Sparc: a sparsity-based consensus algorithm for long erroneous sequencing reads

Chengxi Ye, Sam Ma

Motivation: The third generation sequencing (3GS) technology generates long sequences of thousands of bases. However, its error rates are estimated in the range of 15-40%, much higher than the previous generation (approximately 1%). Fundamental tasks such as genome assembly and variant calling require us to obtain high quality sequences from these long erroneous sequences. **Results:** In this paper we describe a versatile and efficient linear complexity consensus algorithm *Sparc* that builds a sparse *k*-mer graph using a collection of sequences from the same genomic region. The heaviest path approximates the most likely genome sequence (consensus) and is sought through a sparsity-induced reweighted graph. Experiments show that our algorithm can efficiently provide high-quality consensus sequences with error rate <0.5% using both PacBio and Oxford Nanopore sequencing technologies. Compared with the existing approaches, *Sparc* calculates the consensus with higher accuracy, uses 80% less memory, and is 5x faster, approximately. **Availability:** The source code is available for download at http://sourceforge.net/p/sparc-consensus/code/ and a testing dataset is available: https://www.dropbox.com/sh/trng8vdaeqywx1e/AAASJesLVAJZcbORkU9f4LuBa?dl=0 (Please copy the link to a browser to access if directly clicking the link fails)

Sparc: a sparsity-based consensus algorithm for long erroneous sequencing reads

Chengxi Ye[1,*], Zhanshan (Sam) Ma[2,*]
[1]Department of Computer Science, University of Maryland, College Park, MD 20740, USA.
[2]Computational Biology and Medical Ecology Lab, State Key Laboratory of Genetic Resources and Evolution, Kunming Institute of Zoology, Chinese Academy of Sciences, Kunming, Yunnan 650223, China.

ABSTRACT

Motivation: The third generation sequencing (3GS) technology generates long sequences of thousands of bases. However, its error rates are estimated in the range of 15-40%, much higher than the previous generation (approximately 1%). Fundamental tasks such as genome assembly and variant calling require us to obtain high quality sequences from these long erroneous sequences.

Results: In this paper we describe a versatile and efficient linear complexity consensus algorithm *Sparc* that builds a sparse *k*-mer graph using a collection of sequences from the same genomic region. The heaviest path approximates the most likely genome sequence (consensus) and is sought through a sparsity-induced reweighted graph. Experiments show that our algorithm can efficiently provide high-quality consensus sequences with error rate <0.5% using both PacBio and Oxford Nanopore sequencing technologies. Compared with the existing approaches, *Sparc* calculates the consensus with higher accuracy, uses 80% less memory, and is 5x faster, approximately.

Availability: The source code is available for download at http://sourceforge.net/p/sparc-consensus/code/ and a testing dataset is available: https://www.dropbox.com/sh/trng8vdaeqywx1e/AAASJesLVAJZcbORkU9f4LuBa?dl=0 (Please copy the link to a browser to access if directly clicking the link fails)

1 INTRODUCTION

Three generations of DNA sequencing technologies have been developed in the last three decades, and we are at the crossroads of the second and third generation of the sequencing technologies. Compared with the previous generations, the third generation sequencing (3GS) can provide reads from 5-120 kilo-bases in one fragment. However, at present, the reported error rates are in the range of 15%-40%; this poses arguably the most significant challenge for assembling genome with the 3GS reads.

Genome assembly is the computational problem of producing longer and high quality fragments, known as *contigs*, by analyzing the overlap relations between the reads (Myers, et al., 2000; Pevzner, et al., 2001). With the 3GS data, genome assembly needs to pass through three major bottlenecks: finding overlaps (Berlin, et al., 2014; Ye, et al., 2014), sequence alignment (Chaisson and Tesler, 2012; Myers, 2014) and sequence polishing/error correction. Efficiently correcting these long erroneous reads is a non-trivial problem (Au, et al., 2012; Hackl, et al., 2014; Koren, et al., 2012; Salmela and Rivals, 2014). Given a target genomic region, fast algorithms are required to collect the query sequences that can be aligned to the target region, and an accurate aligner is necessary to exploit the layout relations of the query sequences. Finally, another polishing/consensus algorithm takes the layout information to infer the 'ground truth' sequence.

