Marek KOKOT*, Sebastian DEOROWICZ

Chapter 6. OPTIMIZING SIGNATURE SELECTION SCHEME FOR *K*-MER COUNTING PROBLEM

6.1. Motivation

6.1.1. *k*-mer counting

The amount of data produced by next-generation sequencing instruments is enormous. A single experiment may result in hundreds of billions of symbols (bases) grouped in reads, each of length up to hundreds of letters. Reads are short (in comparison with the genome length) and are not error-free. To be able to obtain any knowledge from such a noisy data, it is required to have each genome base represented into multiple reads. The average number of reads in which a single base is represented, is called depth of coverage. The higher the depth of coverage, the better for quality of analyses. Managing such amounts of data is challenging and requires development of high-performance algorithms. Many bioinformatics analyses are realized in a form of a pipeline constituted of a couple of steps. Often the first step is so-called *k*-mer counting. Formally, *k*-mer (*k*-length string) counting algorithm may be defined as follows. The input of the algorithm are a set of sequences (usually reads) and the value of *k*. The output is a set of pairs, where the first element is a *k*-mer being a substring of at least one input sequence and the second element is the number of occurrences of this *k*-mer in all the sequences. The strand (forward or reverse) a read originates from is unknown. Thus, usually canonical *k*-mers are counted. A canonical *k*-mer is the lexicographically smaller of a *k*-mer and its reverse complement. A reverse complement of a *k*-mer is obtained by reversing its symbols and changing them according to the mapping: A→T, C→G, G→C, T→A. For example, a reverse complement of 5-mer **ATGCT** is **AGCAT**, and

* Corresponding author: marek.kokot@polsl.pl, ul. Akademicka 16, 44-100 Gliwice, PL. Department of Algorithmics and Software, Silesian University of Technology.

as the reverse complement is lexicographically smaller, it is also a canonical *k*-mer. The ordering of *k*-mer does not need to be lexicographic, for example, DSK [1] uses A<C<T<G as an ordering of symbols.

Some examples of bioinformatics analyses where *k*-mer counting is used are de novo assembly [2], fast multiple sequence alignment [3], repeat detection [4], correction of sequencing reads [5]. There are also alignment-free analyses in which sets of *k*-mers of multiple samples are combined to get some biological knowledge. For example, in a DIAMUND [6] algorithm, *k*-mers of healthy parents and a child with a genetic disease are combined to find novel mutations. Alternatively, the same strategy may be applied to healthy and diseased (e.g., cancerous) tissues. Another examples of similar approaches are NIKS [7], which is also designed to find a mutation in closely related samples and GenomeTester4 [8], which allows to find group-specific *k*-mers to identify bacteria.

There were many approaches designed to solve *k*-mer counting problem presented in the literature. Very often they utilize specialized, dictionary-based data structures tuned to work well in a parallel environment. Jellyfish [12], DSK [1], Gerbil [16], MSPKmerCounter [15] use hash tables as a main data structure. In KMC 1 [17], KMC 2 [18], KMC 3 [19] sorting is used to identify identical *k*-mers. The khmer [9, 10], BFCounter [13] and Turtle [14] are examples of algorithms utilizing Bloom filters. In a KCMBT [20] multiple tries are used. Tallymer [21] exploits suffix arrays. Some of the mentioned algorithms are so-called disk-based, which means they utilize disk space to reduce memory consumption. Those are KMC 1, KMC 2, KMC 3, DSK, Gerbil, MSPKmerCounter. The main idea of these algorithms is to split the data into many bins, in a way that allows to process each bin independently and consuming less memory than in the case of processing the whole input at once. The best in terms of memory usage and computation time are KMC 3, DSK, and Gerbil. A detailed comparison of *k*-mer counting software may be found in [11].

6.1.2. Disk-based approaches and optimization criteria

Disk-based *k*-mer counters are usually two-stage algorithms. In the first stage, all *k*-mers are extracted from the input sequences and assigned to one of n bins. Each bin is, in fact, a disk file. The *k*-mers are assigned to bins using some function f , so it is assured that each instance of the same *k*-mer in the input will be stored in the same bin. The number of bins, n, is a couple of hundred which is much less than the number of distinct *k*-mers,

so in one bin there are many distinct *k*-mers. This way each bin may be processed independently in the second stage. Concatenation of k-mers counted per each bin is a final result of the algorithm.

