Extraction of long k-mers using spaced seeds

Leinonen, Miika

2022-11

Leinonen , M & Salmela , L 2022 , ' Extraction of long k-mers using spaced seeds ' , IEEE/ACM Transactions on Computational Biology and Bioinformatics , vol. 19 , no. 6 , pp. 3444-3455 . https://doi.org/10.1109/TCBB.2021.3113131

http://hdl.handle.net/10138/351833
https://doi.org/10.1109/TCBB.2021.3113131

Extraction of Long k-mers Using Spaced Seeds

Miika Leinonen and Leena Salmela

Abstract—The extraction of k-mers from reads is an important task in many bioinformatics applications, such as all DNA sequence analysis methods based on de Bruijn graphs. These methods tend to be more accurate when the used k-mers are unique in the analyzed DNA, and thus the use of longer k-mers is preferred. When the read lengths of short read sequencing technologies increase, the error rate will become the determining factor for the largest possible value of k. Here we propose LoMeX which uses spaced seeds to extract long k-mers accurately even in the presence of sequencing errors. Our experiments show that LoMeX can extract long k-mers from current Illumina reads with a similar or higher recall than a standard k-mer counting tool. Furthermore, our experiments on simulated data show that when the read length further increases enabling even longer k-mers, the performance of standard k-mer counters declines, whereas LoMeX still extracts long k-mers successfully.

Index Terms—k-mers, k-mer counting, spaced seeds

1 Introduction

Counting and extracting k-mers from reads is a frequently used technique in bioinformatics applications and many tools have been developed to solve this task [28]. A k-mer counter needs to enumerate all different subsequences of length k that occur in the reads and report the frequency of each such k-mer.

Counting k-mers has several applications in bioinformatics. In the overlap-layout-consensus approach to genome assembly, k-mers can be used to identify candidate pairs of overlapping reads by finding reads that share a substantial amount of k-mers [2], [32]. These candidate read pairs are then further verified for actual overlaps by aligning them. In the de Bruijn graph-based approaches [12], [37], [42], [46], k-mer counting is the first step as it identifies the distinct k-mers occurring in the reads that will become the edges of the de Bruijn graph, whereas the $k-1$-mers will form the nodes.

Correction of sequencing errors in reads is another application where k-mer counting plays an important role. The correction procedure may be entirely based on the k-mer spectrum of the reads [17], [25], [30] or k-mers can be used to filter reads for multiple alignments [41]. Other approaches rely on de Bruijn graphs which are built on k-mer sets [40]. Other applications of k-mer counting include metagenomic classification [45], repeat classification [6], [21], and SNV calling directly from read data [33], [44].

It is generally beneficial to be able to count long k-mers because the longer the k-mers are, the more likely they are unique in the genome. For example, a de Bruijn graph will be simpler with fewer branches when the k-mers are longer. If the k-mers are short, the de Bruijn graph will be more complex having multiple branching paths, which makes it difficult to infer with high confidence which long sequences are present in the genome. However, if k is too big, the unique k-mer abundance starts to drop and the graph becomes too fragmented. The optimal choice of k has been studied [8], but it is a difficult task to estimate the best choice of k. Nevertheless, the usage of long k-mers can be helpful in many bioinformatics applications.

The accurate short read sequencing technologies, such as Illumina, nowadays reach read lengths of 300 bp, which allow the use of longer k-mers. However, although the error rate of these technologies is low, the sequencing errors will become a limiting factor for determining the largest possible value of k in the standard k-mer counting methods when the read lengths further increase. The standard methods involve only counting how many times each k-mer appears in the data. A common approach is to require a k-mer to occur at least twice for it to be counted as a real k-mer existing within the data. With short enough k-mers one expects to find enough error-free occurrences of the k-mers but the likelihood of finding at least two error-free occurrences of a k-mer decreases as k increases. A higher coverage increases the likelihood of finding at least two error-free occurrences of a k-mer but producing high coverage data sets is more costly and might be infeasible if the required coverage is very high.

Development of k-mer counting methods has largely concentrated on time and memory efficiency. Much less attention has been given to improving the quality, i.e., getting long and accurate k-mers. Here we propose LoMeX (LOng k-MEr eXtraction) to extract long k-mers with high precision and recall. To increase the accuracy of the k-mer extraction process, we want to diminish the influence of sequencing errors. To achieve this, LoMeX uses string patterns called *spaced seeds* of matching length k. A spaced seed specifies which characters of a string are relevant. We call the relevant character positions *fixed positions*, and the remaining ones are called *gap positions* (sometimes known as "don't care"

- *The authors are with the Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki, 00014 Helsinki, Finland. E-mail: {miika.leinonen, leena.salmela}@helsinki.fi.*

Manuscript received 23 Oct. 2020; revised 23 Apr. 2021; accepted 9 Sept. 2021. Date of publication 16 Sept. 2021; date of current version 8 Dec. 2022.
This work was supported by the Academy of Finland under Grants 308030 and 323233 to LS.
(Corresponding author: Leena Salmela.)
Digital Object Identifier no. 10.1109/TCBB.2021.3113131

positions). We will denote the number of fixed positions with q (also known as the weight of the spaced seed), and the number of gap positions will be $k - q$. When we are searching for matches between k-mers using a spaced seed, only the characters in the q fixed positions are required to match. If two k-mers have the same characters in the fixed positions, they are treated as the same *spaced k-mer*. Afterward, the consensus of the k-mers associated with the same spaced k-mer are used to fill in the gaps, resulting in a *consensus k-mer*. This process allows us to ignore incorrect characters in the reads when they fall in a gap position because they are unlikely to affect the consensus of multiple k-mers.

We compared LoMeX to DSK [38] which is a standard k-mer extraction tool. Our results show that on current Illumina data LoMeX typically has similar or higher recall than DSK with a small drop in precision. Extracting k-mers with a high recall is valuable for downstream applications. While a missing k-mer cannot be recovered, it is possible to later discard erroneous k-mer information, for example by examining the tip and bubble structures of a de Bruijn graph. Furthermore, our experiments on simulated data show that when read lengths further increase, the error rate will become the limiting factor for choosing a large k in standard k-mer extraction, whereas LoMeX can still extract long k-mers successfully.

LoMeX is freely available at https://github.com/Denopia/LoMeX

2 Related Work

2.1 K-mer Counting

Many strategies have been developed to count the k-mers present in a set of reads. For example KMC3 [18], Turtle [39], and GenomeTester4 [15] use sorting to count k-mers. In this approach, all k-mers are extracted from the reads. The k-mers are then sorted and from the sorted list of k-mers, it is easy to count how many times each k-mer occurs.

An alternative method is to use a data structure to store the k-mers and their counts. Jellyfish [29] implements a lock-free hash table using a compare-and-swap assembly instruction to store the k-mers as keys and the counts as values. The lock-free data structure enables fast, parallel k-mer counting.

Other tools have employed approximate membership query (AMQ) data structures for more efficient k-mer counting. BFCounter [31] uses Bloom filters to filter out singleton k-mers and stores the non-singleton k-mers and their counts in a hash table. To account for the false-positives of the Bloom filter, it reiterates over the reads to correct the wrong counts in the hash table. Squeakr [35] uses counting quotient filters (CQF) to store the k-mer counts. Squeakr supports both exact and approximate k-mer counting.

Other data structures used by k-mer counters include enhanced suffix trees used by Tallymer [20] and burst tries used by KCMBT [27].

MSPKmerCounter [24], KMC3 [18], and Gerbil [10] use minimum string partitioning to further reduce memory usage. They partition the input strings into multiple disjoint partitions based on minimizers and store several consecutive k-mers sharing a minimizer as a single super k-mer. The disjoint partitions can then be processed independently to get the actual k-mer counts.

Many k-mer counters use a disk-based implementation to save memory costs. For example, DSK [38] first calculates the number of k-mer partitions it will need. The k-mers are then distributed to the partitions based on their hash values and an iteration number. The actual counting happens by loading one partition to memory at a time and counting the k-mers assigned to that partition.

Also, GPU computation has been used to speed up k-mer counting [10]. We refer the reader to [28] for a more detailed review of the various k-mer counting methods and their benchmarking.

2.2 Spaced Seeds

Determining if two sequences are similar is a central question in biology. Initially, such problems were solved by pairwise alignment of the two sequences but the quadratic dynamic programming algorithms for pairwise alignment soon became too costly when the number of sequences increased as the number of pairwise comparisons also grows quadratically. The introduction of seeds presented a solution to this problem. The main idea is that similar sequences share identical regions and thus identical seeds can be found in these areas.