The consensus algorithm is critical for genome assembly in multiple ways. First, consensus algorithm is part of the assembler and necessary to produce high quality outputs. Second, the consensus algorithm can also be used to provide higher quality inputs to the off-the-shelf assemblers for the first and next generation sequences (Huang, et al., 2003; Mullikin and Ning, 2003; Myers, et al., 2000), which were designed for highly accurate sequences (often requiring the sequence accuracy >95%). Third, recent assembly advancements resort to an error correction procedure (Au, et al., 2012; Hackl, et al., 2014; Koren, et al., 2012; Salmela and Rivals, 2014) to raise the per-base accuracy in the input sequences. The consensus algorithm can also be used to correct each individual read, and those corrected reads become high quality inputs, and are fed into existing *Overlap-Layout-Consensus* based assemblers. In this third scenario, each long erroneous read is treated as the target; sequences from either NGS or 3GS may be used as the query sequences. Nevertheless, due to the lack of efficient consensus algorithm, this step is usually circumvented by simpler approaches such as replacing regions in the target sequences with the NGS reads or assemblies. Unfortunately, errors in the NGS sequences may corrupt the originally correct 3GS sequences and create unwanted consensus errors in the final assembly. Fourth, it is also noteworthy that the polishing step in genome assembly pipelines often takes the largest portion of the computational time if the consensus is inefficient (Berlin, et al., 2014; Chin, et al., 2013; Lee, et al., 2014). Therefore an efficient consensus algorithm can significantly accelerate the whole genome assembly process.

Most existing consensus algorithms were designed for sequences with error rates lower than 5% (Huang, et al., 2003; Mullikin and Ning, 2003; Rausch, et al., 2009). Traditional multiple sequence alignment, known to be a computationally challenging task is used to find the layout and construct a sequence alignment graph (Edgar, 2004; Larkin, et al., 2007; Lee, et al., 2002; Rausch, et al., 2009). Alignments are refined and clustered to infer the alignment profile as the consensus in the target region. The higher error rates lead to much higher complexities with these traditional approaches. To lower the complexity, researchers have tried to simplify the multiple sequence alignment by aligning all query sequences to a backbone sequence and creating a multigraph representing the alignment graph (Chin, et al., 2013). Graph simplifications are applied to merge the multiple edges and the best scored path is found as the consensus sequence. Each graph node in this work is a nucleotide base. For NGS data, a similar strategy using the *de Bruijn* graph has been developed (Ronen, et al., 2012) to correct the assembly errors in single cell sequence assembly.

In this work, we borrow wisdom from the well-known *de Bruijn*/*k*-mer graph (Hannenhalli, et al., 1996; Pevzner, et al., 2001; Ronen, et al., 2012) and develop a simpler graph formulation of the consensus problem. Improvements upon the *k*-mer graphs

*To whom correspondence should be addressed.

could potentially be used to facilitate the construction of the new graph. Based on this we provide a general and versatile ‘Sparc’ algorithm to polish long erroneous reads. We build a regular graph (*i.e.*, non-multigraph) directly from the sequences and search for the consensus from a sparsity-induced reweighted graph. Each node in our regular graph is a *k*-mer. The graph is allowed to be ‘sparse’ (Ye, et al., 2012) to avoid using excessive memory to store false *k*-mers. The links/edges between the *k*-mers are found by analyzing each read. Edge weight represents the confidence in the link. Intuitively, a path with the highest sum of edge weights is a good approximation of the consensus. We explain this in detail in the next section. We show that the proposed algorithm can provide superb results without utilizing any graph simplification techniques. Sparc also supports hybrid data and provides high quality (error rate <0.5%) results with Oxford Nanopore sequencing reads. Due to the simplicity, the algorithm is five times faster and uses five times less memory space compared to existing programs such as PBdagcon (Chin, et al., 2013).