It is desired to keep the bins evenly distributed, more precisely to keep the biggest (in a term of the number of contained *k*-mers) bin as small as possible. The reason is that all *k*-mers from a bin must be loaded into RAM, so the number of *k*-mers in the largest bin determines the minimal amount of RAM required to finish computations. On the other hand, it is also desired to keep the total size of bins (in bytes this time) as small as possible. The total bins size determines disk space requirements for intermediate files. In most cases, the disk space is not an issue as it is quite cheap (especially in comparison with RAM), but the time access is much higher. For this reason, keeping the total bins size small does not only decrease disk space requirements, but also decreases running time. The main issue with storying *k*-mers in intermediate files is the fact that two adjacent *k*-mers from the input sequence share $k - 1$ common symbols, but when assigned to the different bins, the common symbols must be stored twice. Having *f* increasing chances for a series of adjacent *k*-mers to be assigned to the same bin would help reduce the redundancy. Minimizer of a *k*-mer is a lexicographically smallest *m*-mer ($m < k$) of a *k*-mer. It is likely for two adjacent *k*-mers to have the same minimizers which makes the minimizer approach a good candidate of *f* . If *l.* consecutive *k*-mers share the same minimizer, they may be stored together occupying $k + f. - 1$ symbols in a form of super-*k*-mer. This is much better than storying the *k*-mers in plain and spending $l \times k$ symbols. The example presenting this idea is shown in Fig. 6.1.

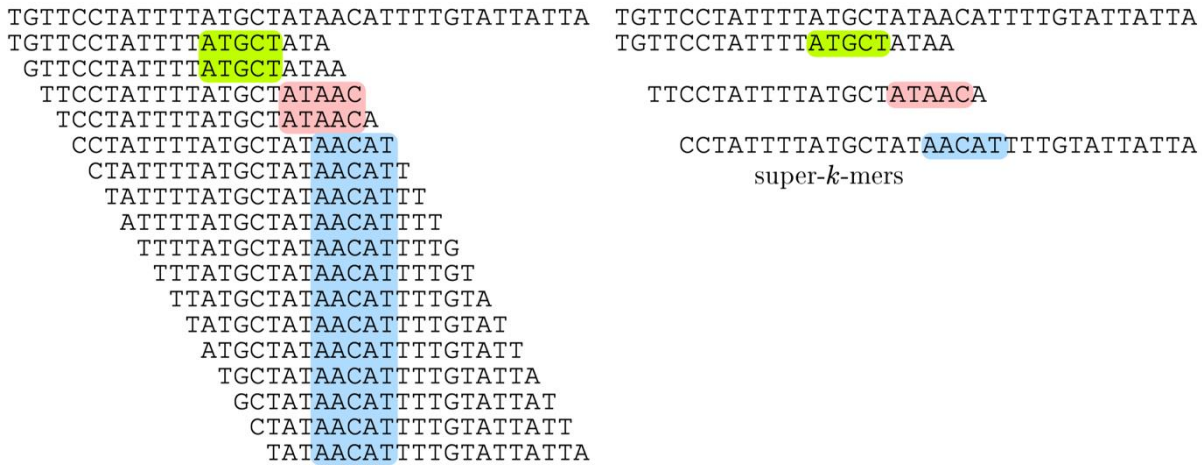

Fig. 6.1. Example of minimizer-based *k*-mer binning. n the left side, the sequence and all its 20-mers are presented. Colors were used to mark minimizers for $m = 5$. All *k*-mers with the same minimizer may be stored as a longer sequence (right side)

Rys. 6.1. Przykład przyporządkowania *k*-merów do kubełków opartego na minimizerach. Po lewej stronie przedstawiono sekwencję i wszystkie zawarte w niej 20-mery. Za pomocą kolorów oznaczono minimizery dla m=5. Wszystkie *k*-mery z tym samym minimizerem mogą być przechowywane w postaci dłuższej sekwencji (prawa strona)