First programs for homology search, such as BLAST [1], used matches of k-mers as seeds and then extended the seed matches to longer alignments. Spaced seeds [5], [9], [13], [16], [26] extend this concept by allowing gaps within the k-mer seeds. PatternHunter [26] proposed to optimize the predefined positions required to match and obtained a significantly better sensitivity than BLAST [1]. Furthermore, Buhler *et al.* [4], Ma *et al.* [26] and Brejová *et al.* [3] noticed that using several spaced seeds further increased the sensitivity. In practice, spaced seeds have been shown to have high sensitivity and specificity for homology search even when the spaced seeds are not optimized [23], [26]. More recently, Leimeister *et al.* [22] have shown that spurious matches can be further reduced by filtering matches based on characters in the gap positions.

Spaced seeds are effective in finding similar sequences when the sequences mainly differ by mismatches. However, as the frequency of insertions and deletions increases, the length and the number of common spaced seeds found in similar sequences decreases. Thus, the spaced seeds are no longer effective for identifying similar sequences which has been seen for example in long read alignment [7].

3 Definitions

We start with a formal definition of the k-mer extraction problem. Then we extend it to spaced k-mer extraction and finally to consensus k-mer extraction. Consensus k-mers are the long and accurate k-mers LoMeX produces as its output.

A *k-mer* is a sequence of k characters. A *canonical k-mer* is a k-mer that is lexicographically smaller than its reverse complement. Canonical k-mers are used in k-mer counting because the reads can originate from either strand of the DNA molecule and we only want to count a k-mer once regardless of its orientation. Suppose we have an 11-mer m = ACTCATAATCA. Its reverse complement is m'=TGATTATGAGT, which is lexicographically bigger than m. Thus m is the canonical k-mer of these two.

Problem 1 (k-mer Extraction). *Given a set of reads R and a threshold S, find all canonical k-mers that occur at least S times in the reads and their reverse complements.*

Now we can extend the notion of k-mers to spaced k-mers with the help of spaced seeds. A *spaced seed* is a pattern of zeros and ones where ones correspond to fixed characters, and zeros correspond to gap characters. The number of fixed positions is q, and the number of gap positions is $k-q$. For example, spaced seed $p = 10010101001$ ($q = 5$, $k-q = 6$) could be used to identify spaced 11-mers. With this, we are ready to define spaced k-mers.

Definition 2 (Spaced k-mer). *A spaced k-mer g adhering to a spaced seed p is a string of characters from {A,C,G,T,*} such that if $p[i] = 0$ then $g[i] = *$, and if $p[i] = 1$, then $g[i] \in \{A, C, G, T\}$.*

Because spaced seeds can be very long, it will sometimes save space to alternatively represent them as a sequence of integers, where the integers at odd positions indicate the number of consecutive fixed positions, and the integers at even positions indicate the number of consecutive gap positions. Thus the spaced seed $p = 10010101001$ could be represented as p =1-2-1-1-1-1-1-2-1.

As an example of spaced k-mers, suppose we have a spaced seed $p = 10010101001$ and an 11-mer m = ACTCATAATCA. Using spaced seed p on this 11-mer m yields spaced k-mer $g =$ A**C*T*A**A. Since the spaced seed is known, we can remove the gap characters without losing information, and represent the spaced k-mer compactly as $g =$ ACTAA.

When the spaced seed is palindromic, we can define a canonical spaced k-mer similar to canonical k-mers. A spaced k-mer is canonical if it is lexicographically smaller than its reverse complement. The spaced seed is required to be palindromic so that the spaced k-mers from both strands have the same fixed and gap positions. With this definition of canonical spaced k-mers, we can now define the spaced k-mer extraction problem:

Problem 3 (Spaced k-mer extraction). *Given a set of reads R, a threshold S, and a spaced seed p, find all canonical spaced k-mers that adhere to the spaced seed p and occur at least S times in the reads and their reverse complements.*

Each spaced k-mer reported by the solution to the spaced k-mer extraction problem has a set of at least S occurrences (regular k-mers) in the reads. These occurrences are used to determine which characters are solid at each position of the spaced k-mer.

Definition 4 (Solid Characters). *Let Q be the set of occurrences of a spaced k-mer g in a set of reads R and let c be the threshold for a character to be solid. Character $n_i \in \{A, C, G, T\}$ is a solid character at position i of spaced k-mer g, if n_i appears at least c times at position i in Q.*

We classify each position of a spaced k-mer as unambiguous, ambiguous, or undecided. The number of solid characters determines how a position is classified. A position is unambiguous if it has only one solid character, and if there is more than one solid character, the position is classified as ambiguous. If there are no solid characters, the position is undecided. We note that all fixed positions of a spaced k-mer are unambiguous, whereas gap positions can be in any of the three categories.

Definition 5 (Undecided, Unambiguous, and Ambiguous Positions). *Position i in spaced k-mer g is undecided if it has no solid characters. The position is unambiguous if it has exactly one solid character. If there is more than one solid character, the position is ambiguous.*

With the help of undecided, unambiguous, and ambiguous positions, we can define consensus k-mers.

Definition 6 (Consensus k-mer). *Consensus k-mer d is a k-mer which corresponds to a spaced k-mer g such that $d[i]$ is a solid character of g at position i. Additionally, for all ambiguous positions j in the spaced k-mer g, we require g to have at least two occurrences (regular k-mers) x and y such that $d[j] = x[j] = y[j]$.*

Because the characters are not independent of each other, we do not want to take all possible combinations of solid characters in ambiguous positions. For this reason, we require that there exist at least two occurrences that have the specific ambiguous position solid character combination before it is used to construct a consensus k-mer.

If a spaced k-mer has any undecided positions, it does not have a consensus k-mer. If all positions are unambiguous, there is only one possible consensus k-mer. If there is at least one ambiguous position, there can be more than one consensus k-mer. Fig. 2 shows examples of different consensus k-mer cases, which will be discussed in more detail in Section 4.1.

Finally, we are ready to define the consensus k-mer extraction problem:

Problem 7 (Consensus k-mer Extraction). *Find all consensus k-mers corresponding to spaced k-mers found in a read set R.*

The reliability of k-mers is measured by *k-mer counts*, i.e., the number of occurrences a k-mer has in the read set. Here we extend the notion of k-mer counts to consensus k-mer *support counts*. Each occurrence of spaced k-mer where all the ambiguous positions match a consensus k-mer exactly contributes to the support count of that consensus k-mer. One such occurrence can only support exactly one consensus k-mer. The remaining occurrences of the spaced k-mer are evenly distributed among all its consensus k-mers.

Definition 8 (Consensus k-mer Support Counts). *Let Q be the set of occurrences of a spaced k-mer g and D the set of consensus k-mers of g. Given a consensus k-mer $d \in D$, we denote by Q_d the set of occurrences where the ambiguous positions match d. The support count for consensus k-mer d is $|Q_d| + |(Q \setminus \bigcup_{d' \in D} Q_{d'})|/|D|$.*

We can see from the definition that the sum of the support counts for consensus k-mers of the same spaced k-mer is equal to $|Q|$, i.e., the number of its occurrences.

4 Methods

LoMeX pipeline can be divided into three steps. The first step solves the spaced k-mer extraction problem, the second step gathers k-mers corresponding to the extracted spaced

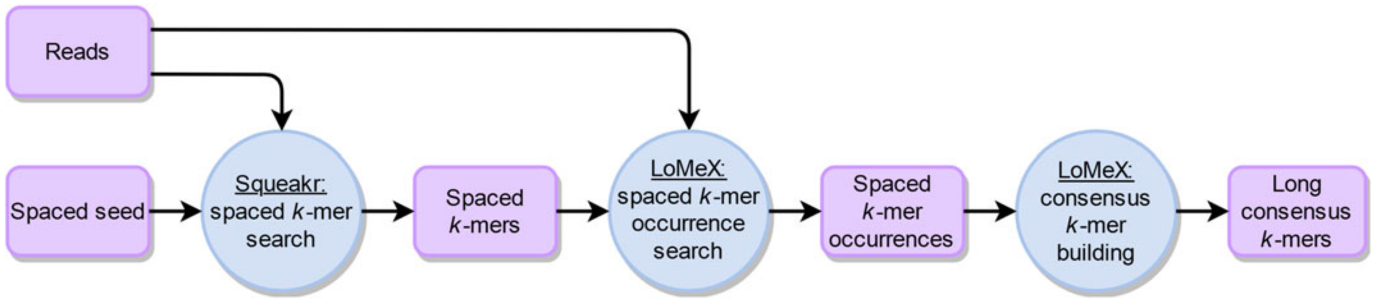
Reads
Spaced seed
Squeakr: spaced k-mer search
Spaced k-mers
LoMeX: spaced k-mer occurrence search
Spaced k-mer occurrences
LoMeX: consensus k-mer building
Long consensus k-mers

Fig. 1. Long k-mer extraction steps. First, the spaced k-mers are extracted from the reads according to the chosen spaced seed using a modified version of Squeakr k-mer counting program. Next, the reads are scanned once more while the extracted spaced k-mers are kept in memory. For each spaced k-mer reported by Squeakr, their occurrences (matching regular k-mers) are stored on disk. Finally, the long k-mers are built based on the consensus of the regular k-mers.

k-mers, and the third step solves the consensus k-mer extraction problem for each extracted spaced k-mer.

