Bioinformatics, 33(17), 2017, 2759–2761
doi: 10.1093/bioinformatics/btx304
Advance Access Publication Date: 4 May 2017
Applications Note

Sequence analysis

KMC 3: counting and manipulating *k*-mer statistics

Marek Kokot, Maciej Długosz and Sebastian Deorowicz*

Institute of Informatics, Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, Gliwice 44-100, Poland

*To whom correspondence should be addressed.

Associate Editor: Bonnie Berger

Received on January 30, 2017; revised on April 19, 2017; editorial decision on May 2, 2017; accepted on May 3, 2017

Abstract

Summary: Counting all *k*-mers in a given dataset is a standard procedure in many bioinformatics applications. We introduce KMC3, a significant improvement of the former KMC2 algorithm together with KMC tools for manipulating *k*-mer databases. Usefulness of the tools is shown on a few real problems.
Availability and implementation: Program is freely available at http://sun.aei.polsl.pl/REFRESH/kmc.
Contact: sebastian.deorowicz@polsl.pl
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

In many applications related to genome sequencing, e.g. *de novo* assembly, read correction, repeat detection and comparison of genomes, the first step is *k-mer counting*. This procedure consists in determining all unique *k*-symbol long strings (usually with counters) in the read collection.

From the conceptual point of view *k*-mer counting is quite simple task. Nevertheless, the problems appear when we deal with real data, which could be huge. There are many articles published in the recent years that discuss this problem, which shows that providing an efficient tool solving this task is far from trivial.

The obtained *k*-mer statistics are of course only a point of departure for following analyzes. In some of them various operations on sets of *k*-mers are necessary. We introduce a new version of KMC2 (Deorowicz *et al.*, 2015), which is more memory frugal and much faster. We also introduce KMC tools for easy manipulation of sets produced by KMC. The usefulness of KMC tools is shown on a few literature case studies in which various utilities were replaced by operations from our package with significant gains in processing time and memory usage.

2 Materials and methods

KMC3 follows the same two-stage processing scheme as KMC2. In the first stage the reads are split into several hundred bins (disk files) according to the signatures (short *m*-symbol long substrings) of *k*-mers. The bins are then sorted one by one to remove duplicates in the second stage. A similar processing was used in several recently published algorithms for *k*-mer counting. Their first stages are similar to KMC2, but then the bins are handled in different ways, e.g. Gerbil (Erbert *et al.*, 2017) employs hash tables, DSK2 (Rizk *et al.*, 2013) uses multistage processing in a case of small disk space. Recently published, KCMBT (Mamun *et al.*, 2016) applies burst tries for *k*-mer storage. Jellyfish (Marcais and Kingsford, 2011) utilizes thread-safe hash tables.

There are several main novelties in KMC3. Concerning the first stage, input files, especially in gzipped FASTQ format, are loaded faster due to better input/output (I/O) subsystem. Signatures are assigned to bins in an improved way, i.e. larger default signature length (nine instead of seven) is picked and larger part of input data is taken in the pre-processing stage in which the signatures are assigned to bins. These lead to better balance of bin sizes (at the cost of slightly larger total size of bins), which results in smaller main memory requirements and faster processing in the second stage. The most significant improvements are in the second stage. Both KMC (Deorowicz *et al.*, 2013) and KMC2 used the least-significant-digit (LSD) radix sort. When the keys to sort are long (which happens when $k > 32$) the LSD radix sort requires unnecessarily many iterations over the data. Therefore, in KMC3 we implemented the fastest existing most-significant-digit radix sort (Kokot *et al.*, 2016). We also improved the parallelization scheme of other routines (not directly related to sorting) at this stage.

The second part of the package consists of various tools for manipulation of sets containing *k*-mers. Figure 1 shows a general scheme of the package (complete description is given in the Supplementary Material). The *filtering* allows to extract from FASTQ files the reads satisfying some criteria, e.g. with sufficiently large number of *k*-mers occurrences. The family of *transform* operations allows to modify the KMC database, e.g. remove too frequent/rare *k*-mers, remove counters, build histogram. The family of *simple* operations is composed of several set operations involving more than one KMC database. For example, the user is able to intersect, union the databases. Finally, the *complex* operations allow to construct compound expressions taking as parameters KMC databases.

3 Results

We used six datasets for KMC3 evaluation. Table 1 shows the results for two representatives. The description of the datasets, the platform used for tests, the remaining results are given in the Supplementary Material. FASTQ files are almost always stored in compressed form. Thus, we provide the running times for both: uncompressed and gzipped FASTQ.

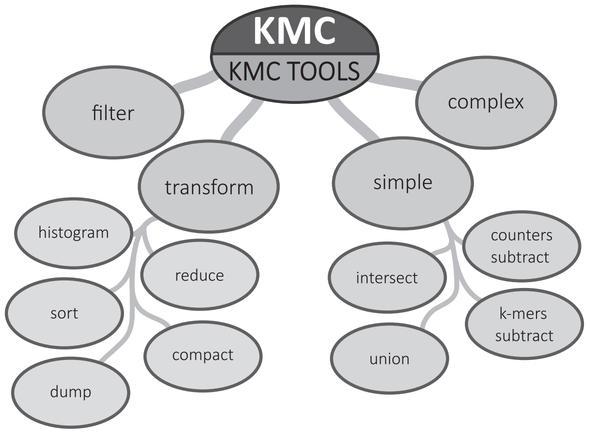

Fig. 1. Scheme of KMC 3 package

As we can observe for the *Gallus gallus* data, KMC3 is usually the fastest, especially for larger *k*. The times for *Homo sapiens* 3, the largest of our dataset, are much longer. Nevertheless, KMC3 was able to complete in <100 min for typical input format for both examined *k*. It can be observed, that the improved I/O subsystem as well as the new sorting routine and better parallelization of the second stage gave substantial benefits comparing to KMC2. It is also worth noting that KMC3 even for the largest dataset used a reasonable amount of memory.

