

CLC Assembly Cell

User manual

User manual for CLC Assembly Cell 4.2

Windows, Mac OS X and Linux

October 9, 2013

This software is for research purposes only.

CLC bio
Silkeborgvej 2
Prismet
DK-8000 Aarhus C
Denmark



Contents

1	Introduction	8
1.1	Overview of Commands	8
1.1.1	Core analysis tools	8
1.1.2	Viewing and reporting tools overview	9
1.1.3	Assembly post-processing tools	9
1.1.4	Sequence preparation tools	9
1.1.5	Format conversion	10
2	System Requirements and Installation	11
2.1	System requirements	11
2.1.1	Limitations on maximum number of cores	12
2.1.2	Supported CPU architectures	12
2.2	Disk space	12
2.3	Downloading and installing the software	12
2.4	Installing Static license	13
2.4.1	Licensing the software on a networked Linux or Mac machine	13
2.4.2	Licensing the software on a networked Windows machine	13
2.4.3	Licensing the software on a non-networked machine	14
2.5	Network Licenses	14
2.5.1	Installing and Running CLC License Server	14
2.5.2	Configuring the software to use a network license	15
2.6	Restricting CPU usage	15
3	The Basics	16
3.1	How to use the programs	16

3.1.1	Getting Help	16
3.1.2	A basic example	17
3.2	Input Files	17
3.3	Cas File Format	17
3.3.1	Cas Format Basics	18
3.3.2	What a cas file contains	18
3.3.3	What a cas file does not contain	18
3.3.4	Considerations and limitations	18
3.3.5	Converting to and from SAM and BAM formats	19
3.4	Paired read Considerations	19
3.4.1	Relative orientation of the reads	20
3.4.2	Measuring the distance between the reads	20
3.4.3	Paired Read File Input	20
4	Read Mapping	22
4.1	Overview of base space mapping	23
4.2	Circular references	23
4.3	Saving and re-using reference index files	24
4.4	Overview of color space mapping	24
4.4.1	Sequencing	25
4.4.2	Error modes	25
4.4.3	Mapping in color space	26
4.4.4	Color space file formats	28
4.5	General information for both read mappers	30
4.5.1	Non-specific matches	30
4.5.2	Placement of Read Pairs	30
4.5.3	Scoring Schemes	30
4.5.4	Mapping quality thresholds	31
4.6	Running Read Mapping Analyses	31
4.7	Mixed base space and color space mappings	32
5	De novo assembly	33
5.0.1	De novo assembly inputs	33

5.0.2	De novo assembly outputs	33
5.1	How it works	34
5.1.1	Resolve repeats using reads	36
	Remove weak edges	37
	Remove dead ends	37
	Resolve repeats without conflicts	37
	Resolve repeats with conflicts	39
5.1.2	Automatic paired distance estimation	39
5.1.3	Optimization of the graph using paired reads	41
5.1.4	AGP export	43
5.1.5	Bubble resolution	43
5.1.6	Converting the graph to contig sequences	45
5.1.7	Summary	45
5.2	Randomness in the results	45
5.3	Specific characteristics of CLC bio's algorithm	46
5.4	SOLiD data support in de novo assembly	46
5.5	Command line options	47
5.5.1	Specifying the data for the assembly and how it should be used	47
5.5.2	Specifying information for the assembly	49
5.5.3	Specifying information about the outputs	50
5.5.4	Other options	50
5.5.5	Example commands	50
6	Viewing and reporting tools	52
6.1	Mapping Viewer	52
6.2	The clc_sequence_info Program	53
6.3	The clc_mapping_table Program	56
6.4	The clc_mapping_info Program	58
7	Assembly post-processing tools	61
7.1	The clc_change_cas_paths Program	61
7.2	The clc_filter_matches Program	62
7.3	The clc_find_variations Program	62

7.4	The <code>clc_join_mappings</code> Program	62
7.5	The <code>clc_submapping</code> Program	63
7.5.1	Specifying Mapping Files	63
7.5.2	Extracting a Subset of Reference Sequences	63
7.5.3	Extracting a Part of a Single Reference Sequence	63
7.5.4	Extracting Only Long Contigs (Useful for De Novo Assembly)	63
7.5.5	Extracting a Subset of Read Sequences	63
7.5.6	Other Match Restrictions	64
7.5.7	Output Reference File	64
7.5.8	Output Read File	64
7.5.9	Handling of non-specific matches	64
7.6	The <code>clc_unmapped_reads</code> Program	65
7.7	The <code>clc_unpaired_reads</code> Program	65
7.8	The <code>clc_agp_join</code> Program	65
8	Sequence preparation tools	66
8.1	The <code>clc_adapter_trim</code> Program	66
8.2	Quality trimming	67
8.2.1	Fastq quality scoring	67
8.3	The <code>clc_remove_duplicates</code>	68
8.3.1	Looking for neighbors	68
8.3.2	Sequencing errors in duplicates	69
8.3.3	Paired data	70
8.3.4	Known limitations	70
8.3.5	Example of duplicate read removal	70
8.4	The <code>clc_sample_reads</code> Program	71
8.5	The <code>clc_sort_pairs</code> Program	71
8.6	The <code>clc_split_reads</code> Program (for 454 paired data)	72
8.7	The <code>clc_overlap_reads</code> Program	73
9	Format conversion tools	74
9.1	The <code>clc_cas_to_sam</code> Program	74
9.2	The <code>clc_sam_to_cas</code> Program	74

9.3 The clc_convert_sequences Program	75
A Updating program names from earlier versions	76
B Options for All Programs	78
Bibliography	79
Index	79

Chapter 1

Introduction

This document describes the CLC Assembly Cell. This package includes command line tools for performing de novo assemblies, read mappings, and basic downstream analysis of the results of these analyses. CLC Assembly Cell also includes utility tools for certain types of data pre-processing and sequence format conversion.

The names of the programs of CLC Assembly Cell have changed since version 3.2.2 and earlier. Please see section [A](#) for more information.

1.1 Overview of Commands

There are many tools within the CLC Assembly Cell. We list the tools included briefly here, with chapters dedicated to details about the core tools, and then other categories of tools, following.

Full usage information for all tools are given in section [B](#). The full usage information for each program can also be viewed by executing it without any options.

1.1.1 Core analysis tools

De novo assembly and mapping reads to reference sequences form the core tools of CLC Assembly Cell. These tools can be accessed using the following commands:

clc_assembler De novo assembly.

clc_read_mapping Used for mapping reads to a reference sequence.

clc_read_mapping_legacy The read mapper included in earlier versions of CLC Assembly Cell. This tool is for mapping sequencing reads from the SOLiD color space platform to a reference sequence.

The output of a de novo assembly is a set of contig sequences in fasta format.

The output of read mapping tools is a file in a special format called cas. The file extension for this file is .cas.

1.1.2 Viewing and reporting tools overview

A basic viewing tool for assemblies is included with CLC Assembly Cell:

clc_mapping_viewer Show a number of mappings in a text based viewer.

The following commands are available for reporting information about cas assembly files as well as contig data:

clc_sequence_info Print overview of any sequence file.

clc_mapping_info Print overview of a mapping.

clc_mapping_table Print details of each read in a mapping.

1.1.3 Assembly post-processing tools

Various operations can be performed on cas assembly files.

clc_change_cas_paths Change the references and / or read file names in a mapping file.

clc_filter_matches Remove matches of low similarity.

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

clc_join_mappings Join a number of assemblies to the same reference.

clc_submapping Extract a part of an assembly

clc_unmapped_reads Extract unassembled reads from an assembly.

clc_unpaired_reads Extract reads from broken pairs.

If more advanced downstream analyses of assemblies are desired, the CLC Genomics Workbench can be used (see <http://www.clcbio.com/genomics>). The Workbench uses the same de novo assembly and read mapping algorithms as the CLC Assembly Cell, so these tools can be directly run in the CLC Genomics Workbench, or alternatively, read mapping or assembly outputs created using CLC Assembly Cell can be imported into the Workbench. However, note that the CLC Genomics Workbench requires a separate license to the CLC Assembly Cell.

1.1.4 Sequence preparation tools

A range of different tools are available for sequence preparation.

clc_adapter_trim Trim adapters from sequences.

clc_quality_trim Trim reads based on quality.

clc_remove_duplicates Remove duplicate reads from genomic data.

clc_sample_reads Random sampling of reads.

clc_sort_pairs Split paired read files into paired and unpaired files.

clc_split_reads Remove linker from 454 paired data and extracts pairs.

clc_overlap_reads Merge overlapping reads.

1.1.5 Format conversion

clc_cas_to_sam For conversion of cas format mapping files to sam format.

clc_sam_to_cas For conversion of sam format mapping files to cas format.

clc_to_fasta Converts fastq, sff, csfasta and genbank format files into fasta.

Chapter 2

System Requirements and Installation

2.1 System requirements

- Windows XP, Windows Vista, Windows 7, Windows 8, Windows Server 2003 or Windows Server 2008
- Mac OS X 10.6 or later. However, Mac OS X 10.5.8 is supported on 64-bit Intel systems.
- Linux: Red Hat 5.0 or later. SUSE 10.2 or later. Fedora 6 or later.
- 1024 x 768 display recommended
- Intel or AMD CPU required
- **Special requirements for read mapping.** The numbers below give minimum and recommended memory for systems running mapping and analysis tasks. The requirements suggested are based on the genome size. Systems with less memory than specified below will benefit from installing the legacy read mapper plugin (see <http://www.clcbio.com/plugins>). This is slower than the standard mapper but adjusts to the amount of memory available.
 - **E. coli K12 (4.6 megabases)**
 - * Minimum: 500Mb RAM
 - * Recommended: 1Gb RAM
 - **C. elegans (100 megabases)** and **Arabidopsis thaliana (120 megabases)**
 - * Minimum: 1Gb RAM
 - * Recommended: 2Gb RAM
 - **Zebrafish (1.5 gigabases)**
 - * Minimum: 5Gb RAM
 - * Recommended: 8Gb RAM
 - **Human (3.2 gigabases)** and **Mouse (2.7 gigabases)**
 - * Minimum: 16Gb RAM
 - * Recommended: 24Gb RAM

- **Special requirements for de novo assembly.** *De novo* assembly may need more memory than stated above - this depends both on the number of reads, error profile and the complexity and size of the genome. See <http://www.clcbio.com/white-paper> for examples of the memory usage of various data sets.
- 64 bit computer and operating system required to use more than 2GB RAM

2.1.1 Limitations on maximum number of cores

For static licenses, there is a limitation on the number of CPU cores on the computer. If there are more than 64 cores (hyper threaded cores), the CLC Assembly Cell cannot be started. In this case, a network license is needed (read more at <http://www.clcbio.com/desktop-applications/licensing/>).

2.1.2 Supported CPU architectures

Software from CLC bio is developed for and tested on the x86 and x86-64 CPU architectures, which are used in most Intel and AMD CPUs. PowerPC CPUs, such as those used in Apple products until 2006, are not supported. To run CLC bio Assembly Cell the CPU must also support the SSE2 instruction set, which is commonly available in Intel CPUs produced from 2001 and onwards and AMD CPUs produced from 2003 and onwards.

If you are not sure if your CPU is supported, send a mail to support@clcbio.com with all available technical information about your computer.

2.2 Disk space

Data from Next-Generation sequencing machines naturally take up a lot of disk space. Besides the output files, the CLC Assembly Cell will sometimes write temporary files. These files will be written to the directory specified in the TMP variable on Windows and TMPDIR on Linux and Mac.

2.3 Downloading and installing the software

1. Download the distribution from:

<http://www.clcbio.com/products/clc-assembly-cell-direct-downloads/>.

2. Unzip the zip-file and ensure that the resulting folder is placed in the desired final location on your computer.

There are two main types of license for the CLC Assembly Cell software: static and network. Static licenses are tied to the hardware to which they are downloaded. Network licenses are served using a separate piece of software, the CLC License Server, and allow CLC Assembly Cell programs to run on any machine that can contact the License Server¹.

For obtaining an evaluation license, please follow the instructions in the static license section.

¹For running the software on a computer cluster, the most common license type would be a network license, which would then allow you to submit jobs to any node of your computer cluster.

2.4 Installing Static license

For purchased licenses, please ensure you have your License Order ID available, preferably in a form you can copy and paste, before embarking on the instructions in this section.

These instructions assume that the machine you have installed CLC Assembly Cell on is connected to the network and can access outside sites. If this is not the case, please see the section below.

2.4.1 Licensing the software on a networked Linux or Mac machine

1. On the command line, run the **clc_cell_licutil** tool that you will find inside the installation directory of CLC Assembly Cell.

You will need to run this tool as a user that has permissions to write the license file that is downloaded into the **licenses** folder in the installation directory of CLC Assembly Cell. If your software is installed centrally, this may mean running the tool with **sudo**.

2. You will be prompted as to whether you wish to **Request an evaluation license** or **Download license using a License Order ID**.

For an evaluation license, just choose that option. As long as you have not previously trialled the software on your machine, the evaluation license should be downloaded. You will see a message printed to screen about the expiry date of the evaluation license and where the license was downloaded to. You should now be able to trial the software.

3. If you have a License Order ID, please copy it and then paste it in at the prompt.

After a few moments, your license should be downloaded and a message will be written to screen saying that it was successfully downloaded and where it was saved.

