

C C A T T 0 1 0 0 0
G A G G A 0 1 1 0 1
G A A T T 0 0 1 1 0
A C A A G 0 0 1 0 0
T A C C A 0 0 1 1 0
T T A C A 0 1 0 0 0
A C C T C 0 0 0 1 0
A A G G A 0 0 0 0 0
G A T G A 0 1 1 0 0
T A G A T 0 0 1 0 0
G A T G A 1 0 1 0 0
T G T A G 1 0 0 0 0
T A G T A 0 0 0 0 0
G A T A T 1 0 0 0 0
G A G T G 1 1 0 0 0
A G A T T 1 1 0 0 0
G A G T A 1 1 0 0 0
T G A T G 1 1 0 0 0
A T T A G 1 1 0 0 0
T A G A T 1 1 0 0 0
G A G A 1 1 0 0 0
G T A 1 1 0 0 0
G A T 1 1 0 0 0
T A G 1 1 0 0 0
A G A 1 1 0 0 0
G A 1 1 0 0 0
A 1 1 0 0 0
T 1 1 0 0 0

White Paper

White paper on reference assembly in CLC Assembly Cell 3.0

May 10, 2010





Contents

1 Introduction	3
2 Algorithms	3
2.1 Short reads assembly	3
2.1.1 Scoring	3
2.1.2 Multiple matches	4
2.1.3 Paired reads	4
2.1.4 Gapped alignment	4
2.1.5 Reported Results	5
2.2 Long reads assembly	5
3 Benchmarks	5
3.1 Data	5
3.2 Comparison with other assembly software	6
3.2.1 Results	6
3.3 Gapped alignments	7
3.4 Paired benchmarks	7
3.5 General performance benchmarks	8
3.6 Color space benchmarks	9

1 Introduction

This white paper explains how the reference assembly programs of the *CLC Assembly Cell* work and how they perform under different circumstances.

First, the details of the algorithms are explained, and next follows a series of benchmarks comparing with other assembly software and using various parameters and data sets.

2 Algorithms

There are two variations of the assembly programs: one for long reads (*clc_ref_assemble_long*) and one for short reads (*clc_ref_assemble_short*).

2.1 Short reads assembly

By default, *clc_ref_assemble_short* does local alignment of reads to a set of reference sequences. The advantage of performing local alignment instead of global alignment is that the ends are automatically removed if there are sufficiently many sequencing errors there.

Like Maq, Soap, and other similar programs for short read assembly, the default setting is to use ungapped alignment. Also, a threshold is set for the alignment quality for each read. With Maq and Soap, this is done by specifying the number of allowed mismatches. This works because those programs do global alignment. For local alignments it is a little more complicated.

The default alignment scoring scheme of *clc_ref_assemble_short* is +1 for matches and -2 for mismatches. The limit for accepting an alignment is given as the alignment score relative to the read length. For example, if the score limit is 8 below the length, up to two mismatches are allowed as well as two ending nucleotides not assembled (remember that a mismatch costs 2 points, but when there is a mismatch, a potential match is also lost). Alternatively, with one mismatch, up to 5 unaligned positions are allowed. Or finally, with no mismatches, up to 8 unaligned positions are allowed. See figure 1 for examples. The default setting is exactly this limit of 8 below the length.

It is also possible to use global alignment in CLC's reference assembler (see below under parameters).

2.1.1 Scoring

There are two parameters concerning the scores of ungapped alignments: the mismatch score and the score limit. The match score is always +1. If the mismatch cost is changed, the default score limit will also change to:

$$\text{score limit} = 3 \times (1 + \text{mismatch cost}) - 1 \quad (1)$$

The default mismatch score of -2 equals a mismatch cost of 2 and a score limit of 8 below the read length, as stated above. For any mismatch cost, the default score limit allows any alignment scoring strictly better than 3 mismatches.

2.1.5 Reported Results

For each assembled read, the guaranteed best scoring alignment is found with the reference sequences. The number of such optimally scoring alignments are also found. For each read, the assembly program reports one random optimal alignment as well as the number of alignments scoring equally well. Reads that do not reach the chosen score cutoff are reported as not assembling.

2.2 Long reads assembly

The long read assembler is very similar to the short read assembler. The differences are:

- The reads may have any length up to 8000 bp long
- Alignments are always gapped
- The criteria for accepting a match is different (see below)

As described above, the short read assembler accepts matches based on the score of the alignment. This makes sense for short reads, where a small number of differences are expected. In contrast, long reads often have varying lengths and more differences in total, so a different approach should be used. Thus, the long read assembler uses a criteria, where a given fraction (default 50 %) of the read must match the reference with a given sequence similarity (default 80 %). If 70 % of a read matches the reference, it is acceptable that the similarity for these 70 % is less than 80 % as long as a part of the match covering 50 % of the read matches with at least 80 % similarity.

