \mathcal{M}, u \in I_{pos}\}|$ is maximized
Remove all pairs $(u, v) \in \mathcal{M}$ whose $u \notin I_{pos}$
if $|\mathcal{M}| < C$ **then**
  return *null*
else
  return $(o, \max(L_e, L))$
end if

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

Input: x_i, x_j: two input strings;
o: estimated shift;
L_e: estimated overlap length
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

$\mathcal{M} \leftarrow \{(p, p') \mid (s, t, r, i, p) \in \Delta_i, (s', t', r', j, p') \in \Delta_j, t = t'\}$
$\mathcal{M}' \leftarrow \{(p, p') \in \mathcal{M} \mid (p - p') \in [o - \epsilon \cdot L_e, o + \epsilon \cdot L_e]\}$
Sort matches $(p, p') \in \mathcal{M}$ using p in the increasing order
$\mathcal{W} \leftarrow \{\}$
$cur \leftarrow (p[1], p[1], p'[1], p'[1])$, where $(p[1], p'[1])$ is the first element in $\mathcal{M}'$, we represent the elements of cur as $cur[i] (i \in [4])$
for each $(p, p') \in \mathcal{M}'$ **do**
  $d_1 \leftarrow p - cur[2], d_2 \leftarrow p' - cur[4]$
  $d \leftarrow |d_1 - d_2|, step \leftarrow \max(d_1, d_2)$
  if $d_2 \geq 0 \wedge (d \leq \epsilon \cdot step)$ **then**
    $cur \leftarrow (cur[1], p, cur[3], p')$
  else
    $\mathcal{W} \leftarrow \mathcal{W} \cup \{cur\}$
    $cur \leftarrow (p, p, p', p')$
  end if
end for
$\mathcal{W}' \leftarrow \{\}$
$cur \leftarrow \mathcal{W}[1]$, where $\mathcal{W}[1]$ is the first element in $\mathcal{W}$
for each $(p_i^s, p_i^e, p_j^s, p_j^e) \in \mathcal{W}$ **do**
  $d_1 \leftarrow p_i^s - cur[2], d_2 \leftarrow p_j^s - cur[4]$
  $l_1 \leftarrow \frac{cur[2]+cur[4]-cur[3]-cur[1]}{2}, l_2 \leftarrow \frac{p_i^e+p_j^e-p_i^s-p_j^s}{2}$
  $d \leftarrow |d_1 - d_2|, step \leftarrow \frac{d_1+d_2}{2}$
  if $(step < \max(l_1, l_2)) \vee (d \leq 2\epsilon \cdot step)$ **then**
  $cur \leftarrow (cur[1], \max(cur[2], p_i^e), \min(cur[3], p_j^s), \max(cur[4], p_j^e))$
  else
    $\mathcal{W}' \leftarrow \mathcal{W}' \cup \{cur\}$
    $cur \leftarrow (p_i^s, p_i^e, p_j^s, p_j^e)$
  end if
end for
return $(p_i^s, p_i^e, p_j^s, p_j^e) \in \mathcal{W}'$ with largest $\frac{p_i^e+p_j^e-p_i^s-p_j^s}{2}$

Archive at the National Center for Biotechnology Information website (https://www.ncbi.nlm.nih.gov). The details of these datasets are described in Supplementary Material S1.3.

We randomly sample 50 000 reads from each dataset in our experiments. The details of the subsampled datasets are shown in Table 2. We test algorithms on subsampled datasets because this can show the accurate performance of the overlap detection task while allowing all algorithms to finish in a reasonable amount of time. See Supplementary Material S1.6 for details.

We also test simulated datasets generated by NanoSim (Yang *et al.*, 2017), a read simulator designed for Oxford MinION sequencers. Given the reference genome and a Nanopore dataset of the same species, NanoSim first analyzes the errors in the dataset to generate its error profile by mapping the dataset to the reference genome, and then utilizes the error profile to produce the set of simulated reads. For each simulated read, we know its exact mapping positions on the reference genome, and thus can use it as ground truth to test the performance of algorithms.

4.1.2 Measurements

We report *recall*, *precision*, *F_1 score*, *CPU time* and *memory usage* in our experiments. To compute the precision and recall, we need to know ground truth, i.e. whether a pair of sequences overlaps or not. For the simulated datasets, we know the mapping of each read to the reference genome. For the real-world datasets, we compute such mapping using Blasr (Chaisson and Tesler, 2012).