2.4.2 Licensing the software on a networked Windows machine

1. Go to the Windows start menu and in the search box, type **cmd**.

2. Click on the **cmd.exe** tool which will launch the windows command prompt.

You need to run this as a user that has permissions to write the license file that is downloaded into the **licenses** folder in the installation directory of the CLC Assembly Cell. If your software is installed centrally, this will likely mean right clicking on the **cmd.exe** option and choosing to **Run as administrator**.

3. Navigate to the installation folder of the CLC Assembly Cell and execute the **clc_cell_licutil.bat** script.

4. You will be prompted as to whether you wish to **Request an evaluation license** or **Download license using a License Order ID**.

For an evaluation license, just choose that option. As long as you have not previously trialled the software on your machine, the evaluation license should be downloaded. You will see a message printed to screen about the expiry date of the evaluation license and where the license was downloaded to. You should now be able to trial the software.

5. If you have a License Order ID, please copy it and then paste it in at the prompt.

After a few moments, your license should be downloaded and a message will be written to screen saying that it was successfully downloaded and where it was saved.

2.4.3 Licensing the software on a non-networked machine

Using the tool distributed with CLC Assembly Cell for downloading a static license, the license will be specific to the machine you download it to. For a machine unable to connect to an outside network, you can follow the steps below to get a license for the software:

1. Get the host id for the machine that CLC Assembly Cell is installed on. To do this, run the **clc_cell_licutil** tool as per the instructions in the Linux and Mac or Windows sections above. (You do not need administrator privileges for this.)
2. Copy the **Host ID(s)** information that is printed near the top of the output.
3. On a machine that is able to reach external sites, go to the webpage <https://secure.clcbio.com/LmxWSv3/GetLicenseFile>
4. Paste in your License Order ID and your host ID information, as well as a host name. The host name is not important, but we recommend it is something that allows you to recognise or identify the machine that's been licensed if needed.
5. Click on the **Save** button.
6. Move the license file onto the machine where CLC Assembly Cell is installed.
7. Save the license file in the folder called **licenses** in the installation directory of CLC Assembly Cell².

2.5 Network Licenses

Network licenses are made available to users of CLC Assembly Cell software by using a separate piece of software called CLC License Server.

In general terms, you need to:

- Download, install and start up CLC License Server on a machine that is accessible to the machines that CLC Assembly Cell will be running on. This would generally be a machine that is left on, with the CLC License Server running as a service.
- Configure the license settings for copies of CLC Assembly Cell that will make use of the network licenses.

2.5.1 Installing and Running CLC License Server

How to install and run CLC License Server is described in our CLC License Server manual available from http://www.clcbio.com/wp-content/uploads/2012/09/CLC_License_Server_User_Manual.pdf.

²Locations for static license files supported in earlier versions of the CLC Assembly Cell can continue to be used. We recommend, however, that you choose to store your static license in the **licenses** folder in the installation directory, as this could help us in troubleshooting any licensing issues you may contact us about.

The CLC License Server software can be downloaded from: <http://www.clcbio.com/products/clc-license-server-direct-download/>

2.5.2 Configuring the software to use a network license

In order to make CLC Assembly Cell contact the license server for a license, you need to create a text file called `license.properties` including the following information:

```
serverip=192.168.1.200
serverport=6200
useserver=true
```

The `serverip` and `serverport` should be edited to match your license server set-up.

This text file then should be placed in the **licenses** folder of the installation area of CLC Assembly Cell.

Locations supported in earlier versions of CLC Assembly Cell can still be used, although we recommend the location above. The full list of locations is:

- in the **licenses** folder of the installation area of CLC Assembly Cell.
- in the *working directory*
- in `/etc/clcbio/licenses` on the executing machine, or
- in `$HOME/.clcbio/licenses/` where `$HOME` is the home directory of the user executing the program.

2.6 Restricting CPU usage

De novo assembly and mapping programs will use all cores available on the system if the job is large enough to warrant this. Should you wish to limit the number of cores to be used by a particular analysis, ‘`-cpus`’ option can be used to set the maximum.

This option is included in the full listing of options for the relevant programs.

Chapter 3

The Basics

The chapter covers the basics of command line use in CLC Assembly Cell, including data format considerations such as supported data formats, the cas assembly file format and conversion to other assembly formats. The chapter ends with an overview of paired data handling in CLC Assembly Cell.

3.1 How to use the programs

The CLC Assembly Cell consists of standard command line tools, where the tool name is provided, followed by any flags or parameters required. All input to the command, including designating input and output files, is done via parameter arguments.

General things to be aware of when setting up a CLC Assembly Cell command include:

- For programs where there are choices between fasta and fastq as output formats, the format that is output is determined based on the filename you specify in the command. For example, for the `clc_remove_duplicates` program, if you provide an output filename ending in `.fq` or `.fastq`, then the output format will be fastq. Otherwise, it will be fasta. Any program with this sort of behaviour should include information about the convention used in the usage information produced by running the command without any arguments.
- When providing paired data in two files, where one file contains one member of a pair and the other file contains the other member of a pair, you must include the `-i` flag in front of **each** input file. More information is provided about this later in this chapter, when paired data input is discussed, as well as in the chapters on read mappings and de novo assembly.
- When providing information about sequences, such as fragment lengths, also referred to as distances, for paired data, the parameter values you enter will apply to **all read files after that point in the command**, until the point in the command where new parameter values are provided. This is discussed further in the chapters on read mappings and de novo assembly.

3.1.1 Getting Help

This manual gives information about the tools included in CLC Assembly Cell.

Full usage information for each program is available in Appendix B of this user manual, and also by running any of the CLC Assembly Cell commands without any arguments. For the core programs, `clc_mapper` and `clc_assembler`, particular parameters are discussed in more detail within the chapters dedicated to those tools.

3.1.2 A basic example

A basic example of a CLC Assembly Cell command would be running the `clc_unpaired_reads` program. This program generates an output file of reads that are not paired within a given mapping. Here, we would need to specify the mapping to look at and the name of the output file. Below is an example of how such a command might look:

```
clc_unpaired_reads -a assembly.cas -o unmapped.fasta
```

3.2 Input Files

The formats in the following table are recognized as valid input formats by one or more of the CLC Assembly Cell tools. Note that not all listed formats are valid for data to be treated as sequence reads, and not all listed formats are valid for data to be treated as reference sequences in the case of read mappings.

Input file formats are automatically detected by the software through consideration of the file contents. The filename is irrelevant with regards to input format.

Format	Reads	References
Fasta	+	+
Fastq	+	-
Scarf	+	-
csfasta*	+	-
Sff**	+	-
GenBank	-	+

*The full sequence of any read containing one or more . symbols, present in a .csfasta format file, will be converted to contain only N characters when used by or output by any of the Assembly Cell tools.

**Please note that paired 454 data needs to be pre-processed using the `clc_split_reads` program.

Read data compressed using `gzip` is supported as input by the CLC Assembly Cell programs except for `clc_remove_duplicates`.

Reference data cannot be in a compressed form.

3.3 Cas File Format

CLC Assembly Cell uses the `cas` file format for read mappings. It is a custom file format that caters to the demands of high throughput sequencing data, while being flexible enough to handle

other sequence data types also. No deep knowledge of this file format is necessary to work with it, but some basics can aid in understanding what this format contains and how it can be used.

3.3.1 Cas Format Basics

The cas format is a binary format. This is space efficient, taking only approximately 8 bytes per read assembled to the human genome. So a cas file with 100 million Solexa reads, each with a length of 35 bases, assembled to the human genome would only take up about 800 MB.

3.3.2 What a cas file contains

In essence, cas format files contain data about the relationships between sequences in other files.

In particular, 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 pair.

3.3.3 What a cas file does not contain

Cas format files do **not** contain any sequence data. Rather than the sequence information itself, cas files contain the names of the corresponding read and reference sequence files. As sequence reads and references already exist, much space can be saved by not generating a second copy of them as part of the assembly output file.

3.3.4 Considerations and limitations

The cas file format is designed with high volume assembly data in mind. However, there are certain considerations that should be kept in mind:

1. There is a limit of one alignment position per read. In other words, a read matching in multiple locations can only be assigned to one of these locations within the cas file. This limitation is in place because when assembling short reads to a large genome, some reads may match hundreds of thousands of locations. Keeping track of all such alignments would be problematic.

2. If you are planning to send your assembly to someone else for viewing or further processing, you need to include your read and reference files in addition to the cas assembly file. This is because the cas file contains information about the assembly, and does not contain any sequence information.
3. If you are planning to send your assembly to someone else, they must put the read and reference files in the same relative location to the cas file, as you did when you ran the assembly. This is because the cas file stores relative file names, and these must match the location of the read and reference files when further processing is undertaken. Please note though that the program `change_assembly_files` can be used to change the file names and locations.
4. If you plan to convert your cas file to SAM or BAM format, which include read information, you need to have the read data used for your mapping, as well as the cas file, available when you run the `clc_cas_to_sam` program.

3.3.5 Converting to and from SAM and BAM formats

CLC Assembly Cell includes a tool called **clc_cas_to_sam** to convert a cas format file to the SAM¹ or BAM² format. Also included is a tool called **clc_sam_to_cas** that converts from SAM or BAM format into the cas format.

These tools are described in more detail in their own sections: section 9.1 and section 9.2.

3.4 Paired read Considerations

You can specify that a read file came from a paired sequencing experiment using the ‘-p’ option. This option is described in detail here, as well as within the read mapping and de novo assembly sections of the manual.

A typical set of information one would provide after the -p flag would look like this

```
-p fb ss 100 200
```

. The meaning of this would be:

- **fb** Specifies the relative orientation of the reads. Here, the first read of the pair is in the forward direction, the second read is in the backward, or reverse, orientation. The allowed values for this are provided below.
- **ss** Specifies the way the distances between the pair members should be measured. Here, the distances are given from the start (5 prime end) of the first read to the start (5 prime end) of the second read. Here, since the relative orientation is set to fb, the second read is reversed, so indicating ss means that the distance specified will include both the read lengths as well as the length of the sequence between the reads.
- **100 200** The range of distances expected between the specified start positions. Here this is between 100 and 200 bases.

¹Sequence Alignment/Map format

²BAM is the binary compact format for SAM

3.4.1 Relative orientation of the reads

For all codes, it is possible to assemble the pair to any of the two reference sequence strands, so 'ff' may mean that both reads are placed in the forward direction or that both reads are placed in the reverse direction. There is still a difference between 'ff' and 'bb', though. For 'bb', the second read is effectively placed before the first read. The 'bb' option is not widely used and is included for the sake of completeness.

The allowed values for the directions and their meanings are summarized in the table below.

Code	Read		Description
	First	Second	
ff	→	→	Both reads are forward.
fb	→	←	Reads point toward each other.
bf	←	→	Reads point away from each other.
bb	←	←	Both reads are backward.

3.4.2 Measuring the distance between the reads

How the distance between the reads should be measured depends on how the sequencing experiment is done. If the reads are sequenced in the upstream to downstream direction, the start of the reads is where the distance should be measured. This is indicated by the 'ss' code for start to start. The allowed values are 'ss', 'se', 'es', and 'ee', where the first letter indicates which end of the first read should be used and the second letter indicates which end of the second read should be used ('s' for start and 'e' for end). The 'ss' option is the most typical.

So, for typical paired end Illumina sequencing protocol, using the 'fb ss' combination ensures the correct relative directions of the reads. It also ensures that the distance is independent of the read length since typical sequencing experiment progress expands the reads toward each other from their starting points.

When the '-p' option is used, it applies to all read files from that point and forward in the command line. If different experiments with different paired properties are combined, the '-p' option can be used several times. To indicate that the following read files are not paired, used '-p no'. This is only necessary if another '-p' option was previously used. An example:

```
clc_mapper -o assembly.cas -d human.gb -q reads1.fasta -p fb ss 180
                                     250 reads2.fasta -p no reads3.fasta
```

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

Note that the clc_sort_pairs and clc_split_reads program can be used to convert data from SOLiD and 454 systems, respectively, into an format accepted by the CLC Assembly Cell tools.

3.4.3 Paired Read File Input

Paired data may be contained in a single file, where the pairs are sorted such that the first two sequences are one pair, the second two sequences the next pair, and so on. Paired data may also exist in two files, with one file containing the first member of all pairs, and the other file

containing the second member of all pairs, with each member appearing in the same ordered position in each file. For example, the 51st sequence in file A is the mate of the 51st sequence in file B.

The CLC Assembly Cell programs assume the single file form for paired data as the default. For paired data with separate files for first and second members of the pair, both files need to be included as input, with each of these files being preceded by the ‘-i’ option (for interleave). The order of the files on the command line matters. The first file should contain the first member of the pair. The second file should contain the second member of the pair.

