Mining K-mers of Various Lengths in Biological Sequences

Jingsong Zhang[1], Jianmei Guo[2], Xiaoqing Yu[3], Xiangtian Yu[1], Weifeng Guo[4], Tao Zeng[1(✉)], and Luonan Chen[1,5(✉)]

[1] Institute of Biochemistry and Cell Biology,
Shanghai Institutes for Biological Sciences,
Chinese Academy of Sciences, Shanghai 200031, China
jasun@dmbio.info, graceyu1985@163.com, {zengtao,lnchen}@sibs.ac.cn
[2] Department of Computer Science and Engineering,
East China University of Science and Technology, Shanghai 200237, China
gjm@ecust.edu.cn
[3] Department of Applied Mathematics, Shanghai Institute of Technology,
Shanghai 201418, China
xqyu@sit.edu.cn
[4] School of Automation, Northwestern Polytechnical University, Xi'an 710072, China
shaonianweifeng@126.com
[5] Collaborative Research Center for Innovative Mathematical Modelling,
Institute of Industrial Science, University of Tokyo, Tokyo 153-8505, Japan

Abstract. Counting the occurrence frequency of each k-mer in a biological sequence is an important step in many bioinformatics applications. However, most k-mer counting algorithms rely on a given k to produce single-length k-mers, which is inefficient for sequence analysis for different k. Moreover, existing k-mer counters focus more on DNA sequences and less on protein ones. In practice, the analysis of k-mers in protein sequences can provide substantial biological insights in structure, function and evolution. To this end, an efficient algorithm, called VLmer (Various Length k-mer mining), is proposed to mine k-mers of various lengths termed *vl-mers* via inverted-index technique, which is orders of magnitude faster than the conventional forward-index method. Moreover, to the best of our knowledge, VLmer is the first able to mine k-mers of various lengths in both DNA and protein sequences.

Keywords: Sequential pattern mining · K-mer counting · K-mers of various lengths · Biological sequence analysis

1 Introduction

K-mer counting, which identifies frequent contiguous subsequences of length-k in a sequence database, is a fundamental data-mining problem with broad applications, including genome assembly [1], error correction of sequencing reads [2],

J. Zhang and J. Guo—Contributed equally to this work.

Z. Cai et al. (Eds.): ISBRA 2017, LNBI 10330, pp. 186–195, 2017.
DOI: 10.1007/978-3-319-59575-7_17

protein-protein interaction prediction [3], finding mutations [4], sequence classification [5], sequence alignment [6]. Many previous studies contributed to efficient k-mer counting, such as Tallymer [7], Jellyfish [8], BFCounter [9], KMC [10], DSK [11], KAnalyze [12], KMC 2 [13] and KCMBT [14]. In recent years, k-mer counting is gaining momentum and has become an active topic in sequence analysis community.

The k-mer counting algorithms developed so far have good performance. Unfortunately, almost all of the previous algorithms rely on a fixed k to split the given sequence(s) and output all possible contiguous subsequences along with their frequencies. This leads to the inefficiency during the sequence analysis for various values of k, since a user usually have to perform the k-mer counting algorithm many times. For example, as shown in [15,16], at least 6 sizes (17-mer, 21-mer, 25-mer, 41-mer, 55-mer and 77-mer) are selected to count the k-mers. In addition, Kurta et al. [7] used k-mers ranging from 10 to 500 to annotate the plant genomes, which needs to manually run the k-mer counter 491 times. Currently, there is no method that is able to automatically generate k-mers of various lengths.

The frequent k-mers in protein sequences are often the conserved composition patterns reflecting structural and functional features [17]. Miranda et al. [18] deeply analyzed the sequence specificity of pentatricopeptide repeat (PPR) proteins by enriched k-mers. However, previous work on k-mer counting focuses more on DNA sequences and less on protein ones, which hinders the identification of conserved regions in protein sequences. Consequently, a major challenge is how to design an effective algorithm to ensure that the k-mer counter outputs all k-mers of various lengths and meanwhile such counter is suitable for both DNA and protein sequences.

