Bioinformatics, 31(19), 2015, 3216–3218
doi: 10.1093/bioinformatics/btv332
Advance Access Publication Date: 28 May 2015
Applications Note

OXFORD

Sequence analysis

ACE: accurate correction of errors using *K*-mer tries

Siavash Sheikhizadeh* and Dick de Ridder

Bioinformatics Group, Wageningen University, 6700 AP Wageningen, The Netherlands

*To whom correspondence should be addressed.

Associate Editor: Inanc Birol

Received on November 23, 2014; revised on May 2, 2015; accepted on May 22, 2015

Abstract

Summary: The quality of high-throughput next-generation sequencing data significantly influences the performance and memory consumption of assembly and mapping algorithms. The most ubiquitous platform, Illumina, mainly suffers from substitution errors. We have developed a tool, ACE, based on K-mer tries to correct such errors. On real MiSeq and HiSeq Illumina archives, ACE yields higher gains in terms of coverage depth, outperforming state-of-the-art competitors in the majority of cases.

Availability and implementation: ACE is licensed under the GPL license and can be freely obtained at https://github.com/sheikhizadeh/ACE/. The program is implemented in C++ and runs on most Unix-derived operating systems.

Contact: siavash.sheikhizadehanari@wur.nl

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Genome sequencing involves reading thousands or millions of genome fragments and reconstructing the original genome, either by assembling these reads in *de novo* assembly projects, or aligning them to a known reference genome in re-sequencing studies. Over the last decade, next-generation sequencing (NGS) technology dramatically increased the ease with which material can be sequenced, yielding millions of short reads in a short time. The lower quality of the data (compared to Sanger sequencing) however significantly influences performance and memory consumption of assemblers and alignment algorithms; as a result, there has been a growing interest in correcting errors in short-read archives. Sequencing errors can result in substitutions, insertions, deletions and unconfirmed nucleotides represented by '*N*' symbols. The most ubiquitous platform, Illumina, mostly suffers from substitution errors while for others, like 454 and Ion Torrent, insertions and deletions are most abundant.

As an error at a specific genomic position occurs infrequently and randomly, an erroneous base can be detected and corrected taking advantage of the high frequency of the reads that cover that position. This is the idea behind all count-based error correction methods which count *K*-mers using various data structures. For example, SHREC (Schroder *et al.*, 2009) constructs a generalized suffix trie while HiTEC (Ilie *et al.*, 2011) uses a suffix array. Built upon SHREC, Hybrid-SHREC (Salmela, 2010) captures InDel errors as well as substitutions. SGA (Simpson and Durbin, 2012) performs error-correction using the FM-index derived from the compressed Burrows-Wheeler transform. BLESS (Heo *et al.*, 2014) employs a bloom-filter and RACER (Ilie and Molnar, 2013) organizes 2-bit-encoded *K*-mers as 64-bit integers and stores them in a hash table. Fiona, based on partial suffix array, is also able to deal with InDel errors (Schulz *et al.*, 2013). Alternatively, *K*-spectrum based error correction methods, like Quake (Kelley *et al.*, 2010) and Musket (Liu *et al.*, 2013) collect all *K*-mers appearing in the set of reads, and align those with a small Hamming distance from each other to achieve the correct consensus. Finally, MSA-based methods, like Coral (Salmela and Schroder, 2011), apply multiple sequence alignment between reads that share *K*-mers to detect errors. A recent survey provides a comprehensive review of error-correction methods, and establishes a common set of benchmark data and evaluation criteria (Molnar and Ilie, 2014).

Here we present ACE, a new *K*-mer count-based algorithm. We employ the *K*-mer trie, a data structure more time/space-efficient than the suffix trees employed in SHREC. *K*-mer tries have been

effective in solving some bioinformatics problems (Brudno *et al.*, 2003; Sheikhizadeh and Hosseini, 2014).

2 Methods

ACE is the C++ implementation of our algorithm, equipped with Open-MP directives to scale with the number of available processors. It organizes the *K*-mers of short reads (and their reverse-complement) in a *K*-mer trie.