To further illustrate this, 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_mapper -o assembly.cas -d human.gb -q -p fb ss 180 250
-i first.fasta second.fasta
```

This is identical to:

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

Note that the ‘-i’ option has to immediately proceed the input files.

Chapter 4

Read Mapping

There are two programs within the CLC Assembly Cell for mapping reads to a reference sequence, or reference sequences:

`clc_mapper`

for mapping in base space and

`clc_mapper_legacy`

for mapping in color space.

The aim of both programs is the same: to map reads to the area of a reference sequence that they are likely to have originated from. In both cases, 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 at 90% identity over 50% of the read length. A gapped alignment is always performed.

By default, read mapping is done with local alignment of reads to a set of reference sequences. The advantage of performing local alignment rather than global alignment is that the ends are automatically removed if there are sufficiently many sequencing errors in those regions. This can also be beneficial if the ends of the reads contain vector contamination or adapter sequences.

An option exists to run global alignment instead of local alignment if this is desired.

In cases where memory consumption is an issue the

`clc_mapper_legacy`

can be used for base space mapping as it has a scalable memory consumption. However, we recommend that the

`clc_mapper`

is used for base space mapping when possible as it has better performance in terms of both quality and speed.

4.1 Overview of base space mapping

The base space read mapping tool is based on an uncompressed suffix array that represents the entire reference genome in a single data structure.

The algorithm iterates over input reads, mapping each read individually by applying the following procedure:

1. A search is carried out for the longest stretches of matching bases between the reference genome and a read by considering each base position of the read as a start position of a seed candidate.
2. End-positions of seeds are then determined by elongating the seeds as long as there are fully matching rows in the suffix array.
3. Finally, a maximum of 100 seeds is examined in detail using a banded Smith-Waterman algorithm.

The seed lengths in this mapping tool is variable but has a minimum size of 15bp. The variable seed length enable identification of short seeds where the alignment score is higher than the alignment score for longer seeds. This leads to a better mapping of some reads, and improves the chance of identifying the optimal mapping, especially for reads with high error rates.

The memory consumption of the `clc_mapper` tool is bounded from below by $5 * N$, where N equals the size of the reference genome. So, for example, the suffix array of the human genome, which is approximately 3 gigabases, consumes 15 gigabytes of main memory. Whilst such a requirement might appear rather large, it then allows for extremely good performance in both seeding and extension stages. The result is a very fast run time. Examples are provided in our white paper on the base space read mapper, available from our website: <http://www.clcbio.com/files/whitepapers/whitepaper-on-CLC-read-mapper.pdf>

4.2 Circular references

You can indicate to the `clc_mapper` tool that you are mapping to a circular reference by using ‘-z’ option in front of the file containing the circular genome. This option indicates that the next reference sequence to be input, using the -d option, is circular.

In the case of human genome and mitochondrial references then, the input command would look like this:

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

This will cause the mapper to map reads spanning the start point of the circular reference sequence as if they were mapped to a contiguous reference sequence. Note that the ‘-d’ option is used twice in the example above to achieve this.

* Please note this functionality is not available for the legacy read mapper.

4.3 Saving and re-using reference index files

The first stage of the base space mapper is to create an index file for the reference sequence(s) being used. For a given reference, the index will be identical each time you run the mapping tool. You can choose to save the index file that is created during one mapping run, and then re-use the index file in subsequent mapping runs.

The

```
-n <file> / --indexoutput <file>
```

parameter causes the index file for a particular reference set to be saved.

This reference index can be used instead of a reference sequences in subsequent mapping runs, i.e. use the name of the index file after the -d flag instead of the reference file name. If you have multiple index files, these can be entered individually as arguments to the -d flag similar to multiple reference files.

* Please note this functionality is not available for the legacy read mapper.

4.4 Overview of color space mapping

Color space mapping is done using the legacy mapping tool, as released in version 3.x of the CLC Assembly Cell. This is based on a four stage seeding approach and a seed index representing the reference genome. The mapper is able to ignore incorrect colors without obscuring the rest of the read alignment. The mapping algorithm iterates over input reads, mapping each read individually by applying the following procedure:

1. Seeding sequences of 30 nucleotides each are sampled from each third position of the input read.
2. These seeds are looked up in the index and the resulting candidate alignment locations are examined using a banded Smith Waterman alignment.
3. If no valid results are found, the mapping is retried three more times with shorter seeds sampled from every individual position of the read.

As soon as any of these four, ordered attempts yields one or more valid mapping result, the procedure is aborted and the highest scoring mapping is reported. If there are multiple mappings sharing the same highest score, one is chosen randomly.

The scoring system for a color space mapping includes the same parameters discussed in the Scoring Schemes section below, and includes one additional parameter to account for color space errors. This additional penalty score has a property whereby if this penalty has been applied for a particular aligned position of your read against your reference, there is an additional effect that the rest of that read will be subject to a phase shift, corresponding to a color correction applied to the remainder of the read. Overall, this will change the score for the mapping of the read to the reference. The mapping of the read to the reference with the highest score will be the one retained. This concept is explained in more detail below.

4.4.1 Sequencing

The SOLiD sequencing technology from Applied Biosystems is different from other sequencing technologies since it does not sequence one base at a time. Instead, two bases are sequenced at a time in an overlapping pattern. There are 16 different dinucleotides, but in the SOLiD technology, the dinucleotides are grouped in four carefully chosen sets, each containing four dinucleotides. The colors are as follows:

Base 1	Base 2			
	A	C	G	T
A	●	●	●	●
C	●	●	●	●
G	●	●	●	●
T	●	●	●	●

Notice how a base and a color uniquely defines the following base. This approach can be used to deduce a whole sequence from the initial nucleotide and a series of colors. Here is a sequence and the corresponding colors.

Sequence T A C T C C A T G C A
Colors ● ● ● ● ● ● ● ● ● ● ●

The colors do not uniquely define the sequence. Here is another sequence with the same list of colors:

Sequence A T G A G G T A C G T
Colors ● ● ● ● ● ● ● ● ● ● ●

But if the first nucleotide is known, the colors do uniquely define the remaining sequence. This is exactly the strategy used in SOLiD sequencing: The first nucleotide is known from the primer used, and the remaining nucleotides are deduced from the colors.

4.4.2 Error modes

As with other sequencing technologies, errors do occur with the SOLiD technology. If a single nucleotide is changed, two colors are affected since a single nucleotide is contained in two overlapping dinucleotides:

Sequence T A C T C C A T G C A
Colors ● ● ● ● ● ● ● ● ● ● ●

Sequence T A C T C C A A G C A
Colors ● ● ● ● ● ● ● ● ● ● ● ●

Sometimes, a wrong color is determined at a given position. Due to the dependence between dinucleotides and colors, this affects the remaining sequence from the point of the error:

Sequence T A C T C C A T G C A
Colors ● ● ● ● ● ● ● ● ● ●

Sequence T A C T C C A A C G T
Colors ● ● ● ● ● ● ● ● ● ●

Thus, when the instrument makes an error while determining a color, the error mode is very different from when a single nucleotide is changed. This ability to differentiate different types of errors and differences is a very powerful aspect of SOLiD sequencing. With other technologies sequencing errors always appear as nucleotide differences.

4.4.3 Mapping in color space

Reads from a SOLiD sequencing run may exhibit all the same differences to a reference sequence as reads from other technologies: mismatches, insertions and deletions. On top of this, SOLiD reads may exhibit color errors, where a color is read wrongly and the rest of the read is affected. If such an error is detected, it can be corrected and the rest of the read can be converted to what it would have been without the error.

Consider this SOLiD read:

Read T A C T C C A A C G T
Colors ● ● ● ● ● ● ● ● ● ●

The first nucleotide (T) is from the primer, so this is ignored in the following analysis. Now, assume that a reference sequence is this:

Reference G C A C T G C A T G C A C
Colors ● ● ● ● ● ● ● ● ● ●

Here, the colors are just inferred since they are not the result of a sequencing experiment.

Looking at the colors, a possible alignment presents itself:

Reference	G	C	A	C	T	G	C	A	T	G	C	A	C
Colors	●	●	●	●	●	●	●	●	●	●	●	●	●
				:			:	:	:	:	:	:	:
Read	A	C	T	C	C	A	A	C	G	T			
Colors	●	●	●	●	●	●	●	●	●	●			

In the beginning of the read, the nucleotides match (ACT), then there is a mismatch (G in reference and C in read), then two more matches (CA), and finally the rest of the read does not match. But, the colors match at the end of the read. So a possible interpretation of the alignment is that there is a nucleotide change in position four of the read and a color space error between positions six and seven in the read. Such an interpretation can be represented as:

Reference	G	C	A	C	T	G	C	A	T	G	C	A	C
				:									
Read	A	C	T	C	C	A	*T	G	C	A			

Here, the * represents a color error. The remaining part of the displayed read sequence has been adjusted according to the inferred error. So this alignment scores nine times the match score minus the mismatch cost and a color error cost. This color error cost is a new parameter that is introduced when performing read mapping in color space.

Note that a color error may be inferred before the first nucleotide of a read. This is the very first color after the known primer nucleotide that is wrong, changing the whole read.

Here is an example from a set of real SOLiD data that was reference assembled by taking color space into account using ungapped global alignments. The `clc_mapping_table` program with the '-a' option reports:

444_1840_767_F3 has 1 match with a score of 35:

```
1046535 GATACTCAATGCCGCCAAAGATGGAAGCCGGGCCA 1046569 reference
      |||
      GATACTCAATGCCGCCAAAGATGGAAGCCGGGCCA reverse read
```

444_1840_803_F3 has 0 matches

444_1840_980_F3 has 1 match with a score of 29:

```
2620828 GCACGAAAACGCCGCGTGGCTGGATGGT*CAAC*GTC 2620862 reference
      |||
      GCACGAAAACGCCGCGTGGCTGGATGGT*CAAC*GTC read
```

444_1840_1046_F3 has 1 match with a score of 32:

```
3673206 TT*GGTCAGGGTCTGGGCTTAGGCGGTGAATGGGGC 3673240 reference
      ||*
      TT*GGTCAGGGTCTGGGCTTAGGCGGTGAATGGGGC reverse read
```

444_1841_22_F3 has 0 matches

444_1841_213_F3 has 1 match with a score of 29:

```
1593797 CTTTG*AGCGCATTGGTCAGCGTGTAATCTCCTGCA 1593831 reference
      ||||*
      CTTTG*AGCGCATTAGTCAGCGTGTAATCTCCTGCA reverse read
```

The first alignment is a perfect match and scores 35 since the reads are all of length 35. The next alignment has two inferred color errors that each count is -3 (marked by * between residues), so the score is $35 - 2 \times 3 = 29$. Notice that the read is reported as the inferred sequence taking the color errors into account. The last alignment has one color error and one mismatch giving a score of $34 - 3 - 2 = 29$, since the mismatch cost is 2.

Running the same reference assembly without allowing for color errors, the result is:

444_1840_767_F3 has 1 match with a score of 35:

```
1046535 GATACTCAATGCCGCCAAAGATGGAAGCCGGGCCA 1046569 reference
      |||
      GATACTCAATGCCGCCAAAGATGGAAGCCGGGCCA reverse read
```

444_1840_803_F3 has 0 matches

444_1840_980_F3 has 0 matches

444_1840_1046_F3 has 1 match with a score of 29:

```

3673206 TTGGTCAGGGTCTGGGCTTAGGCGGTGAATGGGGC 3673240    reference
      ||||||||||||||||||||||||||||||||||
      AAGGTCAGGGTCTGGGCTTAGGCGGTGAATGGGGC    reverse read

```

444_1841_22_F3 has 0 matches

444_1841_213_F3 has 0 matches

The first alignment is still a perfect match, whereas two of the other alignment now do not match since they have more than two errors. The last alignment now only scores 29 instead of 32, because two mismatches replaced the one color error above. This shows the power of including the possibility of color errors when aligning: many more matches are found.

The reference assembly program in CLC Assembly Cell does not directly support alignment in color space only, but if such an alignment was carried out, sequence 444_1841_213_F3 would have three errors, since a nucleotide mismatch leads to two color space differences. The alignment would look like this:

444_1841_213_F3 has 1 match with a score of 26:

```

1593797 CTTTG*AGCGCATT*G*GTCAGCGTGTAATCTCCTGCA 1593831    reference
      |||||*|||||||*|*||||||||||||||||||
      CTTTG*AGCGCATT*G*GTCAGCGTGTAATCTCCTGCA    reverse read

```

So, the optimal solution is to both allow nucleotide mismatches and color errors in the same program when dealing with color space data. This is the approach taken by the assembly program in CLC Assembly Cell.

Note! If you set the color error cost as low as 1 while keeping the mismatch cost at 2 or above, a mismatch will instead be represented as two adjacent color errors. To invoke color space assembly, use the ‘-c’ option. The cost of color errors is set using ‘-y’ (range 1-3, default is 3). Note that the limit is also affected by the color space error cost:

4.4.4 Color space file formats

The .csfasta file format is often used for color space data. That format looks like this:

```

#picked reads from data/reads/SHIRAZ_20080320_MP_2_Sample1_F3.csfasta.original, panel ra
09
>600_50_31_F3
T2222002113300322132112231
>600_50_63_F3
T2330133212130133221033110
>600_50_100_F3
T0130001131012310201000101
>600_50_170_F3

```

```

T1002312103033121321233103
>600_50_174_F3
T0330022330332000323031121
>600_50_241_F3
T2103103103100212123030011
>600_50_256_F3
T0301131010233311200223332
>600_50_329_F3
T1303211033112301303220000
>600_50_342_F3
T2100003012212000310130111

```

...

So, it is very similar to the fasta file format. It does, however, allow one or more lines starting with # before the first sequence. The sequences are specified as a nucleotide followed by the colors encoded as numbers where 0 is blue, 1 is green, 2 is yellow, and 3 is red. So the sequence:

Sequence	T	A	C	T	C	C	A	T	G	C	A
Colors											

Would be coded like this in a `.csfasta` file:

```

>sequence
T3122013131

```

The T is the nucleotide that is known from the primer and the numbers indicate the colors. Because the T came from the primer, it is not part of the sequenced DNA molecule. Thus, this letter should be ignored when analyzing the read. So this sequence would look like this in `.fasta` format:

```

>sequence
ACTCCATGCA

```

So there is one nucleotide for each experimentally determined color (i.e. the numbers in the `.csfasta` file).

The `.csfasta` does not contain any significant information that is not also present in a standard fasta file of the same sequences. The only extra information is the last nucleotide of the primer, which is not useful in later analyses.

So from the viewpoint of software programs analyzing read data, color space is just yet another file format for reads along with `.fasta`, `.fastq`, `.sff`, etc. Thus, in the Assembly Cell programs, color space options for assembly have no connection to file formats. You can choose to assemble SOLiD data in `.csfasta` format without using the color space options for assembly and you can also choose to assemble reads in a normal `.fasta` file using color space assembly options.

4.5 General information for both read mappers

4.5.1 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 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.

Supplying a value for the `-t` option means that you can have multiple hit positions saved in the output of the assembly. Using the `clc_mapping_table` program, these multiple hit positions can be retrieved from each read.

4.5.2 Placement of Read Pairs

Many sequencing technologies allow paired 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 options are found.
- If both reads can be placed independently but no pairs satisfy the paired 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 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.

4.5.3 Scoring Schemes

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 ¹

An ambiguous nucleotide aligned to any other nucleotide including the same ambiguous type is treated as a mismatch.

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.

The restricted values in the scoring scheme allows more efficient algorithms to be used, which can have a large impact on the time required when large data sets are being considered.

4.5.4 Mapping quality thresholds

Once a read has been mapped, a filtering process determines whether this match is good enough for the read to be included in the result. The filtering threshold is determined by two fractions:

Length fraction The minimum length fraction of a read that must match the reference sequence. Setting a value at 0.5 means that at least half the read needs to match the reference sequence for the read to be included in the final mapping. This is set using the ‘-l’ option. The default value is 0.5.

Similarity The minimum fraction of identity between the read and the reference sequence. If you want the reads to have e.g. at least 90% identity with the reference sequence in order to be included in the final mapping, set this value to 0.9. Note that the similarity fraction does not apply to the whole read; it relates to the Length fraction. So with a length fraction set to 0.5, entering 0.9 for the similarity level has the meaning that *at least 50 % of the read must have at least 90 % identity*. This is set using the ‘-s’ option. The default value is 0.8.

4.6 Running Read Mapping Analyses

The key features to a read mapping command are to supply the input read data along with any pairing information, the reference sequences, and the output file name.

A basic example of a read mapping command is the following:

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

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_mapper -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 `clc_join_mappings` program.

For paired data, reads can be input either as single file where reads are interleaved or as two files which are automatically interleaved. In the following example paired reads are input as a single file:

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

In the next example, paired reads are input as two files using the `-i` option: The above command would be identical to the following command, if the same paired data were in an interleaved file.

```
{ \footnotesize
\begin{verbatim}
clc_mapper -o assembly.cas -d human.gb -q -p fb ss 180 250
                                                    -i first.fasta second.fasta
```

The above command would be identical to the following command, if the same paired data was in an interleaved file.

Please refer to the section on paired data input earlier in the manual for further details about that topic.

You can save index files for a reference set and then use the index file directly, instead of the reference sequence, in subsequent mappings.

Save the reference index:

```
clc_mapper -o assembly.cas -d human.gb -q -p fb ss 180 250
                                                    joint.fasta -n human-reference.index
```

Use the reference index:

```
clc_mapper -o assembly.cas -d human-reference.index
                                                    -q -p fb ss 180 250 joint.fasta
```

Further details are provided in Appendix [B](#).

4.7 Mixed base space and color space mappings

For mixed data sets consisting of base space data and color space data, we recommend the following approach:

1. Run a mapping using the `clc_mapper` tool using the base space data.
2. Run a mapping using the `clc_mapper_legacy` tool using the color space data and the exact same reference sequence set used for the base space mapping.
3. Join these mappings using the `clc_join_mappings` tool.

Chapter 5

De novo assembly

The `clc_assembler` program performs assembly of reads without a known reference. The input data consists of files containing read sequences.

5.0.1 De novo assembly inputs

Any number of read files can be input to a de novo assembly. These includes files containing paired reads and files containing single reads. Different types of read data can be input to a single de novo assembly. Below is a table of the accepted formats for data input.

Format	option	
Fasta	+	+
Fastq	+	-
Scarf	+	-
csfasta	+	-
Sff *	+	-
GenBank	-	+

* Please note that paired 454 data needs to be pre-processed using the `clc_split_reads` program

5.0.2 De novo assembly outputs

The output of the `clc_assembler` is a fasta file containing all the contig sequences.

This means that there is no information about where the reads are placed, how they align, coverage levels etc. If this information is desired, you can use the `clc_mapper` or `clc_mapper_legacy` program and use the newly created contig sequences as references. The `cas` format file created using the mapping program will contain this sort of information.

If the `-f` option has been used, then a file containing features related to scaffolding will be generated. Choosing to name the file given as an argument to the `-f` option with a `.agp` suffix will generate an AGP format file. This format specification can be found online https://www.ncbi.nlm.nih.gov/projects/genome/assembly/agp/AGP_Specification.shtml.

Choosing to name the file given as an argument to the `-f` option with a `.gff` suffix will generate a

GFF format file. The columns of this file contain the following information:

Column 1: Name of contig Column 2: Source program Column 3: Annotation type (see below)
Column 4: Start position Column 5: End position Column 6: Score (see below) Column 7, 8 and
9: no meaning: there to conform to the GFF format.

Further details about Annotation types (column 3)

There are three annotation types that can appear in the third column:

1) Alternatives Excluded: More than one path through the graph was possible in this region but evidence from paired data suggested the exclusion of one or more alternative routes in favor of the route chosen.

2) Contigs Joined: More than one route was possible through the graph such that an unambiguous choice of how to traverse the graph cannot be made. However evidence from paired data supports one of these routes and on this basis, this route is followed to the exclusion of the other(s).

3) Scaffold: The route through the graph is not clear but evidence from paired data supports the connection of two contigs. A single contig is then reported with N characters between the two connected regions. This entity is also known as a scaffold. The number of N characters represents the expected distance between the regions, based on the evidence the paired data.

If one chooses not to scaffold, a resulting gff annotation file will still report any "Contigs joined" and "Alternatives excluded" optimizations, as these are still performed in this case.

Further details about Scores (column 6)

For annotation type Scaffold, the size of the gap that has been estimated between scaffolded sections of the contig is reported in the score column.

For annotation type Alternatives Excluded, the score is reported as the (word size + 1). This value merely serves as a reminder that the region reported for this event is associated with the word size used for the assembly.

For annotation type Contigs Joined, the value in the score column is 0.

5.1 How it works

CLC bio's de novo assembly algorithm works by using de Bruijn graphs. This is similar to how most new de novo assembly algorithms work [Zerbino and Birney, 2008, Zerbino et al., 2009, Li et al., 2010, Gnerre et al., 2011]. The basic idea is to make a table of all sub-sequences of a certain length (called words) found in the reads. The words are relatively short, e.g. about 20 for small data sets and 27 for a large data set (the word size is determined automatically, see explanation below).

Given a word in the table, we can look up all the potential neighboring words (in all the examples here, word of length 16 are used) as shown in figure 5.1.

Typically, only one of the backward neighbors and one of the forward neighbors will be present in the table. A graph can then be made where each node is a word that is present in the table and edges connect nodes that are neighbors. This is called a de Bruijn graph.

For genomic regions without repeats or sequencing errors, we get long linear stretches of connected nodes. We may choose to reduce such stretches of nodes with only one backward

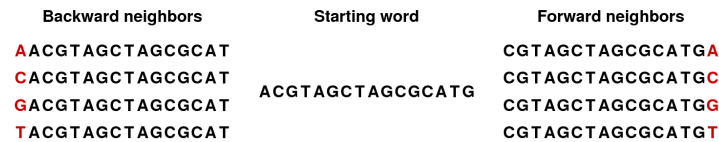


Figure 5.1: The word in the middle is 16 bases long, and it shares the 15 first bases with the backward neighboring word and the last 15 bases with the forward neighboring word.

and one forward neighbor into nodes representing sub-sequences longer than the initial words.

Figure 5.2 shows an example where one node has two forward neighbors:



Figure 5.2: Three nodes connected, each sharing 15 bases with its neighboring node and ending with two forward neighbors.

After reduction, the three first nodes are merged, and the two sets of forward neighboring nodes are also merged as shown in figure 5.3.

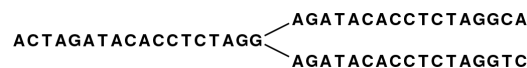


Figure 5.3: The five nodes are compacted into three. Note that the first node is now 18 bases and the second nodes are each 17 bases.

So bifurcations in the graph leads to separate nodes. In this case we get a total of three nodes after the reduction. Note that neighboring nodes still have an overlap (in this case 15 nucleotides since the word length is 16).

Given this way of representing the de Bruijn graph for the reads, we can consider some different situations:

When we have a *SNP* or a *sequencing error*, we get a so-called bubble (this is explained in detail in section 5.1.5) as shown in figure 5.4.



Figure 5.4: A bubble caused by a heterozygous SNP or a sequencing error.

Here, the central position may be either a C or a G. If this was a sequencing error occurring only once, we would see that one path through the bubble will only be words seen a single time. On the other hand if this was a heterozygous SNP we would see both paths represented more or less equally. Thus, having information about how many times this particular word is seen in all the reads is very useful and this information is stored in the initial word table together with the words.

The most difficult problem for de novo assembly is repeats. Repeat regions in large genomes often get very complex: a repeat may be found thousands of times and part of one repeat may also be part of another repeat. Sometimes a repeat is longer than the read length (or the paired distance when pairs are available) and then it becomes impossible to resolve the repeat. This is simply because there is no information available about how to connect the nodes before the repeat to the nodes after the repeat.

In the simple example, if we have a *repeat sequence* that is present twice in the genome, we would get a graph as shown in figure 5.5.



Figure 5.5: The central node represents the repeat region that is represented twice in the genome. The neighboring nodes represent the flanking regions of this repeat in the genome.

Note that this repeat is 57 nucleotides long (the length of the sub-sequence in the central node above plus regions into the neighboring nodes where the sequences are identical). If the repeat had been shorter than 15 nucleotides, it would not have shown up as a repeat at all since the word length is 16. This is an argument for using long words in the word table. On the other hand, the longer the word, the more words from a read are affected by a sequencing error. Also, for each extra nucleotide in the words, we get one less word from each read. This is in particular an issue for very short reads. For example, if the read length is 35, we get 16 words out of each read if the word length is 20. If the word length is 25, we get only 11 words from each read.

To strike a balance, CLC bio's de novo assembler chooses a word length based on the amount of input data: the more data, the longer the word length. It is based on the following:

```
word size 12: 0 bp - 30000 bp
word size 13: 30001 bp - 90002 bp
word size 14: 90003 bp - 270008 bp
word size 15: 270009 bp - 810026 bp
word size 16: 810027 bp - 2430080 bp
word size 17: 2430081 bp - 7290242 bp
word size 18: 7290243 bp - 21870728 bp
word size 19: 21870729 bp - 65612186 bp
word size 20: 65612187 bp - 196836560 bp
word size 21: 196836561 bp - 590509682 bp
word size 22: 590509683 bp - 1771529048 bp
word size 23: 1771529049 bp - 5314587146 bp
word size 24: 5314587147 bp - 15943761440 bp
word size 25: 15943761441 bp - 47831284322 bp
word size 26: 47831284323 bp - 143493852968 bp
word size 27: 143493852969 bp - 430481558906 bp
word size 28: 430481558907 bp - 1291444676720 bp
word size 29: 1291444676721 bp - 3874334030162 bp
word size 30: 3874334030163 bp - 11623002090488 bp
etc.
```

This pattern (multiplying by 3) continues until word size of 64 which is the max. Please note that the range of word sizes is 12-24 on 32-bit computers and 12-64 on 64-bit computers. The word size can also be specified manually using the `-w` option. Using the `-v` (verbose) option, you can see the word size that is automatically calculated by the assembler

5.1.1 Resolve repeats using reads

Having build the de Bruijn graph using words CLC bio's de novo assembler removes repeats and errors using reads. This is done in the following order:

- Remove weak edges
- Remove dead ends
- Resolve repeats using reads without conflicts
- Resolve repeats with conflicts
- Remove weak edges
- Remove dead ends

Each phase will be explained in the following subsections.

Remove weak edges

The de Bruijn graph is expected to contain artifacts from errors in the data. The number of reads agreeing upon an error is likely to be low especially compared to the number of reads without errors for the same region. When this relative difference is large enough, it's possible to conclude something is an error.

In the remove weak edges phase we consider each node and calculate the number c_1 of edges connected to the node and the number of times k_1 a read is passing through these edges. An average of reads going through an edge is calculated $avg_1 = k_1/c_1$ and then the process is repeated using only those edges which have more than or equal avg_1 reads going through it. Let c_2 be the number of edges which meet this requirement and k_2 the number of reads passing through these edges. A second average $avg_2 = k_2/c_2$ is used to calculate a limit,

$$limit = \frac{\log(avg_2)}{2} + \frac{avg_2}{40}$$

and each edge connected to the node which has less than or equal *limit* number of reads passing through it will be removed in this phase.

Remove dead ends

Some read errors might occur more often than expected, either by chance or because they are systematic sequencing errors. These are not removed by the "Remove weak edges" phase and will cause "dead ends" to occur in the graph, which are short paths in the graph that terminate after a few nodes. Furthermore, the "Remove weak edges" sometimes only removes a part of the graph, which will also leave dead ends behind. Dead ends are identified by searching for paths in the graph where there exists an alternative path containing four times more nucleotides. All nodes in such paths are then removed in this step.

Resolve repeats without conflicts

Repeats and other shared regions between the reads lead to ambiguities in the graph. These must be resolved otherwise the region will be output as multiple contigs, one for each node in the region.

The algorithm for resolving repeats without conflicts considers a number of nodes called the *window*. To start with, a window only contains one node, say R . We also define the *border nodes*

as the nodes outside the window connected to a node in the window. The idea is to divide the border nodes into sets such that border nodes A and C are in the same set if there is a read going through A, through nodes in the window and then through C. If there are strictly more than one of these sets we can resolve the repeat area, otherwise we expand the window.

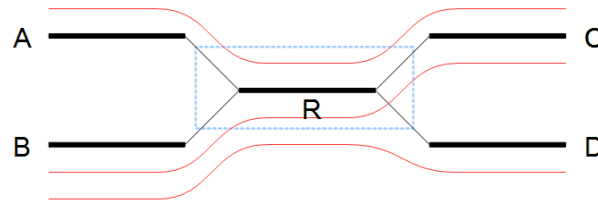


Figure 5.6: A set of nodes.

In the example in figure 5.6 all border nodes A, B, C and D are in the same set since one can reach every border nodes using reads (shown as red lines). Therefore we expand the window and in this case add node C to the window as shown in figure 5.7.

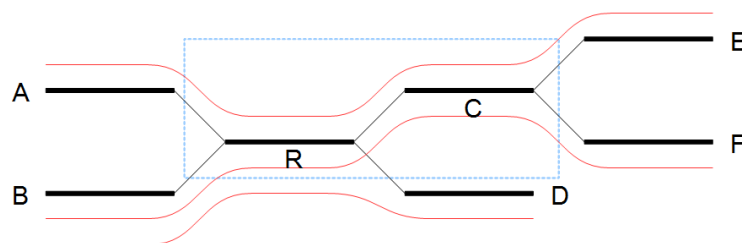


Figure 5.7: Expanding the window to include more nodes.

After the expansion of the window, the border nodes will be grouped into two groups being set A, E and set B, D, F. Since we have strictly more than one set, the repeat is resolved by copying the nodes and edges used by the reads which created the set. In the example the resolved repeat is shown in figure 5.8.

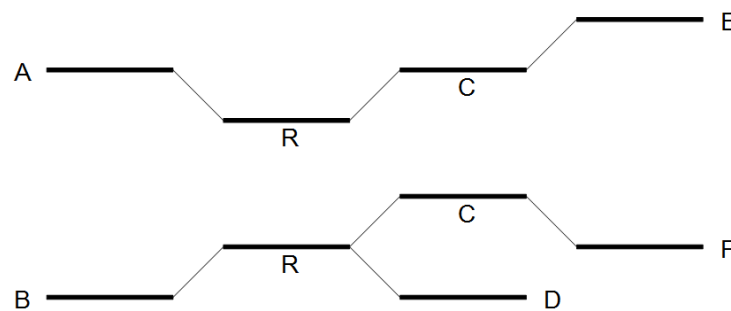


Figure 5.8: Resolving the repeat.

The algorithm for resolving repeats without conflict can be described the following way:

1. A node is selected as the window
2. The border is divided into sets using reads going through the window. If we have multiple sets, the repeat is resolved.

3. If the repeat cannot be resolved, we expand the window with nodes if possible and go to step 2.

The above steps are performed for every node.

Resolve repeats with conflicts

In the previous section repeats were resolved without excluding any reads that goes through the window. While this lead to a simpler graph, the graph will still contain artifacts, which have to be removed. The next phase removes most of these errors and is similar to the previous phase:

1. A node is selected as the initial window
2. The border is divided into sets using reads going through the window. If we have multiple sets, the repeat is resolved.
3. If the repeat cannot be resolved, the border nodes are divided into sets using reads going through the window where reads containing errors are excluded. If we have multiple sets, the repeat is resolved.
4. The window is expanded with nodes if possible and step 2 is repeated.

The algorithm described above is similar to the algorithm used in the previous section, except step 3 where the reads with errors are excluded. This is done by calculating an average $avg_1 = m_1/c_1$ where m_1 is the number of reads going through the window and c_1 is the number of distinct pairs of border nodes having one (or more) of these reads connecting them. A second average $avg_2 = m_2/c_2$ is calculated where m_2 is the number of reads going through the window having at least avg_1 or more reads connecting their border nodes and c_2 the number of distinct pairs of border nodes having avg_1 or more reads connecting them. Then, a read between two border nodes B and C is excluded if the number of reads going through B and C is less than or equal to $limit$ given by

$$limit = \frac{\log(avg_2)}{2} + \frac{avg_2}{16}$$

An example where we resolve a repeat with conflicts is given in 5.9 where we have a total of 21 reads going through the window with $avg_1 = 21/3 = 7$, $avg_2 = 20/2 = 10$ and $limit = 1/2 + 10/16 = 1.125$. Therefore all reads between border nodes B and C are excluded resulting in two sets of border nodes A, C and B, D . The resolved repeat is shown in figure 5.10.

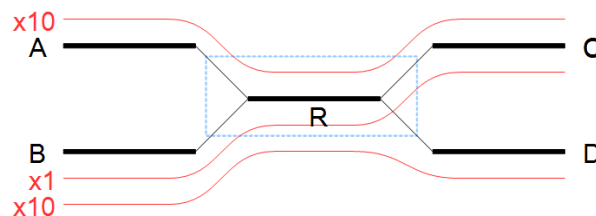


Figure 5.9: A repeat with conflicts.

5.1.2 Automatic paired distance estimation

The default behavior of the de novo assembler is to use the paired distances provided by the user. If the automatic paired distance estimation is enabled, the assembler will attempt to

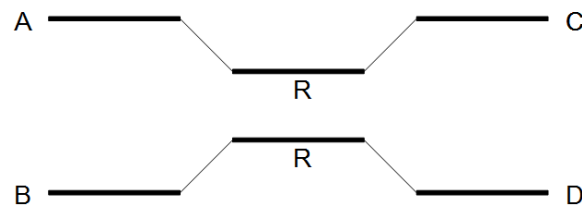


Figure 5.10: Resolving a repeat with conflicts.

estimate the distance between paired reads. This is done by analysing the mapping of paired reads to the long unambiguous paths in the graph which are created in the read optimization step described above. The distance estimation algorithm creates a histogram (H) of the paired distances between reads in each set of paired reads (see figure 5.11). Each of these histograms are then used to estimate paired distances as described in the following.

We denote the average number of observations in the histogram $H_{avg} = \frac{1}{|H|} \sum_d H(d)$ where $H(d)$ is the number of observations (reads) with distance d and $|H|$ is the number of bins in H . The gradient of H at distance d is denoted $H'(d)$. The following algorithm is then used to compute a distance interval for each histogram.

- Identify peaks in H as $\max_{i \leq d \leq j} H(d)$ where $[i, j]$ is any interval in H where $\{H(d) \geq \frac{H_{avg}}{2} | i \leq d \leq j\}$.
- For the two largest peaks found, expand the respective intervals $[i, j]$ to $[k, l]$ where $H'(k) < 0.001 \wedge k \leq i \wedge H'(l) > -0.001 \wedge j \leq l$. I.e. we search for a point in both directions where the number of observations becomes stable. A window of size 5 is used to calculate H' in this step.
- Compute the total number of observations in each of the two expanded intervals.
- If only one peak was found, the corresponding interval $[k, l]$ is used as the distance estimate unless the peak was at a negative distance in which case no distance estimate is calculated.
- If two peaks were found and the interval $[k, l]$ for the largest peak contains less than 1% of all observations, the distance is not estimated.
- If two peaks were found and the interval $[k, l]$ for the largest peak contain $< 2X$ observations compared to the smaller peak, the distance estimate is only computed if one peak was at a positive distance and the other was at a negative distance. If this is the case the interval $[k, l]$ for the positive peak is used as a distance estimate.
- If two peaks were found and the largest peak has $\geq 2X$ observations compared to the smaller peak, the interval $[k, l]$ corresponding to the largest peak is used as the distance estimate.

If a distance estimate for a data set is deemed unreliable, the estimate is ignored and replaced by the distance supplied by the user using the ‘-p’ option for that data set. The ‘-e’ option requires a file name argument, which is used to output the result of the distance estimation for each dataset. The output is a tab-delimited file containing the estimated distances, if any, and a status code for each data set. The possible status codes are:

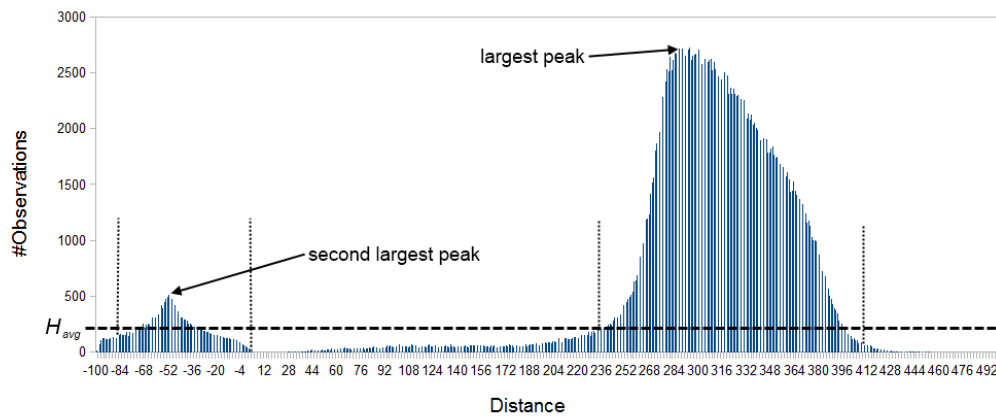


Figure 5.11: Histogram of paired distances where H_{avg} is indicated by the horizontal dashed line. There is two peaks, one is at a negative distance while the other larger peak is at a positive distance. The extended interval $[k, l]$ for each peak is indicated by the vertical dotted lines.

- **DISTANCE_ESTIMATED** The distance interval was estimated and used for scaffolding.
- **NO_DATA** No or very few reads were mapped as paired reads.
- **NOT_ENOUGH_DATA** Not enough reads were mapped as paired reads to give a reliable distance estimate.
- **NEGATIVE_DISTANCE** The distance interval was in the negative range which is usually caused by either wrong orientation of the reads or paired-end contamination in a mate-pair data set.
- **AMBIGUOUS_DISTANCE** Several possible distance intervals were detected but there was not enough data to select the correct one.
- **WRONG_DIRECTION** The orientation of the reads was not set correctly.

Only distance estimates with the DISTANCE_ESTIMATED status code is used for the assembly. In general we do not recommend that the automatic paired distance estimation is used on mate-pair reads where the expected distance is larger than 10Kbp as the distance estimate will often either fail or be inaccurate.

5.1.3 Optimization of the graph using paired reads

When paired reads are available, we can use the paired information to resolve large repeat regions that are not spanned by individual reads, but are spanned by read pairs. Given a set of paired reads that align to two nodes connected by a repeat region, the repeat region may be resolved for those nodes if we can find a path connecting the nodes with a length corresponding to the paired read distance. However, such a path must be supported by a minimum of four sets of paired reads before the repeat is resolved.

If it's not possible to resolve the repeat, scaffolding is performed where paired read information is used to determine the distances between contigs and the orientation of these. Scaffolding is only considered between contigs with a minimum length of 120 to ensure that enough paired

read information is available. An iterative greedy approach is used when performing scaffolding where short gaps are closed first, thus increasing the paired read information available for closing gaps (see figure 5.12).

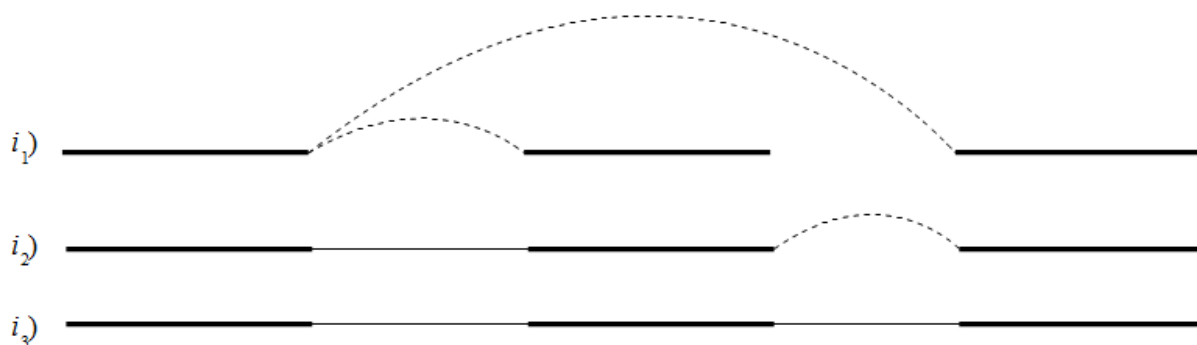


Figure 5.12: Performing iterative scaffolding of the shortest gaps allows long pairs to be optimally used. i_1 shows three contigs with dashed arches indicating potential scaffolding. i_2 is after first iteration when the shortest gap has been closed and long potential scaffolding has been updated. i_3 is the final results with three contigs in one scaffold.

Contigs in the same scaffold are output as one large contig with Ns inserted in between. The number of Ns inserted correspond to the estimated distance between contigs, which is calculated based on the paired read information. More precisely, for each set of paired reads spanning two contigs a distance estimate is calculated based on the supplied distance between the reads. The average of these distances is then used as the final distance estimate. The distance estimate will often be negative which happens when the paired information indicate that two contigs overlap. The assembler will attempt to align the ends of such contigs and if a high quality overlap is found the contigs are joined into a single contig. If no overlap is found, the distance estimate is set to two so that all remaining scaffolds have positive distance estimates.

Furthermore, Ns can also be present in output contigs in cases where input sequencing reads themselves contain Ns.

Please note that in CLC Genomics Workbench 6.0.1, Genomics Server 5.0.1, Assembly Cell 4.0.2 and all earlier versions of these products a performance optimization gave rise to Ns being inserted in certain non-scaffold regions, which in the current version can be solved with reads covering such specific regions.

Additional information on how paired reads have been used to in the scaffolding step can be printed by using `-f` to specify an output file for GFF or AGP 2.0 formatted annotations.

The annotations in table format can be viewed by clicking the "Show Annotation Table" icon () at the bottom of the viewing area. "Show annotation types" in the side panel allows you to select the annotation "Scaffold" among a list of other annotations. The annotations tell you about the scaffolding that was performed by the de novo assembler. That is, it tells you where particular contigs, those areas containing complete sequence information, were joined together across regions without complete sequence information.

For the GFF format there are three types of annotations:

- **Scaffold** refers to the estimated gap region between two contigs where Ns are inserted.
- **Contigs joined** refers to the join of two contigs connected by a repeat or another ambiguous

structure in the graph, which was resolved using paired reads. Can also refer to overlapping contigs in a scaffold that were joined using an overlap.

- **Alternatives excluded** refers to the exclusion of a region in the graph using paired reads, which resulted in a join of two contigs.

5.1.4 AGP export

The AGP annotations describe the components that an assembly consists of. This format can be validated by the NCBI AGP validator.

If the exporter is executed on an assembly where the contigs have been updated using a read mapping, the N's in some scaffolds might be resolved or removed. If the exporter encounters such a region, it will give a warning but not stop. If the exporter is executed on an assembly from a previous version of the GWB, it will often stop with an error saying that it encountered more than 10 N's which wasn't marked as a scaffold region. In this case the user would have to rerun the assembly with the current version of the de novo assembler if they wish to be able to export to AGP.

Currently we output two types of annotations in AGP format:

- **Contig** a non-redundant sequence not containing any scaffolded regions.
- **Scaffold** the estimated gap region between two contigs.

5.1.5 Bubble resolution

Before the graph structure is converted to contig sequences, bubbles are resolved. As mentioned previously, a bubble is defined as a bifurcation in the graph where a path furcates into two nodes and then merge back into one. An example is shown in figure 5.13.



Figure 5.13: A bubble caused by a heterozygous SNP or a sequencing error.

In this simple case the assembler will collapse the bubble and use the route through the graph that has the highest coverage of reads. For a diploid genome with a heterozygous variant, there will be a fifty-fifty distribution of reads on the two variants, and this means that the choice of one allele over the other will be arbitrary. If heterozygous variants are important, they can be identified after the assembly by mapping the reads back to the contig sequences and performing standard variant calling. For random sequencing errors, it is more straightforward; given a reasonable level of coverage, the erroneous variant will be suppressed.

Figure 5.14 shows an example of a data set where the reads have systematic errors. Some reads include five As and others have six. This is a typical example of the homopolymer errors seen with the 454 and Ion Torrent platforms.

When these reads are assembled, this site will give rise to a bubble in the graph. This is not a problem in itself, but if there are several of these sites close together, the two paths in the graph will not be able to merge between each site. This happens when the distance between the sites is smaller than the word size used (see figure 5.15).