K-mer counting has been studied extensively, yet still efficient implementations take several hours or even days on large sequence databases. The inverted index, in computer science, is an important data structure that stores a mapping from content to its locations in a database file. Compared to the forward index structure, the inverted index restructures the representative format of files and contributes to a high efficiency of information retrieval, especially on large databases. Inspired by the well applications of inverted index [19], this technique can be explored to alleviate the efficiency problem encountered by previous k-mer counting methods.

In this paper, we propose VLmer that has four steps to efficiently mine k-mers of various lengths (i.e., vl-mers) in both DNA and protein sequences. In the first step, the initial biological sequences are transformed to structure like "(vl-mer, positions)" by the inverted index technique. In the second step, VLmer uses a pattern-growth scheme to generate the candidates of frequent vl-mers. In the third step, a few pruning techniques are explored to prune the unpromising vl-mers. In our experiments, we compare VLmer to ConSpan, which is a modified version of both CCSpan [20] and ConSgen [21] algorithms that are most related to our work, in terms of mining efficiency.

2 Methods

2.1 Preliminaries

The term "k-mer" typically refers to the fixed-length contiguous subsequences. Unfortunately, it's quite difficult to pre-specify a proper length k to count k-mers which have the promise to provide biological insights. We introduce an alternative term "vl-mer" to extend the traditional definition of k-mer as follows.

Definition 1 (vl-mer). *Given two sequences $S_1 = a_1 a_2 \cdots a_i$ and $S_2 = b_1 b_2 \cdots b_j$, where $|S_1| \leq |S_2|$, S_1 is a vl-**mer** of S_2 if S_1 is a contiguous subsequence (i.e., substring) of S_2.*

Based on this definition, the term vl-mer refers to all the possible contiguous subsequences (i.e., all the k-mers of various lengths) of a given sequence. Note that, this extended definition of k-mer uses non-fixed-length instead of fixed-length adopted in traditional definition to constraint the subsequences.

For a vl-mer s, it can be represented by a tuple (s, idx), where idx is the index (i.e., position) of s in sequence database D. For example, assume the database D consists of only a sequence $S = CCTCCCGCCTCA$, the tuple representation of vl-mer $s = CCTC$ is set $\{(CCTC, 0),\ (CCTC, 7)\}$.

Definition 2 (frequent vl-mer). *Given a minimum support threshold σ, a vl-mer s is frequent in sequence database D if $Sup_D(s) \geq \sigma$.*

For simplicity, we use notation *vl-mer* to denote *frequent vl-mer* if not explicitly stated.

We are now ready to formulate our problem. Given a sequence database D and a support threshold σ, discover a complete set of vl-mers such that each of which is frequent.

2.2 Inverted Projection

As we discussed earlier, the inverted index data structure is preferable to the forward index one in terms of the speed of the query in information retrieval domain, which motivates us to utilize such inverted index technique to transform the input sequence for vl-mer mining. For clarity, we call inverted index data structure *inverted projection* and revise the notion of it in sequence analysis community as follows.

Definition 3 (inverted projection). *Given a single sequence S as the input sequence database D, suppose all distinct characters of S be a set $I = \{i_1, i_2, \cdots, i_m\}$. The inverted projection of S is a set R expressed in tuples, denoted as $(f, <f\_indexes>)$ such that (1) each tuple consists of a character f ($f \in I$) and its position index(es) appearing in S, and (2) there is a one-to-one correspondence between the f of tuple $(f, <f\_indexes>)$ in R and the element $i_k \in I$, where $1 \leq k \leq m$.*

Note that, unlike the forward index data structure, the inverted projection uses a set of $(f, <f\_indexes>)$ pairs to equivalently represent the input sequence. The element $<f\_indexes>$ is a list consisting of one or more non-negative integers, each of which corresponds to a position number of vl-mers f in the original sequence. Table 1 shows the inverted projection of the sequence $S = CCTCCCGCCTCAGTTCGCGCCGCGCCTCGGCTTGGAACGC$. The shift of input sequence significantly reduces the number of scans of the sequence, so as to minimize the computation cost during the mining process.

