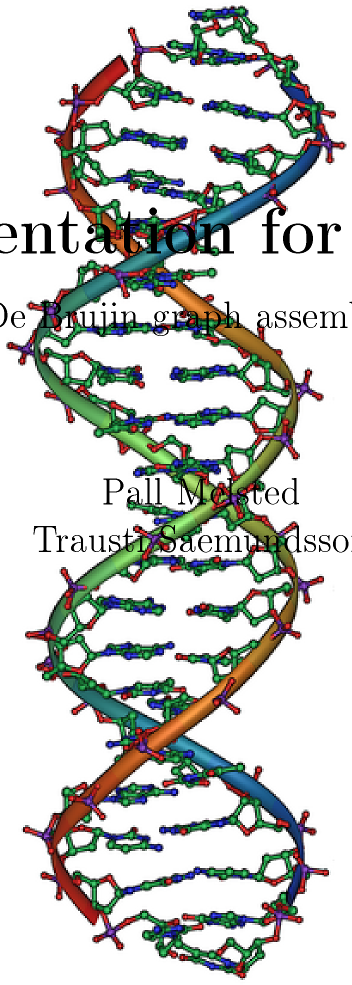

Documentation for BFGraph

A memory efficient De Bruijn graph assembler using Bloom Filters

Pall Melsted
Trausti Saemundsson

BFGraph Documentation August 13, 2012

Questions and bug reports: pmelsted@gmail.com

Contents

Cover photo is from Wikipedia[8]

Chapter 1

Introduction

This is a documentation for the program BFGraph. BFGraph is a De Brujin graph assembler. Currently it makes the pregraph but does not simplify it like SOAPdenovo[6] and Velvet[7]. It makes the pregraph by using Bloom Filters in contrast to most other assemblers, that use hash tables. This saves a lot of memory and the result is independent of the false positive rate of the Bloom Filter.

A Bloom Filter is quite similar to a hash table but. If a Bloom Filter is queried for a specific key, it answer correctly if the key is not stored but answers incorrectly according to the false positive rate if the key is stored within it.
The first phase of BFGraph makes the pregraph according to a Bloom Filter but then fixes the graph afterwards which makes it independent of the probabilistic nature of the Bloom Filter. The memory usage of BFGraph is a lot lower than that of most other assemblers, and it is compared in the paper about this program.

Chapter 2

Definitions

kmer[9]: string of A,C,G,T which has length k (k is often 31)
twin: interchange A <-> T and C <-> G in a string and then reverse it
contig: string of A,C,G,T which has length greater or equal to the kmer size

Chapter 3

Dependencies

Inside the program directory are a few programs made by others:

- **sparse_hash**[2] is inside the directory google, a memory efficient hash table made by Google used in this program for storing common kmers.
- **libdivide**[5] in the file *libdivide.h* is used for fast integer division.
- **kseq**[1] in the file *kseq.h* is used for fasta/fastq file reading.

Chapter 4

Usage

First the program has to be compiled. Run `‘make‘` to do that. The directory *example* contains two small read files in fastq format: *tinyread_1.fq* and *tinyread_2.fq*.

Here follows a guide on how to run the program on these two files with one thread and kmer-size 31. The script *example.sh* does the same as described below.

4.1 Part I: Filter the reads

This command filters the reads and saves the result to a new file, *example/output/tiny.bf* (it will contain a Bloom Filter).

```
$ ./BFGraph filter example/tinyread_*.fq -k 31 -t 1 -o example/output/tiny.bf -n 8000 -N 4000 -v
```

The parameter **-n** is an upper bound of the number of kmers from the read files and the parameter **-N** is an upper bound of the number of *different* kmers from the read files. The parameter **-v** is for verbose mode.

(a) Values for -N and -n

On the TODO list for BFGraph is to write a program to estimate these numbers from the reads. But until it has been written the user has to estimate them by himself.

The first number is easy to calculate by hand. The files *tinyread_1.fq* and *tinyread_2.fq* have in total 2000 reads of length 70. Since the kmer-size is 31, we will get 40 kmers from every read ($70 - 31 + 1 = 40$). Thus the number of kmers from the files is: $40 \cdot 2000 = 8000$.

NOTE: These number do not have to be accurate, but the program runs faster if they are close to correct values.

We can expect that an average kmer will be seen at least twice so 4000 is not a bad value for **-N**.

