

CLC NGS Cell

User manual

User manual for CLC NGS Cell 1.0

Windows, Mac OS X and Linux

September 17, 2008

CLC bio
Finlandsgade 10-12
DK-8200 Aarhus N
Denmark



Contents

1	Introduction	6
1.1	Notation	6
1.2	Overview of Commands	6
2	System requirements	8
2.1	Operating system platforms	8
2.2	Supported Intel CPU architectures	8
2.3	Supported AMD CPU architectures	9
2.4	How do I determine my CPU type?	9
2.4.1	CPU info: Windows XP	9
2.4.2	CPU info: Mac OS X	9
2.4.3	CPU info: Linux	10
3	Cas File Format	12
3.1	Sequence Data	12
3.2	Binary Format	12
3.3	Contained Data	12
3.4	Limitations	13
4	Command Line Options	14
4.1	Input Files	14
4.2	Paired Ends	15
4.3	Interleaved Read Files for Paired Ends	16
5	Reference Assembly	17
5.1	Non-specific matches	17

5.2	Placement of Read Pairs	18
5.3	Scoring Schemes	18
5.4	Short Read Reference Assembly	19
5.5	Long Read Reference Assembly	19
6	Printing File Information	21
6.1	The sequence_info Program	21
6.2	The assembly_table Program	21
6.3	The assembly_info Program	23
7	Working with Assemblies	25
7.1	The change_assembly_files Program	25
7.2	The join_assemblies Program	25
7.3	The sub_assembly Program	25
7.3.1	Specifying Assembly Files	26
7.3.2	Extracting a Subset of Reference Sequences	26
7.3.3	Extracting a Part of a Single Reference Sequence	26
7.3.4	Extracting a Subset of Read Sequences	26
7.3.5	Other Match Restrictions	26
7.3.6	Output Reference File	26
7.3.7	Output Read File	27
7.3.8	Non-specific matches	27
7.4	The find_variations Program	27
7.5	The unassembled_reads Program	28
8	Assembly Viewer	29
A	Options for All Programs	33
A.1	Options for clc_ref_assemble_long	33
A.2	Options for clc_ref_assemble_short	34
A.3	Options for assembly_info	36
A.4	Options for assembly_table	36
A.5	Options for change_assembly_files	37
A.6	Options for find_variations	37

A.7 Options for <code>join_assemblies</code>	38
A.8 Options for <code>sequence_info</code>	38
A.9 Options for <code>sub_assembly</code>	39
A.10 Options for <code>unassembled_reads</code>	40
A.11 Options for <code>clc_assembly_viewer</code>	40
Bibliography	41
Index	41

Chapter 1

Introduction

This document describes CLC bio's command line tools for performing sequence assembly and for basic analysis of such assemblies. If more advanced analyses of assemblies are desired, the CLC Genomics Workbench can be used (see <http://www.clcbio.com/genomics>). You can either import assembly files to the Workbench or make the assemblies directly within the Workbench. The Workbench uses the same assembly algorithms as the CLC NGS Cell.

1.1 Notation

We distinguish between *reference assembly* where the target sequences are known and *de novo assembly* where the goal is to find the sequences that the reads came from. Other words for reference assembly used outside this document is alignment and mapping. De novo assembly is sometimes just called assembly, but in this document the general term *assembly* covers both reference assembly and de novo assembly.

To keep notation consistent, the sequences that reads are aligned to are always called reference sequences. This is the case even if the sequences were formed in a de novo assembly process.

1.2 Overview of Commands

The following commands are available for creating assemblies:

clc_ref_assemble_short Short read reference assembly.

clc_ref_assemble_long Long read reference assembly.

Assembly files are in a special format called cas files because the extension is .cas. The following commands are available for analyzing these files as well as sequence files:

sequence_info Print overview of fasta file.

assembly_info Print overview of assembly.

assembly_table Print details of assembly.

Apart from printing the contents of cas files in different ways, it is also possible to perform various operations on them using these commands:

change_assembly_files Change the sequence file names in an assembly file.

join_assemblies Join a number of assemblies to the same reference.

sub_assembly Extract a part of an assembly

find_variations Find the positions where the reads differ from the reference sequences.

unassembled_reads Extract unassembled reads from an assembly.

Chapter 2

System requirements

2.1 Operating system platforms

The system requirements of CLC NGS Cell are these:

- Windows XP or Windows Vista
- Mac OS X 10.3 or newer
- Linux: Redhat or SuSE
- CPU architectures as described below

2.2 Supported Intel CPU architectures

The Cell uses the SSE2 extension of the Intel CPU instruction set. It was introduced in 2001.

Intel uses a number of different CPU microarchitectures with different performance characteristics. The recent ones are:

- The NetBurst microarchitecture:
 - Pentium 4 (670, 661, 660, 651, 650, 641, 640, 631, 630, 551, 541, 531, 524, 521)
 - Pentium D
 - Xeon (7150N, 7140M, 7140N, 7130M 7130N, 7120M, 7120N, 7110M, 7110N, 7041, 7040, 7030, 7020, 5080, 5063, 5060, 5050, 5030)
- The Pentium M microarchitecture:
 - Pentium M (780, 770, 765, 760, 755, 750, 745, 740, 735, 730, 725, 715, 705, 778, 758, 738, 718, 773, 753, 733J, 733, 723, 713)
 - Pentium Core Solo (T1400, T1300, U1500, U1400, U1300)
 - Pentium Core Duo (T2700, T2600, T2500, T2400, T2300, T2300E, L2500, L2400, L2300, U2500, U2400)

- The Core microarchitecture:
 - Pentium Core 2 Duo (E6700, E6600, E6400, E6300, E4300, T7600, T7400, T7200, T5600, T5500, L7400, L7200)
 - Pentium Core 2 Extreme (X6800, QX6700)
 - Xeon (3070, 3060, 3050, 3040, X3220, X3210, X5355, L5320, L5310, E5345, E5335, E5320, E5310, 5160, 5150, 5148 LV, 5140, 5130, 5120, 5110)

As shown, the Pentium Core processors have the Pentium M microarchitecture, while Pentium Core 2 processors have the Pentium Core microarchitecture.

The highest performance per GHz is with the Core microarchitecture while Pentium M has a lower performance and NetBurst is slightly lower.

2.3 Supported AMD CPU architectures

AMD introduced the SSE2 extension in 2003, so recent AMD architectures are supported and their performance is generally a little better than Intel Pentium M but not as high as the Intel Core microarchitecture.

2.4 How do I determine my CPU type?

If you do not know the type of your CPU, use this guide to find out:

2.4.1 CPU info: Windows XP

- Click **Start**
- Right-click **My computer**
- Click **Properties**

You will now see a dialog similar to the one shown in figure 2.1:

The red circle indicates the CPU information. Check with the list of CPU types above to see if your CPU is supported. If the CPU is not in the list, please send an email to support@clcbio.com with the information from this dialog.

2.4.2 CPU info: Mac OS X

- Click the **Apple** at the upper left corner of the screen
- Right-click **About This Mac**

You will now see a dialog similar to the one shown in figure 2.1:

The red circle indicates the CPU information. Check with the list of CPU types above to see if your CPU is supported. If the CPU is not in the list, please send an email to support@clcbio.com with the information from this dialog.