In the first step, a spaced seed is used to extract spaced k-mers from the read set. The purpose of this step is to filter out spaced k-mers that do not have enough occurrences. In the second step, the reads are scanned once more to find all occurrences of the spaced k-mers that appear often enough. The idea of this step is to group k-mers in the reads based on the chosen spaced seed. In other words, k-mers that yield the same spaced k-mer belong to the same group. In the third step, the long consensus k-mers are built with the help of the grouped k-mers. Specifically, we use the consensus of the grouped k-mers to fill the gap positions of the spaced k-mers to produce consensus k-mers, which are then reported as the output. Fig. 1 depicts all the steps of this process, which are explained in more detail in the following sections.

LoMeX aims to diminish the significance of erroneous characters i.e., substitution errors. If such an error occurs in a gap position of a k-mer, it does not affect the grouping of that k-mer. Furthermore, in the consensus step, one wrong character is unlikely to affect which characters are solid and used for filling the gap positions. This method also gets around unclear bases (such as N characters) if they appear in a gap position. If such a character appears in a fixed position, the k-mer will not be used.

4.1 Spaced K-mer Extraction

In the first step, LoMeX extracts all canonical spaced k-mers that adhere to a given spaced seed. As noted before, we are only considering palindromic spaced seeds. Furthermore, we use an odd number of fixed characters so that the spaced k-mers have an odd weight. This ensures that a sequence and its reverse complement cannot be the same, so only one of them can be the canonical spaced k-mer.

To find all occurrences of the spaced k-mers in the input reads, we use a modified version of an existing k-mer counting system called Squeakr [35]. Given a set of reads, threshold S, and length k, Squeakr solves the k-mer extraction problem and outputs all the k-mers that appear in the reads at least S times. A k-mer is required to appear more than once to get rid of some of the spurious k-mers that arise due to the sequencing errors in the reads. By default, Squeakr reports k-mers that appear at least $S = 2$ times.

We modified Squeakr so that instead of basing the search on k-mer length, it is based on a spaced seed. Squeakr uses the spaced seed to report the corresponding spaced k-mers that appear at least S times. Only characters in the fixed positions are reported, and Squeakr's exact k-mer counting implementation supports only k-mers up to length 32. Because of this and the fact that we want to use odd length spaced k-mers, we are limited to using at most 31 fixed positions. There exists other k-mer counting tools that could work with longer k-mers, but Squeakr was the easiest one for us to modify for our needs, so we decided to use it over the other programs.

4.2 Consensus K-mer Construction

After Squeakr output has been obtained, the reported spaced k-mers are given to LoMeX which solves the consensus k-mer extraction problem. Before that, for each reported spaced k-mer, LoMeX must find its occurrences in the input reads. This is done by looking at all the regular k-mers, and checking if they match a spaced k-mer reported by Squeakr. If this is the case, the regular k-mer is stored in memory associated with the matched spaced k-mer. Essentially, the k-mers in the read set are split into groups where the k-mers have the same characters in the fixed positions. If a k-mer does not match any spaced k-mer, it is not used.

After occurrences of the spaced k-mers have been found, LoMeX fills the gap positions with solid characters to construct consensus k-mers. For every spaced k-mer position, the occurrences of the four possible characters (A, C, G, and T) in the associated k-mers are counted. Then, the counts are used to determine which characters are considered solid at each position. A character is solid if it appears at least c times, where c is the minimum character count threshold. To make LoMeX work on different sized read sets, we take the number of regular k-mers into account when deciding a suitable value for c. LoMeX requires that c is at least ten percent of the number of all characters at the position, i.e., the character appears at least in ten percent of the regular k-mers at the specified position. Additionally, LoMeX has a hard minimum threshold N for the number of required occurrences, set to two. The minimum character count threshold for spaced k-mer g is defined in LoMeX as $c_g = \max(N, r \cdot |Q_g|)$, where $N = 2$ is the absolute minimum number of required character occurrences, $r = 0.1$ is the required proportion with respect to the number of regular k-mers, and Q_g is the set of regular k-mers corresponding to g. The user can set different values for the parameters N and r. The effect of N on the accuracy of LoMeX is explored more in Section 5.

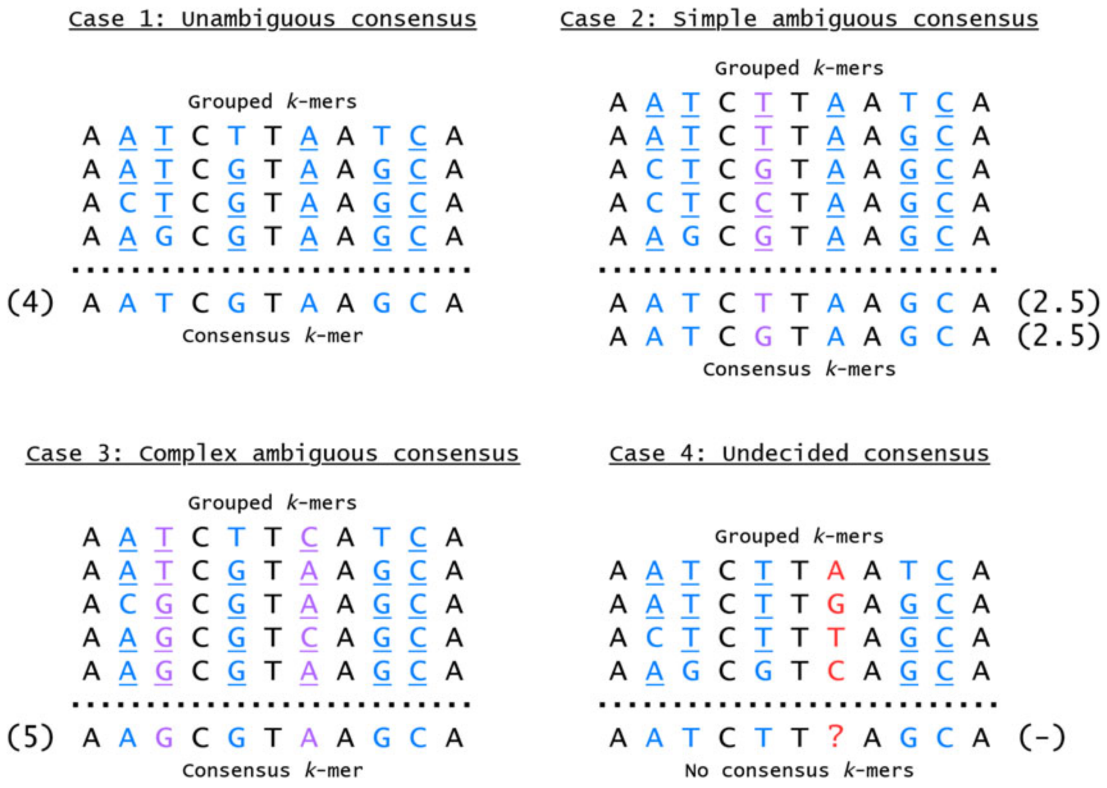

Fig. 2. Four different gap-filling cases. In this example, we are filling the gaps of the spaced k-mer A**C*T*A**A. Fixed characters are colored black. Gap characters are blue if only one character appears at least twice in its column. Gap characters are colored purple if there is more than one character that appears at least twice in its column. If none of the gap characters appears twice in its column, all the characters are colored red. A gap character is underlined if it appears more than once in its column. The numbers in brackets next to the consensus k-mers are their support counts.