```

:AGATGACCAGGGTGTGCGATAAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGATAAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGATAAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,
:AGATGACCAGGGTGTGCGAT - AAAAAATGCCAATCATCTGGAC,

```

Figure 5.14: Reads with systematic errors.

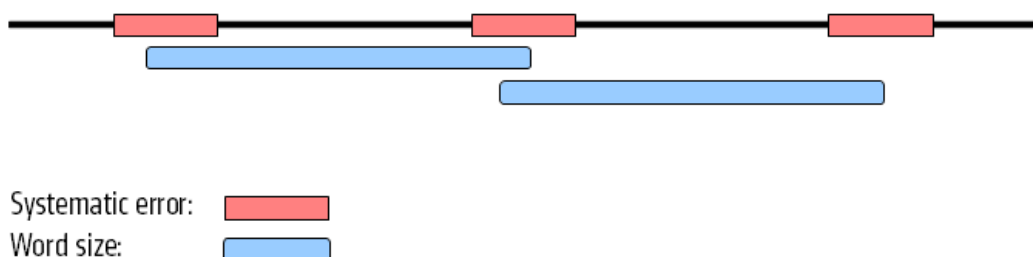


Figure 5.15: Several sites of errors that are close together compared to the word size.

In this case, the bubble will be very large because there are no complete words in the regions between the homopolymer sites, and the graph will look like figure 5.16.

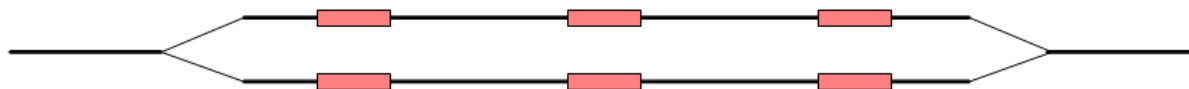


Figure 5.16: The bubble in the graph gets very large.

If the bubble is too large, the assembler will have to break it into several separate contigs instead of producing one single contig.

The maximum size of bubbles that the assembler should try to resolve can be set by the user. In the case from figure 5.16, a bubble size spanning the three error sites will mean that the bubble will be resolved (see figure 5.17).

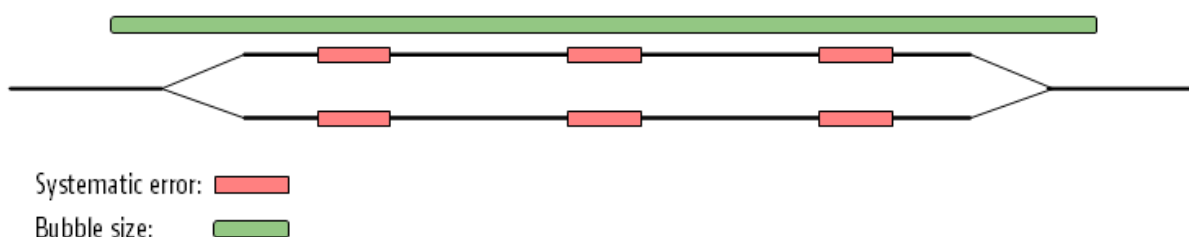


Figure 5.17: The bubble size needs to be set high enough to encompass the three sites.

While the default bubble size is often fine when working with short, high quality reads, considering the bubble size can be especially important for reads generated by sequencing platforms yielding long reads with either systematic errors or a high error rate. In such cases, a higher bubble size is recommended. For example, as a starting point, one could try half the length of the average

read in the data set and then experiment with increasing and decreasing the bubble size in small steps. For data sets with a high error rate it is often necessary to increase the bubble size to the maximum read length or more. Please keep in mind that increasing the bubble size also increases the change of misassemblies.

5.1.6 Converting the graph to contig sequences

The output of the assembly is not a graph but a list of contig sequences. When all the previous optimization and scaffolding steps have been performed, a contig sequence will be produced for every non-ambiguous path in the graph. If the path cannot be fully resolved, Ns are inserted as an estimation of the distance between two nodes as explained in section 5.1.3.

5.1.7 Summary

So in summary, the de novo assembly algorithm goes through these stages:

- Make a table of the words seen in the reads.
- Build a *de Bruijn* graph from the word table.
- Use the reads to resolve the repeats in the graph.
- Use the information from paired reads to resolve larger repeats and perform scaffolding if necessary.
- Output resulting contigs based on the paths, optionally including annotations from the scaffolding step.

These stages are all performed by the assembler program.

5.2 Randomness in the results

Different runs of the de novo assembler can result in slightly different results. This is caused by multi-threading of the program combined with the use of probabilistic data structures. If you were to run the assembler using a single thread, the effect would not be observed. That is, the same results would be produced in every run. However, an assembly run on a single thread would be very slow. The assembler should run quickly. Thus, we use multiple threads to accelerate the program.

The main reason for the assembler producing different results in each run is that threads construct contigs in an order that is correlated with the thread execution order, which we do not control. The size and "position" of a contig can change dramatically if you start building a contig from two different starting points (i.e. different words, or k-mers), which means that different assembly runs can lead to different results, depending on the order in which threads are executed. Whether a contig is scaffolded with another contig can also be affected by the order that contigs are constructed. In this case, you could see quite large differences in the lengths of some contigs reported. This will be particularly noticeable if you have an assembly with reasonably few contigs of great length.

We are working on addressing the fact that slightly different output is returned with different runs of the de novo assembler without appreciably affecting the speed of the assembler. For the moment, the output of runs may vary slightly, but the overall information content of the assembly should not be markedly different between runs.

5.3 Specific characteristics of CLC bio's algorithm

There are some advantages and some disadvantages of CLC bio's algorithm when compared to other programs such as Velvet [Zerbino and Birney, 2008] and SOAPdenovo [Li et al., 2010]. The advantages are:

- `clc_assembler` does not use as much RAM as other programs
- `clc_assembler` program is quite fast
- `clc_assembler` readily uses data from mixed sequencing platforms (Sanger, 454, Illumina, SOLiD¹, etc).

The reason that we are able to use little RAM compared to other programs is that we have a very strong focus on keeping the data structures very compact. When appropriate, we also use the hard drive for temporary data rather than using RAM.

The speed of the assembly program has been achieved by threading many parts of the program to use all available CPU cores. Also, some parts of the program are done using assembler code including SIMD vector instructions to get the optimal performance.

5.4 SOLiD data support in de novo assembly

SOLiD sequencing is done in color space. When viewed in nucleotide space this means that a single sequencing error changes the remainder of the read. An example read is shown in figure 5.18.

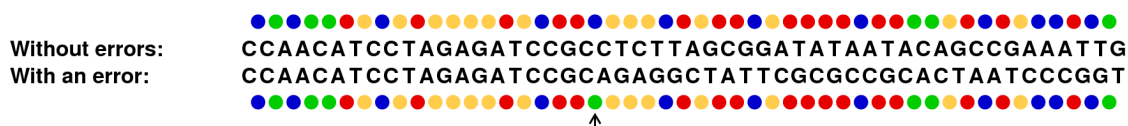


Figure 5.18: How an error in color space leads to a phase shift and subsequent problems for the rest of the read sequence

Basically, this color error means that C's become A's and A's become C's. Likewise for G's and T's. For the three different types of errors, we get three different ends of the read. Along with the correct reads, we may get four different versions of the original genome due to errors. So if SOLiD reads are just regarded in nucleotide space, we get four different contig sequences with jumps from one to another every time there is a sequencing error.

Thus, to fully accommodate SOLiD sequencing data, the special nature of the technology has to be considered in every step of the assembly algorithm. Furthermore, SOLiD reads are fairly short and often quite error prone. Due to these issues, we have chosen not to include SOLiD support in

¹See how SOLiD is supported in section 5.4

the first algorithm steps, but only use the SOLiD data where they have a large positive effect on the assembly process: when applying paired information. Thus, the `clc_assembler` program has a special option ("`-p d`") to indicate that a certain data set should be used only for its paired information. This option should always be applied to SOLiD data. It is also useful for data sets of other types with many errors. The errors might have the effect of confusing the initial graph building more than improving it. But the paired information is still valuable and can be used with this option.

5.5 Command line options

This section provides details of the command line options available for the `clc_assembler` command. As with all other programs in the CLC Assembly Cell, full usage information are given in Appendix B and can be viewed by executing the command without any arguments.

Note that you can use the `clc_sequence_info` program with the '`-n`' option to get statistics on the result of a de novo assembly. This is described in the **Viewing and reporting tools section of the manual**.

5.5.1 Specifying the data for the assembly and how it should be used

The parameters described in this section can be used multiple times within a single `clc_assembler` command.

Information provided using flags that can be specified multiple times in a single command pertains to **all inputs that follow** until you specify otherwise. For example, if you state you wish reads to be used for guidance only, all datasets entered via the command from that point forward will be used only for guidance, until a point in the command where you choose to indicate that reads should be used at all stages of the assembly. Please see the examples section below for further details on this.

Some of these options are most easily explained by considering the de novo as consisting of four steps:

1. Create fragments from reads.
2. Connect fragments based on all reads to form a graph.
3. Optimize the graph based on all reads.
4. Optimize the graph based on paired reads.