Table 1. An inverted projection of sequence S

Character	Index list
C	0, 1, 3, 4, 5, 7, 8, 10, 15, 17, 19, 20, 22, 24, 25, 27, 30, 37, 39
T	2, 9, 13, 14, 26, 31, 32
G	6, 12, 16, 18, 21, 23, 28, 29, 33, 34, 38
A	11, 35, 36

2.3 Candidate Generation of *vl*-mers

Most k-mer counting algorithms, due to the forward index data structure, need to split all single-length contiguous subsequences as k-mers and compute their occurrence frequencies, rendering the counting operation to be costly. To the best of our knowledge, instead of the forward index, an alternative but clever data structure is the inverted projection. Therefore, we propose a pattern-growth approach based on the inverted projection to generate candidates so as to efficiently mine vl-mers. Based on this approach, each length-k candidate is easily produced by the original sequence, frequent $(k-1)$-mers and their indexes, rather than using a n-gram model [22,23] to enumerate all the possible snippets of input sequences.

2.4 Pruning Techniques

Upon generating a new candidate by the pattern-growth approach based on the inverted projection, we want to immediately check whether or not it is a distinct yet frequent vl-mer. We study some properties of frequent vl-mers in this subsection, which underpin the design of pruning scenarios.

Definition 4 (max-suffix). *Given two sequences $s_1 = a_1 a_2 \cdots a_i$ and $s_2 = b_1 b_2 \cdots b_j$, s_1 is a max-suffix of s_2, denoted as $s_1 \sqsubset_{suf} s_2$, if (1) $|s_1| \geqslant 1$, $|s_2| - |s_1| = 1$; and (2) $a_1 = b_2, a_2 = b_3, \cdots, a_i = b_j$.*

Theorem 1. *Given a sequence s ($|s| \geqslant 2$), suppose s is frequent in sequence database D, then the max-suffix of s is frequent in D.*

Proof (PROOF). Letting F be a set consisting of all subsequences of s, then each element of F, i.e., $\forall s' \in F$ is frequent by virtue of Theorem 1 in [21]. Letting the max-suffix of s be s_{suf}, then s_{suf} satisfies $s_{suf} \in F$. Thus, s_{suf} is frequent. The theorem holds immediately.

Lemma 1. *Given a sequence s ($|s| \geqslant 2$) and a sequence database D, if there exists no frequent max-suffix of s, i.e., $Sup_D(s_{suf}) < \sigma$ holds, then s can be safely pruned.*

Proof (PROOF). The proof of the above lemma is obvious according to Theorem 1, and thus it is omitted here.

By exploring some properties of *vl*-mers above, three effective pruning techniques, repeated candidate pruning, max-suffix pruning, and support pruning, are introduced to prune the unpromising *vl*-mers.

2.5 VLmer Algorithm

For delineating VLmer, we first introduce two data structures. First, the inverted index data structure, namely (f, <$f\_indexes$>) is employed for storing the temporary output *vl*-mers, where f is a *vl*-mer itself and $f\_indexes$ represent its indexes in original sequence database. The size of f's indexes indicates the frequency of f. Second, the triple data structure (f, $f.count$, B) from [21] formalizes the final output *vl*-mers, where f is also a *vl*-mer, $f.count$ is the actual support of f, and the last attribute variable "B" takes on the value "Y" by default. The *vl*-mers of F can be organized into a set $\{\{F_1\}, \{F_2\}, \cdots, \{F_i\}\}$ consisting of i different partitions, each of which is a subset of *vl*-mers.

Algorithm 1. VLmer(S, σ)

Input: sequence S, support threshold σ

$F_{k-1} \leftarrow \emptyset$; // initialize F_{k-1} to store $(k-1)$-mers
$F_k \leftarrow \emptyset$; // initialize F_k to store k-mers
$F_{sta} \leftarrow \emptyset$; // initialize F_{sta} to store all *vl*-mers
$F \leftarrow \emptyset$; // initialize F to store all patterns
1: $F_{k-1} \leftarrow$ 1-mer-gen(S, σ); // inverted projecting
2: $F_{sta} \leftarrow$ sta-gen(F_{k-1}); // standard 1-mers
3: $F \leftarrow F_{sta}$; // add 1-mers
4: **while** $F_{k-1}.count > 0$ **do**
5: $F_k \leftarrow k$-mer-gen(S, F_{k-1}, σ); //generate k-mers
6: **if** $F_k.count > 0$ **then**
7: $F_{sta} \leftarrow$ sta-gen(F_k); // standard k-mers
8: $F \leftarrow \cup_k F_{sta}$; // add standardized k-mers
9: **end if**
10: $F_{k-1} \leftarrow F_k$;
11: **end while**

