BIOINFORMATICS APPLICATIONS NOTE Vol. 29 no. 5 2013, pages 652–653
doi:10.1093/bioinformatics/btt020

Sequence analysis Advance Access publication January 16, 2013

DSK: *k*-mer counting with very low memory usage

Guillaume Rizk[1], Dominique Lavenier[2] and Rayan Chikhi[2,*]

[1]Algorizk, 75013 Paris and [2]ENS Cachan Brittany/IRISA, Campus de Beaulieu, 35700 Rennes, France

Associate Editor: Michael Brudno

ABSTRACT

Summary: Counting all the *k*-mers (substrings of length *k*) in DNA/RNA sequencing reads is the preliminary step of many bioinformatics applications. However, state of the art *k*-mer counting methods require that a large data structure resides in memory. Such structure typically grows with the number of distinct *k*-mers to count. We present a new streaming algorithm for *k*-mer counting, called DSK (disk streaming of k-mers), which only requires a fixed user-defined amount of memory and disk space. This approach realizes a memory, time and disk trade-off. The multi-set of all *k*-mers present in the reads is partitioned, and partitions are saved to disk. Then, each partition is separately loaded in memory in a temporary hash table. The *k*-mer counts are returned by traversing each hash table. Low-abundance *k*-mers are optionally filtered. DSK is the first approach that is able to count all the 27-mers of a human genome dataset using only 4.0 GB of memory and moderate disk space (160 GB), in 17.9 h. DSK can replace a popular *k*-mer counting software (Jellyfish) on small-memory servers.

Availability: http://minia.genouest.org/dsk

Contact: rayan.chikhi@ens-cachan.org

Received on October 19, 2012; revised on December 26, 2012; accepted on January 9, 2013

1 INTRODUCTION

Determining the abundance of each distinct *k*-mer in a set of sequencing reads is a conceptually simple yet fundamental task. It is used in many bioinformatics applications related to sequencing, e.g. genome and transcriptome assembly, variants detection and read error correction. For *de novo* assembly, one is often interested in counting *k*-mers to discard those with low abundance, which likely stem from sequencing errors.

State of the art methods for *k*-mer counting rely on hash tables (Jellyfish; Marçais and Kingsford, 2011) and/or Bloom filters (BFCounter; Melsted and Pritchard, 2011). These structures need to reside in memory for random access. Sequencing errors induce erroneous *k*-mers, in a volume typically greater or comparable with that of correct *k*-mers. Hence, counting *k*-mers for a human dataset with either a single hash table or a Bloom filter is a task that requires tens of gigabytes of memory. In Section 2, we describe a fixed-memory and fixed-disk space streaming algorithm, DSK (disk streaming of k-mers), and its worst-case complexity is analysed in function of the desired memory and disk usage. In Section 3, DSK is used to count all the 27-mers of a whole-genome human dataset. The trade-off between memory and disk space is analysed on two smaller datasets. We conclude with a discussion of the advantages of DSK over related methods.

Algorithm 1. The DSK algorithm

1: **Input:** The set $\mathcal{S}$ of sequences, *k*-mer length k, target memory usage M (bits), target disk space D (bits) and hash function $h(\cdot)$
2: $v \leftarrow \sum_{s \in S} (|s| - k + 1)$ (Number of *k*-mers)
3: $n_{\text{iters}} \leftarrow \left\lceil v \cdot 2^{\lceil \log_2(2k) \rceil} / D \right\rceil$ (Number of iterations)
4: $n_p \leftarrow \left\lceil \frac{v(2^{\lceil \log_2(2k) \rceil} + 32)}{0.7 n_{\text{iters}} M} \right\rceil$ (Number of partitions)
5: **for** each iteration $i = 0..n_{\text{iters}}$ **do**
6: Initialize a set of empty lists $\{d_0, ..., d_{n_p}\}$ stored on disk
7: **for** each sequence s in S **do**
8: **for** each *k*-mer m in s **do**
9: **if** $(h(m) \bmod n_{\text{iters}}) = i$ **then**
10: $j \leftarrow h(m)/n_{\text{iters}} \bmod n_p$
11: Write m to disk in d_j
12: **for** each index $j = 0..n_p$ **do**
13: Initialize a hash table T with M bits of memory
14: **for** each *k*-mer m in d_j **do**
15: $T[m] \leftarrow \begin{cases} T[m] + 1, & \text{if } m \text{ is present in } T \\ 1, & \text{otherwise} \end{cases}$
16: **output** $(m, T[m])$ for each m in T
17: Delete T
18: Delete $\{d_0, ..., d_{n_p}\}$