Minimizers are not perfect, for example, usually the size of the biggest bin is much larger than the size of the second biggest. Also, the total size of bins seems not as good as possible. The with minimizers were previously noticed and described in [18]. It has been shown, that some *m*-mers tend to produce short super-*k*-mers (which corresponds to higher total bins size, which in turn leads to higher disk usage and slower computations) and also causes bin sizes imbalance. As a solution signatures were proposed. The idea is to treat some *m*-mers as disallowed, meaning they are skipped when finding the lowest *m*-mer in a *k*-mer. There is a special bin for a *k*-mers built with only disallowed *m*-mers. This approach allowed to reduce both total bins sizes and the size of the largest bin.

It may be observed that the greater m is the lower is the probability that two adjacent *k*- mers share the same minimizer (or signature). It leads to a higher number of super-*k*-mers and therefore to higher disk usage. n the other hand, the higher *m* the better *k*-mers are distributed into bins. As a consequence, some compromise must be made and typically $5 \leq m \leq 12$. In practice, when canonical *k*-mers are counted, only canonical *m*-mers are taken into account to assure the correctness of the result. The number of possible canonical *m*-mers (and also the number of bins) is 2^{2m-1} if m is even and $2^{m-1}(2m + 1)$ if *m* is odd. As it was mentioned, it is desired to keep the number of bins couple of hundreds. For this reason, in practice, some bins are merged.

In [23] the frequency-based ordering was proposed. The idea is to count all *m*-mers from a part of the input data and sort the pairs (*m*-mers and counters) in non-decreasing order of counters. The resulting order of *m*-mers is used instead of a lexicographic one. Although this approach decreases the size of the largest bin significantly, it does not take into account the aspect of the total number of super-*k*-mers. In [24, 25, 26] a universal hitting set, UHS, was presented. UHS is defined for specific values of *m* and *k*. It is a set of *m*-mers that for each possible *k*-mer it is assured that at least one of its *m*-mers is in UHS. In the simplest case, it is a set of all possible *m*-mers, but the presented idea is to find minimal UHS, i.e., containing as low *m*-mers as possible (without violating the UHS definition). Having UHS the idea to count *k*-mers is to treat all *m*-mers $\notin$ UHS as disallowed.

6.2. Methods

Our goal is to find a minimizer-based binning scheme that minimizes two criteria:

1. The total number of super-k-mers and.
2. The total number of k-mers assigned to the largest bin.

Usually, these criteria are exclusive, but as it turns out to some extent they may be both optimized. All presented heuristics starts with minimizers (each canonical m-mer allowed). During optimization, some m-mer will be disabled or re-enabled. The work is focused on human sequencing data. The first approaches were performed on human chromosome 22, but the results did not generalize to sequencing data. The same effect occured for a bigger chromosome 11. In consequence as a training set 1,500,000 reads from real human sequencing data were taken.

The starting approach was to iteratively disable m-mer related to the largest bin. Single iteration was to find such *m*-mer and block it The number of iterations was set to 1,000. At each iteration, the number of super-*k*-mers, the total number of *k*-mers in the largest bin, and the total number of *k*-mers in the bin for disabled *m*-mers was recorded.
The main approach was based on simulated annealing, SA, [22]. The cost function was:

$$C = w_s n_s + w_m n_m, \tag{1}$$

where n_s is a total number of super-*k*-mers, nm is the number of *k*-mers in the largest bin, w_s and w_m are weights set to $w_s = 1$, $w_m = 250$. The starting temperature, t_0, was set to $-0.01C_s/\ln(P_0)$, where C_s is the value of cost function for starting solution (all *m*-mers allowed), P_0, an initial probability of acceptance of a worse solution, was set to 0.5. Let us denote the current temperature by *t*, the number of iteration by *N*, terminate temperature by t_k. At the end of each iteration, the temperature is decreased to αt, where $\alpha = \sqrt[N]{t_k/t_0}$. At each iteration, a random *m*-mer is chosen and its state (allowed/disallowed) is reversed. It is more likely that lexicographically smaller *m*-mers will be selected as minimizers, consequently, lexicographically larger *m*-mers will tend to have only little impact on the value of cost function. For this reason, *m*-mers are not selected with uniform distribution. Instead, one of three geometric distributions is used at each iteration. The probability of success of those distributions are respectively 0.005, 0.002, 0.0005.