Output: F;

Algorithm 1 sketches VLmer that performs the frequent *vl*-mer mining. As shown, two parameters include a sequence S as the input database and a support threshold σ. Global variable F_{k-1} and F_k store the frequent $(k-1)$-mers and k-mers respectively. Each pattern of both F_{sta} and F is represented by the inverted index structure $(f, <f\_indexes>)$, while F_{sta} and F store *vl*-mers with the triple data structure $(f, f.count, B)$. During the mining process, F_{sta} is the currently generated single-length *vl*-mers, and F saves all output *vl*-mers. Function 1-mer-gen() first produces an inverted projection by projecting the original input sequence, and then generates the frequent 1-mers conforming to the $(f, <f\_indexes>)$ structure (line 1). Such 1-mers are transformed into the standard triple structure $(f, f.count, B)$ (line 2), and then delivered to set F (line 3). Those generated 1-mers are viewed as frequent $(k-1)$-mers to feed function k-mer-gen() for checking the longer k-mers (2-mers). For each non-empty set F_{k-1}, i.e., $F_{k-1}.count > 0$ (line 4), k-mer-gen() produces a set of length-specified frequent k-mers based on S, σ, and the generated $(k-1)$-mers (line 5). Function k-mer-gen() continues its scan until the output set F_k is empty. When a non-empty F_k is generated, each k-mer of them is transformed into the above triple data structure and such a set F_k as a whole is added into set F (line 8). The output of VLmer is such a pattern set $F = \{(f, f.count, B) | f.count \geq \sigma\}$. The main ideas of the above functions are detailed in Subsects. 2.2 to 2.4, and thus we do not recount them here.

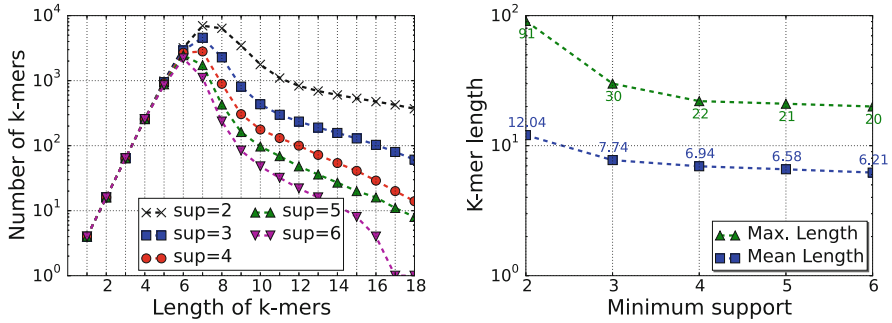
3 Results

3.1 Datasets

In our experiments, we used both DNA and protein sequences to study the performance of the VLmer algorithm.