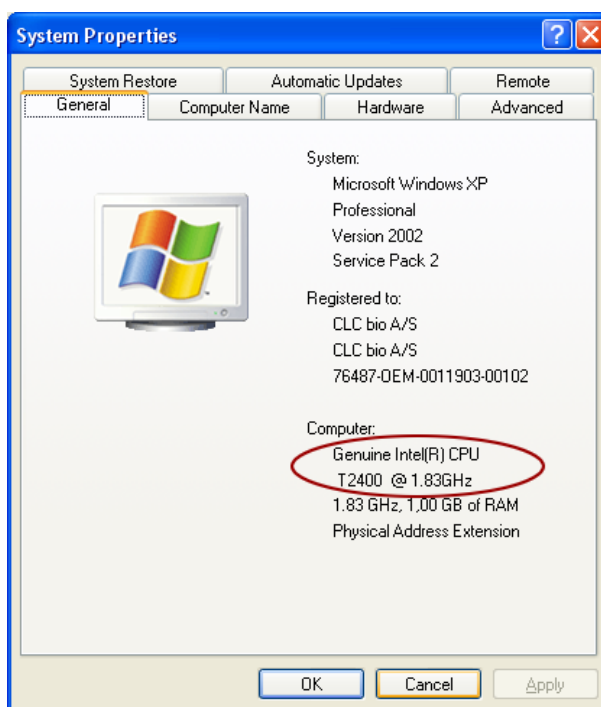


Figure 2.1: Information about CPU on Windows XP.



Figure 2.2: Information about CPU on Mac OS X.

2.4.3 CPU info: Linux

Enter this:

```
cat /proc/cpuinfo
```

You will now see information about your CPU similar to figure 2.3:

Check with the list of CPU types above to see if your CPU is supported. If the CPU is not in the list, please send an email to support@clcbio.com with the information from this dialog.

```
laptop-27:~% cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 14
model name    : Genuine Intel(R) CPU          T2400 @ 1.83GHz
stepping      : 8
cpu MHz       : 1000.000
cache size    : 2048 KB
physical id   : 0
siblings      : 2
core id       : 0
cpu cores     : 2
fdiv_bug     : no
hlt_bug      : no
f00f_bug     : no
coma_bug     : no
fpu          : yes
fpu_exception: yes
cpuid level   : 10
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat clflush dts acpi mmx fxsr
r_sse sse2 ss ht tm pbe nx constant_tsc pni monitor vmx est tm2 xtpr
bogomips     : 3663.65
clflush size : 64

processor       : 1
vendor_id     : GenuineIntel
cpu family    : 6
model         : 14
model name    : Genuine Intel(R) CPU          T2400 @ 1.83GHz
stepping      : 8
cpu MHz       : 1000.000
cache size    : 2048 KB
physical id   : 0
siblings      : 2
core id       : 1
cpu cores     : 2
fdiv_bug     : no
hlt_bug      : no
f00f_bug     : no
coma_bug     : no
fpu          : yes
fpu_exception: yes
cpuid level   : 10
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat clflush dts acpi mmx fxsr
r_sse sse2 ss ht tm pbe nx constant_tsc pni monitor vmx est tm2 xtpr
bogomips     : 3661.30
clflush size : 64

laptop-27:~% █
```

Figure 2.3: Information about CPU on Linux.

Chapter 3

Cas File Format

With CLC bio's command line assembly tools, the cas file format is used. It is a custom file format made with next generation sequencing data in mind (but works fine for any kind of sequencing data). It is not necessary to know everything about this format to use it, but a few basics will help.

3.1 Sequence Data

The most important thing to notice is that cas files do not contain any sequence data. They only contain data about relations between sequences available in other files. Instead of actual sequence data, the cas files contain the names of the corresponding read and reference sequence files. This approach was chosen to save space. There is no reason to keep all the sequences in two places.

3.2 Binary Format

The cas files are in a binary format. Again, the reason for this is to save space. Due to this design, the size of a cas file is only about 8 bytes per read assembled to the human genome. So a cas file with 100 million Solexa reads of length 35 assembled to the human genome is only about 800 MB in size. This is significantly smaller than assemblies in other file formats.

3.3 Contained Data

Cas files contain the following information:

- General info such as: program that made the file, its version and its parameters.
- The file names for the reference sequences.
- The file names for the read sequences.
- Information about the reference sequences: their number, lengths, etc.
- The scoring scheme used when making the file.

- Information about each read:
 - Whether it matches anywhere.
 - Which reference sequence does it match to.
 - Alignment between the reference sequence and the read.
 - The number of places the read matches.
 - Whether the read is part of a paired end pair.

3.4 Limitations

As previously noted, cas files do not contain the actual sequences. This means that you have to be careful to include all the files when sending an assembly to someone. You also have to be careful when moving assembly files, since relative file names may not match any more. The program `change_assembly_files` can be used to change the file names.

There is also a limit of one alignment per read. So a read matching in multiple locations can only have one of these locations described. When assembling short reads to the human genome, some reads may match in over 100,000 locations, so keeping track of all those alignments would be problematic.

Chapter 4

Command Line Options

This chapter describes some general command line options. More specific options are given in the sections for individual programs (chapters 5–8). Finally, appendix A gives details for all the options for all the programs.

4.1 Input Files

The assembly programs support the following input file formats:

Format	Reads	References
Fasta	+	+
Sff (not paired ends)	+	-
GenBank	-	+

The formats are automatically detected.

The ‘-d’ option indicates that the following files contain reference sequences and the ‘-q’ option indicates that the following files contain read sequences. Both of these options may be used repeatedly. For example:

```
clc_ref_assemble_short -o assembly.cas -d human.gb -q reads1.fasta reads2.fasta
                                                                -d mito.gb
```

This command assembles the reads in the files `reads1.fasta` and `reads2.fasta` to the reference sequences in the two files `human.gb` and `mito.gb`. The assembly may be done on one read file at a time and then later joined using the `join_assembly` program.

It is a good idea to include all the reference files in one assembly operation, rather than assembling to different references independently. Consider a reference assembly to the human genome as an example. If reference assembly was performed independently to each chromosome, many reads would not match anything in a given run (because the reads match another chromosome). This results in longer execution time since the reference assembly program then has to look harder for possible matches without any success.

Here, we have three read files, where reads1.fasta and reads3.fasta are unpaired, while reads2.fasta are paired end reads.

4.3 Interleaved Read Files for Paired Ends

In general, paired end data are expected to be in a single file in the form of two sequences from one pair, then two sequences from the next pair, etc. Some sequencing technologies use separate files for the paired reads. In this case, the '-i' option (for interleaved) can be used followed by the two separate files, one with the first reads of the pairs and one with the second reads.