We use the evaluation module in MHAP to calculate the recall and precision. It takes mapping positions of the reads as input and treats the read pairs sharing at least Γ bps region on the reference genome as true overlaps. To compute the recall, if a true overlap is reported by an overlap detection algorithm and the length difference between the reported and true overlaps is within 30% of the true overlap length, we consider this pair as a true positive. The evaluation program verifies all the true overlap pairs to compute recall. To compute the precision, the evaluation program first randomly samples 10 000 reported overlaps. Each sampled overlap pair is considered as true positive if the edit distance between the reported overlap region is not greater than 30% of the reported overlap length. The 30% criteria on both measurements comes from the 15% error rate in sequencing procedure since, in the worse case, two overlapping reads may contain a 30% fraction of error. We set the parameter Γ to be 500. We also compute the recall and precision for overlaps with lengths $[500, 2000]$ bps to show the advantage of SmoothQGram on short overlaps.

We compute F_1 score to measure the overall performance of algorithms: $F_1 = 2 \times (\text{precision} \times \text{recall})/(\text{precision} + \text{recall})$, which is an integrated metric evaluating both precision and recall.

We also report the running time and memory usage of all the algorithms. Since all algorithms use multiple threads in execution, we set the number of threads to be 16 for each algorithm and measure the CPU time for comparison. All experiments were conducted on Dell PowerEdge T630 server with two Intel Xeon E5-2667 v4 3.2 GHz CPU with eight cores each and 256 GB memory.

4.1.3 Algorithms and parameter selections

We compare SmoothQGram presented in Algorithm 2 with the state-of-the-art algorithms MHAP (Berlin *et al.*, 2015), Minimap2 (Li, 2016, 2018) and DALIGNER (Chaisson and Tesler, 2012). (The implementation of MHAP is obtained from https://github.com/marbl/MHAP. The implementation of Minimap2 is obtained from

Table 2. Statistics of subsampled datasets

Dataset	#Reads	Coverage	Avg. length	Median length	Substitution error (%)	Insertion error (%)	Deletion error (%)
	PacBio datasets						
E.coli	47 910	84	8283.8	7477.5	2.11	8.35	3.37
S.cerevisiae	50 000	24	6038.8	5113.0	2.50	8.16	4.12
D.melanogaster	50 000	3	9324.8	8028.5	2.45	7.98	3.66
Human	50 000	0.11	7459.4	6533.0	3.03	8.88	4.63
	Nanopore datasets						
E.coli	50 000	121	11407.7	9082.5	6.46	5.52	8.98
S.cerevisiae	50 000	30	7421.8	6440.0	7.38	5.35	8.30
D.melanogaster	50 000	3	7009.0	4675.0	5.71	4.43	6.23
Human	50 000	0.10	6633.3	6763.0	4.12	4.90	4.84

https://github.com/lh3/minimap2. The implementation of DALIGNER is obtained from https://github.com/thegenemyers/DALIGNER.) We note that all these algorithms are using 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 main difference between SmoothQGram and the other algorithms is that we have *relaxed* the strict q-gram matches to approximate q-gram matches (via smooth q-gram) to improve the sensitivity of overlap detection.

In the experiments, we run SmoothQGram with parameters $q = 14$, $m = 16$, $\kappa = 35$, $\alpha = 0.2$, $K = 2$, $C = 5$, $\epsilon = 0.15$ and $L = 500$. We run MHAP with the parameter '-num-hash 1256', which corresponds to its sensitive mode that achieves higher sensitivity and F_1 scores. For DALIGNER, we add the parameter '-H500', which considers reads with the length at least 500 bps to improve its efficiency. Minimap2 provides two different sets of parameters for PacBio and Nanopore datasets. We use its default parameter set '-x ava-pb' for PacBio datasets and '-x ava-ont' for Nanopore datasets.

Under the default parameter set, Minimap2 runs extremely fast, but the recall is not satisfactory. We find that the performance of Minimap2 is mostly influenced by two parameters: the filter threshold ('-f') and the window size ('-w'). The filter threshold controls the fraction of frequent signatures that the algorithm ignores, and the window size controls the size of consecutive q-grams in which one signature is generated. The lower these two parameters are, the more signatures are retained in the indexing. We try different combinations of these two parameters and find '-f = 1e−7, -w = 3' achieves the highest F_1 scores on most datasets. This is also the set of parameters that achieves the highest recall with the precision no smaller than 80%. Thus, beside the default parameters, we also compare the results from this alternative parameter set for Minimap2. We call the default version MM2-default and the alternative version MM2-alternative. Minimap2 under the alternative parameter set has the best recall compared with all other existing 'seed-extension' methods, but SmoothQGram still outperforms Minimap2.

