Bioinformatics, 31(24), 2015, 3988–3990
doi: 10.1093/bioinformatics/btv487
Advance Access Publication Date: 26 August 2015
Applications Note

Genome analysis

EPGA2: memory-efficient *de novo* assembler

Junwei Luo[1,2], Jianxin Wang[1,*], Weilong Li[1], Zhen Zhang[1], Fang-Xiang Wu[3], Min Li[1] and Yi Pan[4]

[1]School of Information Science and Engineering, Central South University, ChangSha, 410083, China, [2]College of Computer Science and Technology, Henan Polytechnic University, JiaoZuo, 454000, China, [3]Division of Biomedical Engineering, University of Saskatchewan, Saskatchewan, S7N 5A9, Canada and [4]Department of Computer Science, Georgia State University, Atlanta, GA 30302, USA

*To whom correspondence should be addressed.
Associate Editor: John Hancock

Received and revised on June 19, 2015; accepted on August 13, 2015

Abstract

Motivation: In genome assembly, as coverage of sequencing and genome size growing, most current softwares require a large memory for handling a great deal of sequence data. However, most researchers usually cannot meet the requirements of computing resources which prevent most current softwares from practical applications.

Results: In this article, we present an update algorithm called EPGA2, which applies some new modules and can bring about improved assembly results in small memory. For reducing peak memory in genome assembly, EPGA2 adopts memory-efficient DSK to count K-mers and revised BCALM to construct De Bruijn Graph. Moreover, EPGA2 parallels the step of Contigs Merging and adds Errors Correction in its pipeline. Our experiments demonstrate that all these changes in EPGA2 are more useful for genome assembly.

Availability and implementation: EPGA2 is publicly available for download at https://github.com/bioinfomaticsCSU/EPGA2.

Contact: jxwang@csu.edu.cn

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

1 Introduction

Genome assembly is one of the most important tasks in numerous applied fields (He *et al.*, 2013). When using one software to reconstruct complete genome sequence from sequence data, researchers not only emphasize assembly results but also memory efficiency. Although many softwares have been developed for genome assembly, their balance between accuracy and memory efficiency are not satisfactory due to complex data structures.

We previously published EPGA (Luo *et al.*, 2015), one *de novo* assembler which can resolve some problems caused by complex repetitive sequence regions. Although EPGA can get satisfactory assembly results, it does not have advantage about peak memory comparing with other popular assemblers. The bottleneck of EPGA's memory efficiency primarily exists in two steps: K-mers Counting and De Bruijn Graph Constructing, because EPGA requires that all reads and K-mers reside in memory. Such storage strategy ends up with the memory consumption growing dramatically as the number of reads is increasing.

DSK (Rizk *et al.*, 2013) is one K-mer counting tool which partitions reads, and each partition is separately loaded in memory. BCALM (Chikhi *et al.*, 2014) is one algorithm for building simple paths in De Bruijn Graph which clusters K-mers and iteratively loads each cluster in memory. For resolving memory problem in EPGA, we present EPGA2, which replaces some components in EPGA with DSK and BCALM. In addition, EPGA2 adds Errors Correction in its pipeline and parallels the step of Contigs Merging. The experimental results demonstrate that EPGA2 can produce more satisfactory contigs and scaffolds using small memory.

2 Methods

The EPGA2 pipeline consists of seven steps: (i) Errors Correction: there will usually be some errors in sequencing data, EPGA2 adopts

Table 1. Assembly results

	EPGA								EPGA2								Assemblers
Genome	C.Num	C.CN50 (kb)	C.Cov (%)	S.Num	S.CN50 (kb)	S.Cov (%)	Time (m)	PM (G)	C.Num	C.CN50 (kb)	C.Cov (%)	S.Num	S.CN50 (kb)	S.Cov (%)	Time (m)	PM (G)	PM (G)
S.aureus	22	220	99.68	5	597	99.69	15	9	28	189	99.66	7	753	99.68	35	0.9	2.6 (Abyss)
E.coli	38	198	99.98	19	823	99.98	40	28	33	184	99.98	9	1379	99.99	89	1.7	5.3 (Abyss)
S.pombe	334	70	98.44	103	495	98.47	261	97	355	70	98.73	116	743	98.75	342	6.1	6.6 (Abyss)
N.crassa	6206	10	90.08	4651	21	90.14	2830	198	5711	11	91.11	3632	39	91.11	1721	15	25.6 (Abyss)

C.Num, the number of contigs; C.CN50, the CN50 of contigs; C.Cov, the coverage of contigs; S.Num, the number of scaffolds; S.CN50, the CN50 of scaffolds; S.Cov, the coverage of scaffolds; time, running time; PM, peak memory. Last column is the smallest peak memory and corresponding assembler about other popular assemblers.

BLESS (Heo *et al.*, 2014) to correct errors in reads. (ii) K-mers Counting: DSK is applied to count K-mers. (iii) De Bruijn Graph Constructing: the De Bruijn Graph is constructed based on the K-mers produced previously. (iv) Contigs Assembly: EPGA2 selects long nodes (whose lengths longer than insert size) in De Bruijn Graph as start nodes and iteratively determines their successor nodes and precursor nodes according to one scoring function. (v) Contigs Merging: one contig will be removed if it is included in another contig. Two contigs can be merged together if they have overlap and get sufficient support from paired-end reads information. (vi) Scaffolding: the orientation and order of contigs are determined using paired-end reads. (vii) Gap Filling: the gap regions will be filled by mate reads of reads in the ends of contigs.