Consider a situation where we have two fasta files like this (first.fasta):

```
>pair_1/1
ACTGTCTAGCTACTGCATTGACTGCGAC
>pair_2/1
TAGCGACGATGCTACTACTCTACTCGAC
>pair_3/1
GATCTCTAGGACTACGCTACGAGCCTCA
```

and this (second.fasta):

```
>pair_1/2
GGATCATCTACGTCATCGACTAGTACAC
>pair_2/2
AAGCGACACCTACTCATCGATCATCAGA
>pair_3/2
TATCGACTCAGACACTCTATACTACCAT
```

where pair_1/1 and pair_1/2 belong together, pair_2/1 and pair_2/2 belong together, etc. The programs expect to see these sequences as one fasta file like this (joint.fasta):

```
>pair_1/1
ACTGTCTAGCTACTGCATTGACTGCGAC
>pair_1/2
GGATCATCTACGTCATCGACTAGTACAC
>pair_2/1
TAGCGACGATGCTACTACTCTACTCGAC
>pair_2/2
AAGCGACACCTACTCATCGATCATCAGA
>pair_3/1
GATCTCTAGGACTACGCTACGAGCCTCA
>pair_3/2
TATCGACTCAGACACTCTATACTACCAT
```

This is accomplished using the '-i' option like this:

```
clc_ref_assemble_short -o assembly.cas -d human.gb -q -p fb ss 180 250
                        -i first.fasta second.fasta
```

This is identical to:

```
clc_ref_assemble_short -o assembly.cas -d human.gb -q -p fb ss 180 250
                        joint.fasta
```

Chapter 5

Reference Assembly

When the reads come from a set of known sequences with relatively few variations, reference assembly is often the right approach to assembling the data. CLC bio offers two programs for reference assembly: `clc_ref_assemble_short` and `clc_ref_assemble_long`, which are for short and long reads, respectively.

The short read program can be used for reads of length 55 and less. For short reads, it is possible to make reference assembly with a guarantee of finding all alignment locations for all the reads, given a certain quality threshold. Such a threshold can for example be to find all reads with at most two mismatches. The short read assembly program works under the assumption that many alignments of reads to the reference sequences are without gaps. By default, gapped alignments are also found, but only after ungapped alignment has been tried. Gapped alignments can be completely turned off for improved speed ('-u' option).

The long read program is used when the requirements of the short read program are not met. For long reads, the alignment quality threshold is given as a certain fraction of the read that must match in a certain fraction of its positions. E.g., the threshold may be set as 50 % of the read must have 90 % identity. The long read assembly program works under the assumption that many alignments have gaps, so gapped alignment is always performed.

By default, reference assembly is done with 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. If the ends of the reads contain vector contamination or adapter sequences, local alignment is also desirable.

Note that the aligned region has to be greater than the length threshold set.

The following sections contain some general information about options for reference assembly. This is followed by sections on specific options for short and long read assembly.

5.1 Non-specific matches

In some cases it may not be possible to uniquely assign a read to a specific optimal position in a reference sequence. This for example happens when a part of a sequence is repeated a number of times among the references. A read that falls entirely within the repeat sequence is impossible to place uniquely. Using longer reads or paired end sequencing alleviates the problem, but if the

repeat is long enough, some reads will still be impossible to place uniquely.

The reference assembly programs allow two options for how to treat these non-specific matches: They can either be randomly placed or not placed at all. This is controlled by the '-r' option which has random placement as default. Since non-specific matches can always be removed later, there is usually little reason to change this option.

Note that it is not possible to record all the positions of the reads since this would sometimes lead to very large amounts of results.

5.2 Placement of Read Pairs

Many sequencing technologies allow paired end sequencing of reads. In such experiments, the reads come in pairs with certain restrictions on their relative placement and orientation.

The approach taken for determining the placement of read pairs is the following:

- First, all the optimal placements for the two individual reads are found.
- Then, the allowed placements according to the paired end options are found.
- If both reads can be placed independently but no pairs satisfies the paired end criteria, the reads are treated as independent and not marked as a pair.
- If only one pair of placements satisfy the criteria, the reads are placed accordingly and marked as uniquely placed even if either read may have multiple optimal placements.
- If several placements satisfy the paired end criteria, the read is treated according to the above described option for ambiguously placed reads. The number of places for the reads are reported as the possible number of placements of the whole pair, not the individual reads.

5.3 Scoring Schemes

For both reference assembly programs, the alignments are scored using Smith Waterman alignment with a linear gap cost. A linear gap cost means that an insertion or deletion of length two costs twice as much as an insertion or deletion of length one. This corresponds to individual insertion and deletion events occurring independently, even if adjacent.

The parameters are:

Parameter	Option	Restrictions
Match score	-	Always 1
Mismatch cost	'-x'	Between 1 and 3. Default is 2
Gap cost	'-g'	Between 1 and 3. Default is 3

It is the relative scores and costs that determine an alignment, so multiplying all the scores by a common factor would give the same alignment. Thus, having the match score fixed to one does not significantly reduce the flexibility in the scoring scheme since the other values can be adjusted. An ambiguous nucleotide aligned to any other nucleotide including the same ambiguous type is treated as a mismatch.

The limitations in the scoring scheme allows more efficient algorithms to be used which is important considering the large data sets being assembled.

5.4 Short Read Reference Assembly

Given a certain quality threshold, it is possible to guarantee that all optimal ungapped alignments are found for each read. Alignments of short reads to reference sequences usually contain no gaps, so the short read assembly operates with a strict scoring threshold to allow the user to specify the amount of errors to accept.

With other short read assembly programs like Maq and Soap, the threshold is specified as 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 5.1 for examples. The default setting is exactly this limit of 8 below the length.

By default, gapped alignment is also allowed with `clc_ref_assemble_short`. Contrary to ungapped alignments, it is very difficult to guarantee that all gapped alignments of a certain quality is found. The scoring limit discussed above applies to both gapped and ungapped alignments and there is a guarantee that there are no ungapped exceeding the limit, but there is no such guarantee for gapped alignments. This being said, the program does a good effort to find the best gapped alignments and usually succeeds.

5.5 Long Read Reference Assembly

For long read assembly, there is no option to perform ungapped alignment because gaps occur easier for longer reads. Because of this, there is no inherent guarantees of finding the optimal alignments according to some scheme. To guarantee finding all optimal alignments, full Smith Waterman alignment would have to be carried out against the whole set of reference sequences. This would take too much computation time to be practical for most data sets.

Instead, a best effort is done to find all the best alignments and this usually succeeds. The quality threshold is determined as a certain fraction of the read matching over a certain identity threshold. The default is that at least half the read must match in at least 90 % of its positions.

CGTATCAATCGATTACGCTATGAATG ATCAATCGATTACGCTATGA	20	CGTATCAATCGATTACGCTATGAATG TTCAATCGATTACGCTATGA	19
CGTATCAATCGATTACGCTATGAATG ATCAATCGGTTACGCTATGA	17	CGTATCAATCGATTACGCTATGAATG TTCAATCGGTTACGCTATGA	16
CGTATCAATCGATTACGCTATGAATG CTCAATCGGTTACGCTATGA	15	CGTATCAATCGATTACGCTATGAATG ATCAACCGGTTACGCTATGA	14
CGTATCAATCGATTACGCTATGAATG TTCAATCGGTTACCCTATGA	13	CGTATCAATCGATTACGCTATGAATG ATCAATCGATTGCGCTCTTT	12
CGTATCAATCGATTACGCTATGAATG TTCAATCGGTTACCCTATGC	12	CGTATCAATCGATTACGCTATGAATG AGCTATCGATTACGCTCTTT	12

Figure 5.1: Examples of ungapped alignments allowed for a 20 bp read with a scoring limit of 8 below the length using the default scoring scheme. The scores are noted to the right of each alignment. For reads this short, a limit of 5 would typically be used instead, allowing up to one mismatch and two unaligned nucleotides in the ends (or no mismatches and five unaligned nucleotides).