Only solid characters are used to fill a gap. If there is only one character that has a high enough count to be solid, the position is unambiguous, and LoMeX simply uses that character to fill the position. If there are multiple solid characters in a single position, it becomes ambiguous, making the whole consensus k-mer building case ambiguous. If there is a position with no solid characters, the consensus k-mer building is left undecided. Consensus k-mer cases can be divided into four categories; unambiguous consensus, simple ambiguous consensus, complex ambiguous consensus, and undecided consensus. Examples of these cases are illustrated in Fig. 2.

1) *Unambiguous consensus.* Every position is unambiguous, i.e., there is only one character with a count greater than the required threshold c. The only solid characters are used to fill the gap positions, resulting in a single consensus k-mer.
2) *Simple ambiguous consensus.* Only one position is ambiguous, i.e., there are multiple solid characters for that position. All positions with only one solid character are filled as in the previous case. The ambiguous position is filled with all the solid characters, leading to multiple (two, three, or four) different consensus k-mers.
3) *Complex ambiguous consensus.* There are multiple ambiguous positions. The consensus k-mers are constructed by first filling the unambiguous gap positions. Then LoMeX looks for regular k-mers that have the same characters at the ambiguous gap positions. If LoMeX finds at least two k-mers that share the same characters at the ambiguous positions, those characters are used to fill the remaining gaps to produce a consensus k-mer. This case gives us at most $\lfloor\frac{|Q_g|}{2}\rfloor$ consensus k-mers, where Q_g is the set of regular k-mers associated with spaced k-mer g.
4) *Undecided consensus.* There is at least one position with no solid characters. This can happen when the number of regular k-mers corresponding to a spaced k-mer is very low. This results in zero reported consensus k-mers.

Even though there are three different cases where a consensus k-mer is formed, they all still follow our consensus k-mer definition in Section 3. The cases are separated because LoMeX handles them differently in practice based on the number of ambiguous positions. In the complex ambiguous consensus case, the regular k-mers must be analyzed to determine which solid characters can be used. In the unambiguous consensus and simple ambiguous consensus cases, we can determine how the gaps are filled just based on the solid characters.

The gaps in the spaced k-mers are filled with solid character to produce long consensus k-mers, which LoMeX reports as its output. Despite the fact that non-canonical spaced k-mers are discarded, this does not mean that the reported consensus k-mers are necessarily canonical. However, LoMeX only reports k-mers which correspond to canonical spaced k-mers and thus only a k-mer or its reverse complement is reported but never both. Therefore it is easy to transform the output of LoMeX so that only canonical k-mers are reported. One simply needs to check if a reported consensus k-mer is canonical and if it is not, report its reverse complementary sequence instead.

As their name implies, k-mer counters produce the extracted k-mers and how many times they appear in the input reads. We implemented a similar feature for LoMeX but instead of exact k-mer counts, LoMeX reports the support counts of the consensus k-mers as defined in Section 3. The support counts tell how strongly a produced consensus k-mer is supported by the spaced k-mer occurrences in the input data.

Here we define how support counts are calculated for the four different consensus cases:

1) *Unambiguous consensus.* Because all spaced k-mer occurrences support the same consensus k-mer, the support count is the number of spaced k-mer occurrences.
2) *Simple ambiguous consensus.* In this case, the produced k-mers differ only in a single position. The base support count becomes the number of spaced k-mer occurrences that have the matching character at this position. On top of these occurrences, some might have a non-solid character at the ambiguous position. The number of these occurrences is then divided equally between the produced k-mers and added to their base support counts.
3) *Complex ambiguous consensus.* This case is similar to the previous one. Here the base support count is the number of spaced k-mer occurrences that have identical ambiguous position characters as the produced k-mer. The leftover occurrences are again split equally between the produced k-mers and added to their base support counts.
4) *Undecided consensus.* No consensus k-mer is produced, so there is no need to calculate support counts.

Fig. 2 shows examples of consensus k-mers and their support counts. In cases 1 and 3, there is only one built consensus k-mer, so the support count is the number of regular k-mers corresponding to the spaced k-mer. In case 2, both consensus k-mers are supported by two regular k-mers, and the last regular k-mer is split between them, so the support count for both is 2.5. In case 4, a consensus k-mer could not be built, so there is no need to calculate the support count.

4.3 Memory-Efficient and Parallel Implementation

For each spaced k-mer, LoMeX needs to find its occurrences in the reads. Therefore, before constructing the consensus k-mers, all the reads have to be scanned through. Time-wise, it is not efficient to read the input reads separately for all the different spaced k-mers. Instead, LoMeX goes through the input reads just once, trying to match the k-mers in the reads one by one with all spaced k-mers reported by Squeakr. If the read set is reasonably small, LoMeX could store all the reads in memory. It would then be easy to just store the regular k-mers as pointers to specific read positions. The stored spaced reads and regular k-mer pointers could then be used to find the solid characters and build the consensus k-mers. Unfortunately, this would mean that LoMeX worked only with small genomes and small read sets, limiting its usability.

In order to make LoMeX scale to larger genomes, we cannot store the reads in memory nor can we keep all the regular k-mers occurring in the reads in memory. To solve this problem, LoMeX has a buffer where the regular k-mers associated with the matching spaced k-mers are stored, and once the buffer becomes full, the information is stored on disk in temporary files. Then the buffer is emptied, and the scanning of reads for regular k-mers continues. The information in the buffer is written to the temporary files in lexicographic order by the matching spaced k-mers. In other words, the regular k-mers for the lexicographically smallest spaced k-mer come first, and the lexicographically largest last. The size of the buffer B is the number of regular k-mers that are kept in the buffer. B is a parameter of LoMeX and can thus be set appropriately by the user depending on the available memory.

The regular k-mers are stored temporarily as a binary file, which makes it easy to use only two bits for every nucleotide character, instead of eight bits if the file was written in a human-readable format. As the usage of two bits enables us to store only four different characters, we are unable to mark unclear nucleotide characters such as the N characters. For this reason, if a gap position in a regular k-mer has a character other than A, C, G, or T one from these four is randomly chosen as a replacement. This imitates the case where the unclear character was just read incorrectly. A fixed position can never have characters other than A, C, G, or T, because Squeakr reports spaced k-mers that have only these characters in the fixed positions.

After all the reads are scanned and the regular k-mer information has been written to the disk, LoMeX starts building the consensus k-mers. Because the information is written to multiple temporary files in lexicographic order, we can use a technique similar to the multiway merge algorithm. LoMeX starts to read all the files simultaneously, byte by byte. The files are read until all the information (regular k-mers) matching the first spaced k-mers of each file is stored in memory. Next, LoMeX determines which spaced k-mer among those is lexicographically smallest, combines the regular k-mers for that spaced k-mer, and builds the consensus k-mers. After that, the regular k-mers matching the spaced k-mer are discarded, and for every file that contained it, LoMeX continues reading bytes until all the information for the next lexicographically smallest spaced k-mer in memory. This process continues until no file has more content to be read. The produced consensus k-mers are not kept in memory, as they are immediately written to an output file.

The benefit of this approach is that it is possible to find long k-mers even if the input data is too large to be kept in memory. As a drawback, this requires more disk space and increases runtime due to the increased disk IO. At some point, the available disk space may become an issue, which we address by splitting the search and consensus steps into multiple iterations. The spaced k-mers are partitioned between these iterations so that each iteration only cares about k-mers specific to it. The temporary files are deleted at the end of the iterations and thus the space usage is decreased. The number of iterations affects the runtime because the reads must be read in every iteration to find the iteration-specific k-mers. Therefore, there is a trade-off between the runtime and available disk space, which can be optimized by the user with a parameter that controls the number of iterations.

LoMeX thus executes sequential iterations with two distinct steps: search step and consensus step. The spaced k-mer search with Squeakr is not split into multiple iterations. In

TABLE 1
Data Sets Used in the Experiments

Accession number	Organism	Ref. seq. accession	Genome length	Coverage	Number of reads	Avg. read length
ERR654976	*E. coli*	GCA_000005845.2	4,641,652	50	928,274	243
ERR654976	*E. coli*	GCA_000005845.2	4,641,652	111	2,120,290	243
SRR5216995	*A. thaliana*	GCA_000001735.2	119,668,634	50	20,781,062	289
SRR5216995	*A. thaliana*	GCA_000001735.2	119,668,634	130	53,786,130	289

the search step LoMeX goes through the input reads and finds occurrences of specific spaced k-mers and writes them to temporary files for the consensus step. We have parallelized the search step so that reading the input and associating the regular k-mers to spaced k-mers is split into multiple threads. The input reads are split into equally sized blocks and each thread is responsible for a single read block.