6.3. Results

In the experiments, three data sets were used, each containing sequencing reads of the human genome. The first one, denoted as HS1, is of size 1.35G bases, the second one, denoted as HS2, is of size 736.4G bases. In both sets, the average read length was 101. The third set, denoted as HS1 part, was a part of HS1 data set containing 1.5M reads. Results of the simple approach based on disallowing m-mer related to the largest bin are present in Fig. 6.2.

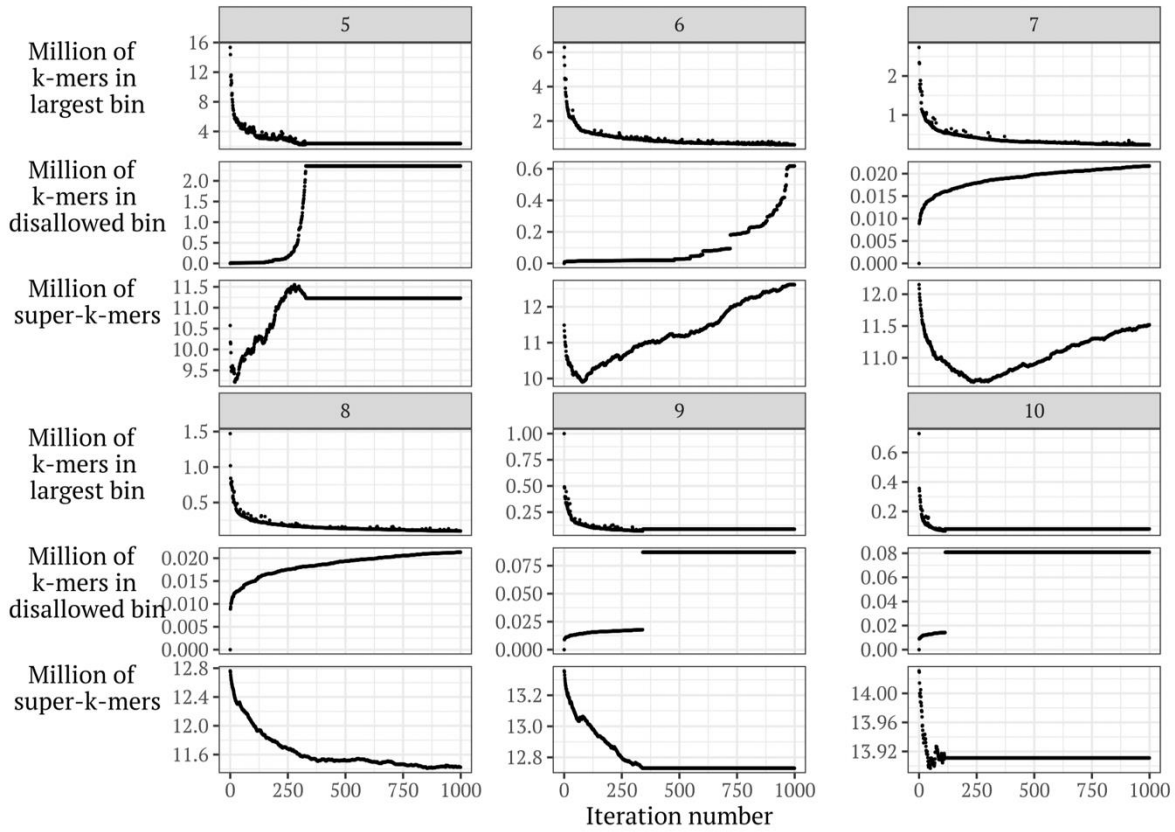

Fig. 6.2. Results of optimization based on iterative disallowing *m*-mer related to the largest bin, $k = 28$, *m* is shown in the gray rectangle