Chapter 6

Printing File Information

6.1 The sequence_info Program

The sequence_info program gives some basic information about the sequences in a fasta file:

```
File                data/paired.fasta
Number of sequences          47356
Residue counts:
  Total                  11114027
Sequence length:
  Minimum                170
  Maximum                240
  Average                234.69
```

Using the '-r' options include counts of the different types of nucleotides, with all ambiguous nucleotides counted as N's. The '-a' option used together with the '-r' option does the counts for amino acids.

The lengths of the sequences can be printed or summarized using the '-l' and '-k' options, respectively.

It is also possible to get various sequence length statistics. Using the '-n' option, the N50 value of the sequences is calculated, assuming that the sequences cover a whole genome without any overlaps.

Use the '-c' option to disregard all sequences under a certain length from being considered in the statistics.

6.2 The assembly_table Program

The assembly_table program takes a single cas file as input and prints assembly information for each read. By default, assembly_table makes a table with one read per row. The columns are:

- Read number (starting from 0).

- Read name (enable using the '-n' option).
- Read length.
- Read position for alignment start.
- Read position for alignment end.
- Reference sequence number (starting from 0).
- Reference position for alignment start.
- Reference position for alignment end.
- Whether the read is reversed (0 = no, 1 = yes).
- Number of optimal locations for the read.
- Alignment score (enable using the '-s' option).

If a read does not match, all columns except the read number and name are '-1'. If a read is reverse, the read positions for the alignment start and end are given after the reversal of the read. The sequence positions start from 0 indicating before the first residue and end at the sequence length indicating after the last residue. So a read of length 35 which matches perfectly will have an alignment start position of 0 and an alignment end position of 35.

Here is part of an example output using both the '-n' and the '-s' option:

```

208      SLXA-EAS1_89:1:1:622:715/1      35  0  35  0      89385      89420  0  1  35
209      SLXA-EAS1_89:1:1:622:715/2      35  0  35  0      89577      89612  1  1  35
210      SLXA-EAS1_89:1:1:201:524/1      35  0  32  0      4829       4861  0  1  29
211      SLXA-EAS1_89:1:1:201:524/2      -1 -1 -1 -1      -1         -1 -1 -1 -1
212      SLXA-EAS1_89:1:1:662:721/1      35  0  35  0      38254      38289  1  1  35
213      SLXA-EAS1_89:1:1:662:721/2      35  0  35  0      38088      38123  0  1  32
214      SLXA-EAS1_89:1:1:492:826/1      35  0  35  0      81872      81907  1  1  35
215      SLXA-EAS1_89:1:1:492:826/2      35  0  35  0      81685      81720  0  1  35

```

As the read names indicate, the data are from a paired end experiment. Read 211 does not match at all and only the first 32 out of the 35 positions in read 210 matches. The score for this read is 29, indicating that a mismatch is also present ($31 - 2 = 29$). Read 213 also has a mismatch while the rest of the sequences match perfectly. We can also see that the pairs are located close together and on opposite strands.

Use the '-a' option to get a very detailed output ('-n' and '-s' are without effect here):

SLXA-EAS1_89:1:1:622:715/1 has 1 match with a score of 35:

```

      89385 TTGCTGTGGAAAATAGTGAGTCATTTTAAAACGGT 89419      coli
      |||
      TTGCTGTGGAAAATAGTGAGTCATTTTAAAACGGT      read

```

SLXA-EAS1_89:1:1:622:715/2 has 1 match with a score of 35:

```

      89577 AAACCTCTTTCAGTGGGAAATTGTGGGGCAAAGTG 89611      coli
      |||
      AAACCTCTTTCAGTGGGAAATTGTGGGGCAAAGTG      reverse read

```

SLXA-EAS1_89:1:1:201:524/1 has 1 match with a score of 29:

```

4829 ATCCAGGCGAATATGGCTTGTTCCCTCGGCACC 4860 coli
      |||
      ATCCAGGCGAATATGGCTTTTTTCCTCGGCACCCCG read

```

SLXA-EAS1_89:1:1:201:524/2 has 0 matches

SLXA-EAS1_89:1:1:662:721/1 has 1 match with a score of 35:

```

38254 AGGGCATTTCGATACGGTGGATAAGCTGAGTGCCTT 38288 coli
      |||
      AGGGCATTTCGATACGGTGGATAAGCTGAGTGCCTT reverse read

```

SLXA-EAS1_89:1:1:662:721/2 has 1 match with a score of 32:

```

38088 ACTGAGTGATTGATTTCGCGAGCCACATACTGTGGA 38122 coli
      |||
      ACTGAGTGATTGATTTCGCGAGCCACATACTCTGGA read

```

SLXA-EAS1_89:1:1:492:826/1 has 1 match with a score of 35:

```

81872 GCATCCAGCACTTTCAGCGCCTGGGTCATCACTTC 81906 coli
      |||
      GCATCCAGCACTTTCAGCGCCTGGGTCATCACTTC reverse read

```

SLXA-EAS1_89:1:1:492:826/2 has 1 match with a score of 35:

```

81685 TTCIGGTTGCTGGTCTGGTGGTAAATGTTCCCACT 81719 coli
      |||
      TTCIGGTTGCTGGTCTGGTGGTAAATGTTCCCACT read

```

6.3 The assembly_info Program

Whereas `assembly_table` outputs detailed information about individual matches, the `assembly_info` program instead gives an overview:

General info:

```

Program name      clc_ref_assemble_short
Program version   1.00.31043
Program parameters -o tmp.cas -d data/paired.fasta -q data/paired_reads.fasta

```

Contig files:

```
data/paired.fasta
```

Read files:

```
data/paired_reads.fasta
```

Read info:

```

Contigs          1
Reads            108420
  Unassembled reads 1506
  Assembled reads  106914
  Multi hit reads  0

```

Alignment info:

Number of inserts	13
Number of deletes	42
Number of mismatches	9253

Coverage info:

Total sites	100000
Average coverage	37.29
Sites covered 0 times	0
Sites covered 1 time	0
Sites covered 2 times	3
Sites covered 3+ times	99997

Contig info:

Contig	Sites	Reads	Coverage
1	100000	106914	37.29

It is possible to make an analysis of paired end distances using the `assembly_info` program. This is done with the standard `'-p'` option and results in output like this:

Paired end info:

Pairs	2478655
Average distance	215.44
99.9 % of pairs between	175 - 253
99.0 % of pairs between	191 - 241
95.0 % of pairs between	197 - 234
Not pairs	143727
Both seqs not matching	21946
One seq not mathing	62938
Both seqs matching	58843
Different contigs	0
Wrong directions	40524
Too close	663
Too far	17656

Note that for paired end analysis `assembly_info` assumes that read one pairs with read two, read three with read four, etc. Thus, it is crucial that the reads are from a paired end experiment and that they are assembled in the right order, possibly using the `interleaved` option for creating the assembly. If an assembly has a mixture of paired and unpaired data, use `sub_assembly` to make an assembly with only the paired end data before analyzing.

When a data set contains paired end data of unknown distances, a good approach is to make an initial reference assembly without using paired end information. Then the `assembly_info` program can be used to investigate the paired end distance properties of the data using wide limits for the distances. Finally, a reference assembly run can be performed with the estimated paired end distances at a suitable distance interval. To get a quicker result, the initial reference assembly run may be done on only a part of the data, using ungapped alignments, and/or using stricter scoring criteria. These factors will usually not affect the paired end distance properties of the results, but a smaller fraction of the reads might match.