2 METHODS

Sparc consists of the following four simple steps (Fig. 1): (*i*) Build an initial position specific (sparse) *k*-mer graph (Ye, et al., 2012) using the draft assembly/backbone sequence. (*ii*) Align sequences to the backbone to continue the construction of the graph. (*iii*) Adjust the edge weights using a sparse penalty. (*iv*) Search for a heaviest path and output the consensus sequence.

2.1 Building the initial graph

We use *k*-mers to encode the local structure of the genomic region. Sparc takes a preassembled draft assembly/backbone to build an initial *k*-mer graph (Fig. 1a). The *k*-mers in *different* positions of the backbone are treated as *independent* nodes. Note that this is the major difference between the position specific *k*-mer graph and the prevalent *de Bruijn* graphs in genome assembly, in which the same *k*-mers in different locations are collapsed. In the consensus context, *k*-mers located at different positions are treated independently. It is noteworthy that allocating *k*-mers in each location can take a large amount of memory, especially in the next stage. To circumvent this problem, we construct a sparse *k*-mer graph (Ye, et al., 2012) by storing a *k*-mer in every *g* bases, which reduces the memory consumption up to 1/*g*. We also record the edge links between the *k*-mer nodes. The edge weight represents the confidence in the corresponding path. We use coverage to represent confidence for simplicity. A generalization using quality score is straightforward.

2.2 Aligning sequences to the backbone and building the whole graph

Sequences that align to the backbone sequence provide rich information about the ground truth sequence. Ideally, we should search for a most likely genome sequence as the consensus given all the input sequences. However, utilizing the multi-sequence information comprehensively requires computationally expensive operations such as pair-wise alignment of all the related sequences (Edgar, 2004; Larkin, et al., 2007; Lee, et al., 2002; Rausch, et al., 2009). Here we adopt a simpler strategy as in PBdagcon (Chin, et al., 2013), by aligning all the sequences to the backbone, and modify the existing graph according to the alignments. Rather than creating an intermediate graph that needs to be refined or simplified (Chin, et al., 2013; Rausch, et al., 2009), we construct a graph that can be used directly. We borrow the wisdom from constructing a *de Bruijn*/*k*-mer graph (Pevzner, et al., 2001; Ye, et al., 2012): (1) if a query region suggests a novel path/variant, we create a branch and allocate new *k*-mer nodes and edges that link these nodes (Fig. 1b, upper half). (2) If a query region perfectly aligns to an existing region in the graph, we merge all the nodes and edges in the region and increase the edges weights (Fig. 1b, bottom half). As previously mentioned, this construction process shares similarity with constructing a *de Bruijn* graph, but the nodes in our graph are “position specific”. In addition, if a sequence (such as from NGS sequencing, in a hybrid assembly setting) is of higher quality, we assign higher weights to the corresponding edges. This position specific *k*-mer graph contains rich information about the underlying genomic region. Next we describe another simple technique to extract the most likely sequence as the consensus.

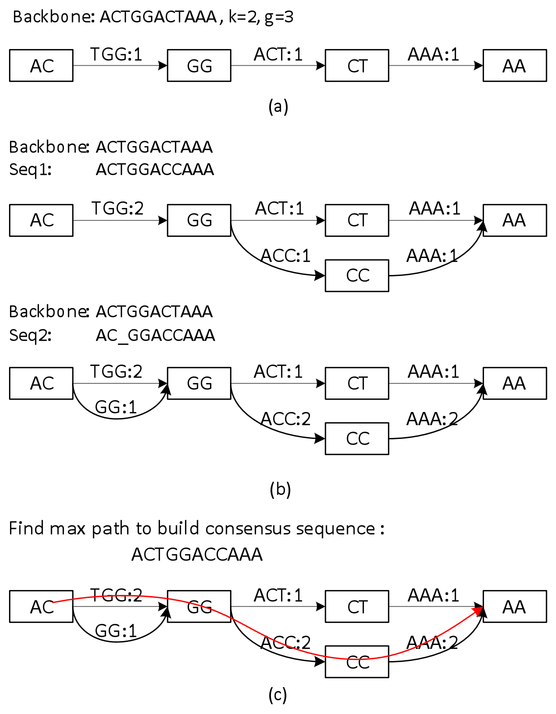

Fig. 1. A toy example of constructing the position specific sparse *k*-mer graph. (a) The initial *k*-mer graph of the backbone. (b) Adding two sequences to the graph. (c) The heaviest path representing the consensus is found by graph traversal (original weights are used in this example).

