BIOINFORMATICS

Vol. 00 no. 00 2011
Pages 1–3

Gossamer - A Resource Efficient *de novo* Assembler

Thomas Conway*, Bryan Beresford-Smith, Jeremy Wazny, Andrew Bromage and Justin Zobel

NICTA Victoria Research Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Parkville, Australia

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: The *de novo* assembly of short read high-throughput sequencing data poses significant computational challenges. The volume of data is huge; the reads are tiny compared to the underlying sequence; and there are significant numbers of errors. There are numerous software packages that allow users to assemble short reads, but, with most, either it is only possible to assemble relatively small genomes (e.g., bacteria); or large computing infrastructure is required; or the algorithms are greedy and thus unlikely to yield good results.

Results: We have developed an implementation of the de Bruijn approach to assembly that requires close to the theoretical minimum of memory, but still allows efficient processing. Our software is able to assemble even a human genome on a single inexpensive computer, while producing high quality results.

Availability: Gossamer is available for non-commercial use from http://www.genomics.csse.unimelb.edu.au/product-gossamer.php.

Contact: tom.conway@nicta.com.au

1 INTRODUCTION

High throughput sequencing technologies have enabled researchers to produce unprecedented volumes of short read data. The *de novo* assembly of such data is a core problem in bioinformatics with numerous applications in analysis of genomes, metagenomes, and transcriptomes. There are several common approaches to the *de novo* assembly of short read data, including those based on greedy extension (Warren *et al.* (2006)), overlap layout extension (Hernandez *et al.* (2008)), and de Bruijn graphs (Chaisson *et al.* (2009); Zerbino *et al.* (2009)). Our assembler, *Gossamer* , is an extension of a prototype based on the succinct representation of de Bruijn assembly graphs as a bitmap or set of integers (Conway and Bromage (2011)). It supports assembly of base-space paired reads such as those from an Illumina sequencing platform.

2 METHODS

Gossamer operates in a series of explicit passes to give the user control of the assembly process. Broadly, assembly proceeds through the following phases: graph construction, graph 'cleaning' to remove spurious edges, alignment of pairs to the Eulerian super-graph, Eulerian super-path lifting, and finally contig production.

For a given node degree k, with our method the de Bruijn graph is constructed by extracting from the input all edges of the graph of length $\rho = k + 1$, the ρ-mers , and their reverse complements. Gossamer accepts FASTA and FASTQ input, and will uncompress files on the fly. The current version of *Gossamer* allows for values of $\rho \leq 62$.

Gossamer provides multiple operations for removing spurious edges from the graph. These fall in to two classes: spectral and structural. The spectral error removal operation is the trimming of low frequency edges. The structural error removal operations are the pruning of *tips*, and the elimination of *bubbles*, both based on the algorithms present in Velvet Zerbino and Birney (2008). Unlike the Euler family of assemblers, *Gossamer* does not in this or other passes attempt to correct errors, and simply removes them from the graph.

The trimming of low frequency edges is done in a context free manner, and is implemented using a sequential pass over the set of edges in the graph, which performs well even if the graph is larger than RAM, and in practice results in a graph that is smaller than RAM, even for human-sized genomes. The graph building process yields a ρ-mer frequency histogram that can be used to guide the choice of trimming cut-off.

The structural passes perform random accesses on the graph, doing path traversal, and require that the succinct graph representation fit in memory. In the case of both tips and bubbles, their elimination tends to remove a small number of edges (a few percent, typically), but leads to a large increase in distance between branches.

After these graph-cleaning passes, the resulting de Bruijn graph contains many fewer spurious edges, and the unbranched paths can be read off as preliminary contigs. Read pair information is utilized by aligning both ends of the pair to the de Bruijn graph to find pairs of 'anchors' into parts of the graph judged to be most likely unique (i.e. copy-number 1), in the underlying genome. For each pair of anchors with sufficient support, a search is performed to find a unique path that is consistent with the bounds defined by the distribution of insert sizes. The bounds are given in terms of the expected insert size, standard deviation (in percent), and number of standard deviations to allow. Where such paths are found, an Eulerian super-path is constructed.

*to whom correspondence should be addressed

3 RESULTS

To demonstrate the applicability of *Gossamer* we present results for assembling four simulated data sets (generated with `wgsim`) and also for assembling two experimental data sets. The simulated data sets have been chosen to facilitate comparison with other work Lin *et al.* (2011). Each was generated with a coverage of 70x and a uniform base error rate of 0.6%. The first two of the synthetic data sets (*Ecoli35* and *Ecoli100*) are from E. coli (Genbank accession NC_009800), a 4.64Mbp genome with read length 35bp and 100bp, and insert size 200bp and 570bp respectively. The remaining two synthetic data sets (*Hsap35* and *Hsap100*) are from Human chromosome three (Genbank accession NT_005612), a 100.5Mbp contig with the same two read-length and insert-size combinations. The assemblies were done with $\rho = 26$ for all data sets, and $\rho = 56$ for the data sets with 100bp reads.

The two experimental data sets are for E. coli and for H. sapiens. Note that the assemblies we have reported here are for proof-of-efficiency, not for final assessment of the assemblies. We have not, for example, filtered the read sets in any way. The first read set (*EcoliExp*) is a sample of Escherichia coli str. K-12 substr. MG1655 (accession ERR022075). The data is 22.7 million paired reads with 100bp on each end, sequenced on an Illumina IIx. The insert size (including adapters) is reported as 600bp. The nominal genome coverage is about 980 times. [[Is this in the table - HSapExp not present?]] The second read set (*HsapExp*) is a human genome (accession NA12878) Gnerre and MacCallum... (2011). We assembled, using $\rho = 28$, the fragment library of this study, of approximately 900 million paired end reads of length 100bp with insert size 155bp and a genome coverage of about 50 times. For paired end data we used XXX short jumping library reads with length 100bp and insert size about 2500bp.