A *K*-mer trie of a sequence *s* (or set of sequences) is a trie of depth *K* which contains all *K*-mers of the sequence. Each edge has a label from the alphabet Σ; the concatenation of edge labels along the path from root to a node is called the *spelled string* of that node. Each leaf corresponds to one or more *K*-mers of *s*, and each node can contain the number of times that its spelled string appears in the sequence. A *K*-mer trie gives constant-time access to all patterns of length at most *K*, useful for counting and storing *K*-mers, detecting zygosity, determining ploidy and genome assembly.

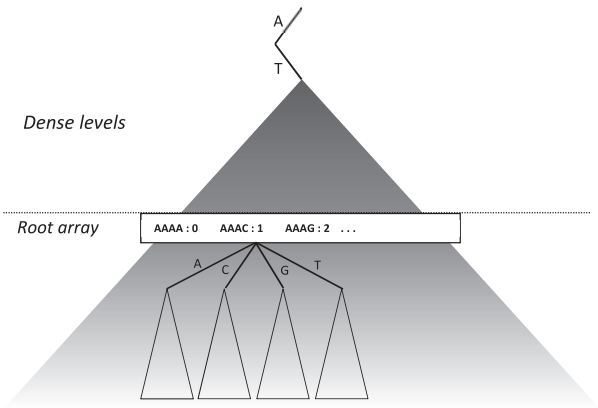

Fig. 1. The AT subtrie of the *K*-mer trie

To cope with large datasets, ACE constructs *K*-mer sub-tries one by one (Fig. 1) by applying a prefix-based classification on *K*-mers and shortening them to *k*-mers where $k = K - p$ and *p* is the prefix length, determined based on the amount of available memory and the size of the input data. As the *K*-mer trie is very dense in the top levels, ACE further reduces memory consumption (and the size of search space) by building a root array instead of constructing the top triangle of the trie. Moreover, to efficiently organize billions of *K*-mers in the trie ACE applies another prefix-based division to allow parallel construction of the branches of each subtrie. In Figure 1, a branch has been divided into four sub-branches to be constructed and scanned for errors by four independent parallel threads. Branches are divided into 16, 64 or more sub-branches if more cores are available. More details on the algorithm, including a pseudo-code description, can be found in the Supplementary Material.

3 Results

We experimentally compared the performance of ACE in increasing the coverage depth/breadth of reads/*K*-mers to those of seven state-of-the-art tools, using the benchmark data and following the same evaluation procedure as presented in a recent survey (Molnar and Ilie, 2014). To be consistent with the result of Molnar *et al.*, we chose the same value of $K = 20$ for evaluations. The specifications of the MiSeq (M1–M9) and the HiSeq datasets (H1-H13) are presented in Supplementary Table 1. All experiments were conducted on a Linux server (SUSE 3.8.6-2) with an Intel® Xeon® ES-2667 CPU, exploiting 16 logical cores running at 2.9 GHz and 256 GB RAM.

Table 1. The gain of ACE in increasing the depth/breadth of reads/*K*-mers, compared to that of the best tool in Molnar and Ilie (2014)