Rys. 6.2. Wyniki optymalizacji bazującej na iteracyjnym blokowaniu *m*-mera związanego z największym kubełkiem, $k = 28$, w szarym prostokącie przedstawiona jest wartość *m*

At the beginning (iteration 0; all *m*-mers are allowed) when *m* increases the number of *k*-mers in the largest bin decreases, and the number of super-*k*-mers increases. It is in line with theoretical expectations. The fastest decrease of the number of *k*-mers in the largest bin is in the first 250 iterations. Interestingly, in this range also the number of super-*k*-mers decreases. The decrease is especially significant for $m < 8$. It may be noticed that with further iterations the number of super-*k*-mers starts to grow and for $m \in \{5, 6\}$ it even exceeds the number of super-*k*-mers at iteration 0. For $m \in \{5, 9, 10\}$

there is a saturation after a relatively small number of iterations, which is because the bin for disallowed *m*-mers becomes the largest one. For $m = 5$, the total number of *m*-mers is low, and after a couple of hundreds of iterations a large fraction of *m*-mers is disabled, which leads to many *k*-mers containing only disallowed *m*-mers. The case of $m > 8$ is more complex, the size of the largest bin is relatively small. What is interestring, the last disallowed *m*-mer is **TCCATTCCA** ($m = 9$), **TGGAATGGAA** ($m = 10$), **CCATTCCATTC** ($m = 11$, not presented on the plot). It may be speculated that the largest bin contains k-mers with tandem repeats.

Although this simple optimization technique allows optimizing both criteria to some point, it is not perfect. The SA and simple approach were run for HS 1 part, and resulting binning were used for evaluation on all data sets. The number of iterations in SA was set to 2 M. The evaluation was performed using a modified version of KMC, in which original binning was replaced with the new ones. For HS 1 part, the number of *k*-mers in the largest bin was 280.90 *k* for $m = 7$, which is a minimal value from all tested heuristics, but at the same time, the total number of super-*k*-mers was 11.53 M. It is worse than in the case of KMC (10.32 M) and only slightly better than in the case of pure minimizers (12.15 M). SA for $m = 7$ leads to 9.84 M super-*k*-mers which is the best value found in experiments. The number of *k*-mers in the largest bin was 384.25 k, which is better than KMC (711.39 k). For HS 1 also simple optimization for $m = 7$ leads to the best result in terms of the size of the largest bin (216.46 M), but again at a cost of the total number of super-*k*-mers (10.41 G), while in case of SA it is only 8.89 G (and the size of largest bin is 348.09 M). Results of SA for $m = 8$ are also interesting as the total number of super-*k*-mers is 9.35 M and the number of *k*-mers in the largest bin is 225.84 M (which is close to the best). In the case of HS 3 the best result in terms of the largest bin are for SA for $m = 8$ (1.19 G), the total number of super-*k*-mers, in this case, is 51.02 G. The best result in terms of the total number of super-*k*-mers is SA for $m = 7$ (48.48 G), the number of *k*-mers in the largest bin, in this case, is 2.04. One dimensional model was predicated using described geometry and parameters to predict creation of rings. However, problem is only described on one coordinate, rings aren’t noticeable – only where product precipitate.