Chapter 7

Working with Assemblies

7.1 The `change_assembly_files` Program

This program allows you to change the file names in an assembly file. It is useful if you have moved the sequence files after the assembly was made. Or if you for example made the assembly with relative file names and want to change the file names to absolute names (or vice versa). It is also possible to change the file format, for example from fasta to GenBank format if you wish a richer representation of the sequence. For the operation to be a success, however, the actual sequences and their order must remain unchanged.

With the `change_assembly_files` program, file names are specified like they are when making the original reference assembly, i.e. using the `'-d'`, `'-q'`, and `'-i'` options. The output assembly file is specified with the `'-o'` option and the input assembly file is specified with the `'-a'` option.

To make the change in place, use the same assembly file name for input and for output. It is of course slightly safer to use different file names, so a backup of the original is kept.

By default, the program compares the sequence files to make sure they contain the same data. This takes some time, so the `'-n'` option is included to avoid this check. The `'-n'` option is also useful if the old sequence files does not exist any more.

7.2 The `join_assemblies` Program

Using this program, it is possible to join two or more `cas` assembly files into one. It is sometimes convenient to perform reference assemblies on different sets of reads as independent runs. These runs can then be joined later with the `join_assemblies` program. It is a requirement that the assemblies have exactly the same reference sequence files in the same order to join them.

7.3 The `sub_assembly` Program

The `sub_assembly` program allows the user to make a new assembly containing only part of the original assembly.

7.3.1 Specifying Assembly Files

The ‘-a’ options specifies the input assembly and the ‘-o’ option specifies the output assembly.

7.3.2 Extracting a Subset of Reference Sequences

The ‘-s’ option is used for making a new assembly with only matches to a single reference sequence. The ‘-d’ option makes a new assembly with only matches to the reference sequences of a single file. The sequence or file must be specified as its number in the list of reference sequences or files in the input assembly. You can use `assembly_info` to see the contents of the input assembly is needed.

These options are useful when working with a large assembly such as the human genome. Extracting sub assemblies for each chromosome may make it easier to work with.

7.3.3 Extracting a Part of a Single Reference Sequence

If a single reference sequence is specifies using the ‘-s’ option or if the input assembly contains only a single reference sequence, the ‘-b’ option may be used to specify a position range to extract. The output assembly will then only contain matches to this specific region. If a match is partially located in the region, only the part of the match inside the region is kept.

This option is useful for studying a particular section of a long reference sequence. It could for example be a single gene in the whole human genome.

7.3.4 Extracting a Subset of Read Sequences

Using the ‘-q’ option, you can make an assembly with only the reads from one of the read files. The read file is specified by its number in the input assembly. If reads are interleaved, the output assembly will refer to the two interleaved files instead of just one file.

This is for example useful if you wish to study how the reads from a particular experiment behaved even is the full assembly contains reads from several experiments.

7.3.5 Other Match Restrictions

The ‘-u’ option ensures that only uniquely placed matches are kept. The ‘-l’ option specifies a minimum length of a read sequence that must be part of its match alignment for it to be kept. Mismatches within the alignment does not affect the length measurement.

7.3.6 Output Reference File

By default, the output assembly refers to one or all of the reference files in the input assembly. It refers to just one of the files when it has been selected using the ‘-d’ option or when a single reference sequence has been selected with the ‘-s’ option.

If the ‘-g’ option is used, an output file is made with only the reference sequences of the output assembly. The new assembly automatically refers to this reference sequence file. This is typically useful when selecting only a single reference sequence and the input alignment contains may reference sequences in the same file. That way the output assembly only contains the relevant

reference sequence instead of many references with no matches. It makes the output assembly easier and faster to work with.

If a position range was specified, the output reference file only contains these positions.

7.3.7 Output Read File

By default, the output assembly refers to one or all of the read files in the input assembly. It refers to just one of the files when it has been selected using the '-q' option.

Using the '-f' option, a new read file is made instead containing only the reads that match. The output assembly automatically refers to this new read file instead of the originals.

This is very useful when making a sub assembly that only covers a small part of the original reference sequences. That way a much smaller number of reads come into play when working with the sub assembly, making subsequent analyses more efficient.

When the reads are from a paired end experiment, the assembly analysis programs expect read one to pair with read two, read three to pair with read four, etc. If one read out of a pair is removed with the sub_assembly program, the paired end read order is disrupted. Because of this, the '-p' option should be used when the reads are from a paired end experiment. It works by retaining reads that do not match the sub_assembly criteria if the counterpart does match the criteria. Without the '-p' option, the read file will contain no unassembled reads, but with this option some reads may be unassembled because the other member of their pair is part of the assembly.

7.3.8 Non-specific matches

If an assembly contains non-specific matches reads and a sub assembly is made from it, the non-specific matches will still be marked as such even if there is only a single place they match in the chosen subset of the reference sequences. The reason for this is that the sub_assembly program is meant to make it simpler to study a small region of a large assembly, so the original characteristics of the larger assembly are kept.

7.4 The find_variations Program

This program makes a new reference sequence file containing all the original data but with changes made so the references reflect the read sequences of an assembly. The new reference file is always in fasta format. It is also possible to run the program so it only prints a list of differences instead of actually making a new file.

The find_variations program is used after reference assembly to get an estimate of the actual sequences that was studied in the sequencing experiment.

If you wish to see the reads matched to the new reference sequences, a new round of reference assembly has to be performed. The reason for this is that the changes to the references may significantly change the optimal locations of the reads in the changed regions. So a complete new reference assembly is necessary. Sometimes the new read alignments may suggest a few more changes to the reference sequences, so another run of find_variations may be in order.

7.5 The `unassembled_reads` Program

This program extracts the unassembled read sequences from an assembly. They are output in fasta file. By default the only output sequences are the ones that does not match at all. Using the options it is also possible to output the unaligned ends of reads. A minimum length of unassembled sequence can also be specified.

This program is useful for investigating the sequences that were not part of the expected reference sequences used in a previous assembly.

Chapter 8

Assembly Viewer

The assembly viewer program shows assemblies in a text based terminal window. It is useful for getting a quick overview of the data and for investigating interesting places.

The program takes one or more assembly files as parameters. For large assemblies, it may take a little while to start since the reads have to be sorted for viewing. The key bindings are as follows:

Key	Description
Arrows	Move view.
0-9	Any (possibly multi digit) number followed by any other key: move to that position. Follow by 'K' to multiply by 1,000, or 'M' to multiply by a million.
Z	Center vertical position on reads.
V	Scroll left and center horizontally.
B	Scroll right and center horizontally.
C	Toggle color scheme.
M	Toggle position marks.
E	Toggle how to show unaligned ends.
R	Toggle between reference sequences.
P	Move to same position as for last contig.
H	Show help screen.
Q	Quit.

Using shift together with one of the toggle keys ('C', 'E', 'R' and 'M') cycles the other direction. Using shift with one of the movement keys (including arrows) makes the movement faster. This also applies to the 'K' and 'M' keys for sequence positions. Figures 8.1-8.4 show some screen shots and examples.