	Depth Gain						Breadth Gain					
	Read			K-mer			Read			K-mer		
		Best	ACE		Best	ACE		Best	ACE		Best	ACE
M1	*SGA*	26.30	56.10	*RACER*	37.42	58.90	*BLESS*	6.75	9.13	*SGA*	0.00	−202.26
M2	*RACER*	58.06	57.05	*RACER*	49.48	51.73	*RACER*	6.31	6.23	*Coral*	0.01	−11.48
M3	*RACER*	13.09	14.06	*RACER*	19.05	23.01	*RACER*	1.94	2.08	*Coral*	−0.01	−3.92
M4	*RACER*	74.96	83.91	*RACER*	58.72	73.90	*RACER*	7.73	8.60	*Coral*	0.00	−0.94
M5	*RACER*	0.88	0.89	*RACER*	4.00	4.39	*RACER*	0.098	0.096	*SGA*	−0.95	−13.53
M6	*RACER*	16.46	18.03	*RACER*	21.09	26.01	*RACER*	3.47	3.75	*Coral*	0.00	−5.35
M7	*RACER*	86.11	90.26	*RACER*	65.59	79.48	*RACER*	28.41	29.80	*Coral*	0.00	−2.56
M8	*HiTEC*	37.78	73.18	*HiTEC*	47.65	69.47	*BLESS*	67.96	79.83	*HiTEC*	5.00	−5.00
M9	*RACER*	0.39	0.45	*HiTEC*	6.71	7.90	*BLESS*	0.97	0.49	*Coral*	0.00	−45.94
H1	*BLESS*	61.72	63.55	*Musket*	51.28	67.67	*BLESS*	5.66	5.43	*Coral*	−1.74	−7.49
H2	*BLESS*	44.30	44.93	*HiTEC*	33.03	34.86	*BLESS*	4.54	4.56	*SGA*	−1.73	−1.96
H3	*RACER*	34.13	34.69	*RACER*	19.26	22.15	*BLESS*	2.28	2.25	*SGA*	−0.88	−1.20
H4	*RACER*	92.27	93.71	*HiTEC*	85.83	88.77	*BLESS*	49.49	49.67	*Coral*	−0.36	−1.72
H5	*BLESS*	13.70	13.80	*HiTEC*	9.27	9.48	*BLESS*	4.63	4.58	*SGA*	−9.83	−12.55
H6	*BLESS*	86.16	92.35	*HiTEC*	82.83	90.23	*BLESS*	82.87	85.46	*Coral*	0.00	−8.12
H7	*SGA*	52.89	54.26	*RACER*	47.35	50.35	*BLESS*	4.69	4.70	*SGA*	−1.99	−24.40
H8	*RACER*	26.77	27.83	*RACER*	11.92	13.74	*BLESS*	4.06	4.10	*SGA*	−2.84	−280.70
H9	*BLESS*	18.14	20.35	*RACER*	16.19	19.70	*BLESS*	4.50	4.62	*SGA*	−5.70	−27.11
H10	*BLESS*	26.13	27.53	*Musket*	19.64	21.67	*BLESS*	7.98	8.03	*SGA*	−6.49	−31.77
H11	*SGA*	61.66	57.52	*RACER*	38.11	46.19	*SGA*	12.92	12.03	*SGA*	−4.89	−11.59
H12	*SGA*	65.46	62.61	*RACER*	42.42	48.73	*SGA*	16.15	14.95	*SGA*	−5.63	−13.81
H13	*SGA*	27.86	27.70	*RACER*	35.49	40.35	*SGA*	3.18	3.01	*SGA*	−4.32	−11.29

Note: Highlights indicate the level of improvement.

Table 2. Time and memory requirements of tools able to successfully correct all datasets

	Time (s/Mb)				Space (MB/Mb)			
	Musket	RACER	SGA	ACE	Musket	RACER	SGA	ACE
M1	0.05	0.55	2.07	10.22	3.64	17.80	10.02	30.92
M2	0.14	0.18	0.85	3.14	4.31	10.00	5.40	5.30
M3	0.07	0.27	1.12	3.76	3.47	7.20	4.52	6.21
M4	0.07	0.35	1.16	3.92	3.18	8.15	3.95	7.01
M5	0.07	0.25	0.56	3.48	2.95	6.24	4.02	5.52
M6	0.07	0.34	1.25	3.90	2.35	6.78	3.10	5.70
M7	0.06	0.23	1.05	3.29	1.53	3.18	2.03	2.73
M8	0.06	0.43	1.82	8.29	0.26	2.24	0.90	12.86
M9	0.06	0.33	1.83	5.64	0.24	1.21	0.75	4.94
Average	0.07	0.32	1.30	5.07	2.44	6.98	3.86	9.02
H1	0.27	0.17	0.87	4.28	4.49	9.40	5.99	5.16
H2	0.23	0.22	0.82	5.78	4.28	9.68	6.10	4.23
H3	0.20	0.16	0.75	5.21	2.93	7.59	4.28	5.12
H4	0.28	0.31	1.20	5.88	1.58	4.03	2.37	5.67
H5	0.20	0.24	0.92	4.77	1.17	3.01	1.73	2.45
H6	0.48	0.25	1.08	3.62	0.66	2.04	1.09	4.48
H7	0.29	0.30	1.11	4.97	0.70	2.83	0.85	6.70
H8	0.27	0.31	1.09	4.56	0.52	2.36	0.56	4.83
H9	0.42	0.32	1.14	4.29	0.61	2.70	0.55	4.75
H10	0.32	0.31	1.22	4.17	0.62	2.26	0.46	4.64
H11	0.24	0.74	1.55	3.93	0.20	1.66	0.23	1.39
H12	0.28	0.47	1.69	3.92	0.19	2.28	0.23	1.36
H13	0.26	0.47	1.62	3.92	0.16	1.37	0.22	1.32
Average	0.29	0.33	1.16	4.56	1.39	3.94	1.90	4.01