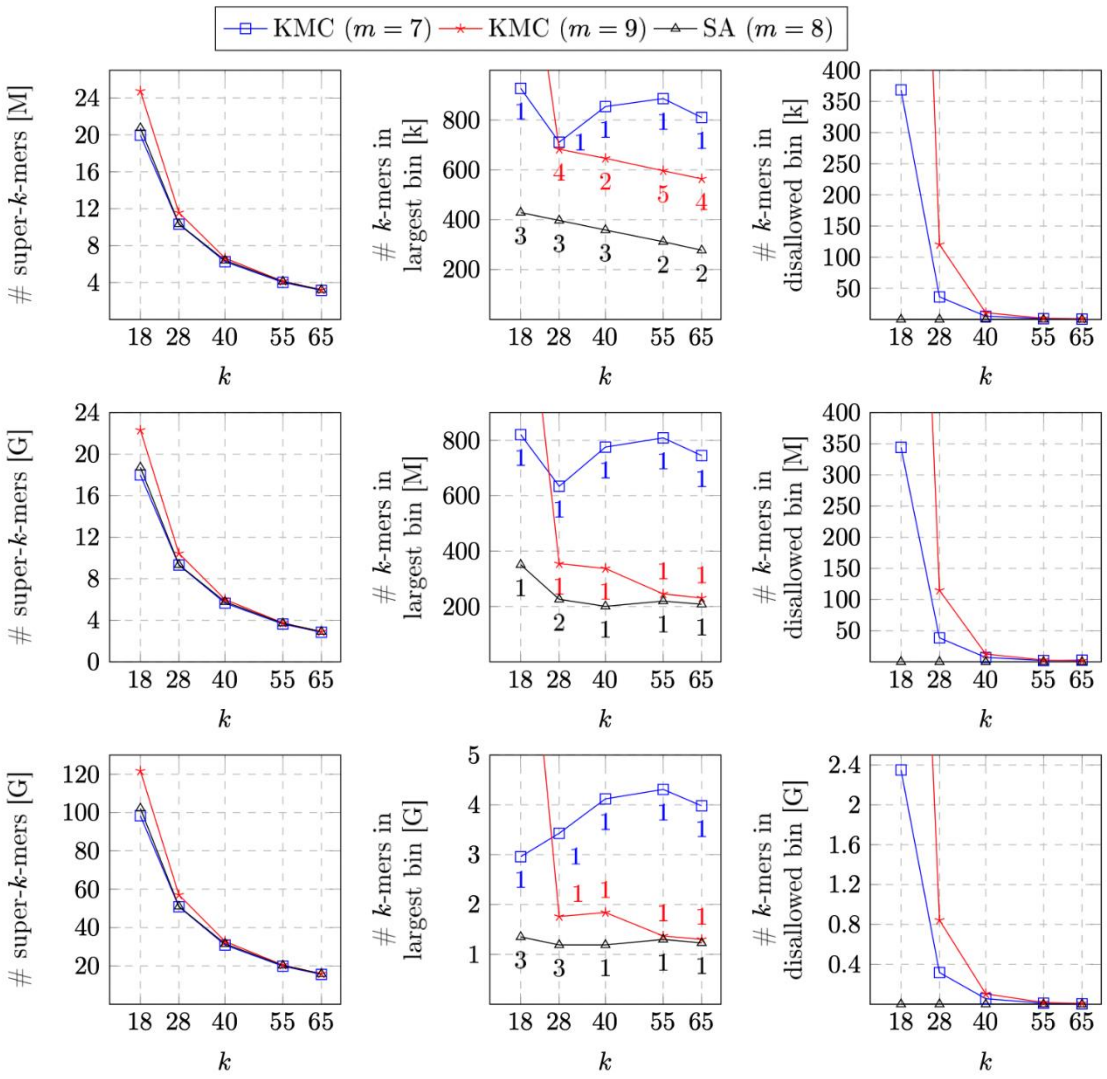

Fig. 6.3. Comparison of signature binning used in KMC and developed with simulated annealing for various *k* values. The first row of plots shows results for a part of HS 1 data set used as an input of simulated annealing algorithm, the second row shows the result on whole HS 1 data set, the third shows results for HS 2 data set. The first column presents the total number of super *k*-mers, the second one presents the total number of *k*-mers in the largest bin, the third one presents the total number of *k*-mers in the bin of disallowed *m*-mers. Additionally, the number of *m*-mers assigned to the biggest bins are shown on plots in the second column. In case of KMC (m = 9, k = 18) some values are excluded from charts to improve readability. This is caused by the fact that the disabled bin became the largest one. Its sizes were 1.78M, 1.6G, 9.83G *k*-mers for HS 1 part, HS 1 and HS 2 respectively

