Cortex Tutorial

Ricardo H. Ramrez-Gonzlez, Mario Caccamo

April 1, 2011

1 Tutorial files

This tutorial is based on two files.

E. Coli. Reference The complete reference genome of Escherichia coli str. K-12 substr. MG1655 (Accession U00096.2)saved as `ecoli.fasta`. Note that any small reference genome can be used instead.

E. Coli. Samples The illumina runs stored with accession number SRR001666 named as `SRR001666_1.fastq` and `SRR001666_2.fastq`. This dataset is paired end with an insert site of 500bp

2 Compiling cortex

Inside the cortex

```
$ make MAXK=31
```

MAXK can be 31, 63 or 95. That is the maximum k-mer size that the specific compilation of cortex will use.

Cortex is designed to run on 64-bits machines, however it is possible to compile it on 32 bits machines with the flag 32_BITS=1

```
$ make MAXK=31  32_BITS=1
```

To enable the use of read pairs, the flag ENABLE_READ_PAIRS=1

```
$ make MAXK=31 ENABLE_READ_PAIRS=1
```

Please note that currently Cortex only compiles with compilers that allow nested functions. It has been tested with GCC 4.2.*.

3 Simple assembly on reference genome

To test if cortex is running, we will load the E.Coli. reference into cortex and just output the contigs.

1. Create a text file with the path to `ecoli.fasta` and name it `ecoli_reference.txt`
2. Execute cortex as follows:

```
cortex_con_31
        --input_format fasta
        --input ecoli_reference.txt
        --mem_height 16
        --kmer_size 27
        --output_supernodes supernodes.fa
```

3.1 To think about...

1. How many sequences does the reference file has?
2. How many sequences does supernodes.fa has?
3. Is there any reason for the difference?
4. What happens when you increase the kmer size to 31, or 63?

For each file of contigs created in the tutorial think about:

- The amount of assembled sequence
- The N50.
- The number of contigs.

Beware, this is not a comprehensive list of quality metrics, but are enough to give a rough idea.

4 A more realistic example

Now, we will assemble the illumina sample dataset. We had printed the the contigs without any ambiguity only once. Now we will output contigs which are still reliable, printing different flanking sequences to a repetitive region.

1. Create a text file with the paths to `SRR001666_1.fastq` and `SRR001666_2.fastq.`, each path in its own line. Save it as `ecoli_samples.txt`
2. create the graph and store it as it is in memory.

```
cortex_con_31
        --input_format fastq
        --input ecoli_samples.txt
        --mem_height 18
        --kmer_size  27
        --hash_output_file ecoli_from_reads1.mem
```

3. Load the graph in to memory and print the supernodes.

```
cortex_con_31
         --input_format hash
         --input ecoli_from_reads1.mem
         --output_contigs sample_supernedes.fa
```

4.1 Cleaning

So far ,we had assembled all the reads bases in the raw files. However, the reads always contain noise and we need to clean them. The following commands show how the cleaning can be done.

4.1.1 Clp tipping

As the end of the read is more prone to have sequencing errors, some paths in the graph may be completely disconnected. If the path is short enough, it can be removed. In the following example, the minimum length a tip can have to be real is 100 kmers.

```
cortex_con_31
         --input_format hash
         --tip_clip 100
         --input ecoli_from_reads1.mem
         --output_contigs sample_tc100_supernedes.fa
```

4.1.2 Remove low coverage supernodes

A chimeric read or a sequencing error could potentially lead to paths joining two different contigs. To reduce the chances of this happening,

```
cortex_con_31
         --input_format hash
         --remove_low_coverage_supernodes 1
         --input ecoli_from_reads1.mem
         --output_contigs sample_tc100_supernedes.fa
```

4.1.3 Remove bubbles

A single base error or a SNP lead to a bubble shaped structure in the graph. If the aim is to find a consensus sequence, removing the bubbles extend the length of the contigs.

```
cortex_con_31
         --input_format hash
         --remove_bubbles
         --input ecoli_from_reads1.mem
         --output_contigs sample_rb_supernedes.fa
```

5 Large genomes

One way to speed up the assembly process and to cope with a large amount of reads is to create the graph for each fastq file and merge them at a later stage.

1. Create a file with the path to `SRR001666_1.fastq` called `ecoli_sample1.txt`
2. Create a binary file just for `SRR001666_1.fastq` using the flag `--dump_binary`

   ```
   cortex_con_31
           --input_format fastq
           --input ecoli_sample1.txt
           --mem_height 18
           --kmer_size  27
           --dump_binary ecoli_from_reads1.ctx
   ```

3. Repeat 1 and 2 with the reads from `SRR001666_2.fastq`
4. Create a file with the file to both `ctx` files.

   ```
   cortex_con_31
           --input_format binary
           --input ecoli_ctx_both_samples.txt
           --mem_height 19
           --kmer_size 27
           --hash_output_file ecoli_from_reads.mem
           --dump_binary ecoli_from_reads.ctx
           --output_supernodes contigs_from_reads.fa
   ```

Note that you can either dump to a new ctx if you want to keep adding kmers, to a hash file, if you don't want to keep adding files. This strategy can be used to do a partial clean when you have memory constrains too.

6 Large genomes with huge coverage

When dealing with datasets larger than your infrastructure can deal in a reasonable time, it is possible to do a divide and conquer strategy. This is specially useful when dealing with the HiSeq platform where the fastq files are significantly larger than on the previous generation.

1. Create a file of files with the sections of the fastq you want to assemble.

   ```
   scripts/util/build_cortex_entries.pl both_lanes.txt 4 mini_lanes
   ```

```
for i in  {1..8} ; do
./release_1.0.1/bin/cortex_con_31
        --input_format fastq
        --input mini_lanes.$i
        --mem_height 17
        --kmer_size  27
        --dump_binary ecoli_mini_$i.ctx
done;
```

7 FAQ

7.1 What is the difference between the hash and binary formats?

The hash format is a dump of the whole graph, you can't modify it's settings. It is used when you had readed all the input files and you just want to see the impact of different cleaning or printing algorithms. The binary format stores all the kmers only once, however you can merge different binary files to get the full graph. This is slower to load, but it gives you the flexibility to make an incremental load of reads.