We would like to note that we only select a single set of parameters for SmoothQGram to show its robustness. However, one could adopt different parameters according to datasets to further improve the sensitivity or running time of SmoothQGram. For instance, on the *E.coli* dataset, we could decrease α and η to select fewer signatures and reduce the running time, whereas on *Human* dataset, we could increase the α and η to improve our precision and recall.

4.2 Experimental results

4.2.1 Performance comparison

Table 3 shows the performance comparison (recall, precision, F_1 score, running time and memory usage) of all algorithms. As we can see, SmoothQGram always achieves the best F_1 score and has about a 3.8% advantage over the best competitor on average (over all real datasets). We also achieve the best recall on all datasets and the second-best precision on all real datasets except PacBio *Human* which ranks third. Among all the state-of-the-art algorithms, MM2-alternative always achieves the best F_1 score, followed by MHAP in most cases.

Our advantage is greater when we consider the short overlaps with lengths $[500, 2000]$ where we gain on average about a 7.0% improvement on recall compared with the best competitor. Short overlaps are relatively not easy to detect as they contain fewer matched q-grams. Compared with existing algorithms, SmoothQGram using smooth q-grams as seeds is more sensitive and can capture short overlaps with higher probability. We observe that short overlap is the *bottleneck* for all the overlap detection algorithms: the recall on short overlaps is typical 20% lower than the overall recall for all algorithms.

For different datasets and species, we find that *E.coli* is the easiest one for all algorithms while *D.melanogaster* and *Human* are much more challenging, because there are larger numbers of repeated regions in these genomes. SmoothQGram is more robust on difficult datasets than the competitors. Moreover, we find that Nanopore datasets are more challenging than PacBio datasets for SmoothQGram. We notice that the overlaps in Nanopore are more likely to be broken into smaller sub-regions, which are more difficult to handle by our verification algorithm (Algorithm 4).

SmoothQGram's running time is similar to MHAP but is worse than MM2-default. The reason is that instead of q-gram, SmoothQGram considers smooth q-gram, and thus it takes more time to Generate-Smooth-q-Gram signatures and verify candidate overlaps. On the other hand, this is also why SmoothQGram significantly improves the sensitivity for overlap detection. We note that in our experimental studies, we mainly focused on the accuracy; our codes were not optimized for running time.

We present the experimental results on simulated datasets in Supplementary Material S1.4.

4.2.2 The detection of short overlaps

When we focus on the detection of short overlaps (length $500 - 2000$ bps), we observe from Table 3 that SmoothQGram consistently outperforms other algorithms on all datasets. We would like to emphasize that the detection of short overlaps is critical in sequence assembly. This is because for long overlaps, the common mistake in the overlap detection is outputting overlaps with wrong lengths/locations, which may be corrected in the subsequent layout step. However, if short overlaps are missed in the overlap detection stage, they will not be recovered in future stages, which may induce more contigs in the final assembly, especially when the sequence coverage is not high.

4.3 Summary

In this section, we provide a thorough experimental study on smooth q-gram and its application to overlap detection. We observe that the smooth q-gram-based approach achieved better accuracy than the conventional q-gram-based approaches for overlap detection, which is mainly due to the fact that smooth q-gram is capable of capturing approximate matches between two sequences. Our algorithm is stable and robust on genome sequences that we have