Rys. 6.3. Porównanie przyporządkowania *k*-merów do kubełków użytego w algorytmie KMC i opracowanego z wykorzystaniem symulowanego wyżarzania dla różnych wartości *k*. Pierwszy wiersz wykresów przedstawia wyniki dla wycinka zestawu HS 1, który był użyty jako wejście algorytmu symulowanego wyżarzania, drugi wiersz przedstawia wyniki dla całego zestawu HS 1, trzeci wiersz przedstawia wyniki dla zestawu HS 2. Pierwsza kolumna przedstawia całkowitą liczbę super-*k*-merów, druga przedstawia całkowitą liczbę *k*-merów w największym kubełku, trzecia przedstawia całkowitą liczbę *k*-merów w kubełku zabronionych *m*-merów. Dodatkowo liczba *m*-merów przypisanych do największego kubełka jest przedstawiona na wykresach w drugiej kolumnie. W przypadku KMC (m=9, k=18) niektóre wartości zostały usunięte z wykresów dla poprawy czytelności. Jest to spowodowane tym, że kubełek zablokowanych *m*-merów staje się największym kubełkiem. Jego rozmiary to 1,78 M, 1,6 G, 9,83 G *k*-merów dla zestawów wycinek HS 1, HS 1, HS 2 odpowiednio

According to these results SA with $m = 8$ was chosen as the best and compared to original KMC m-mers binning for various values of k on all data sets. The results are presented in Fig. 6.3. For $k = 18$, $m = 9$ the largest bin is a bin of disallowed m-mers. As expected, in the case of KMC increasing m causes a reduction of the size of the largest bin and an increase of the total number of super-k-mers. It may be noticed that the number of super-k-mers for SA is only slightly higher than in the case of KMC with $m = 7$. The size of the largest bin is relevantly lower in the case of SA than in the case of KMC which compensates a slightly higher total number of super-k-mers. It is also worth noting that results presented on the charts expose the number of k-mers per file and, in some cases, the largest file refiects multiple m-mers. It means that the size of the largest file could be further lowered in KMC by allowing more temporary files (in KMC the default number of intermediate files is 512).

6.4. Discussion

In this paper two optimization methods for m-mer binning scheme for k-mer counting problem were presented. The first method was very simple and was intended to show some properties of the minimizer-based approach in k-mer counting. The second approach, simulated annealing, was more complex and allowed to optimize both criteria (total number of super-k-mers as well as the number of k-mers in the largest bin). It confirms that the parameters of this metaheuristic were suitable. The results show that a trained binning scheme leads to satisfactory results when used as a part of KMC algorithm. In future work, the impact on computational time, RAM, and disk usage should be measured.

Bibliography

1. G. Rizk, D. Lavenier, R. Chikhi, DSK: k-mer counting with very low memory usage, Bioinformatics (2013) 29(5):652-653.
2. P.E.C. Compeau, P.A. Pevzner, G. Tesler, How to apply de Bruijn graphsto genome assembly, Nature Biotechnology (2011) 29(11):987.
3 R.C. Edgar, MUSCLE: multiple sequence alignment with high accuracy and high throughput, Nucleic Acids Research (2004) 32(5):1792-1797.

4 S. Kurtz, A. Narechania, J. C. Stein, D. Ware, A new method to compute K- mer frequencies and its application to annotate large repetitive plant genomes, BMC Genomics (2008) 9(1):517.

5. Y. Liu, J. Schroder, B. Schmidt, Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data, Bioinformatics (2012) 29(3):308-315.

6. S.L. Salzberg, M. Pertea, J.A. Fahrner, N. Sobreira, DIAMUND: Direct comparison of genomes to detect mutations, Human Mutation (2014) 35(3):283-288.

7. K.J.V. Nordstrom, M.C. Albani, G.V. James, C. Gutjahr, B. Hartwig, F. Turck, U. Paszkowski, G. Coupland, K. Schneeberger, Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers, Nature Biotechnology (2013) 31(4):325.

8. L. Kaplinski, M. Lepamets, M. Remm, GenomeTester4: a toolkit for performing basic set operations-union, intersection and complement on k-mer lists, Gigascience (2015) 4(1):58.

9. Q. Zhang, J. Pell, R. Canino-Koning, A.C. Howe, C.T. Brown, These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure, PloS one (2014) 9(7):e101271.