As for short read assembly with gaps, it is difficult to guarantee finding all optimal hits without performing a very time consuming full alignment calculation. Thus, in a few rare cases, an optimal alignment may be missed by the long read assembler.

3 Benchmarks

We have performed a number of benchmarks showing different aspects of the *CLC Assembly Cell*. First of all, we compare it with other assembly softwares. Second, we compare the performance using gaps and no gaps. Next, we show how paired data affect performance of the *clc_ref_assemble_short*. Finally, different data sets are benchmarked on different hardware set-ups.

3.1 Data

The benchmarks are done on several data sets. An overview of the data sets used are given in table 1.

The first *E. Coli* data set is sequenced on the 454 platform, and the second on Illumina's Genome Analyzer. They are both commensal strain K-12 of *E. Coli*. Both data sets are available from <http://www.clcbio.com/ngsexampledta>.

The *C. Elegans* data set is sequenced on the Illumina Genome Analyzer platform downloaded from the Short Reads Archive at NCBI ¹, and the reference sequences are the six *C. Elegans*

¹<http://www.ncbi.nlm.nih.gov/Traces/sra>, IDs SRS000837 and SRS000838

Name	Reference (bp)	Reads	Reads (bp)	Length	Paired
454 <i>E. Coli</i>	4,686,137	436,142	102,245,955	Long	Single
Illumina <i>E. Coli</i>	4,686,137	5,244,764	183,566,740	Short (35 bp)	Paired
<i>C. Elegans</i>	100,267,632	5,216,050	172,129,650	Short (33 bp)	Single
Small human	3,080,436,051	8,597,824	300,923,840	Short (35 bp)	Paired
Large human	3,080,436,051	85,978,232	3,009,238,120	Short (35 bp)	Paired

Table 1: The data sets used for benchmarks.

chromosomes downloaded from RefSeq.

The human data set is sequenced on an Illumina Genome Analyzer. The data are from a paired end run done at Beijing Genomics Institute (BGI) and were kindly put at our disposal for these benchmarks.

3.2 Comparison with other assembly software

We compared *clc_ref_assemble_short* with *Maq* version 0.6.7 and *Soap* version 1.10 (see <http://maq.sourceforge.net/> and <http://soap.genomics.org.cn/>). Both were compiled on 64 bit Linux. The comparison was done using ungapped alignment of reads treated as unpaired.

The benchmarks comparing with other software were run on a computer with two quad core 2.50 GHz Intel Xeon E5420 CPUs (i.e. a total of 8 cores) with 32 GB of memory. The operating system was 64 bit Fedora Core 9 Linux.

3.2.1 Results

Table 2 shows time and memory consumption for reference assembly of the two human data sets using the three programs described here.

Program	Read set	Time	Memory	Assembled
Maq	Small human	3:57:48	5 GB	83.13 %
Soap	Small human	2:55:15	14 GB	83.04 %
CLC	Small human	0:28:14	5 GB	85.29 %
Maq	Large human	39:37:56*	5 GB	83.14 %*
Soap	Large human	27:52:19*	14 GB	83.05 %*
CLC	Large human	2:14:53	7 GB	85.30 %

Table 2: Approximate results. Note that Maq is not multi threaded, so the data was split in 8 parts to use all 8 cores of the computer. The time and memory requirements shown here are for such a split of the data. Each process uses 620 MB of memory for a total of 5 GB.

* These numbers are estimates.

The reason that the CLC assembly program assembles more reads than Maq and Soap is that it does local alignment instead of global alignment. This allows reads to align with more than two mismatches at the ends.

3.3 Gapped alignments

With the `clc_ref_assemble_short`, choosing to perform assembly using ungapped alignment is faster than using gapped alignments. To give an indication of the difference in speed, we have performed an assembly of the same data set using gapped and un-gapped alignment, respectively. The parameter used when running the `clc_ref_assemble_short` is `-u`.

The data set used is sequence data from the Illumina *E. coli* data set as well as the big human data set (as described in table 1).

Table 3 shows the difference in speed.

	Time	Memory	Assembled
<i>E. Coli</i> with gaps	1m 07s	0.2 GB	95.38 %
<i>E. Coli</i> without gaps	54.1s	0.2 GB	95.36 %
Human with gaps	6h 29m 50s	5.7 GB	85.47 %
Human without gaps	2h 16m 02s	5.6 GB	85.32 %

Table 3: Performing the same assembly using gapped and ungapped alignment.

Generally, it is slower to use gapped alignment. For small data sets the difference is not so great, but for large assemblies the difference is significant. For the large data set of one time coverage of the human genome, it means that an assembly of 2h 16m will be 6h 30m using gapped alignment. You can also see a very small rise in the amount of assembled reads but it is not significant.