Table 3. Comparison of overlap detection algorithms on real datasets

	Recall< 2000 (%)	Precision< 2000 (%)	Recall (%)	Precision (%)	F_1 score	CPU (s)	Memory (G)
	PacBio *E.coli*						
MHAP	34.52	99.88	60.93	99.77	0.757	3786	20.4
DALIGNER	26.36	95.24	58.73	97.41	0.733	1476	8.4
MM2-default	61.11	99.96	81.12	99.97	0.896	195	5.1
MM2-alternative	67.79	99.85	85.01	99.89	0.919	332	7.5
SmoothQGram	73.11	99.85	87.98	99.93	**0.936**	6550	30.6
	PacBio *S.cerevisiae*						
MHAP	39.86	96.67	53.97	93.09	0.683	4176	22.2
DALIGNER	10.07	89.90	31.97	93.21	0.476	1588	5.8
MM2-default	8.00	99.83	12.42	99.79	0.221	74	3.4
MM2-alternative	50.80	95.64	69.35	93.28	0.796	1230	9.0
SmoothQGram	59.49	99.25	75.00	99.34	**0.855**	4274	24.3
	PacBio *D.melanogaster*						
MHAP	28.52	93.16	41.11	90.63	0.566	3971	22.3
DALIGNER	21.88	69.94	48.93	76.26	0.596	60 643	9.3
MM2-default	17.45	98.64	28.48	98.33	0.442	123	5.1
MM2-alternative	47.52	92.94	59.66	88.02	0.711	49 459	32.8
SmoothQGram	57.00	95.55	65.87	96.98	**0.785**	7976	35.4
	PacBio *Human*						
MHAP	16.14	74.70	29.41	72.79	0.419	4152	22.3
DALIGNER	11.21	61.34	31.71	64.47	0.425	2841	7.3
MM2-default	15.09	99.45	27.21	99.21	0.427	100	4.6
MM2-alternative	28.30	97.77	46.19	95.62	0.623	354	6.0
SmoothQGram	33.26	96.43	51.36	95.4	**0.667**	5551	29.5
	Nanopore *E.coli*						
MHAP	28.87	97.06	65.33	96.93	0.781	5887	21.8
DALIGNER	10.79	88.62	32.71	92.86	0.484	4928	12.0
MM2-default	33.38	99.77	73.89	99.41	0.848	390	8.4
MM2-alternative	38.81	96.82	76.53	97.06	0.856	880	11.9
SmoothQGram	48.33	95.80	79.63	97.40	**0.876**	14 402	45.7
	Nanopore *S.cerevisiae*						
MHAP	31.98	98.69	56.04	99.21	0.716	5251	22.0
DALIGNER	3.32	95.85	14.83	97.83	0.258	3941	5.4
MM2-default	4.80	99.29	7.65	99.26	0.142	235	5.6
MM2-alternative	33.48	98.00	70.61	97.32	0.818	3490	22.1
SmoothQGram	41.78	95.77	71.25	97.45	**0.823**	6555	33.4
	Nanopore *D.melanogaster*						
MHAP	53.05	90.71	56.39	89.52	0.692	3713	22.2
DALIGNER	14.97	79.26	31.97	83.21	0.462	18 259	8.1
MM2-default	15.97	98.30	29.19	97.79	0.450	140	5.2
MM2-alternative	57.09	95.00	65.57	92.03	0.766	6416	22.2
SmoothQGram	61.54	93.95	67.47	95.49	**0.791**	4812	27.1
	Nanopore *Human*						
MHAP	28.59	39.62	52.04	42.21	0.466	11 170	22.4
DALIGNER	19.67	71.21	44.46	75.34	0.559	3642	7.6
MM2-default	23.33	99.38	33.03	99.45	0.496	113	5.1
MM2-alternative	32.61	84.28	65.40	81.03	0.724	2587	11.1
SmoothQGram	38.16	93.56	67.36	93.36	**0.783**	5480	25.9

Note: Recall< 2000 and precision< 2000 show the measurements for overlaps of length 500 to 2000 bps. Recall and precision show the measurements for all true overlaps.

Highest F_1 score should be highlighted in Bold.

tested. The performance of our algorithm is more notable for short overlaps, and may improve the overall qualify of long-reads assembly.

5 Discussions

In this paper, we present SmoothQGram, a highly sensitive overlap detection algorithm for long, error-prone sequencing reads. We first propose smooth q-gram, a variant of q-gram which can capture q-grams with small edit distances, and then show that by using smooth q-gram-based signatures, the sensitivity of the overlap detection algorithm can be significantly improved. We also observe that SmoothQGram has greater advantage in detecting short overlaps, which has strong implications in improving the overall quality of genome assembly using long-reads.

SmoothQGram achieves robust performance on all the datasets with a single set of parameters. We may obtain even better sensitivity or running time if we adopt parameters for different datasets. In contrast, the commonly used Minimap2 used different parameters for PacBio and Nanopore data, and we have to select additional

parameters to make its sensitivity satisfying. We observe that its results are highly sensitive with parameters.