In the consensus step, LoMeX reads all the temporary files to access the necessary information for consensus k-mer building. Our experiments suggested that in this step the bottleneck is the disk IO instead of the actual consensus k-mer construction. Because all the temporary files must be read simultaneously due to the multiway merge -like nature of the disk storing implementation, this task cannot be efficiently split between threads. For this reason, we did not utilize parallelism in this step.

4.4 Time and Space Complexity

Here we analyze the time and space complexities of LoMeX. The whole k-mer extraction process can be divided into three steps, and some of them are performed multiple times according to the iteration count. The steps are:

1) Squeakr spaced seed search
2) LoMeX search step
3) LoMeX consensus step

Squeakr Spaced Seed Search. Extracting all spaced k-mers from a read set with total length L takes $O(Lq)$ time where q is the weight of the spaced k-mer i.e., the number of fixed characters. Inserting a spaced k-mer into the counting quotient filter (CQF) takes $O(1)$ time and the contents of the CQF can be enumerated in linear time [34] so the total complexity of this step is $O(Lq)$. The space complexity of Squeakr spaced seed search is $O(|G|)$ where G is the set of distinct spaced k-mers.

LoMeX *Search Step.* First, a hashtable is initialized to support membership queries to the set of spaced k-mers returned by Squeakr which takes $O(|G|)$ time where G is the extracted spaced k-mer set. In each iteration of the search step, LoMeX extracts all the k-mers from the reads in $O(Lk)$ time. For each k-mer, we check if it corresponds to a stored spaced k-mer and if so, it is inserted to the set of k-mers for that spaced k-mer. This takes $O(k)$ time per k-mer. Therefore the total time complexity of the search step is $O(iLk)$ where i is the number of iterations. In this step, the memory contains the hashtable of $|G|$ spaced k-mers and the buffer of B regular k-mers. Thus the total space complexity is $O(|G| + B)$.

LoMeX *Consensus Step.* During the consensus step in total $O(L)$ regular k-mers each of size k are read from disk. Let n be the maximum number of regular k-mers associated with a spaced k-mer. The worst case for consensus k-mer generation occurs when there are multiple ambiguous positions. In this case, we compare the regular k-mers to each other, which takes $O(n^2k)$ time. There are at most $\lfloor n/2 \rfloor$ consensus k-mers to report and thus the total complexity of this step is $O(Lk + gn^2k)$. The space complexity of this step is $O(nk)$ since at any given time we have the regular k-mers associated with the currently processed spaced k-mer in memory.

During an iteration, the temporary files on disk contain the regular k-mers associated with the iteration-specific spaced k-mers. Assuming the regular k-mers are evenly distributed over the spaced k-mers and the spaced k-mers are evenly distributed over the iterations, the maximum total number of k-mers stored on disk at any given time is $O(L/i)$.

5 Experiments and Results

We ran experiments on Illumina read set sequenced from an *E. coli* genome with 111x coverage, and a smaller sampled read set with 50x coverage. We also ran experiments on a larger *A. thaliana* genome, with two read sets of coverage 130x and 50x, where the smaller set was sampled from the larger one. The details of the data sets are shown in Table 1. Additionally, we simulated read sets from an *E. coli* reference genome. Reads were simulated with four different read lengths, 250 bp, 500 bp, 1,000 bp, and 2,000 bp, and four different substitution error rates, 0.0001, 0.005, 0.010, and 0.020. All combinations of the read lengths and the error rates were used, totaling 16 different data sets. Only substitution errors were simulated, in a simple uniformly random manner.

We compare LoMeX to DSK [38], one of the state-of-the-art k-mer extraction tools. DSK is a low memory usage program, that supports k-mer search for large k-mers. This is the reason we chose to compare against DSK as it is also able to extract long k-mers. *E. coli* experiments were executed on a machine with 8 cores and 16GB memory, and for the *A. thaliana* experiments we used a more powerful machine with 8 cores and 64 GB memory. LoMeX was run with the default parameters (Squeakr abundance $S = 2$, solid character minimum count $N = 2$, solid character minimum proportion $r = 0.10$), with the exception of using 8 threads, the number of iterations being 32, and having buffer size of 1 000 000 for *E. coli* and 5 000 000 for *A. thaliana*. For LoMeX we used the following hand-picked spaced seeds:

- *121-mers:* 6-30-6-15-7-15-6-30-6
- *221-mers:* 6-55-6-40-7-40-6-55-6
- *321-mers:* 6-80-6-65-7-65-6-80-6
- *421-mers:* 6-105-6-90-7-90-6-105-6
- *521-mers:* 6-130-6-115-7-115-6-130-6

We evaluated the correctness of both programs by comparing the sets of k-mers extracted from the reads to the set

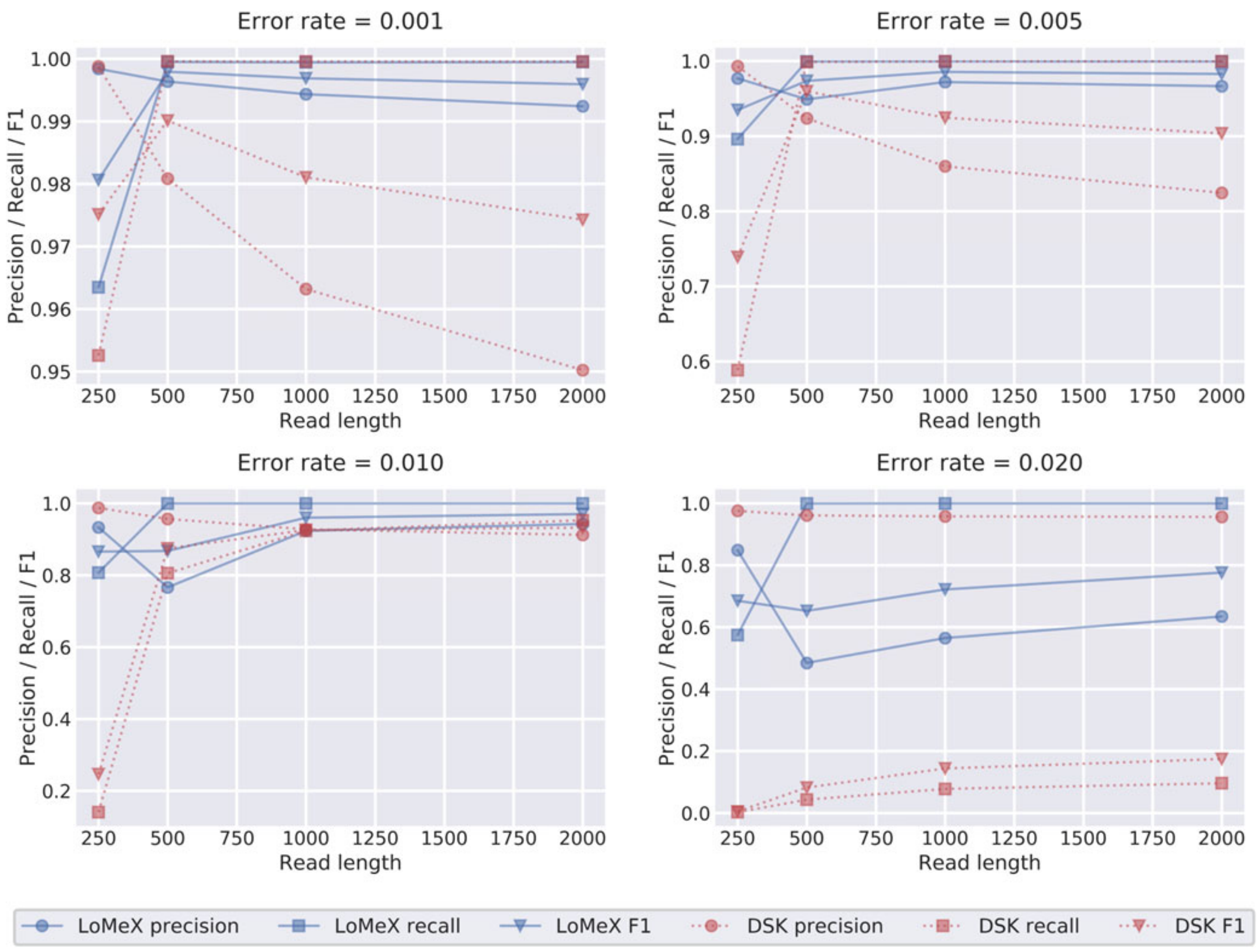

Fig. 3. Effect of different error rates and read lengths on LoMeX and DSK 221-mer extraction. This experiment was performed with simulated *E. coli* reads.

of k-mers occurring in the reference genome, which we refer to as the real k-mers. We call the intersection of real k-mers and k-mers extracted from the read set the extracted real k-mers. With these sets of k-mers we can compute the precision and recall for the tools

$$\text{Precision} = \frac{|\text{Extracted real } k - \text{mers}|}{|\text{Extracted } k - \text{mers}|}$$

$$\text{Recall} = \frac{|\text{Extracted real } k - \text{mers}|}{|\text{Real } k - \text{mers}|}.$$

Finally, the F1 score is the harmonic mean of precision and recall

$$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

We used LoMeX and DSK to extract 221-mers from the simulated data sets to evaluate how well they fared with varying read lengths and error rates. The results of these experiments can be seen in Fig. 3.