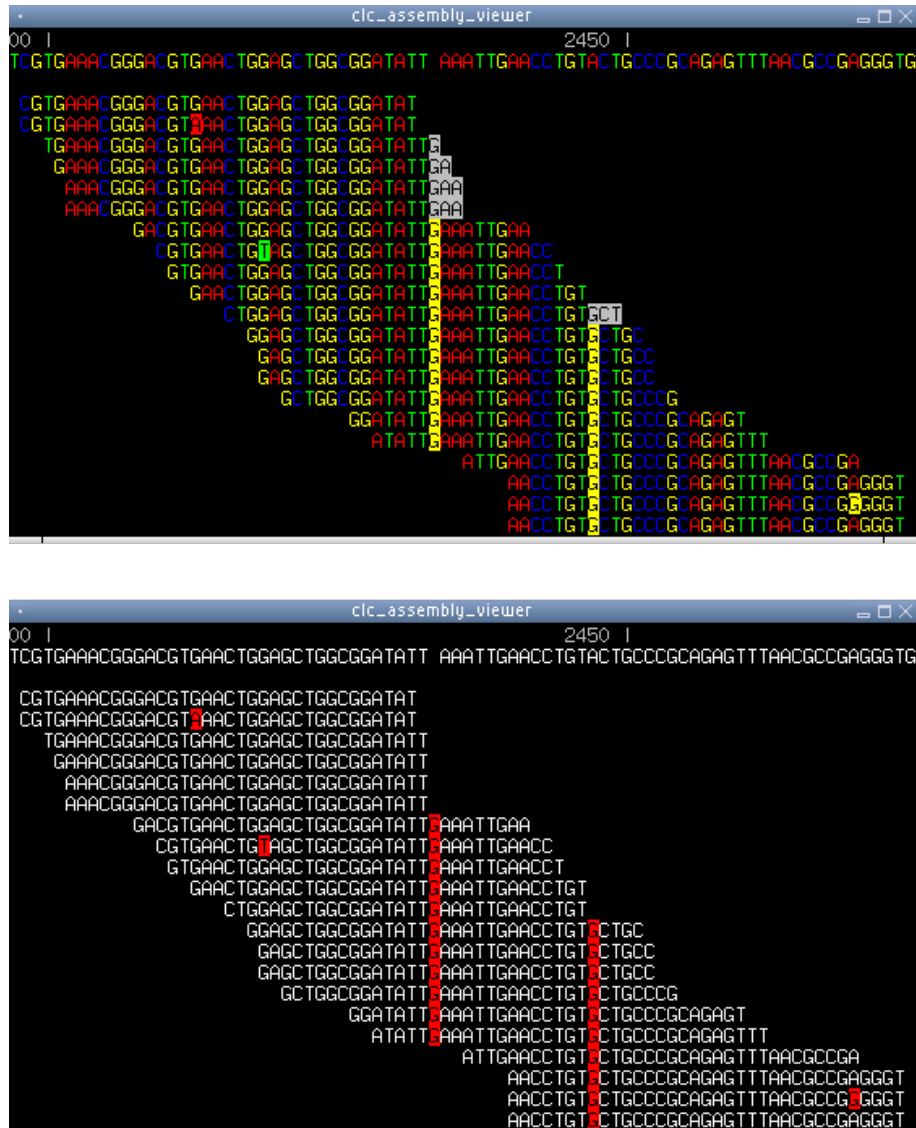


Figure 8.1: Two screen shots from the assembly viewer. Top) Residue coloring. Residues differing from the reference are highlighted. The first column of highlighted G's is an insertion, the second is a mutation (the reference residue is A in that position). The reversed gray residues at the end of some of the reads are not aligned. Bottom) Another color scheme, where differences are easier to spot. Here the unaligned residues have also been turned off.

The figure consists of two screenshots of the 'clic_assembly_viewer' application. The top screenshot shows a window with a title bar 'clic_assembly_viewer' and a status bar '2450 |'. The main content area displays a DNA sequence alignment. The top line is 'TCGTGAAACGGGACGTGAAC TGGAGCTGGCGGATATT AAATTGAACCTGTAC TGCCCGCAGAGTTTAAACGCCGAGGGTG'. Below it, several reads are shown, some in green and some in red. The green reads are: 'CGTGAACCGGGACGTGAAC TGGAGCTGGCGGATAT', 'GAAACCGGGACGTGAAC TGGAGCTGGCGGATATTGA', 'AAACCGGGACGTGAAC TGGAGCTGGCGGATATTGAA', and 'AAACCGGGACGTGAAC TGGAGCTGGCGGATATTGAA'. The red reads are: 'GACGTGAAC TGGAGCTGGCGGATATTGAAATTGAA', 'CGTGAACCTGAGCTGGCGGATATTGAAATTGAACC', 'GTGAAC TGGAGCTGGCGGATATTGAAATTGAACCT', 'GAAC TGGAGCTGGCGGATATTGAAATTGAACCTGT', 'CTGGAGCTGGCGGATATTGAAATTGAACCTGTGCT', 'GGAGCTGGCGGATATTGAAATTGAACCTGTGCTGC', 'GAGCTGGCGGATATTGAAATTGAACCTGTGCTGCC', 'GAGCTGGCGGATATTGAAATTGAACCTGTGCTGCC', and 'GCTGGCGGATATTGAAATTGAACCTGTGCTGCCCG'. The bottom screenshot shows a window with a title bar 'clic_assembly_viewer' and a status bar '66750 | 66800 |'. The main content area displays a DNA sequence alignment. The top line is 'GGCGTTGAATGCCGGATGCGCTTTGCTTATCCGGCC TACAAAA TCGCAGCGTG TAGGCCAGATAAGACCGGTCAGCGTCGG'. Below it, several reads are shown, some in blue and some in red. The blue reads are: 'ATGCCGGATGCGCTTTGCTTATCCGGCC TACAAAA', 'ATGCCGGATGCGCTTTGCTTATCCGGCC TACAAAA', 'GCCGGATGCGCTTTGCTTATCCGGCC TACAAAAATC', and 'CGGATGCGCTTTGCTTATCCGGCC TACAAAA TCGC'. The red reads are: 'TTTGGCTTATCCGGCC TACAAAA TCGCAGCGGTAG', 'TTTGGCTTATCCGGCC TACAAAA TCGCAGCGGTAG', 'TTTGGCTTATCCGGCC TACAAAA TCGCAGCGGTAG', 'TTGGCTTATCCGGCC TACAAAA TCGCAGCGGTAGG', 'GCTTATCCGGCC TACAAAA TCGCAGCGGTAGGCC', 'CTTATCCGGCC TACAAAA TCGCAGCGGTAGGCCA', 'CTTATCCGGCC TACAAAA TCGCAGCGGTAGGCCA', 'CTTATCCGGCC TACAAAA TCGCAGCGGTAGGCCA', 'TTATCCGGCC TACAAAA TCGCAGCGGTAGGCCAG', 'TCCGGCC TACAAAA TCGCAGCGGTAGGCCAGATA', 'GCC TACAAAA TCGCAGCGGTAGGCCAGATAAGAC', 'CCTACAAAA TCGCAGCGGTAGGCCAGATAAGACG', 'AAATCGCAGCGGTAGGCCAGATAAGACCGGTCAG', 'AATCGCAGCGGTAGGCCAGATAAGACCGGTCAGC', 'AATCTCAGCGGTAGGCCAGATAAGACCGGTCAGC', 'ATCGCAGCGGTAGGCCAGATAAGACCGGTCAGCG', and 'ATCGCAGCGGTAGGCCAGATAAGACCGGTCAGCG'.