Although we have only studied the overlap detection problem in this paper, we believe that smooth q-gram can be used in other cases, e.g. the other steps in fragment assembly (e.g. error corrections, consensus sequence generation) as well as other similarity search problems (e.g. sequence classification, sequence clustering), where sensitive and robust q-gram signatures are needed. We leave these applications as future work.

Funding

This work has been supported in part by the NSF CCF-1619081, IIS-1633215 and CCF-1844234.

Conflict of Interest: none declared.

References

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Belazzougui,D. and Zhang,Q. (2016) Edit distance: sketching, streaming, and document exchange. In: *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pp. 51–60.

Berlin,K. *et al.* (2015) Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.*, **33**, 623–630.

Brudno,M. *et al.* (2003) Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.*, **13**, 721–731.

Burkhardt,S. and Kärkkäinen,J. (2001) Better filtering with gapped q-grams. In: *Combinatorial Pattern Matching, 12th Annual Symposium, CPM2001 Jerusalem, Israel, July 1–4, 2001, Proceedings (Lecture Notes in Computer Science)*, Vol. **2089**, pp. 73–85.

Burkhardt,S. and Kärkkäinen,J. (2002) One-gapped q-gram filters for Levenshtein distance. In: *Combinatorial Pattern Matching, 13th Annual Symposium, CPM 2002, Fukuoka, Japan, July 3-5, 2002, Proceedings (Lecture Notes in Computer Science)*, Vol. **2373**, pp. 225–234.

Burkhardt,S. and Kärkkäinen,J. (2003) Better filtering with gapped q-grams. *Fundam. Inform.*, **56**, 51–70.

Chaisson,M. and Tesler,G. (2012) Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): theory and application. *BMC Bioinform.*, **13**, 238.

Chakraborty,D. *et al.* (2016) Streaming algorithms for embedding and computing edit distance in the low distance regime. In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC2016, Cambridge, MA, USA, June 18–21, 2016*, pp. 712–725.

Compeau,P.E. *et al.* (2011) How to apply de Bruijn graphs to genome assembly. *Nat. Biotechnol.*, **29**, 987–991.

Jain,M. *et al.* (2015) Improved data analysis for the minion nanopore sequencer. *Nat. Methods*, **12**, 351–356.

Keich,U. *et al.* (2004) On spaced seeds for similarity search. *Discrete Appl. Math.*, **138**, 253–263.

Kurtz,S. *et al.* (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.

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

Li,H. (2018) Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, **34**, 3094–3100.

Ma,B. *et al.* (2002) Patternhunter: faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.

Manning,C.D. and Schütze,H. (1999) *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Mikheyev,A.S. and Tin,M.M. (2014) A first look at the Oxford Nanopore minion sequencer. *Mol. Ecol. Resourc.*, **14**, 1097–1102.

Miller,J.R. *et al.* (2010) Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315–327.

Myers,G. (2014) Efficient local alignment discovery amongst noisy long reads. In: *Algorithms in Bioinformatics – 14th International Workshop, Wroclaw, Poland, Proceedings (Lecture Notes in Computer Science)*, Vol. **8701**, pp. 52–67.

Pevzner,P.A. *et al.* (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. USA*, **98**, 9748–9753.

Qin,J. *et al.* (2011) Efficient exact edit similarity query processing with the asymmetric signature scheme. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece*, June 12–16, 2011, pp. 1033–1044.

Roberts,R.J. *et al.* (2013) The advantages of SMRT sequencing. *Genome Biol.*, **14**, 405.

Schwartz,S. *et al.* (2003) Human–mouse alignments with BLASTZ. *Genome Res.*, **13**, 103–107.

Sović,I. *et al.* (2016) Fast and sensitive mapping of nanopore sequencing reads with graphmap. *Nat. Commun.*, **7**, 11307.

Wang,J. *et al.* (2012) Can we beat the prefix filtering? An adaptive framework for similarity join and search. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20–24, 2012*, pp. 85–96.

Xiao,C. *et al.* (2008) Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB*, **1**, 933–944.

Yang,C. *et al.* (2017) NanoSim: nanopore sequence read simulator based on statistical characterization. *GigaScience*, **6**, 1–6.

Zhang,H. and Zhang,Q. (2017) Embedjoin: efficient edit similarity joins via embedding. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13–17, 2017*, pp. 585–594.