Simulated data sets were also used to evaluate how the value of k affects the performance of these programs. For this experiment, we used the four simulated read sets with 2,000 bp read length and varying error rates. All handpicked spaced seeds were used to extract different length k-mers, which were compared against DSK extracted k-mers. The results of these experiments are found in Fig. 4.

LoMeX was also compared to DSK with real read sets. For these experiments, we used two larger read sets of *E. coli* and *A. thaliana*, and two smaller read sets sampled from the larger ones. In these experiments, only 221-mers were extracted. The results can be seen in Table 2, and the runtimes and memory usages in Table 3.

We also experimented with how the chosen spaced seed affects the k-mer extraction. We chose four random spaced seeds of length 221 and compared them to the handpicked 221 length spaced seed. The random spaced seeds are the following:

- *Random I:* 1-11-1-2-1-27-1-6-2-3-2-6-1-20-2-6-1-1-1-9-1-4-3-4-1-9-1-1-1-6-2-20-1-6-2-3-2-6-1-27-1-2-1-11-1
- *Random II:* 2-9-1-4-1-8-2-1-1-20-2-1-1-6-1-5-1-23-1-7-1-3-1-8-1-8-1-3-1-7-1-23-1-5-1-6-1-1-2-20-1-1-2-8-1-4-1-9-2
- *Random III:* 1-1-1-6-2-32-1-15-1-6-1-5-1-1-1-2-2-7-1-2-1-10-1-6-1-2-1-2-1-6-1-10-1-2-1-7-2-2-1-1-1-5-1-6-1-15-1-32-2-6-1-1-1

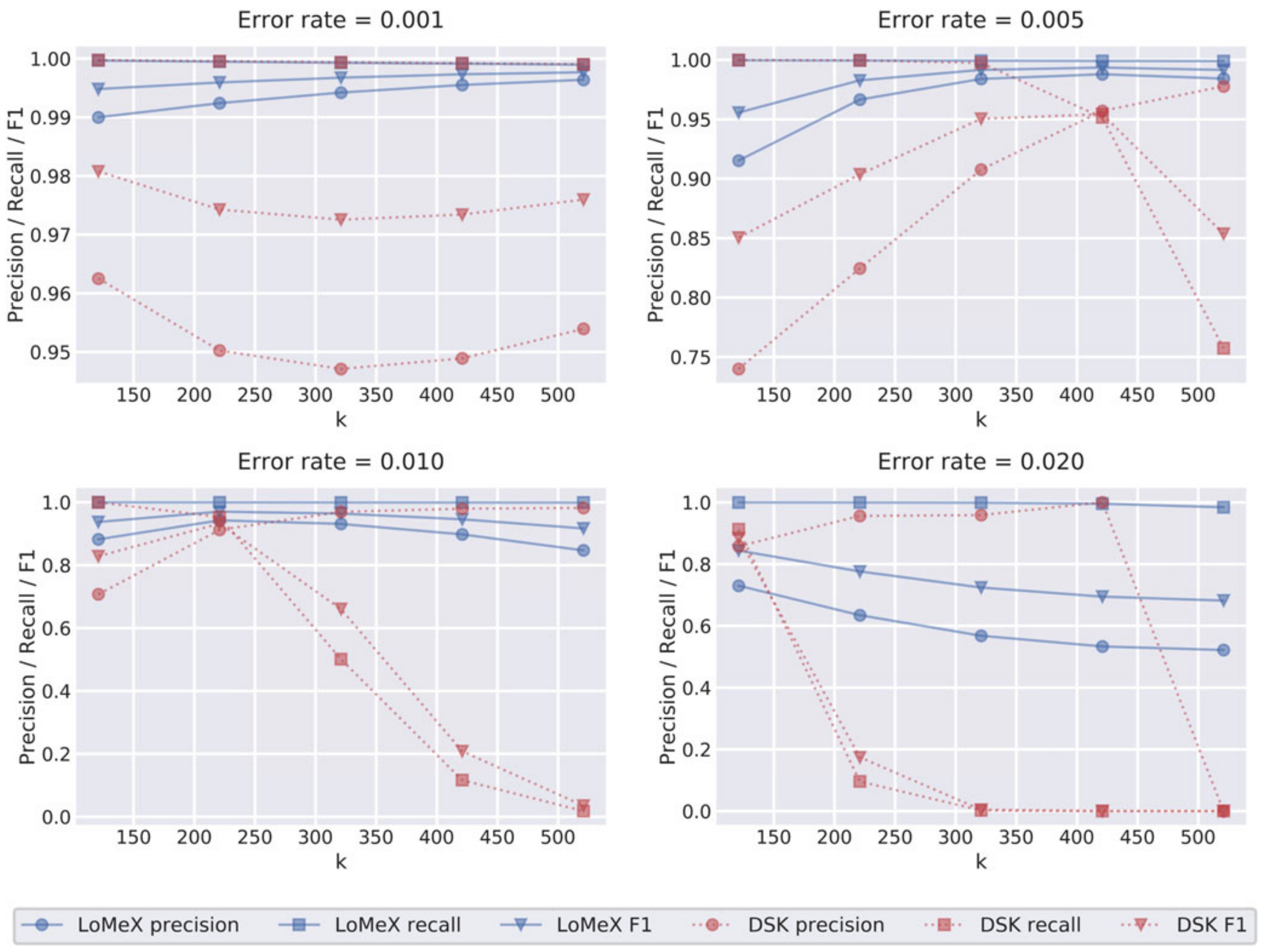

Fig. 4. Effect of different error rates and values of k on LoMeX and DSK k-mer extraction. This experiment was performed with 2,000 bp long simulated *E. coli* reads.

TABLE 2
LoMeX and DSK 221-mer Extraction Results With Different Data Sets

Program	Data set	Coverage	k	Real k-mers	Extracted k-mers	Extracted real k-mers	Precision	Recall	F1 score
LoMeX	*E. coli*	50	221	4,589,073	4,073,788	3,971,588	0.974913	0.865445	0.916923
DSK	*E. coli*	50	221	4,589,073	3,948,038	3,911,563	0.990761	0.852365	0.916367
LoMeX	*E. coli*	111	221	4,589,073	4,798,705	4,486,627	0.934966	0.977676	0.955844
DSK	*E. coli*	111	221	4,589,073	4,574,246	4,466,413	0.976426	0.973271	0.974846
LoMeX	*A. thaliana*	50	221	117,866,955	127,422,253	109,783,497	0.861572	0.931419	0.895135
DSK	*A. thaliana*	50	221	117,866,955	118,800,490	105,014,811	0.883959	0.890961	0.887446
LoMeX	*A. thaliana*	130	221	117,866,955	163,363,095	115,442,220	0.706660	0.979428	0.820981
DSK	*A. thaliana*	130	221	117,866,955	155,947,309	115,468,749	0.740434	0.979653	0.843409

- *Random IV:* 1-10-1-11-1-1-1-2-1-2-1-5-1-9-1-2-1-6-1-7-1-4-1-4-1-19-1-8-1-5-1-5-1-8-1-19-1-4-1-4-1-7-1-6-1-2-1-9-1-5-1-2-1-2-1-1-1-11-1-10-1

The results of these experiments are shown in Table 4. The differences in runtime and memory usage between the spaced seeds were negligible.

LoMeX has parameters that can be tuned to optimize the results of the k-mer extraction. Demanding a greater number of occurrences in Squeakr spaced k-mer search and a higher base count for solid characters will give us fewer reported k-mers. This can discard mostly false k-mers from the results, leading to higher precision. On the other hand, some weakly supported correct k-mers can end up being discarded lowering the recall rate. We experimented with a few different thresholds. In Table 5 we report the results on how these parameters affect the extracted k-mers. Here parameter S is the minimum number of spaced k-mer occurrences required by the modified Squeakr. Parameter N is the absolute minimum base count for a base to be considered solid. Again, the differences in runtime and memory usage with the different parameters were negligible so they are not shown.