2.3 Adjusting the weights of the graph

Intuitively, a path in the *k*-mer graph is likely to be genuine if it is supported by multiple sequences. Based on this intuition, we should search for a path in the graph with the highest confidence, *i.e.* the largest sum of edge weights. However, direct implementation of this strategy may result in erroneous outputs. A simple example is a long insertion error: the sum of weights of this erroneous path is high even though there is only one supporting sequence. To circumvent this type of error, we reduce the edge weight by a small amount. This amount is adaptively determined with consideration of the sequencing coverage in each region. This technique, also known as soft thresholding (Mallat, 2008), is equivalent to put a l^1-penalty on the edge weights. With this sparse penalty, the low coverage long insertion errors will be less likely to be favored compared to the genuine sequences.

2.4 Output the heaviest path as the consensus

With the adjusted weights, we use the breadth-first search to search for a heaviest path that links the start node and the end node of the backbone. We then backtrack from the highest scored node in this path to the first node that has positive weight sum. This sub-path is output as the consensus path (Fig. 1c).

2.5 The complexity of sparc

The complexities of the above procedures described in subsections 2.1-2.4 are linear with the data size. Sparc allows for taking different *k*-mer sizes (*k*) and skip sizes (*g*). Setting *k* to a smaller value makes the consensus more sensitive and can recover weaker alignments, while setting *k* to larger values helps to avoid spurious alignments. In our experiments we found that using k = 1-3, g = 1-5 is sufficient for practical purposes. It is noteworthy due to high error rate in 3GS data, we use very small *k*-mers to have sufficient matching anchors in the *k*-mer graph, unlike in common *de Bruijn* graphs. Assembly using hybrid data can reduce the requirement of coverage of the 3GS data, which significantly reduces the sequencing cost. Since the NGS data has much higher accuracy, including the data into the consensus also improves the quality. Sparc is designed to take advantage of the hybrid sequencing data, and leverage more weight to the high quality paths. Currently we rely on Blasr to provide long read alignment information (Chaisson and Tesler, 2012). Sparc takes as input a backbone file in fasta format and the Blasr alignment results. To avoid multiple placement of a query read, each read is mapped to one best matching region in the backbone. Finally, reusing the consensus result as the input and iteratively running the consensus algorithm helps to improve the accuracy even more (shown in he Results section).

3 RESULTS

Sparc has been tested on a variety of datasets. Here we demonstrate the test results from two PacBio datasets (http://schatzlab.cshl.edu/data/ectools/) and one Oxford Nanopore dataset (http://gigadb.org/dataset/100102). Sparc is designed to be a base-level consensus algorithm, while there are platform-specific ones that take into account signal-level information such as Quiver (for PacBio), and Nanopolish (for Oxford Nanopore). These programs usually take the outputs of the base-level ones as inputs to further improve the accuracy. As a fair comparison, we demonstrate results side by side with the most similar program to ours, which is PBdagcon (Chin, et al., 2013). PBdagcon is the major module that is intensively used in HGAP (Chin, et al., 2013) and MHAP (Berlin, et al., 2014) pipelines to correct reads and generate consensus using base-level information. We therefore show the comparative results of both programs on these datasets. Both programs were fed with the same input data. We generated assembly backbones and collect the related reads for each backbone using DBG2OLC (Ye, et al., 2014). Blasr (Chaisson and Tesler, 2012) was called (with option –m 5) to obtain the alignments. The final consensus error rates were calculated using the dnadiff function in MUMmer 3 (Kurtz, et al., 2004). We conducted all our experiments on a workstation with AMD Opteron 2425 HE CPUs (@ 800MHz frequency). In our hybrid consensus experiments, we used the Illumina assembly contigs by SparseAssembler (Ye, et al., 2012) and increase the edge weights by 5.