For all the test sets except *HsapExp*, we used a single server with 8 AMD Barcelona cores and 32GB RAM running Ubuntu Linux. In the case of *HsapExp* we used three of these same servers for the concurrent building of the initial graph of size 240GB. [[Same graph size for all assemblers?]] All subsequent steps of the assembly used a single server, except for pair resolution, which had a peak memory usage of XXXGB and was performed on YYY.

[[99% of what? ie what is $NG50_{99}$?]]

The results of the assembly (other than for HsapExp) are shown in Table 1. [[So table caption is wrong?]] Of the assemblies that will successfully complete on our 32GB installation, the three assemblers produce assemblies of comparable quality. All three were run with only very modest effort at tuning. For *Gossamer* , we used default parameters, except for those governing the number of threads, maximum buffer sizes, and so on.

A detailed analysis of the performance of each assembler is beyond the scope of this paper. However, we note that Velvet consistently produces the best N50, but its $NG50_{99}$ is much lower — a significant number of the contigs it produces have a number of small errors scattered through them, or spuriously join otherwise correct shorter contigs.

Note that *Gossamer* writes out a new copy of the graph structure after each stage in the assembly process (e.g., after each pass of tip pruning, of which there are several). This repeated writing accounts for a substantial proportion of the total runtime. Further work will allow multiple passes to be run together, which will yield a significant speedup overall.

Table 1. Comparison of assembly results for Gossamer (Go), Velvet (Ve) and SOAPdenovo (SO). All results are for $\rho = 26$ unless specified.

Dataset	Tool	Cov	N50 (%)	Cov_{99} (%)	$NG50_{99}$	Time (s)	RAM (MB)
Ecoli35	Go	97.66	37,755	94.01	**34,753**	350	**586**
	Ve	**97.78**	**71,930**	70.51	33,844	476	1461
	SO	97.50	22,165	**97.50**	21,694	**120**	1099
Ecoli100				$\rho = 26$			
	Go	97.53	42,594	**92.59**	**37,446**	2032	**746**
	Ve	**97.55**	**73,554**	60.53	28,149	888	2001
	SO[a]	0.29	100	0.01	0	**367**	1917
				$\rho = 56$			
	Go	98.05	69,380	81.23	49,528	1536	**523**
	Ve	98.62	**90,404**	67.90	50,003	396	2107
	SO	**98.82**	69,537	**98.35**	**69,537**	**120**	6781
Hsap35	Go	**84.96**	**3358**	84.16	**2669**	10,853	**5854**
	Ve	row1	row1	row1	row1	row1	row1
	SO	84.77	2153	**84.77**	1692	**3180**	15,502
Hsap100				$\rho = 26$			
	Go	82.33	3251	81.37	2438	17,365	4045
	Ve	row1	row1	row1	row1	row1	row1
	SO	row1	row1	row1	row1	row1	row1
				$\rho = 56$			
	Go	row1	row1	row1	row1	row1	row1
	Ve	row1	row1	row1	row1	row1	row1
	SO	row1	row1	row1	row1	row1	row1
EcoliExp	Go	row1	row1	row1	row1	row1	row1
	Ve	row1	row1	row1	row1	row1	row1
	SO	row1	row1	row1	row1	row1	row1

The columns have the following meaning. Size is the total length of the contigs expressed relative to the size of the reference genome; N50 is the conventional N50 statistic computed across all contigs; Cov_{99} is the coverage of the reference genome of contigs that map with 99% identity; $NG50_{99}$ is the NG50 of contigs which map to the reference genome at 99% identity; '—' is given for cases where the 32GB of RAM is insufficient or runtime exceeds 24 hours.

[a] For this experiment, SOAPdenovo produced over 2 million contigs of length 100 or less, which suggests that this assembler can perform badly for seemingly reasonable parameter choices.

ACKNOWLEDGEMENT

National ICT Australia (NICTA) is funded by the Australian Government's Department of Communications; Information Technology and the Arts; Australian Research Council through Backing Australia's Ability; ICT Centre of Excellence programs.

REFERENCES

Chaisson, M. J., Brinza, D., and Pevzner, P. A. (2009). De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research*, **19**(2), 336–46.

Conway, T. C. and Bromage, A. J. (2011). Succinct data structures for assembling large genomes. *Bioinformatics*, **27**(4), 479–86.

Gnerre, S. and MacCallum..., I. (2011). High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the ....*

Hernandez, D., Francois, P., Farinelli, L., Osteras, M., and Schrenzel, J. (2008). De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*, **18**(5), 802–809.

Lin, Y., Li, J., Shen, H., Zhang, L., and Papasian..., C. (2011). Comparative studies of de novo assembly tools for next-generation sequencing technologies. ....

Warren, R. L., Sutton, G. G., Jones, S. J. M., and Holt, R. A. (2006). Assembling millions of short dna sequences using ssake. *Bioinformatics*, **23**(4), 500–501.

Zerbino, D. R. and Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, **18**(5), 821–829.

Zerbino, D. R., McEwen, G. K., Margulies, E. H., and Birney, E. (2009). Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS ONE*, **4**(12), e8407.