```
-p <par> / --paired <par>
```

Here, `par` is a set of parameters, which indicate the pair status of your data, and for paired data, the relative orientations and expected distances between members of the pair. Options after the `-p` flag are also used to indicate if paired data are in two files and also whether any particular data should be used only for its paired distance information during the final phase of the assembly.

Paired status

Data are assumed to be paired by default. To indicate that the data contain single reads, that is, that the data are not paired, the single string value "no" is given. I.e. "-p no" followed by the name of the read file.

Paired information

For paired data, `par` consists of four strings: `<mode>` `<distance_mode>` `<min_dist>` `<max_dist>`

`mode` indicates the relative orientation of the reads in a pair set. These can be **ff**, **fb**, **bf** or **bb**. These are used for "forward-forward" reads, "forward-reverse" reads, "reverse-forward" reads and "reverse-reverse" reads².

`distance_mode` indicates the point on paired reads from which the distance measure you provide is taken. The options are **ss**, **se**, **es** or **ee**. These mean "start-start", "start-end" reads, "end-start" reads and "end-end".

`<min_dist>` and `<max_dist>` give the minimum and maximum distance range for the distances between the pairs. Where to take the start and end points of this distance range is what is specified by the `distance_mode` described above.

So, `-p fb ss 180 250` would indicate that the reads are inverted and pointing towards each other, that the distance range includes the sequences of both the reads as well as the fragment between them, and that the distance range is between 180 and 250 bases.

How data should be used in the assembly

`-p d <mode> <distance_mode> <min_dist> <max_dist>` An additional option, **d**, can be added to the `-p` flag to indicate that the reads should only be used in the fourth step of the assembly, as listed at the top of this section. That is, the paired distance information associated with these reads is used when optimizing the graph. The sequence information itself is not used towards the assembly result. This is useful for data such as SOLiD, where the reads are quite short and may contain many errors. Such reads would not, by themselves, be of great help in building and optimizing the graph initially, but the paired information associated with them can be valuable during the final graph optimization stage.

Please also refer to the information about the `-g` option in the section below.

`-g <mode>/ --fragmentmode <mode>`

You might choose to use this option with longer reads that are expected to contain many errors, for example, in the case of 454 reads.

Here, `mode` can be one of two values: "use" and "ignore". The default is that all data is run with `-g 'use'`.

Providing the `"-g ignore"` before the name of a read set indicates that this read set (and all others after this point in the command, until any point where the `"-g use"` option is entered in the command) is **not** used in the first step of the assembly, as described above, where fragments are generated from the reads. In other words, read sets following `"-g ignore"` in the command, are used in steps 2, 3 and 4 as listed above, where the graph is determined and optimized.

`-n / --no-scaffolding` Pair distance information is used for the creation of contig sequences, but no scaffolding (i.e. making associations between contigs) is performed.

²The letter "b" in the mode strings refers to the word "backwards", but the more common word to use to describe this relative orientation is "reverse".


```
-q / --reads
```

This flag indicates that the information that follows are read filenames. Read files can be in fasta, fastq or sff format.

```
-i <file1> <file2> / --interleave <file1> <file2>
```

To input paired read data that is in two files, where one read of each pair is in one file and the other of each pair is in the second file, the pair of file names should be provided after the -i flag.

Read files for paired data entered without the -i flag are assumed to contain interleaved pairs. That is, the first sequence is the first member of a pair, the second sequence is the second member of that same pair, the third sequence is the first member of the second pair, the fourth sequence is the second member of the second pair, and so on.

Mixed files, that is, single input files that contain both paired and unpaired reads, cannot be used as input with the `clc_assembler` command, unless the intention is to treat all reads as single reads.

5.5.2 Specifying information for the assembly

Options in this section can alter the results of the assembly. The **How it works** section of the manual gives further details that are relevant to how these settings may affect an assembly.

```
-w <n> / --wordsize <n>
```

Set *n* to be the word size for the de Bruijn graph. The default is based on the size of the input as described in the **How it works** section of the manual.

```
-b <n> / --bubblesize <n>
```

Set the maximum bubble size for the de Bruijn graph. The default is 50 bases.

```
-e <file> / --estimatedistances <file>
```

This setting estimates the distances for paired reads as observed within unscaffolded contigs. These distances are then used in the scaffolding step. If multiple sets of paired data have been input, the distances are estimated separately for each data set. The distances calculated will be saved to the file specified as the argument to this parameter.

When this flag is used, the program will aim to identify tight distance intervals from areas containing a substantial number of the mapped reads for each dataset.

There are situations where it is not possible to estimate accurate paired distances from the data, such as:

- No best candidate interval for distance estimation can be specified as the two best candidate intervals to be used for estimating the distances differ only by a factor of two in the number of pairs they contain.
- The best interval suggests a negative average distance for the paired reads in a dataset.
- More than half the reads have the wrong relative orientation.
- The best interval contains less than 1% of the mapped reads in that dataset.

If it is not possible to estimate an accurate distance from the data for any particular paired read set, then the original paired distance entered as part of the parameter settings associated with the **-p flag** will be used. Errors and warnings associated with such situations will be written to the file specified with the **-e** parameter.

5.5.3 Specifying information about the outputs

-o <file> / --output <file> Give the name for the file that will contain the contigs, in fasta format, that are assembled.

This parameter is required.

-m <n> / --min-length <n> Set the minimum length for contigs to be output from the assembly process. The default value is 200 bases.

-f <file> / --feature_output <file>

Providing this option indicates that the annotations associated with scaffolding should be output.

The output can be in GFF format (the default), or in AGP format. The file suffix you provide specifies the output format. For AGP format, use **' .agp '**. For gff format, you can use whatever name you like, but it would be usual to use the filename suffix **' .gff '**.

5.5.4 Other options

--cpus <n> Specify the maximum number of cpus that should be used by the assembly process. If not set explicitly, the process assumes it has access to as much of your computer's cpus as it needs.

-v / --verbose: This option specifies that verbose reporting should be turned on. This results in various information being written to the terminal when the process is running, include the word size used if you have chosen to allow that to be determined by the assembly program.

5.5.5 Example commands

The command below is a mixed assembly, where a set of paired reads, available in two files with one member of each pair in each file, is used for all stages of the assembly, and a set of 454 data is used for guidance only. The output file, containing the fasta formatted assembled contigs, will be called **myContigs.fasta**.

Note the use of the **-p no** before the non-paired, 454 data. This ensures that the assembler knows that the data about to be entered are not paired. It effectively overwrites the earlier **-p** information provided earlier in the command.

```
clc_assembler -o myContigs.fasta -p fb ss 200 400 -q -i pairedRead-member1.fastq
               pairedRead-member2.fastq -g ignore -p no -q 454read.sff
```

The above command is equivalent to:

```
clc_assembler -o myContigs.fasta -g ignore -p no -q 454read.sff -g use
               -p fb ss 200 400 -q -i pairedRead-member1.fastq pairedRead-member2.fastq
```

Examples of undesirable commands

A command like the following would imply that the data in 454reads.sff was paired, in an interleaved file, with the reads having relative orientation of forward-reverse, and a paired distance range of 200 to 400 bases. That is, the earlier information provided to the -p parameter would be used for all the following data entered in the command.

```
clc_assembler -o myContigs.fasta -p fb ss 200 400 -q -i pairedRead-member1.fastq  
pairedRead-member2.fastq -g ignore -q 454read.sff
```

A command like the following would fail because all reads are now used in a guidance only role. This leaves no reads being used to create the graph fragments in the initial stage. The reason here is that the -g ignore parameter is given early in the command and because not -g use parameter is entered later, all read sets are ignored for the building of the fragments.

```
clc_assembler -o myContigs.fasta -g ignore -p no -q 454read.sff -p fb ss 200 400 -q  
-i pairedRead-member1.fastq pairedRead-member2.fastq
```

Chapter 6

Viewing and reporting tools

6.1 Mapping Viewer

The mapping 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 to interesting part and center horizontally.
b	Scroll right to interesting part and center horizontally.
c	Toggle color scheme.
m	Toggle position marks.
e	Toggle how to show unaligned ends.
r	Toggle between contigs.
j	Toggle joint read view.
p	Move to same position as for last contig.
h	Show help screen.
s	Search for a sequence in the reference.
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 6.1-6.4 show some screen shots and examples.

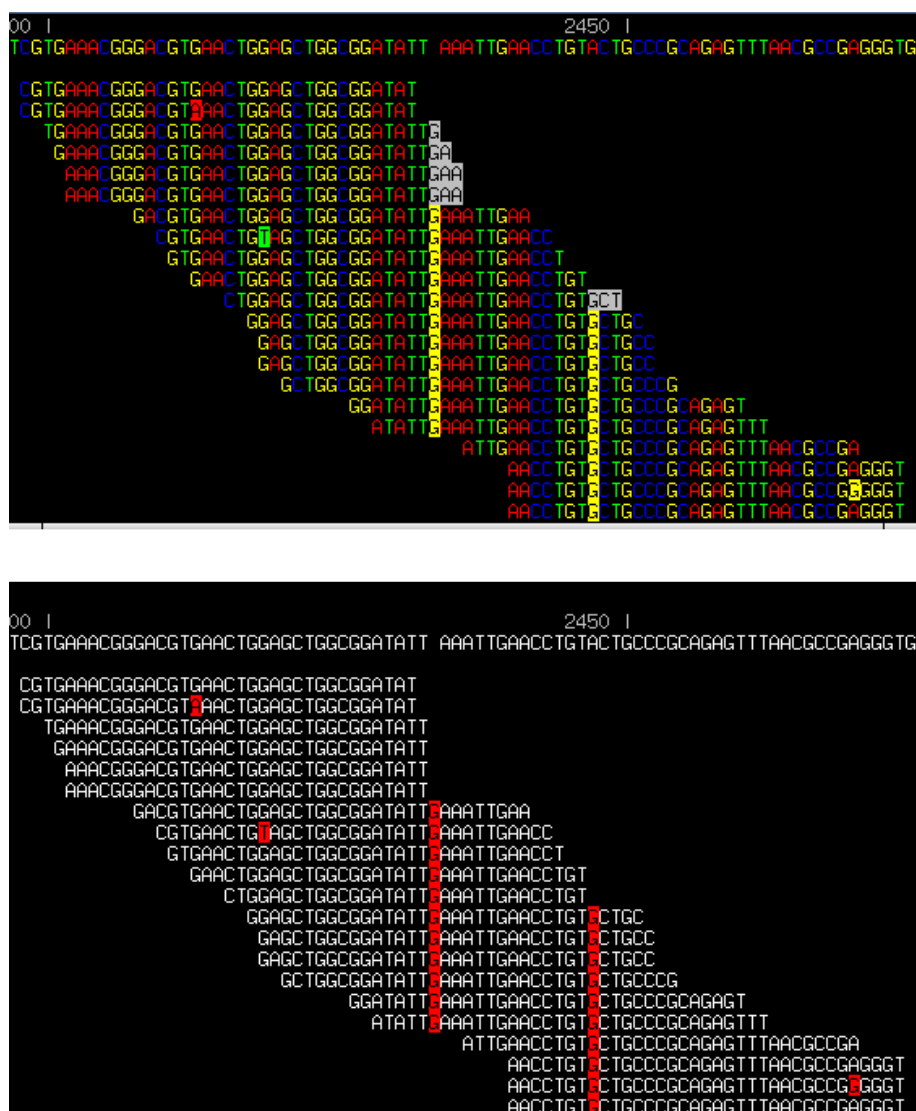


Figure 6.1: Two screen shots from the Mapping viewer. Top) Residue coloring. Residues differing from the reference are highlighted. The first column of highlighted G's is an insertion, the second is a mutation (the reference residue is A in that position). The reversed gray residues at the end of some 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.

6.2 The `clc_sequence_info` Program

The `clc_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
```

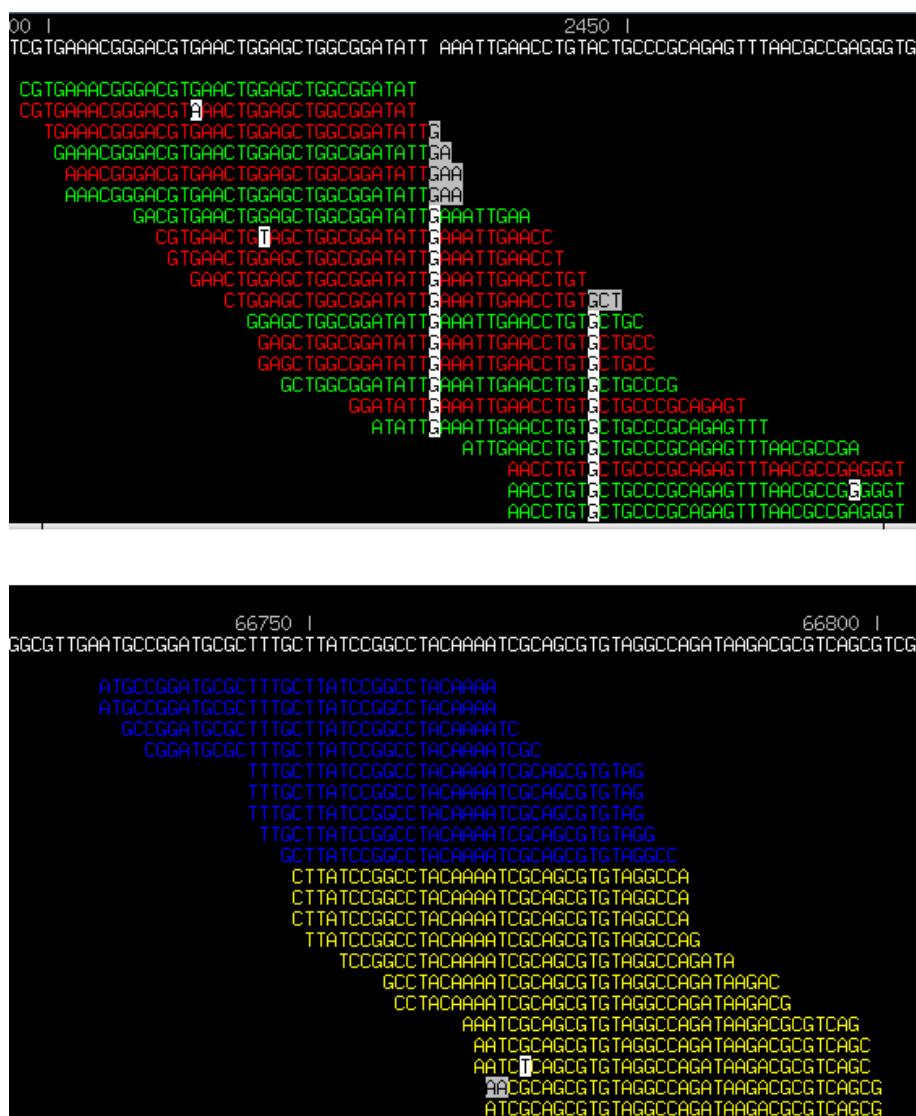


Figure 6.2: Another screen shot from the Mapping viewer. Top) reads are colored according to the direction. Green is forward, red is reverse. Bottom) a yellow color indicate reads that map uniquely while a blue color indicate reads which map ambiguously, i.e. they map with the same score at multiple positions which often indicate a repetitive region.