On PacBio datasets, we set k = 1, g = 1, and ran the consensus algorithms for two rounds. The per-base error rates for the first round and the second round were reported as Err1 and Err2 in Tables 1&2, respectively. In the first experiment, we used an *E. coli* PacBio dataset and tested the accuracy using different sequencing coverages. The longest backbones generated by DBG2OLC using 10x/30x data were 1.3Mb and 4.6Mbp respectively. The *E. coli* genome reference (4.6 Mbp) can be found with accession number NC_000913. Sparc reached an error rate of 0.09% using only 10x data in a hybrid setting in contrast of 0.64% with PBdagcon. As expected, the quality is even better with 30x data (0.02%). The error rates of using PacBio (PB) data only (*i.e.* non-hybrid) were slightly higher as expected.

Table 1. Results on an *E. coli* dataset using PacBio sequencing

Program	Coverage	Time	Memory	Err1	Err2
Sparc	10x PB	0.5	308MB	1.95%	1.51%
PBdagcon	10x PB	3.0	1.10GB	1.95%	1.52%
Sparc	10x Hybrid	0.5	237MB	0.19%	0.09%
PBdagcon	10x Hybrid	3.0	1.23GB	1.02%	0.64%
Sparc	30x PB	1.3	2.30GB	0.41%	0.16%
PBdagcon	30x PB	9.3	7.70GB	0.49%	0.23%
Sparc	30x Hybrid	1.3	2.14GB	0.17%	0.02%
PBdagcon	30x Hybrid	9.7	9.58GB	0.49%	0.18%

Sparc scales well to larger datasets; we show here the performance of Sparc and PBdagcon on a larger 20x PacBio *A. thaliana* dataset (genome size 120 Mbp). The longest backbone generated by DBG2OLC was 7.1 Mbp. Sparc finished with 1/5th time and memory compared with PBdagcon while producing more accurate results. Here we used a pure PacBio full genome assembly generated by MHAP (Berlin, et al., 2014) as the reference to calculate the error rates.

Table 2. Results on an *A. thaliana* dataset using PacBio sequencing

Program	Coverage	Time	Memory	Err1	Err2
Sparc	20x Hybrid	21m	1.7GB	0.36%	0.19%
PBdagcon	20x Hybrid	123m	8.9GB	0.81%	0.53%

On an Oxford Nanopore dataset, we set k = 2, g = 2 and ran the consensus algorithms for four rounds in consideration of the higher error rate. The per base error rates for the first round and the fourth round are reported as Err1 and Err2 in Table 3. The results using Oxford Nanopore (ON) data only and using hybrid data are reported in rows 1,2 and rows 3,4 respectively. We noticed that with Oxford Nanopore data, Sparc obtained significantly lower error rate using hybrid data. Below 0.5% error rate was reached even though the raw error rate could be as high as 40%. In contrast, nonhybrid consensus generated less usable results due to this excessive error rate. The longest backbone in this test was 4.6Mbp.

Table 3. Results on an *E. coli* dataset using Oxford Nanopore sequencing.

Program	Coverage	Time	Memory	Err1	Err2
Sparc	30x ON	2.3m	1.89GB	11.96%	7.47%
PBdagcon	30x ON	10.0m	8.38GB	13.70%	12.86%
Sparc	30x Hybrid	3.3m	1.86GB	0.72%	0.46%
PBdagcon	30x Hybrid	13.2m	9.56GB	11.20%	9.96%

We also tested the memory, time and quality of using different parameters. We varied the parameters in the second round of the 30x PacBio *E. coli* dataset using PacBio data only. We found that the results are comparable with some slight differences: using a slightly larger *k*-mer size increases the per-base accuracy, as more strict matches are enforced. However, this also increases the memory consumption because more branching nodes are created. Setting a larger *g* helps to reduce the memory consumption (Table 4).