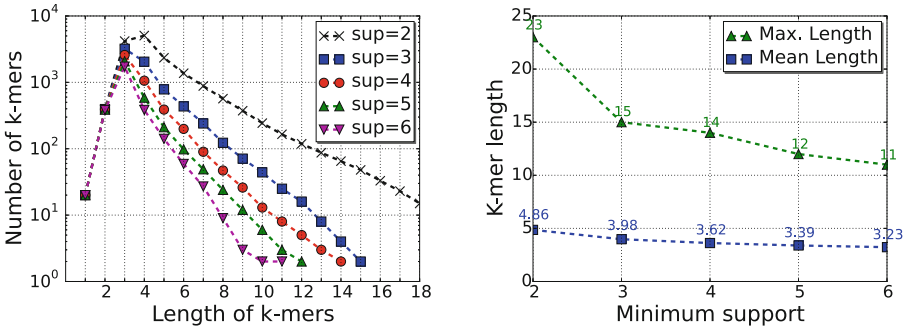
3.2 Effectiveness Study

Unlike previous k-mer counting algorithms, we can conveniently obtain all k-mers of various lengths (i.e., all *vl*-mers) by performing VLmer algorithm only once. Figure 1 depicts the characteristics of the *vl*-mers on DNA sequence AL607040. Figure 1(a) shows the distribution of *vl*-mers against their length for support thresholds (σs) varying from 2 to 6. Note that the smaller σ we choose, the more *vl*-mers will be generated, which is consistent with previous k-mer counting or sequential pattern mining. From Fig. 1(a), we can see that the number of *vl*-mers equals or is close to 4^l when the *vl*-mer-length $l \leq 5$ for all test values of σ. These *vl*-mers are almost the exhaustive enumeration of all possible combinations of the four base-pares. Intuitively, such *vl*-mers may be impossible to reveal any biological significance and can be discarded during some sequence analysis, such as sequence classification and TFBS identification. The peak values of the number of *vl*-mers mainly locate at length 6 ($\sigma \geq 3$) and length 7 ($\sigma = 2$), while they are far less 4^6 and 4^7 respectively. These frequent *vl*-mers with enough lengths may reflect some biological significance that has been demonstrated [24].



(a) Distribution of *vl*-mers (b) Maximal and mean lengths

Fig. 1. *vl*-mer analysis on DNA sequence AL607040.



(a) Distribution of *vl*-mers (b) Maximal and mean lengths

Fig. 2. *vl*-mer analysis on protein sequence XP_011987916.

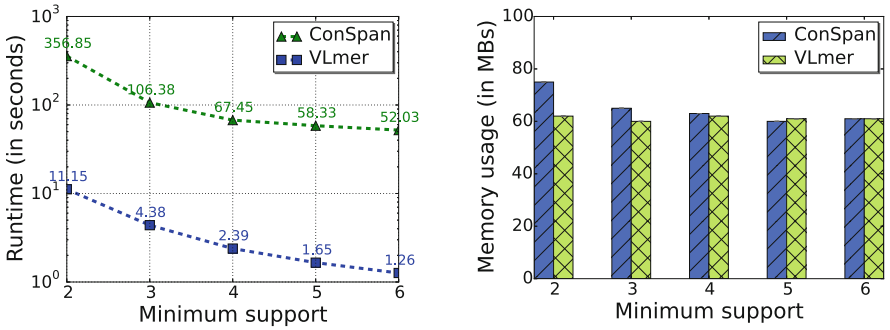
The maximal and mean lengths of *vl*-mers generated by our algorithm are also presented for varied support thresholds. From Fig. 1(b), the maximal length of mined *vl*-mers increases from 20 to 91, with the support thresholds lowering from 6 to 2. An interesting point is that when we set the support threshold as $\sigma = 2$, the maximal length of *vl*-mers reaches 91. These long frequent snippets are often the conserved regions. Figure 1(b) also shows the mean length of *vl*-mers at each σ value, from which one can see that most mean lengths are consistent with their corresponding σ positions of the curve-vertexes.

We also use protein sequence XP_011987916 as the test dataset, to report the characters of mined *vl*-mers. Figure 2(a) shows the distribution of *vl*-mer with varied support thresholds. Except for the snippets of $\sigma = 2$, all protein *vl*-mers are relatively short at 16 and below, while the lengths of DNA *vl*-mers as shown in Fig. 1(a) are far greater than 18. It is easy to see that most protein *vl*-mers fall in the spectrum of 3-mers to 6-mers while DNA *vl*-mers at the range of 6-mers to 10-mers. Like the trend in Fig. 1(b), when the support threshold tends towards a low value, for example, at $\sigma = 2$ as shown in Fig. 2(b), the length of protein *vl*-mers increases dramatically.