10. M.R. Crusoe, H.F. Alameldin, S. Awad, E. Boucher, A. Caldwell, R. Cartwright, A. Charbonneau, B. Constantinides, G. Edvenson, S. Fay et al., The khmer software package: enabling efficient nucleotide sequence analysis, F1000Research (2015) 4.

11. Manekar, Swati C., Shailesh R. Sathe, A benchmark study of k-mer counting methods for high-throughput sequencing, GigaScience (2018) 7.12: giy125.

12. G. Marcais, C. Kingsford, A fast, lock-free approach for efficient parallel counting of occurrences of k-mers, Bioinformatics (2011) 27(6):764-770.

13. P. Melsted, J. K. Pritchard, Efficient counting of k-mers in DNA sequences using a bloom filter, BMC Bioinformatics (2011) 12(1):333.

14. R.S. Roy, D. Bhattacharya, A. Schliep, Turtle: Identifying frequent kmers with cache-efficient algorithms, Bioinformatics (2014) 30(14):1950-1957.

15. Y. Li et al., MSPKmerCounter: a fast and memory efficient approach for k-mer counting, arXiv preprint arXiv:1505.06550 (2015).

16. M. Erbert, S. Rechner, M. Muller-Hannemann, Gerbil: a fast and memoryefficient k-mer counter with GPU-support, Algorithms for Molecular Biology (2017) 12(1):9.

17. S. Deorowicz, A. Debudaj-Grabysz, S. Grabowski, Disk-based k-mer counting on a PC, BMC Bioinformatics (2013) 14(1):160.
18. S. Deorowicz, M. Kokot, S. Grabowski, A. Debudaj-Grabysz, KMC 2: fast and resource-frugal k-mer counting, Bioinformatics (2015) 31(10):1569-1576.
19. M. Kokot, M. Dlugosz, S. Deorowicz, KMC 3: counting and manipulating kmer statistics, Bioinformatics (2017) 33(17):2759-2761.
20. A.A. Mamun, S. Pal, S. Rajasekaran, KCMBT: a k-mer Counter based on Multiple Burst Trees, Bioinformatics (2016) 32(18):2783-2790.
21. S. Kurtz, A. Narechania, J.C. Stein, D. Ware, A new method to compute K- mer frequencies and its application to annotate large repetitive plant genomes, BMC Genomics (2008) 9(1):517.
22. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi et al., Ptimization by simulated annealing, Science (1983) 220(4598):671680.
23. R. Chikhi, A. Limasset, S. Jackman, J.T. Simpson, P. Medvedev, n the representation of de Bruijn graphs, In International conference on Research in computational molecular biology, pages 35-55. Springer (2014).
24. G. Marcais, D. Pellow, D. Bork, Y. Renstein, R. Shamir, C. Kingsford, Improving the performance of minimizers and winnowing schemes, Bioinformatics (2017) 33(14):i110i117.
25. Y. Orenstein, D. Pellow, G. Marcais, R. Shamir, C. Kingsford, Compact universal k-mer hitting sets, In International Workshop on Algorithms in Bioinformatics, pages 257-268. Springer (2016).
26 Y. Orenstein, D. Pellow, G. Marcais, R. Shamir, C. Kingsford, Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing, PLoS Computational Biology (2017) 13(10):e1005777.

OPTIMIZING SIGNATURE SELECTION SCHEME FOR *K*-MER COUNTING PROBLEM

Abstract

The first step of many bioinformatic applications is counting all substrings of specific length occurring in the set of input sequences. Despite the conceptual simplicity of this task, it is far from trivial on real data sets. Therefore, there were a lot of algorithms

proposed in the literature. Some of them are disk-based, where disk space is used to reduce RAM consumption. Efficient algorithms of this group tend to split input data into multiple bins that may be further processed independently. The splitting procedure seems to be crucial in terms of disk and RAM usage. In this paper, we discuss two optimization techniques to improve the splitting procedure. The first one is simple and is used to characterize the data. The second one is more complex as it is based on a simulated annealing metaheuristic.

Keywords: *k*-mer counting, simulated annealing, metaheuristics.