TABLE 3
Memory Usages and Runtimes of LoMeX and DSK in 221-mer Extraction

Program	Data set	Coverage	k	Time usage [hh:mm:ss]	Max memory [MB]
LoMeX	*E. coli*	50	221	00:11:13	8,565
DSK	*E. coli*	50	221	00:00:23	817
LoMeX	*E. coli*	111	221	00:20:46	8,847
DSK	*E. coli*	111	221	00:00:30	1,649
LoMeX	*A. thaliana*	50	221	10:35:51	12,583
DSK	*A. thaliana*	50	221	00:28:21	8,067
LoMeX	*A. thaliana*	130	221	26:29:17	12,583
DSK	*A. thaliana*	130	221	00:54:30	7,065

These resource usages are related to the results in Table 2.

TABLE 4
Comparison Between one Hand-Picked Spaced Seed and Four Randomly Chosen Spaced Seeds for 221-mer Search

Program	k	Real k-mers	Extracted k-mers	Extracted real k-mers	Precision	Recall	F1 score
Hand-picked	221	4,589,073	4,077,016	3,972,459	0.974355	0.865634	0.916782
Random I	221	4,589,073	4,109,510	3,982,071	0.968989	0.867729	0.915568
Random II	221	4,589,073	4,082,556	3,974,210	0.973461	0.866016	0.916601
Random III	221	4,589,073	4,098,478	3,978,745	0.970786	0.867004	0.915965
Random IV	221	4,589,073	4,094,384	3,977,466	0.971444	0.866725	0.916102

These experiments were done with the 50x coverage E. coli *reads.*

TABLE 5
Comparison Between LoMeX Runs for 221-mers With Different Minimum Occurrence and Coverage Thresholds

S	N	k	Real k-mers	Extracted k-mers	Extracted real k-mers	Precision	Recall	F1 score
2	2	221	4,589,073	4,073,790	3,971,590	0.974913	0.865445	0.916923
2	3	221	4,589,073	3,389,853	3,365,283	0.992752	0.733325	0.843543
3	2	221	4,589,073	3,710,129	3,615,914	0.974606	0.787940	0.871388
3	3	221	4,589,073	3,389,853	3,365,283	0.992752	0.733325	0.843543

The experiment was run using the 50x coverage E. coli *read set.*

6 Discussion

The simulated experiments in Fig. 3 give a good overview of how the characteristics of the reads affect LoMeX. As the read length increases, the input data becomes less fragmented and the total number of k-mers in the input data increases, and thus there is more data to construct consensus k-mers. For this reason, the recall of LoMeX increases (or at least stays the same), when the read length increases. The same is true for DSK. On the other hand, precision behaves more interestingly. With short reads, it is the highest, but with error rates 0.005 and higher, the precision of LoMeX first drops and then slightly rises as the read length increases. When the total number of k-mers in the input reads increases, it is more likely that the same error occurs twice in the reads, and thus precision decreases. However, when the number of k-mers in the input data increases further, the minimum character occurrence threshold c in LoMeX starts to increase because LoMeX also requires that a base is present in more than 10% of regular k-mers corresponding to a spaced k-mer. DSK precision always declines as the read length increases.

The most promising results were observed in the simulated read experiments. As seen in Fig. 3, the recall of LoMeX is always identical or better than that of DSK, regardless of read length and error rate. On the data sets with the two lowest error rates, LoMeX also has higher precision with reads at least 500 bp long. On the data sets with the higher error rates, LoMeX does not beat DSK in precision. LoMeX F1 score is also higher or nearly identical compared to DSK with all combinations of read lengths and error rates.

Fig. 4 shows how well LoMeX and DSK can count k-mers of different length with varying error rates and a fixed 2,000 bp read length. With the lowest error rate both LoMeX and DSK have similar recalls, but LoMeX has higher precision and F1 score. With 0.005 error rate LoMeX has a similar recall for the shorter k-mers, but with the longer k-mers DSK recall drops noticeably. DSK starts with low precision but begins to catch up with LoMeX as the value of k increases. On data sets with the two highest error rates, DSK has a very low recall with longer k-mers. On the other hand, LoMeX is still able to find most of the k-mers. With the higher error rates LoMeX starts to lose to DSK in precision. DSK could not find any 521-mers when the error rate was 0.02.

LoMeX performs well with simulated data, but it is also important to assess the performance with real reads. In Table 2 we can see 221-mer extraction results of LoMeX and

DSK with four different read sets. Their recalls are very similar, slightly favoring LoMeX with all sets except the 130x one. On the other hand, DSK performed noticeably better in precision in all cases. LoMeX seems to be able to extract some harder to find k-mers, but as a drawback more false consensus k-mers are also reported. In Table 3 we can see the resources used in these experiments. LoMeX is clearly slower than DSK since our program does much more than just search for existing k-mers in the read set. When filtering spaced k-mers or searching their occurrences, we currently convert each k-mer to a spaced k-mer separately which takes $O(q)$ time where q is the weight of the spaced seed. We could speed this up by using the technique of Petrucci *et al.* [36] which uses the previous overlapping spaced k-mers to construct the next spaced k-mer. However, currently the consensus step of LoMeX is the most time-consuming step so overall this improvement would only have a minor effect.

It is possible that some spaced seeds are better than others for k-mer extraction. We did not try to optimize the spaced seeds, but we checked how an "average" spaced seed would perform by generating random spaced seeds and then comparing them to our hand-picked one. The results of this experiment are in Table 4. There is some slight variation in precision and recall, but none of the seeds is significantly better than the others. As future work, it would be interesting to design optimal spaced seeds for LoMeX.

Designing optimal seeds for similarity search has been researched extensively, and various methods have been proposed to solve this problem [11], [14], [19], [43]. However, these methods are not directly applicable to LoMeX because the studied seeds are much shorter than the ones LoMeX requires, and the proportions between fixed and gap positions is much higher when compared to the spaced seeds in LoMeX. Using more than 31 fixed characters would likely have an effect on the extracted k-mers and could also be explored. Extending Squeakr to allow this would be another potential objective for future work.

From the experiments, we can see that the precision of LoMeX is lower than the precision of DSK with real reads. The precision can be boosted by making the spaced k-mer occurrence threshold S and consensus base count threshold N stricter, but this also reduces recall as shown in Table 5. We believe a high recall is ultimately more important because k-mers lost by a k-mer counter cannot be recovered, whereas erroneous k-mers can be detected for example by examining the tip and bubble structure of a de Bruijn graph built on the k-mers.

7 Conclusion

We have presented LoMeX, a tool for extracting long k-mers from reads. LoMeX uses spaced seeds to successfully find long k-mers even in the presence of sequencing errors. Our experiments show that on current real reads, LoMeX is able to recover some missing k-mers compared to state-of-the-art k-mer counter, DSK. At this point the differences in recall rates are minor, but the method LoMeX utilizes could be a valid choice in future implementations. Furthermore, our experiments on simulated data show that the advantage of LoMeX over DSK increases when the read lengths increase. We expect that the read length of Illumina data will keep increasing and thus LoMeX has the potential to become more practical in the future.

Because of the nature of LoMeX, it is better equipped to handle substitution errors than insertion or deletion errors. A substitution error only disrupts the spaced k-mer matching if the substitution happens at a fixed character position. On the other hand, insertion or deletion will disrupt this regardless of where it occurs. Thus LoMeX is not yet ready to process reads from third generation sequencing machines with high rates of indel errors. Nevertheless, the approach pioneered here opens up the possibility to extract long accurate k-mers also from these high error rate reads.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, 1990.

[2] S. Batzoglou *et al.*, "ARACHNE: A whole-genome shotgun assembler," *Genome Res.*, vol. 12, pp. 177–189, 2002.

[3] B. Brejová, D. G. Brown, and T. Vinař, "Optimal spaced seeds for hidden Markov models, with application to homologous coding regions," in *Proc. 14th Annu. Conf. Combinatorial Pattern Matching*, 2003, pp. 42–54.

[4] J. Buhler, U. Keich, and Y. Sun, "Designing seeds for similarity search in genomic DNA," *J. Comput. Syst. Sci.*, vol. 70, no. 3, pp. 342–363, 2005.

[5] S. Burkhardt and J. Kärkkäinen, "Better filtering with gapped q-grams," *Fundamenta Informaticae*, vol. 56, no. 1/2, pp. 51–70, 2003.