To evaluate KMC tools we picked three studies described recently in the literature. Below we briefly describe the goals of the studies and total gains in time and memory. The detailed results together with the scripts showing KMC tools usage are given in the Supplementary Material.

DIAMUND (Salzberg *et al.*, 2014) is a novel approach for variant detection. It is dedicated to comparison of family trios or normal and diseased samples of the same individual. Instead of mapping the reads onto a reference genome, DIAMUND compares directly the raw reads. We followed this protocol for the Ashkenazim Jewish ancestry trio (Zook *et al.*, 2016). DIAMUND needed 13 h to complete its work and used 107 GB RAM. As most of the stages can be made using KMC tools we replaced the original tools used in DIAMUND and reduced the processing time to 4 h (reducing memory usage to 12 GB RAM).

NIKS (Nordström *et al.*, 2013) is another tool that uses raw sequencing reads for mutation identification taking into account mainly *k*-mer statistics. We replaced its stages related to *k*-mer counting and manipulation of *k*-mer databases by KMC tools operations. In a single stage it was necessary to use KMC API to prepare a short C++ program to reproduce the format of intermediary data used in NIKS. The processing time for *Oryza sativa* dataset (83.3 GB input FASTQ) was reduced from ~40 to 5 min in this case (RAM usage was reduced from 92 to 12 GB).

Finally, we made the same experiment as used in the GenomeTester4 article (Kaplinski *et al.*, 2015), introducing a tool

Table 1. Comparison of *k*-mers counting algorithms

Algorithm	$k = 28$			$k = 55$		
	RAM	Disk	Time/gz-Time	RAM	Disk	Time/gz-Time
G. gallus (35 Gbases in total)						
DSK 2	14	39	783/1180	15	30	719/1123
Gerbil	2	24	607/631	4	17	615/551
GTester4	74	0	2494/—	*unsupported k*		
Jellyfish 2	33	0	909/946	77	0	1048/919
KCMBT	107	0	1335/—	*unsupported k*		
KMC 2	12	25	489/321	12	18	861/672
KMC 3	12	28	492/292	12	18	455/245
H. sapiens 3 (729 Gbases in total)						
Gerbil	28	523	11 994/12 730	62	364	11 968/12 469
Jellyfish 2	84	251	38 338/20 284	104	636	31 783/31 345
KMC 2	64	551	10 777/9036	72	381	13 774/11 804
KMC 3	33	596	9631/5985	34	389	8750/5331

The times are given for: uncompressed input FASTQ file ('Time') and gzipped input FASTQ files ('gz-Time') The units are: seconds (time), GB (Disk and RAM). All programs were executed in 12-threads mode (at the Xeon-based workstation using HDD, see Supplementary Material, Section 2 for details), except for KCMBT which was executed in 8-threads mode due to large main memory consumption (exceeding 128 GB RAM for 12 threads). '—' means that the mode is not supported. There are no results for some programs for *H. sapiens* 3 due to: processing longer than 12 h (DSK 2, GTester4) or memory requirements larger than 128 GB (KCMBT).

that is able to perform similar operations as KMC tools. The investigated problem was to find group-specific *k*-mers to identify bacteria. The processing time was reduced from 5 h to 15 min (RAM usage reduction from 75 to 12 GB).

4 Discussion

We proposed a new version of our *k*-mer counter. It is much faster (even a few times for large *k*) than its predecessor and faster than the existing competitors. The KMC tools offer a number of operations on *k*-mer databases that can be used in projects making use of *k*-mer statistics.

Funding

This work was supported by the Polish National Science Centre under the project DEC-2015/17/B/ST6/01890 and by Silesian University of Technology grant no. (BKM507/RAU2/2016). We used the infrastructure supported by POIG.02.03.01-24-099/13 grant: 'GeCONiI–Upper Silesian Center for Computational Science and Engineering'.

Conflict of Interest: none declared.

References

Deorowicz,S. *et al*. (2013) Disk-based *k*-mer counting on a PC, *BMC Bioinformatics*, **14**, 160.

Deorowicz,S. *et al*. (2015) KMC 2: Fast and resource-frugal *k*-mer counting. *Bioinformatics*, **31**, 1569–1576.

Erbert,M. *et al*. (2017) Gerbil: a fast and memory-efficient *k*-mer counter with GPU-support. *Algorithms Mol. Biol*., **12**, 9.

Kaplinski,L. *et al*. (2015) GenomeTester4: a toolkit for performing basic set operations—union, intersection and complement on *k*-mer lists. *GigaScience*, **4**, 58.

Kokot,M. *et al*. (2016) Sorting data on ultra-large scale with RADULS. In: Kozielski, S. *et al*. (eds) *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation*. Springer, pp. 235–245.

Mamun,A.A. *et al*. (2016) KCMBT: a *k*-mer counter based on multiple burst trees. *Bioinformatics*, **32**, 2783–2790.

Marçais,G. and Kingsford,C. (2011) A fast, lock-free approach for efficient parallel counting of occurrences of *k*-mers. *Bioinformatics*, **27**, 764–770.

Nordström,K.J.V. *et al*. (2013) Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using *k*-mers. *Nat. Biotechnol*., **31**, 325–330.

Rizk,G. *et al*. (2013) DSK: *k*-mer counting with very low memory usage. *Bioinformatics*, **29**, 652–653.

Salzberg,S.L. *et al*. (2014) DIAMUND: Direct comparison of genomes to detect mutations. *Hum. Mutat*., **35**, 283–288.

Zook,J.M. *et al*. (2016) Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, **3**, Article no. 160025.