Figure 8.2: Another screen shot from the assembly viewer. Here, the color scheme is according to the direction of the reads. Green is forward, red is reverse.



Figure 8.3: A screen shot with 454 sequencing data. The directional color scheme is useful for recognizing a particular type of sequencing error with the 454 technology. Notice the position with five inserted G's. They are sequencing errors arising from the stretch of five G's to their left, before the C. These errors tend to occur before a stretch of identical residues, which is why they are only seen in the reverse reads in this case.

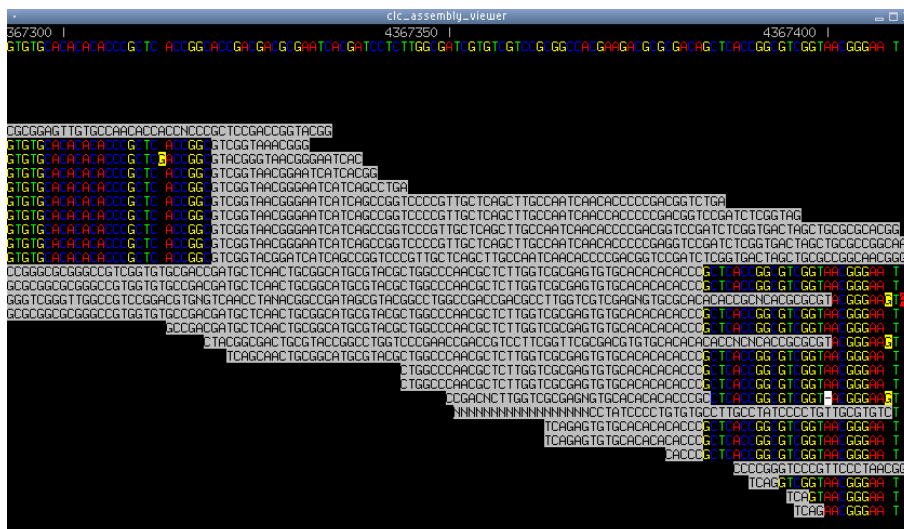


Figure 8.4: A screen shot with 454 sequencing data. This is how a genomic rearrangement looks in a reference assembly. Suddenly the reads do not match any more, and later another set of reads abruptly start matching. These reads may actually be very distant in the real genome (as opposed to the reference).

Appendix A

Options for All Programs

A.1 Options for `clc_ref_assemble_long`

usage: `clc_ref_assemble_long` <options>

Reference assemble some reads to some reference sequences. Mostly used for reads longer than 55 or of varying differences.

Options:

`-h / --help`: Display this message

`-q / --reads`: The files following this option are read files. Fasta, and sff formats are allowed

`-d / --reference`: The files following this option are reference files. Fasta and GenBank formats are allowed. (may be used several times)

`-o <file> / --output <file>`: Give the output assembly file (required)

`-i <file1> <file2> / --interleave <file1> <file2>`: Interleave the sequences in two files, alternating between the files when reading the sequences. Only valid for read files. (may be used several times)

`-x <n> / --mismatchcost <n>`: Set the mismatch cost (range 1 to 3, default 2)

`-g <n> / --gapcost <n>`: Set the gap cost (range 1 to 3, default 3)

`-e <n> / --deletioncost <n>`: Set the deletion cost in which case the gap cost setting only applies to insertions. (range 1 to 3, default 3)

`-r <mode> / --repeat <mode>`: Set the behavior for reads that match more than once, i.e. ignore such reads or place them randomly among the valid locations (ignore / random) (default random)

`-l <n> / --lengthfraction <n>`: Set the fraction of the read that must match. A real number between 0.0 and 1.0 (default 0.5).

`-s <n> / --similarity <n>`: Set the limit for the similarity in the fraction of the read that must match (according to "-l" option). A real number between 0.0 and 1.0 (default 0.8).

`-p <par> / --paired <par>`: Set the paired read mode for the read files

following this option. (may be used several times)

par consists of four strings: <mode> <dist_mode> <min_dist> <max_dist>

mode is ff, fb, bf, bb and sets the relative orientation of read one and two in a pair (f = forward, b = backward)

dist_mode is ss, se, es, ee and sets the place on read one and two to measure the distance (s = start, e = end)

A typical use would be "-p fb ss 180 250" which means that the reads are inverted and pointing towards each other. The distance includes both the reads and the sequence between them. The distance may be between 180 and 250, both included.

To explicitly say that the following reads are not paired, use "no" for par, i.e. "-p no".

For paired end reads split in two files, use the -i option.

-m <n> / --memory <n>: Set the maximum amount of memory to use as a fraction of the available memory (default is 1.0).

Examples:

Reference assemble a single file with reads to a single file with reference sequences:

```
clc_ref_assemble_long -o assembly.cas -q reads.fasta -d reference.fasta
```

Reference assemble reads from two unpaired runs and a paired end run split across two files. Use two reference sequences:

```
clc_ref_assemble_long -o assembly.cas -q unpaired1.fasta unpaired2.fasta
-p fb ss 180 250 -i paired_1.qf paired_2.qf -d
reference1.gb reference2.gb
```

Version: 1.00.32891

A.2 Options for `clc_ref_assemble_short`

usage: `clc_ref_assemble_short` <options>

Reference assemble some reads to some reference sequences. Maximum read length is 55.

Options:

-h / --help: Display this message

-q / --reads: The files following this option are read files. Fasta and sff formats are allowed

-d / --reference: The files following this option are reference files. Fasta and GenBank formats are allowed. (may be used several times)

-o <file> / --output <file>: Give the output assembly file (required)

-i <file1> <file2> / --interleave <file1> <file2>: Interleave the sequences in two files, alternating between the files when reading the sequences. Only valid for read files. (may be used several times)

-x <n> / --mismatchcost <n>: Set the mismatch cost (range 1 to 3, default 2)

-g <n> / --gapcost <n>: Set the gap cost (range 1 to 3, default 3)

-e <n> / --deletioncost <n>: Set the deletion cost in which case the gap cost setting only applies to insertions. (range 1 to 3, default 3)

-u / --ungapped: Used ungapped alignment (default is gapped alignment)

-r <mode> / --repeat <mode>: Set the behavior for reads that match more than once, i.e. ignore such reads or place them randomly among the valid locations (ignore / random) (default random)

-s <n> / --scorelimit <n>: Set the limit for the score. The limit is defined as the number of points below the read length to accept (default is 8 for default scoring scheme).

-p <par> / --paired <par>: Set the paired read mode for the read files following this option. (may be used several times)

par consists of four strings: <mode> <dist_mode> <min_dist> <max_dist>

mode is ff, fb, bf, bb and sets the relative orientation of read one and two in a pair (f = forward, b = backward)

dist_mode is ss, se, es, ee and sets the place on read one and two to measure the distance (s = start, e = end)

A typical use would be "-p fb ss 180 250" which means that the reads are inverted and pointing towards each other. The distance includes both the reads and the sequence between them. The distance may be between 180 and 250, both included.

To explicitly say that the following reads are not paired, use "no" for par, i.e. "-p no".

For paired end reads split in two files, use the -i option.

-m <n> / --memory <n>: Set the maximum amount of memory to use as a fraction of the available memory (default is 1.0).