2 METHODS

Algorithm 1 describes the DSK *k*-mer counting algorithm. The hash function $h(\cdot)$ maps a *k*-mer to a numeric value in $[0; H]$, where H is a large integer (typically 2^{64}). In the following analysis, we make a simplifying assumption. Let d be the total number of distinct *k*-mers in the input; we assume that the number of distinct *k*-mers having a given hash value $x \in [0; H]$ is at most $\lceil d/H \rceil$. In other words, the set of distinct *k*-mer values can be uniformly partitioned by this hash function. Each *k*-mer is encoded using the classical 2 bits representation in the smallest available integer type, i.e. using $2^{\lceil \log_2(2k) \rceil}$ bits. The abundance of each *k*-mer is stored as a 32 bits integer. For convenience, let $b = 2^{\lceil \log_2(2k) \rceil}$.

Each *k*-mer m present in S is examined $n_{\text{iters}} = \lceil vb/D \rceil$ times (once per iteration) and is written to disk only once, at the $(h(m) \bmod n_{\text{iters}})$-th iteration. Using the uniform repartition hypothesis, a multi-set of $v/n_{\text{iters}} \leq \lceil D/b \rceil$ *k*-mers are written to disk at each iteration. As each *k*-mer is encoded using b bits, the maximal disk usage of the algorithm is D bits.

The maximal memory usage of the algorithm is M bits, as Steps 7–11 require constant memory, and Steps 12–17 load a single partition in T that requires exactly M bits. With an open-addressing mechanism, each distinct *k*-mer occupies exactly $(b + 32)$ bits in T. To prove that the algorithm terminates, it suffices to show that T never overflows, i.e. strictly

*To whom correspondence should be addressed.

© The Author 2013. Published by Oxford University Press. All rights reserved. For Permissions, please e-mail: journals.permissions@oup.com

Downloaded from http://bioinformatics.oxfordjournals.org/ at Georgia Institute of Technology on May 27, 2014

Table 1. Wall-clock time and memory usage for counting 27-mers in whole-genome human data

Program	Time (h)	Memory (GB)	Disk (GB)
DSK	17.9	4	160
DSK-SSD[a]	3.5	4	240
BFCounter	41.2	56	0
Jellyfish	3.5	70	211

The dataset used is the NA18507 human genome (SRX016231), unfiltered, consisting of 1.4 billion reads of average length 100 bp (160 GB file size). Jellyfish used eight threads, DSK-SSD used four threads and DSK and BFCounter are single-threaded. The disk column indicates the temporary amount of disk space used by each method.

[a]Executed on a desktop computer equipped with two hard drives, including an SSD.

less than $M/(b+32)$ distinct k-mers are inserted in T. At each iteration (v/n_{iters}), k-mers are split into n_p partitions. Each partition contains at most $v/(n_{\text{iters}}n_p) \leq \lceil 0.7M/(b+32) \rceil$ k-mers. In the worst case, all these k-mers are distinct; thus, the load factor is upper-bounded by 0.7 (a classical threshold above which hash table performance degrades).

The time complexity of Steps 7–11 (including the iteration loop) is $O(v^2b/D)$. The algorithm creates $(n_{\text{iters}}n_p) \leq \lceil v(b+32)/(0.7M) \rceil$ temporary hash tables, inserting at most $\lceil (0.7M/(b+32) \rceil$ elements in each. Hash tables accesses and insertions (Step 15) are done in constant expected time with open-addressing, as long as the load factor is strictly <1 (which was proved earlier in the text). Hence, the expected time complexity of Steps 12–17 (including the iteration loop) is $O(v)$. Thus, Algorithm 1 runs in expected time $O(v^2b/D)$. The algorithm runs in expected linear time with respect to v when $D = \Theta(v)$, e.g. setting D equal to the sum of input bases. In practice, the simplifying assumption on the uniform repartition of the hash function h does not hold exactly. Some partitions contain a slightly larger number of distinct k-mers than $\lceil v/H \rceil$. Hence, the actual disk usage of the algorithm is slightly above D, and the load factor of T could, in theory, be >0.7 (because of high k-mer redundancy, this is not the case in practice).