Note: Time is given in seconds and memory in megabytes, both per input mega base pair.

Table 1 compares the gain of ACE to that of its best competitor among seven state-of-the-art read cleaners: BLESS, Coral, HiTEC, Musket, RACER, SGA and SHREC; more detailed results can be found in Supplementary Tables S3–S11. In these evaluations, *Depth of coverage* indicates the average number of times each base is covered by reads/K-mers and *Breadth of coverage* indicates the proportion of the genome covered by reads/K-mers (Molnar and Ilie, 2014). The first criterion is useful for quantitative applications and overlap-layout-consensus assembly, while the second is more applicable for de Bruijn graph-based genome assembly.

ACE outperforms most other tools in terms of coverage depth gain, improving on the best competitor on 18 resp. 22 out of 22 datasets for reads resp. *K*-mers. In particular for MiSeq data, which contains more errors, the improvements can be significant. For coverage breadth, the picture is less clear: ACE outperforms the best alternative tool on 13 datasets on read coverage breadth gain, whereas *K*-mer coverage breadth gain was generally worse. However, all tools actually yield low read coverage breadth on most MiSeq data (as low as 0.25%) and decrease *K*-mer coverage breadth compared to the raw data (see Supplementary Tables S5–S6, S9–S10).

Table 2 compares the time and memory consumption of ACE to those of the three competitors which were able to successfully correct all datasets. While for most datasets memory consumption is reasonable, ACE has higher computational cost than most other tools, trading speed for accuracy.

4 Conclusion

We developed ACE, a command-line tool to accurately correct substitution errors in Illumina short-read archives. ACE generally outperforms the best among seven state-of-the-art read cleaners in terms of coverage depth, at higher computational cost. This makes it a useful tool for small to medium-sized datasets or applications where accuracy requirements warrant the investment in computational resources.

In future work, we aim to lower the runtime of ACE by updating the K-mer trie instead of rebuilding it for each round of execution. This future version should also be able to handle InDel errors to extend its application to all sequencing platforms.

Conflict of Interest: none declared.

References

Brudno,M. *et al.* (2003) LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.*, **13**, 721–731.

Heo,Y. *et al.* (2014) BLESS: bloom-filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, **30**, 1354–1362.

Ilie,L. *et al.* (2011) HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*, **27**, 295–302.

Ilie,L. and Molnar,M. (2013) RACER: rapid and accurate correction of errors in reads. *Bioinformatics*, **29**, 2490–2493.

Kelley,D.R. *et al.* (2010) Quake: quality-aware detection and correction of sequencing errors. *Genome Biol.*, **11**, R116.

Liu,Y. *et al.* (2013) Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, **29**, 308–315.

Molnar,M. and Ilie,L. (2014) Correcting Illumina data. *Briefings in Bioinformatics*, adv. online publication.

Salmela,L. (2010) Correction of sequencing errors in a mixed set of reads. *Bioinfomatics*, **26**, 1284–1290.

Salmela,L. and Schroder,J. (2011) Correcting errors in short reads by multiple alignments. *Bioinformatics*, **27**, 1455–1461.

Schroder,J. *et al.* (2009) SHREC: a short-read error correction method. *Bioinformatics*, **25**, 2157–2163.

Schulz,M.H. *et al.* (2013) Fiona: a parallel and automatic strategy for read error correction. *Bioinformatics*, **30**, 356–363.

Sheikhizadeh,S. and Hosseini,S. (2014) SMOTER: a structured motif finder based on an exhaustive tree-based algorithm. *Current Bioinformatics*, **9**, 34–43.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. Genome Res., **22**, 549–556.