The EPGA2 includes several improvements relative to EPGA. First, errors in reads usually cause erroneous edges and lead to lost correct edges in De Bruijn Graph, which increases the difficulty of Contig Assembly. EPGA directly employs raw read libraries to count K-mers and construct De Bruijn Graph which increases the difficulty of the following steps. To improve the correctness of De Bruijn Graph and facilitate the following steps, EPGA2 adds correction step using BLESS.

Second, when counting K-mers, EPGA uses a hash table, where keys are the K-mers and the values are the counts. This simple strategy needs large memory. EPGA2 employs DSK to count K-mers and (K + 1)-mers, which only requires a fixed user-defined amount of memory. EPGA2 only keeps solid K-mer whose frequency is larger than one and solid (K + 1)-mer whose frequency is larger than zero. EPGA loads all reads in memory to construct De Bruijn Graph which requires too much memory. When using BCALM to generate simple paths, EPGA2 introduces one additional condition that each edge in De Bruijn Graph should be one solid (K + 1)-mer. After simple paths created by this revised BCALM, EPGA2 transforms these simple paths to optimized De Bruijn Graph (each simple path is merged to one node and each tip is removed). In this step, the memory will be reduced more effectively. Thirdly, EPGA2 parallels the procedure of Contigs Merging.

3 Experiment

We evaluate the performance of EPGA2 on four real datasets which include two bacteria (*Staphylococcus aureus* and *Escherichia coli*) and two fungi (*Schizosaccharomyces pombe* and *Neurospora crassa*) provided by AllPath2 (MacCallum *et al.*, 2009). Details about these real datasets are shown in Supplementary Table S1. We compare EPGA2 with other popular assemblers: Abyss (Simpson *et al.*, 2009), Velvet (Zerbino and Birney, 2008), SOAPDenvo2 (Luo *et al.*, 2012), PE-Assembly (Ariyaratne and Sung, 2011) and AllPath2. To provide unbiased benchmarks, we use evaluation tool GAGE (Salzberg *et al.*, 2012) which provides corrected analysis. GAGE splits contigs and scaffolds at every error position and provides corrected results. CN50 is N50 of corrected contigs or scaffolds.

Assembly results are listed in Table 1, and the explicit results are listed in Supplementary Tables S2–S5. We can get that EPGA2 offers substantial improvements over the original EPGA. CN50 usually can represent the accuracy of assembly results, EPGA2 gets longer S.CN50 in all real datasets. For coverage, EPGA2 mostly acquires higher coverage than EPGA for contigs and scaffolds. The improvements are caused by adding Errors Correction step which enhances the function of assembly strategies in EPGA.

For the four genomes, EPGA2 only requires 0.9G, 1.7 G, 6.1 G and 15 G memory for assembly which are smaller than EPGA and other popular assemblers. This improvement is due to partition strategies of DSK and BCALM which partition reads and K-mers, and each partition is separately loaded in memory. EPGA2 parallels the step of Contigs Merging which can save time, especially for large datasets. Because the first three datasets are relatively small and the decrease of running time in Contigs Merging is smaller than the increase of running time about BLESS, EPGA2 runs longer time than EPGA in the three datasets. Paralleling Contigs Merging can save more time for large datasets, and therefore EPGA2 runs shorter time than EPGA for the last dataset.

4 Conclusion

In this article, to resolve the memory efficiency problem in EPGA, we present EPGA2, which updates some modules in EPGA. In addition, for reducing running time, EPGA2 parallels Contigs Merging. For improving accuracy of assembly results, EPGA2 adds Errors Correction using BLESS. The experimental results demonstrate the balance between assembly results and memory efficiency of EPGA2 is satisfactory. EPGA2 should be particularly appropriate for researchers with limited computing resources.

Funding

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61232001, No. 61420106009, No. 61379108 and the Program for New Century Excellent Talents in University under Grant NCET-12-0547.

Conflict of interest: none declared.

References

Ariyaratne,P. and Sung,W.K. (2011) PE-assembler: de novo assembly using short paired end reads. *Bioinformatics*, **27**, 167–174.

Chikhi,R. *et al.* (2014) On the representation of de Bruijn graphs. In: Sharan,R. (ed), *RECOMB*, Lecture Notes in Computer Science, Springer International Publishing, Pittsburgh, vol. 8394, pp. 35–55.

He,Y. *et al.* (2013) De novo assembly methods for next generation sequencing data. *Tsinghua Sci. Technol.*, **5**, 500–514.

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

Luo,R. *et al*. (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, **1**, 18.

Luo,J. *et al*. (2015) EPGA: de novo assembly using the distributions of reads and insert size. *Bioinformatics*, **31**, 825–833.

MacCallum,I. *et al*. (2009) ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biol*., **10**, R103.

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

Salzberg,S.L. *et al*. (2012) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res*., **22**, 557–567.

Simpson,J.T. *et al*. (2009) ABySS: a parallel assembler for short-read sequence data. *Genome Res*., **19**, 1117–1123.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short-read assembly using de Bruijn graphs. *Genome Res*., **18**, 821–829.