[6] D. Campagna *et al.*, "RAP: A new computer program for de novo identification of repeated sequences in whole genomes," *Bioinformatics*, vol. 21, no. 5, pp. 582–588, 2005.

[7] M. J. Chaisson and G. Tesler, "Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): Application and theory," *BMC Bioinf.*, vol. 13, 2012, Art. no. 238.

[8] R. Chikhi and P. Medvedev, "Informed and automated k-mer size selection for genome assembly," *Bioinformatics*, vol. 30, no. 1, pp. 31–37, Jun. 2013.

[9] K. P. Choi, F. Zeng, and L. Zhang, "Good spaced seeds for homology search," *Bioinformatics*, vol. 20, no. 7, pp. 1053–1059, 2004.

[10] M. Erbert, S. Rechner, and M. Müller-Hannemann, "Gerbil: A fast and memory-efficient k-mer counter with GPU-support," *Algorithms Mol. Biol.*, vol. 12, no. 1, 2017, Art. no. 9.

[11] L. Hahn, C.-A. Leimeister, R. Ounit, S. Lonardi, and B. Morgenstern, "rasbhari: Optimizing spaced seeds for database searching, read mapping and alignment-free sequence comparison," *PLoS Comput. Biol.*, vol. 12, no. 10, pp. 1–18, Oct. 2016.

[12] R. M. Idury and M. S. Waterman, "A new algorithm for DNA sequence assembly," *J. Comput. Biol.*, vol. 2, no. 2, pp. 291–306, 1995.

[13] L. Ilie and S. Ilie, "Multiple spaced seeds for homology search," *Bioinformatics*, vol. 23, no. 22, pp. 2969–2977, 2007.

[14] L. Ilie, S. Ilie, and A. Mansouri Bigvand, "SpEED: Fast computation of sensitive spaced seeds," *Bioinformatics*, vol. 27, no. 17, pp. 2433–2434, 2011.

[15] L. Kaplinski, M. Lepamets, and M. Remm, "GenomeTester4: A toolkit for performing basic set operations - Union, intersection and complement on k-mer lists," *GigaScience*, vol. 4, no. 1, 2015, Art. no. 58.

[16] U. Keich, M. L. Bin Ma, and J. Tromp, "On spaced seeds for similarity search," *Discrete Appl. Math*, vol. 138, no. 3, pp. 253–263, 2004.

[17] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, "Quake: Quality-aware detection and correction of sequencing errors," *Genome Biol.*, vol. 11, 2010, Art. no. R116.

[18] M. Kokot, M. Dlugosz, and S. Deorowicz, "KMC 3: Counting and manipulating k-mer statistics," *Bioinformatics*, vol. 33, no. 17, pp. 2759–2761, 2017.

[19] G. Kucherov, L. Noé, and M. Roytberg, "A unifying framework for seed sensitivity and its application to subset seeds," *J. Bioinf. Comput. Biol.*, vol. 4, pp. 553–569, 2006.

[20] S. Kurtz, A. Narechania, J. C. Stein, and D. Ware, "A new method to compute K-mer frequencies and its application to annotate large repetitive plant genomes," *BMC Genomics*, vol. 9, no. 1, 2008, Art. no. 517.

[21] A. Lefebvre, T. Lecroq, H. Dauchel, and J. Alexandre, "FORRepeats: Detects repeats on entire chromosomes and between genomes," *Bioinformatics*, vol. 19, no. 3, pp. 319–326, 2003.

[22] C.-A. Leimeister, S. Sohrabi-Jahromi, and B. Morgenstern, "Fast and accurate phylogeny reconstruction using filtered spaced-word matches," *Bioinformatics*, vol. 33, no. 7, pp. 971–979, 2017.

[23] M. Li, B. Ma, D. Kisman, and J. Tromp, "PatternHunter II: Highly sensitive and fast homology search," *J. Bioinf. Comput. Biol.*, vol. 2, no. 3, pp. 417–439, 2004.

[24] Y. Li and X. Yan, "MSPKmerCounter: A fast and memory efficient approach for k-mer counting," 2015, *arXiv:1505.6550*.

[25] Y. Liu, J. Schröder, and B. Schmidt, "Musket: A multistage k-mer spectrum-based error corrector for Illumina sequence data," *Bioinformatics*, vol. 29, no. 3, pp. 308–315, 2013.

[26] B. Ma, J. Tromp, and M. Li, "PatternHunter: Faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002.

[27] A.-A. Mamun, S. Pal, and S. Rajasekaran, "KCMBT: A k-mer counter based on multiple burst trees," *Bioinformatics*, vol. 32, no. 18, pp. 2783–2790, 2016.

[28] S. C. Manekar and S. R. Sathe, "A benchmark study of k-mer counting methods for high-throughput sequencing," *GigaScience*, vol. 7, no. 12, 2018, Art. no. giy125.

[29] G. Marçais and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, vol. 27, no. 6, pp. 764–770, 2011.

[30] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner, "Error correction of high-throughput sequencing datasets with non-uniform coverage," *Bioinformatics*, vol. 27, no. 13, pp. i137–i141, 2011.

[31] P. Melsted and J. K. Pritchard, "Efficient counting of k-mers in DNA sequences using a bloom filter," *BMC Bioinf.*, vol. 12, no. 1, 2011, Art. no. 333.

[32] J. R. Miller *et al.*, "Aggressive assembly of pyrosequencing reads with mates," *Bioinformatics*, vol. 24, no. 24, pp. 2818–2824, 2008.

[33] F.-D. Pajuste, L. Kaplinski, M. Möls, T. Puurand, M. Lepamets, and M. Remm, "FastGT: An alignment-free method for calling common SNVs directly from raw sequencing reads," *Sci. Rep.*, vol. 7, 2017, Art. no. 2537.

[34] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "A general-purpose counting filter: Making every bit count," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 775–787.

[35] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "Squeakr: An exact and approximate k-mer counting system," *Bioinformatics*, vol. 34, no. 4, pp. 568–575, 2017.

[36] E. Petrucci, L. Noé, C. Pizzi, and M. Comin, "Iterative spaced seed hashing: Closing the gap between spaced seed hashing and k-mer hashing," *J. Comput. Biol.*, vol. 27, no. 2, pp. 223–233, 2020.

[37] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proc. Nat. Acad. Sci. USA*, vol. 98, no. 17, pp. 9748–9753, 2001.

[38] G. Rizk, D. Lavenier, and R. Chikhi, DSK: K-mer counting with very low memory usage," *Bioinformatics*, vol. 29, no. 5, pp. 652–653, 2013.

[39] R. S. Roy, D. Bhattacharya, and A. Schliep, "Turtle: Identifying frequent k-mers with cache-efficient algorithms," *Bioinformatics*, vol. 14, no. 30, pp. 1950–1957, 2014.

[40] L. Salmela and E. Rivals, "LoRDEC: Accurate and efficient long read error correction," *Bioinformatics*, vol. 30, no. 24, pp. 3506–3514, 2014.

[41] L. Salmela and J. Schröder, "Correcting errors in short reads by multiple alignments," *Bioinformatics*, vol. 27, no. 11, pp. 1455–1461, 2011.

[42] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. M. Jones, and I. Birol, "ABySS: A parallel assembler for short read sequence data," *Genome Res.*, vol. 19, no. 6, pp. 1117–1123, 2009.

[43] Y. Sun and J. Buhler, "Designing multiple simultaneous seeds for dna similarity search," *J. Comput. Biol.*, vol. 12, no. 6, pp. 847–861, 2005.

[44] R. Uricaru *et al.*, "Reference-free detection of isolated SNPs," *Nucleic Acids Res.*, vol. 43, no. 2, 2015, Art. no. e11.

[45] D. E. Wood and S. L. Salzberg, "Kraken: Ultrafast metagenomic sequence classification using exact alignments," *Genome Biol.*, vol. 15, 2014, Art. no. R46.

[46] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de bruijn graphs," *Genome Res.*, vol. 18, no. 5, pp. 821–829, 2008.

Miika Leinonen received the MSc degree in computer science from the University of Helsinki, Finland, in 2019, and is currently working toward the postgraduate degree. His current research focuses on developing algorithms for bioinformatics, especially for sequencing data.

Leena Salmela received the DSc(Tech) degree in computer science from the Helsinki University of Technology, Finland, in 2009. Since 2009 she has worked in the University of Helsinki, Finland, where she currently holds the position of Academy of Finland research fellow. Her research interests include string algorithms and bioinformatics.