3.3 Efficiency Study

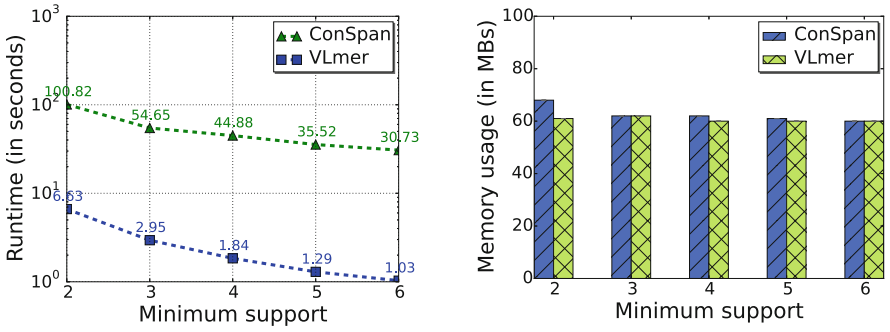
We assessed VLmer efficiency in both runtime and memory usage for the two real datasets in terms of the support threshold. We compare our approach with ConSpan, which is a modified version of both CCSpan [20] and ConSgen [21] algorithms that are most related to our work. Figure 3(a) presents the running time of the two algorithms at different support thresholds on dataset AL607040. The execution time of ConSpan increases from 52.02 to 356.85 s, while VLmer only takes from 1.26 to 11.15 s. At a low support ($\sigma = 2$), VLmer can be 32 times faster than ConSpan. When we raise the σ, for example, at $\sigma = 6$, our algorithm obtain a better performance that reaches 41 times compared to ConSpan.

Figure 3(b) shows the comparison of the memory consumption between the two algorithms on DNA dataset AL607040 shared with the above experiments. In most cases, VLmer and ConSpan have very similar performance in memory usage, while our algorithm requires a smaller memory space in comparison with ConSpan when the σ value is lowered to 2.



(a) Runtime (b) Memory usage

Fig. 3. Runtime and memory usage comparison on DNA sequence AL607040.



(a) Runtime (b) Memory usage

Fig. 4. Runtime and memory usage comparison on protein sequence XP_011987916.

We also compare the running time and memory consumption between VLmer and ConSpan on protein sequence XP_011987916. In Fig. 4(a), the execution time of the two algorithms is illustrated. One can see that, the time consumption of ConSpan ranges from 52.02 ($\sigma = 6$) to 356.85 ($\sigma = 2$) s, while VLmer only from 1.03 to 6.63 s. Obviously, VLmer is significantly faster than ConSpan. As shown in Fig. 4(b), the two algorithms occupy a similar memory space.

From the above efficiency study, we conclude that VLmer has better overall performance for both DNA and protein sequences compared to ConSpan.

4 Conclusion

In this paper, we introduced the problem of mining k-mers of various lengths, i.e., vl-mers, in biological sequences. We presented a novel algorithm, VLmer, which efficiently mines all distinct vl-mers. VLmer first utilizes the inverted index technique to project the original sequences. Then, a pattern-growth approach is adopted to generate potential vl-mers, each of which accurately records their occurrence positions in the original sequences. Three pruning techniques, i.e., repeated candidate pruning, max-suffix pruning, and support pruning, are explored to remove the unpromising candidate vl-mers. All possible vl-mers are generated by running VLmer only once. We used both DNA and protein sequences to evaluate the the performance of VLmer. Our experimental results demonstrated that VLmer is able to analyze both DNA and protein sequences. In the future, we plan to study how to push gap constraint into VLmer in order to mine conserved patterns.

Acknowledgements. This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDB13040700); the National Natural Science Foundation of China (Nos. 91439103; 91529303; 61602460; 31200987); Shanghai Municipal Natural Science Foundation (Nos. 17ZR1406900; 2016M601660); the China Postdoctoral Science Foundation (Nos. 2016M600338; 2016M601660); and the JSPS KAKENHI Grant (No. 15H05707).

References

1. Li, W., Freudenberg, J., Miramontes, P.: Diminishing return for increased mappability with longer sequencing reads: implications of the k-mer distributions in the human genome. BMC Bioinform. **15**(1), 2 (2014)
2. Bremges, A., Singer, E., Woyke, T., Sczyrba, A.: MeCorS: metagenome-enabled error correction of single cell sequencing reads. Bioinformatics **32**(14), 2199–2201 (2016)
3. Hamp, T., Rost, B.: Evolutionary profiles improve protein-protein interaction prediction from sequence. Bioinformatics **31**(12), 1945–1950 (2015)
4. Zhou, J., Troyanskaya, O.G.: Predicting effects of noncoding variants with deep learning-based sequence model. Nat. Methods **12**(10), 931–934 (2015)
5. Kim, D., Song, L., Breitwieser, F.P., Salzberg, S.L.: Centrifuge: rapid and sensitive classification of metagenomic sequences. Genome Res. **26**(12), 1721–1729 (2016)