Table 4. Memory and quality comparisons using different parameters

K	g	Time	Memory	Error rate
1	1	43s	2.3GB	0.16%
2	1	55s	3.5GB	0.14%
1	2	59s	1.6GB	0.18%
2	2	68s	2.3GB	0.13%

CONCLUSION

Consensus module is a critical component in the Overlap-Layout-Consensus assembly framework. With the introduction of the third generation sequencing technology, its importance is further raised. In this work we demonstrated a simple but efficient consensus algorithm that uses *k*-mers as building blocks and produces high quality consensus from a position-specific *k*-mer graph. The proposed method is expected to significantly expand its applications in error correction and variant discovery. The consensus quality can also be increased further by incorporating platform specific signal-level information.

ACKNOWLEDGEMENTS

We thank Chris Hill and Jue Ruan for help and fruitful discussions. *Funding*: The research received funding from the following sources: NSFC (Grant No: 61175071 & 71473243) and “Exceptional Scientists Program of Yunnan Province, China.”

REFERENCES

Au, K.F., *et al.* (2012) Improving PacBio long read accuracy by short read alignment, *PLoS One*, **7**, e46679.

Berlin, K., *et al.* (2014) *Assembling Large Genomes with Single-Molecule Sequencing and Locality Sensitive Hashing*.

Chaisson, M. and Tesler, G. (2012) Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory, *BMC bioinformatics*, **13**, 238.

Chin, C.S., *et al.* (2013) Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data, *Nat Methods*, **10**, 563-569.

Edgar, R.C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput, *Nucleic acids research*, **32**, 1792-1797.

Hackl, T., *et al.* (2014) proovread: large-scale high-accuracy PacBio correction through iterative short read consensus, *Bioinformatics*.

Hannenhalli, S., *et al.* (1996) Positional sequencing by hybridization, *Comput Appl Biosci*, **12**, 19-24.

Huang, X.Q., *et al.* (2003) PCAP: A whole-genome assembly program, *Genome Res*, **13**, 2164-2170.

Koren, S., *et al.* (2012) Hybrid error correction and de novo assembly of single-molecule sequencing reads, *Nature biotechnology*, **30**, 693-700.

Kurtz, S., *et al.* (2004) Versatile and open software for comparing large genomes, *Genome Biology*, **5**, R12.

Larkin, M.A., *et al.* (2007) Clustal W and Clustal X version 2.0, *Bioinformatics*, **23**, 2947-2948.

Lee, C., Grasso, C. and Sharlow, M.F. (2002) Multiple sequence alignment using partial order graphs, *Bioinformatics*, **18**, 452-464.

Lee, H., *et al.* (2014) *Error correction and assembly complexity of single molecule sequencing reads*.

Mallat, S. (2008) *A wavelet tour of signal processing: the sparse way*. Academic press.

Mullikin, J.C. and Ning, Z.M. (2003) The phusion assembler, *Genome Res*, **13**, 81-90.

Myers, E.W., *et al.* (2000) A whole-genome assembly of Drosophila, *Science*, **287**, 2196-2204.

Myers, G. (2014) Efficient Local Alignment Discovery amongst Noisy Long Reads. In, *Algorithms in Bioinformatics*. Springer, pp. 52-67.

Pevzner, P.A., Tang, H. and Waterman, M.S. (2001) An Eulerian path approach to DNA fragment assembly, *Proceedings of the National Academy of Sciences*, **98**, 9748-9753.

Rausch, T., *et al.* (2009) A consistency-based consensus algorithm for de novo and reference-guided sequence assembly of short reads, *Bioinformatics*, **25**, 1118-1124.

Ronen, R., *et al.* (2012) SEQuel: improving the accuracy of genome assemblies, *Bioinformatics*, **28**, i188-i196.

Salmela, L. and Rivals, E. (2014) LoRDEC: accurate and efficient long read error correction, *Bioinformatics*, btu538.

Ye, C., *et al.* (2014) DBG2OLC: Efficient Assembly of Large Genomes Using the Compressed Overlap Graph, *arXiv preprint arXiv:1410.2801*.

Ye, C., *et al.* (2012) Exploiting sparseness in de novo genome assembly, *BMC bioinformatics*, **13 Suppl 6**, S1.