3 RESULTS

In Table 1, we compared the execution time and memory usage of DSK with Jellyfish (version 1.1.5) and BFCounter (version 0.2) on a human genome Illumina dataset. The target disk usage of DSK was set to 160 GB, equal to the size of the reads file. As the algorithm relies heavily on I/O to the disk, we also tested DSK with a solid-state drive (DSK-SSD). The reads file was placed on a standard hard disk drive, and partitions of redundant k-mers were written on a 256 GB SSD. In this configuration, we noticed the algorithm is no longer limited by disk I/O and could benefit from multi-threading. The two for loops lines 7 and 12 were parallelized using openMP (four threads). DSK-SSD ran for 3.5 h using 4×1 GB of memory. Although this experiment required specific hardware, it is worth noting that the running time of DSK can be greatly reduced with an SSD and multi-core parallelism.

To further assess the trade-off between time, memory and disk usage, we executed DSK (using a standard hard drive) on two smaller *Escherichia coli* and *Drosophila ananassae* datasets, with various target memory and disk usage parameters (Figure 1). For the executions with 100 MB and 1 GB memory usage, the running time of DSK on both datasets decreases as the target disk

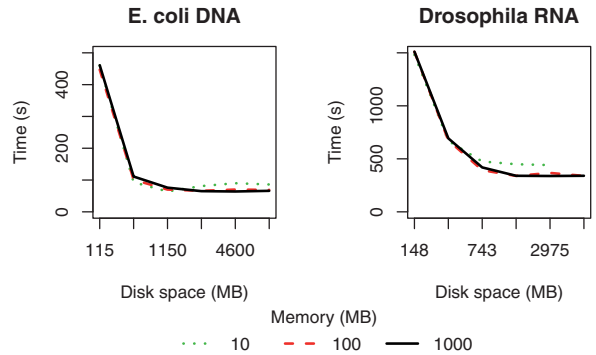

Fig. 1. Execution time of DSK ($k = 21$) as a function of memory and disk usage, on the *E.coli* (Illumina DNA SRR001665, $20.8 \cdot 10^6$ reads of average length 36 bp) and *D.ananassae* datasets (Illumina RNA-Seq SRR332538, $9.1 \cdot 10^6$ reads of average length 150 bp)

space increases. This is a consequence of the decreasing number of iterations n_{iters}. The running times reach a plateau at roughly the reads file size (where $n_{\text{iters}} = 1$). The execution time generally seems to be unaffected by the target memory usage. However, at the smallest tested memory usage (10 MB), the execution time on both datasets is slightly higher, possibly because of consecutive disk writes to a large number of partitions. Note that in practice, the memory usage of DSK cannot be arbitrarily low: it is limited by the number of files that can be simultaneously opened on the system (partitions $\{d_0, \ldots, d_{n_p}\}$ are all opened simultaneously).

4 DISCUSSION

Contrary to other methods, DSK does not provide random access to k-mer counts. However, it benefits from three strong points:

- **Low-memory usage:** Only an arbitrarily small subset of k-mers is loaded in memory at any time. In contrast, BFCounter stores all the k-mers with count ≥ 2 in a hash table. In principle, Jellyfish can use arbitrarily small hash tables; however, storing the intermediate results requires a prohibitive amount of disk (≥ 1 TB for human genome reads using a hash table of size 5 GB).
- **Parameters are automatically inferred:** The only mandatory argument is the k-mer length. Optionally, target memory and disk usages can be specified. Jellyfish and BFCounter require the user to specify a hash table size and an upper-bound on the number of distinct k-mers, respectively.
- **Supports arbitrarily large values of k:** As opposed to up to 32 for Jellyfish (unbounded for BFCounter).

Funding: ANR MAPPI, ANR-10-COSI-0004.

Conflict of Interest: none declared.

REFERENCES

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

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

Downloaded from http://bioinformatics.oxfordjournals.org/ at Georgia Institute of Technology on May 27, 2014