Along with an increase in time spent on the assembly, using gapped alignment also increases the amount of memory used.

3.4 Paired benchmarks

This section reports how the use of paired data affect speed and results of an assembly. We use the same data sets as for testing gapped alignment (see table 3), and the assembly has been run with and without using paired information (the `-p` option).

The results are shown in table 4.

In contrast to the gapped vs. ungapped comparison, the increase in time spent going from not paired to paired assembly is not dramatic. The big data set goes from 2h 16m without using paired information to 2h 44m using paired. The most significant difference is in memory consumption. The paired run for the large data set uses 35 % more memory. This is because more intermediate results need to be stored while performing the assembly.

	Time	Memory	Assembled
<i>E. Coli</i> paired	1m 20s	0.4 GB	95.38 %
<i>E. Coli</i> not paired	1m 06s	0.2 GB	95.38 %
Human paired	2h 44m 01s	8.6 GB	85.32 %
Human not paired	2h 16m 02s	5.6 GB	85.32 %

Table 4: Benchmarks with and without using paired information.

3.5 General performance benchmarks

Given the wide variety in the amount of data and hardware set-ups, we have conducted a number of benchmarks against different data sets on different computers. This makes it easier to get an idea of how the *CLC Assembly Cell* would perform on your own data and your own computer set-up.

We have tested on two computers:

Computer A Two quad core 2.50 GHz Intel Xeon E5420 CPUs (i.e. a total of 8 cores) with 32 GB of memory. Operating system is 64 bit Fedora Core 9 Linux.

Computer B An Intel 2.13 GHz Core2 Duo E6400 processor with 2 GB of memory. Operating system is Fedora Core 6 Linux.

We have chosen these two computer set-ups because they show how the different data sets will perform on a fast computer with a lot of memory (Computer A), and what it means if you scale down to a standard desktop computer with moderate amount of memory.

The results of the benchmarks are shown in figure 5.

On average, the Computer A is between 2 and 5 times as fast as Computer B. For the smaller *E. coli* data sets, this is mainly because it has more processing power (Computer A has eight cores and Computer B has two cores). We have not tried to run the large human data set on the small computer, since this is probably not a realistic scenario.

An important thing shown by this table is that although the amount of memory has an impact on speed, it will not run out of memory. If you look at the *C. Elegans* and the small human data set, then the memory consumption is greater on Computer A than on Computer B. This is because the assembly program adjusts to the amount of available memory. This also means that the assembly could have run faster on Computer B if it had more memory

Note that there is always an overhead reading and writing files etc. which is more or less the same for the two computers. Together with differences in the CPU speeds and amount of memory it explains why you do not see a speed-up of 4 going from 2 to 8 cores. The overhead is clearly illustrated when you compare the speeds of the small and the large human data sets. The small data set is run with paired information and ungapped alignment. The amount of data is 10 times as much in the big data set as in the small. But you do not see an increase in the time of 10 times. Rather the time increase is about 4 (going from 40m to 2h 44m) and this is because the great size of the reference genome imposes a big overhead.

Data set	Computer A		Computer B		Assembled
	Time	Memory	Time	Memory	
454 <i>E. Coli</i>	19s	0.7 GB	1m 29s	0.1 GB	97.24 %
Illumina <i>E. Coli</i>	26s	0.5 GB	53.8s	0.2 GB	95.36 %
<i>C. Elegans</i>	1m 18s	2.3 GB	3m 19s	1.7 GB	74.45 %
Small human	40m 01s	5.8 GB	2h 35m 19s	1.9 GB	93.80 %
Large Human:					
ungapped and no pair	2h 16m 02s	5.6 GB	N/A	N/A	85.32 %
gapped and no pair	6h 29m 50s	5.7 GB	N/A	N/A	85.47 %
ungapped and paired	2h 44m 01s	8.6 GB	N/A	N/A	85.32 %
gapped and paired	7h 19m 22s	9.2 GB	N/A	N/A	85.47 %

Table 5: Benchmark on different data sets on two computers.

3.6 Color space benchmarks

For the SOLiD platform from Applied Biosystems, the original data format does not consist of sequences. A special color system is used to read the sequence information, and in order to get the best analysis results, assembly should be conducted in this so-called *color space*.

Below are the benchmarks of a color space assembly.

The test computer was "computer A" described above.

The data set is provided by Applied Biosystems and it's a human ChIP-seq data set with 82,777,712 reads of 50 bp.

Assembly was done with gapped local alignment against the human genome allowing up to two differences between a read and the reference.

Read length	Reads	Mapped reads	Time
50	82,777,712	33,984,215	10:03:25

Table 6: Color space assembly benchmarks. Time is shown in hours:minutes:seconds.