Examples:

Reference assembly a single file with reads to a single file with reference sequences:

```
clc_ref_assemble_short -o assembly.cas -q reads.fasta -d reference.fasta
```

Reference assemble reads from two unpaired runs and a paired end run split across two files. Use two reference sequences:

```
clc_ref_assemble_short -o assembly.cas -q unpaired1.fasta unpaired2.fasta
-p fb ss 180 250 -i paired_1.qf paired_2.qf -d
reference1.gb reference2.gb
```

A.3 Options for assembly_info

usage: assembly_info <assembly file>

Print information about an assembly.

Options:

-h / --help: Display this message.

-c / --coverage: Show more detailed coverage information

-d <file> / --coveragefile <file>: Output coverage as a function of position for each reference sequence to different files called <file>.001.dat, <file>.002.dat, etc.

-p <par> / --paired <par>: Set the paired read mode.

par consists of four strings: <mode> <dist_mode> <min_dist> <max_dist>

mode is ff, fb, bf, bb and sets the relative orientation of read one and two in a pair (f = forward, b = backward).

dist_mode is ss, se, es, ee and sets the place on read one and two to measure the distance (s = start, e = end).

A typical use would be "-p fb ss 180 250" which means that the reads are inverted and pointing towards each other. The distance includes both the reads and the sequence between them. The distance may be between 180 and 250, both included.

Only read pairs satisfying these criteria are counted in the distance statistics.

-q <file> / --pairedfile <file>: Output file for distance histogram for paired end data.

-f / --fast: No coverage information for a fast result.

Version: 1.00.32891

A.4 Options for assembly_table

usage: assembly_table <assembly file>

Print information about each match in an assembly file. The columns are:

Read number
Read name (enable using the '-n' option)
Read length
Read position for alignment start
Read position for alignment end
Reference sequence number

Reference position for alignment start
Reference position for alignment end
Whether the read is reversed (0 = no, 1 = yes)
Number of matches
Whether the read is paired with the next one (0 = no, 1 = yes) (enable
using the '-p' option)
Alignment score (enable using the '-s' option)

Options:

-h / --help: Display this message.
-n / --names: Include the read names.
-s / --scores: Include the alignment scores.
-p / --paired: Include pair information.
-a / --alignments: Print the full alignments, including names and scores.

Version: 1.00.32891

A.5 Options for change_assembly_files

usage: change_assembly_files <options>

Change the sequence file names in an assembly file. Can be used for the reference files, the read files, or both.

Options:

-h / --help: Display this message.
-a <file> / --assembly <file>: Give the assembly file (required).
-o <file> / --output <file>: Give the output assembly file (required).
-q / --reads: The files following this option are read files. Fasta and sff formats are allowed
-d / --reference: The files following this option are reference files. Fasta and GenBank formats are allowed (may be used several times).
-i <file1> <file2> / --interleave <file1> <file2>: Interleave the sequences in two files, alternating between the files when reading the sequences. Only valid for read files (may be used several times).
-n / --nocheck: Do not check if the sequence files match. This is useful if the old files do not exist any more, or to get a fast result if the files are known to match.

Version: 1.00.32891

A.6 Options for find_variations

usage: find_variations <options>

Find positions where the reads indicate a consistent difference from the reference sequences. Optionally, updated versions of the reference sequences can be output to a fasta file.

Options:

-h / --help: Display this message

-a <file> / --assembly <file>: Specify the assembly file (required).

-o <file> / --output <file>: Specify the output fasta file.

Version: 1.00.32891

A.7 Options for join_assemblies

usage: join_assemblies <options> <input assembly 1> <input assembly 2> ...

Joins any number of assemblies with identical reference files into one.

Options:

-h / --help: Display this message.

-o <file> / --output <file>: Set the output assembly file (required).

Version: 1.00.32891

A.8 Options for sequence_info

usage: sequence_info [options] <sequence file>

Print some information about a sequence file. The accepted formats are fasta, sff, and GenBank.

Options:

-h / --help: Display this help

-l / --lengths: Print length of each sequence

-k / --lengthcounts: Print number of sequences of each length

-n / --n50: Calculate the N50 value

-c <n> / --cutoff <n>: Ignore all sequences below a minimum sequence length

-r / --residues: Include residue counts

-a / --aminoacids: Residue are amino acids (only relevant when including residue counts)

Version: 1.00.32891

A.9 Options for sub_assembly

usage: sub_assembly <options>

Extract part of an assembly into a new assembly file.

Options:

- h / --help: Display this message
- a <file> / --assembly <file>: Set the input assembly file (required).
- o <file> / --output <file>: Set the output assembly file (required).
- d <n> / --reffile <n>: Restrict matches to a single reference file denoted by its number.
- s <n> / --refseq <n>: Restrict matches to a single reference sequence denoted by its number.
- q <n> / --readfile <n>: Restrict matches to a single read file (or two if interlaced) denoted by its number.
- b <m-n> / --subsequence <m-n>: Restrict matches to a position range. The positions start from 1. The '-s' option must also be specified if more than one reference sequence is present in the assembly.
- u / --unique: Restrict to uniquely placed matches.
- l <n> / --minlength <n>: Restrict to matches where a minimum of n read positions are aligned (but not necessarily matching).
- f <file> / --readoutput <file>: Output file for reads. Only matching reads are output. With this option the output assembly refers to this read file instead of the original read files.
- g <file> / --refoutput <file>: Output file for references. With this option the output assembly refers to this reference file instead of the original reference files.
- p / --paired: Keep read pairs together when making read output file. Should be used when the reads are from a paired end experiment. May only be used with the '-f' option.

Examples:

Make an assembly containing only reference sequence two of an existing assembly. Also make a new file for the reads matching this sequence:

```
sub_assembly -a assembly.cas -o new.cas -s 2 -f new_reads.fasta
```

The same but only the first 100,000 positions of the reference sequence and also make a file for the new partial reference sequence

```
sub_assembly -a assembly.cas -o new.cas -s 2 -b 1-100000 -f new_reads.fasta  
-g new_ref.fasta
```

Make an assembly without ambiguously placed reads:

```
sub_assembly -a assembly.cas -o new.cas -u
```

Version: 1.00.32891

A.10 Options for unassembled_reads

usage: unassembled_reads <options>

Make a fasta file with the unassembled reads from an assembly.

Options:

-h / --help: Display this message.

-a <file> / --assembly <file>: Specify the assembly file (required).

-o <file> / --output <file>: Specify the output fasta file (required).

-l <n> / --minlength <n>: Output only sequences with a certain minimum length.

-u / --unaligned: For matching reads with sufficiently long unaligned parts, output these parts as individual sequences. Two parts may be output if both ends are long enough. Must be used with the '-l' option.

Example:

Make a fasta file with all the unassembled reads along with all read parts that were unaligned and has a length of at least 100 bp:

```
unassembled_reads -a assembly.cas -o unassembled.fasta -l 100 -u
```

Version: 1.00.32891

A.11 Options for clc_assembly_viewer

usage: clc_assembly_viewer <assembly files>

Show a number of assemblies in a text viewer. Type H to show an overview of the key bindings.

Version: 1.10.32631

Bibliography

Index

AMD architectures, system requirements, 9

Bibliography, 41

Intel architectures, 8

Linux, 8

Mac OS X, 8

NetBurst microarchitecture, 8

Pentium, system requirements, 8

Platforms supported, 8

References, 41

Supported CPU architectures, 8

System requirements, 8

Windows, 8