4.2 Part II: Create the contigs

This phase of the program reads the file *example/output/tiny.bf*, which is a Bloom Filter, and creates contigs from the kmers. The program must be run with the same kmer-size as the Bloom Filter file was created with, in this case 31. This command creates the contigs in one thread and saves the results into files with the prefix: *example/output/tiny*.

```
$ ./BFGraph contigs example/tinyread_*.fq -k 31 -t 1 -f example/output/tiny.bf -o example/output/tiny -v
```

4.3 Extra: Visualize the De Brujin graph

For small read files the Python program *make_graph.py* can create a **.dot** file with Graphviz[3] to visualize the De Brujin graph.
If you ran the commands above you can now run this program with the prefix from Part II above.

```
$ ./make_graph.py example/output/tiny
```

This creates the file *example/output/tiny.dot*.

This file can be read a native **.dot** file reader like ZGRViewer[10] or converted to **.PNG** if Graphviz is installed with the following command:

```
$ dot -Tpng example/output/tiny.dot -o example/output/tiny.png
```

Chapter 5

Structure of the program

The file *BFGraph.cpp* is compiled into the executable file *BFGraph* when ‘make‘ is run. *BFGraph* runs the correct functions based on the input.

When the command ’filter’ is given, the method *FilterReads* is called, (implemented in *FilterReads.cpp*).
When the command ’contigs’ is given, the method *BuildContigs* is called, (implemented in *BuildContigs.cpp*).

The program flow is quite similar for those two commands. Parameters are first validated and an appropriate error message is given on any error. If all the parameters are valid, either *FilterReads_Normal* or *BuildContigs_Normal* is called depending on the command name. Both those methods go through all the reads (in a while loop) in as many threads as given by the parameter **-t**.

The OpenMP[4] library is used for parallel programming in BFGraph.

Chapter 6

Files

6.1 Unused files

Every file in BFGraph's base directory is used by the program except the following files:

- *CountBF.hpp*
- *CountBF.cpp*
- *DumpBF.hpp*
- *DumpBF.cpp*
- *KmerIntPair.hpp*
- *KmerIntPair.hpp*
- *bloom_filter.hpp*
- *BlockedBloomFilter.hpp*

6.2 Used files

The following classes reside in files by a similar name except ContigRef, it is located in *KmerMapper.hpp* and FastqFile is located in *fastq.hpp*

(a) Classes

- BloomFilter
- CompressedCoverage
- CompressedSequence
- Contig
- ContigRef
- FastqFile
- Kmer

- KmerIterator
- KmerMapper

(b) Structs

- *FilterReads_ProgramOptions* is used as the name indicates to store parameter values from the user for the subcommand **filter**, located in *FilterReads.cpp*
- *BuildContigs_ProgramOptions* is used to store parameter values from the user for the subcommand **contigs**, located in *BuildContigs.cpp*7
- *CheckContig*
- *FindContig*
- *MakeContig*
- *NewContig*
- *KmerHash*

Chapter 7

Important methods in the program

Chapter 8

Final words

Thanks.

Bibliography

[1] attractivechaos. *Generic stream buffer plus a FASTA_Q parser*. [Online; accessed 10-August-2012]. 2012. URL: https://github.com/attractivechaos/klib/blob/master/kseq.h.

[2] Google. *sparsehash - An extremely memory-efficient hash_map implementation*. [Online; accessed 10-August-2012]. 2012. URL: http://code.google.com/p/sparsehash/.

[3] Graphviz. *Graphviz - Graph Visualization Software*. [Online; accessed 13-August-2012]. 2012. URL: http://www.graphviz.org/.

[4] OpenMP. *The OpenMP API specification for parallel programming*. [Online; accessed 13-August-2012]. 2012. URL: http://openmp.org/wp/.

[5] ridiculous_fish. *libdivide is an open source library for optimizing integer division*. [Online; accessed 10-August-2012]. 2012. URL: http://libdivide.com/.

[6] SEQanswers - SEQwiki. *SOAPdenovo - SEQwiki*. [Online; accessed 10-August-2012]. 2012. URL: http://seqanswers.com/wiki/SOAPdenovo.

[7] SEQanswers - SEQwiki. *Velvet - SEQwiki*. [Online; accessed 10-August-2012]. 2012. URL: http://seqanswers.com/wiki/Velvet.

[8] Michael Ströck. *An overview of the structure of DNA*. [Online; accessed 9-August-2012]. 2006. URL: http://commons.wikimedia.org/wiki/File:DNA_Overview2.png.

[9] Wikipedia. *k-mer — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-August-2012]. 2012. URL: http://en.wikipedia.org/wiki/K-mer.

[10] ZGRViewer. *ZGRViewer, a GraphViz/DOT Viewer*. [Online; accessed 13-August-2012]. 2012. URL: http://zvtm.sourceforge.net/zgrviewer.html.