6. Horwege, S., Lindner, S., Boden, M., Hatje, K., Kollmar, M., Leimeister, C.-A., Morgenstern, B.: Spaced words and KMACS: fast alignment-free sequence comparison based on inexact word matches. Nucleic Acids Res. **42**, W1–W7 (2014)
7. Kurtz, S., Narechania, A., Stein, J.C., Ware, D.: A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes. BMC Genom. **9**(1), 517 (2008)
8. Marçais, G., Kingsford, C.: A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics **27**(6), 764–770 (2011)
9. Melsted, P., Pritchard, J.K.: Efficient counting of k-mers in DNA sequences using a bloom filter. BMC Bioinform. **12**(1), 1 (2011)
10. Deorowicz, S., Debudaj-Grabysz, A., Grabowski, S.: Disk-based k-mer counting on a PC. BMC Bioinform. **14**(1), 1 (2013)
11. Rizk, G., Lavenier, D., Chikhi, R.: DSK: k-mer counting with very low memory usage. Bioinformatics **29**(5), 652–653 (2013)
12. Audano, P., Vannberg, F.: Kanalyze: a fast versatile pipelined k-mer toolkit. Bioinformatics **30**(14), 2070–2072 (2014)
13. Deorowicz, S., Kokot, M., Grabowski, S., Debudaj-Grabysz, A.: KMC 2: fast and resource-frugal k-mer counting. Bioinformatics **31**(10), 1569–1576 (2015)
14. Mamun, A.-A., Pal, S., Rajasekaran, S.: KCMBT: a k-mer Counter based on Multiple Burst Trees. Bioinformatics **32**(18), 2783–2790 (2016)
15. Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., et al.: De novo assembly of human genomes with massively parallel short read sequencing. Genome Res. **20**(2), 265–272 (2010)
16. Shariat, B., Movahedi, N.S., Chitsaz, H., Boucher, C.: HyDA-Vista: towards optimal guided selection of k-mer size for sequence assembly. BMC Genom. **15**(10), S9 (2014)
17. Degnan, P.H., Ochman, H., Moran, N.A.: Sequence conservation and functional constraint on intergenic spacers in reduced genomes of the obligate symbiont buchnera. PLoS Genet. **7**(9), e1002252 (2011)
18. Miranda, R.G., Rojas, M., Montgomery, M.P., Gribbin, K.P., Barkan, A.: RNA binding specificity landscape of the pentatricopeptide repeat protein PPR10. RNA **23**(4), 586–599 (2017)
19. Zhang, R., Xue, R., Yu, T., Liu, L.: Dynamic and efficient private keyword search over inverted index-based encrypted data. ACM Trans. Internet Technol. (TOIT) **16**(3), 21 (2016)
20. Zhang, J., Wang, Y., Yang, D.: CCSpan: mining closed contiguous sequential patterns. Knowl.-Based Syst. **89**, 1–13 (2015)
21. Zhang, J., Wang, Y., Zhang, C., Shi, Y.: Mining contiguous sequential generators in biological sequences. IEEE/ACM Trans. Comput. Biol. Bioinf. **13**(5), 855–867 (2016)
22. Zhang, J., Wang, Y., Wei, H.: An interaction framework of service-oriented ontology learning. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, pp. 2303–2306. ACM (2012)
23. Zhang, J., Wang, Y., Yang, D.: Automatic learning common definitional patterns from multi-domain Wikipedia pages. In: 2014 IEEE International Conference on Data Mining Workshop (ICDMW), pp. 251–258. IEEE (2014)
24. Leung, K.-S., Wong, K.-C., Chan, T.-M., Wong, M.-H., Lee, K.-H., Lau, C.-K., Tsui, S.K.: Discovering protein-DNA binding sequence patterns using association rule mining. Nucleic Acids Res. **38**(19), 6324–6337 (2010)