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. The N50 value means that the sum of sequences of this length or longer is at least 50% of the total length of all sequences. This is useful to get a quick quality overview of a de novo assembly.



Figure 6.3: 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.

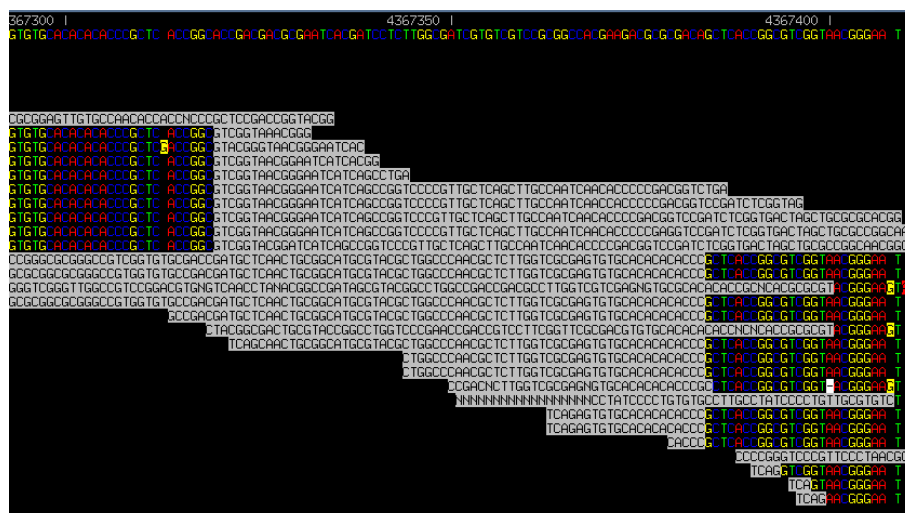


Figure 6.4: A screen shot with 454 sequencing data. This is how a genomic rearrangement looks in a reference assembly. Suddenly the reads do not match any more, and later another set of reads abruptly start matching. These reads may actually be very distant in the real genome (as opposed to the reference).

Use the '-c' option to disregard all sequences under a certain length from being considered in the statistics. This is sometimes useful for analyzing de novo assembly results, where short sequences may not be of interest.

Further details are available in Appendix B.

6.3 The `clc_mapping_table` Program

The `clc_mapping_table` program takes a single `cas` file as input and prints assembly information for each read. By default, `clc_mapping_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 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 AAACCTCCTTTTCAGTGGGAAATTGTGGGGCAAAGTG 89611      coli
      |||||||||||||||||||||||||||||||||||
      AAACCTCCTTTTCAGTGGGAAATTGTGGGGCAAAGTG      reverse read

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

4829 ATCCAGGCGAATATGGCTTGTTCCTCGGCACC      4860      coli
      ||||||||||||||||||| |||||||||||
      ATCCAGGCGAATATGGCTTTTTCCTCGGCACCCCG      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 ACTGAGTGATTGATTGCGGAGCCACATACTGTGGA 38122      coli
      ||||||||||||||||||||||||||| ||||
      ACTGAGTGATTGATTGCGGAGCCACATACTGTGGA      read

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

81872 GCATCCAGCACTTTTCAGCGCCTGGGTCATCACTTC 81906      coli
      |||||||||||||||||||||||||||||||||||
      GCATCCAGCACTTTTCAGCGCCTGGGTCATCACTTC      reverse read

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

81685 TTCTGGTTGCTGGTCTGGTGGTAAATGTTCCCACT 81719      coli
      |||||||||||||||||||||||||||||||||||
      TTCTGGTTGCTGGTCTGGTGGTAAATGTTCCCACT      read

```

Note! The positions in the standard output assumes the reference sequence starts at 0. However, the ‘-a’ option assumes that the reference starts at 1. This is due to the fact that the ‘-a’ option is intended to produce human-readable output whereas the standard option is intended to be used by computer programs.

If multiple hit positions are recorded in the cas file (using the -t option when running the assembly), running the assembly_table with the ‘-m’, the output looks like this:

```

480      35      0      35      0      19625      19660      0      1
481      35      0      35      0      19815      19850      1      1
482      35      0      35      0      2512501      2512536      1      3
-        35      0      35      0      607436      607471      1      3
-        35      0      35      0      15593       15628      1      3
483      35      0      35      0      2512321      2512356      0      3
-        35      0      35      0      607256      607291      0      3
-        35      0      35      0      15413       15448      0      3
484      35      0      35      0      14374       14409      1      1
485      35      0      35      0      14194       14229      0      1

```

Reads 482 and 483 map in three places and they are all printed. The order is random, which has the advantage that using the first match (according to output order) is the same as using a random match. For paired data (like these), the matches are in the same order for two *paired* reads. So the first match for 482 belongs with the first match for 483, etc.

For alignment output in `clc_mapping_table` with `"-m"`, it looks like this:

```
SLXA-EAS1_89:1:1:980:945/1 has 1 paired match with a score of 35: (alignment 1)

    19626 AGCTCCCCCAAAGTTAAGGTGGGGGAGATAGATTA 19660      coli
          |||||||||||||||||||||||||||||||||||
          AGCTCCCCCAAAGTTAAGGTGGGGGAGATAGATTA      read

SLXA-EAS1_89:1:1:980:945/2 has 1 paired match with a score of 35: (alignment 1)

    19816 GATAGTGTTTTATGTTTCAGATAATGCCCGATGACT 19850      coli
          |||||||||||||||||||||||||||||||||||
          GATAGTGTTTTATGTTTCAGATAATGCCCGATGACT      reverse read

SLXA-EAS1_89:1:1:307:821/1 has 3 paired matches with a score of 35: (alignment 1)

    2512502 CGGCCCCGGGGGGATGTCATTACGTGAAGTCACTG 2512536      coli
          |||||||||||||||||||||||||||||||||||
          CGGCCCCGGGGGGATGTCATTACGTGAAGTCACTG      reverse read

SLXA-EAS1_89:1:1:307:821/1 has 3 paired matches with a score of 35: (alignment 2)

    607437 CGGCCCCGGGGGGATGTCATTACGTGAAGTCACTG 607471      coli
          |||||||||||||||||||||||||||||||||||
          CGGCCCCGGGGGGATGTCATTACGTGAAGTCACTG      reverse read

SLXA-EAS1_89:1:1:307:821/1 has 3 paired matches with a score of 35: (alignment 3)

    15594 CGGCCCCGGGGGGATGTCATTACGTGAAGTCACTG 15628      coli
          |||||||||||||||||||||||||||||||||||
          CGGCCCCGGGGGGATGTCATTACGTGAAGTCACTG      reverse read

SLXA-EAS1_89:1:1:307:821/2 has 3 paired matches with a score of 35: (alignment 1)

    2512322 GGTATTACGCCTGATATGATTTAACGTGCCGATGA 2512356      coli
          |||||||||||||||||||||||||||||||||||
          GGTATTACGCCTGATATGATTTAACGTGCCGATGA      read
```

Options for the `clc_mapping_table` are described in Appendix B.

6.4 The `clc_mapping_info` Program

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

General info:

```
Program name      clc_mapper
Program version   1.00.31043
```

```

Program parameters  -o tmp.cas -d data/paired.fasta -q data/paired_reads.fasta -m

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 distances using the `clc_mapping_info` program. This is done with the standard ‘-p’ option and results in an output like this:

```

Paired reads 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 analysis `clc_mapping_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 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 `clc_submapping` to make an assembly with only the paired data before analyzing.

When a dataset contains paired data of unknown distances, a good approach is to make an initial reference assembly without using paired information. Then the `clc_mapping_info` program can be used to investigate the paired distance properties of the data using wide limits for the distances. Finally, a reference assembly run can be performed with the estimated paired 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 distance properties of the results, but a smaller fraction of the reads might match.

Further details can be found in Appendix [B](#).

Chapter 7

Assembly post-processing tools

This chapter covers tools included in CLC Assembly Cell that can be used to further process cas assembly files.

7.1 The `clc_change_cas_paths` Program

Cas files contain information about the files containing the reference and read data used in the mapping. This includes the paths to those files. The `clc_change_cas_paths` program allows you to change the file names and paths for the read and reference data files referred to in a cas file. This is useful if you have moved the sequence data files, and also can be useful when sharing cas files and the constituent data (reads and references) with others.

In addition to changing the locations of the data files, this tool can also be useful for changing relative file paths to absolute paths or vice versa.

The data file information is provided to this tool using the same parameters used with the `clc_mapper` tool. That is, using the `-d`, `-q`, and `-i` options. The input cas file is specified with the `-a` option and the cas file to be generated, containing the updated file locations, is specified with the `-o` option.

By default, the `clc\_change\_cas\_paths` program compares the sequence files to make sure they contain the same data. If the original read files (in their original location) no longer exist, or if you are certain you are working with the correct data files and wish to skip this check, you can use the `n` flag.

For the `clc\_change\_cas\_paths` command to succeed, you must provide the (new) locations of the data in the same order as they were used when creating the cas file originally. If you do not already know what this order is, then you can find out by running the `clc\_mapping\_info` program on the cas file. Providing the `s` flag to that tool skips checks of the contigs and thus can save some time here.

```
clc_mapping_info -s myassembly.cas
```

The order of the reads and reference files listed in that output is the same order they should be provided to the `clc\_change\_cas\_paths` tool.

Generally speaking it is a good idea to use different file names for the input cas file and the output cas file so the original is retained as backup. However changes can be made in place if

the same cas file name is used for both the input and output.

For a full list of parameters, please refer to Appendix B.

7.2 The `clc_filter_matches` Program

The `clc_filter_matches` program removes matches of low similarity from a cas file. The limits for low similarity are expressed as a minimum sequence similarity required over a minimum fraction of the read length. These parameters are set using the `-s` and `-l` options, respectively. The limits work just like for `clc_ref_assemble_long`.

For further details see Appendix B.

7.3 The `clc_find_variations` Program

This program makes it possible to detect variants between a reference sequence and the reads. It operates on a cas file produced by the reference assembly programs.

It makes a new consensus sequence file containing all the original data but with changes made so the references reflect the read sequences of an assembly. The new consensus 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.

There is an option `-c` to determine minimum coverage for read differences to be reported.

The `-r` option will determine how conflicts in the reads should be resolved in the consensus sequence. The default is a simple vote (the majority of the reads determine the consensus base), but it is also possible to get ambiguity characters as well (note that this will mean that sequencing errors will also reflect in the consensus sequence, so it should be used with caution).

Using the `w` option, the program will output a list of zero-coverage regions in the assembly.

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 `clc_find_variations` may be in order.

There is also an option `-i` that will ignore insertions and deletions completely. This can be an advantage when looking for variations in data sets from sequencing platforms producing many indel sequencing errors.

See Appendix B for further details.

7.4 The `clc_join_mappings` Program

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

Options for `clc_join_mappings` can be found in Appendix B.

7.5 The `clc_submapping` Program

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

Options for `clc_submapping` can be found in Appendix B.

7.5.1 Specifying Mapping Files

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

7.5.2 Extracting a Subset of Reference Sequences

The ‘-s’ option is used for making a new mapping with only matches to a single reference sequence. The ‘-d’ option makes a new mapping 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. You can use `clc_mapping_info` to see the contents of the input mappings is needed.

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

7.5.3 Extracting a Part of a Single Reference Sequence

If a single reference sequence is specified using the ‘-s’ option or if the input mapping contains only a single reference sequence, the ‘-b’ option may be used to specify a position range to extract. The output mapping 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.

7.5.4 Extracting Only Long Contigs (Useful for De Novo Assembly)

If you map reads against contigs created by de novo assembly, it can be useful to extract the mappings of the longest contigs only. This can be done using the ‘-r’, specifying the minimum length of the reference sequence.

7.5.5 Extracting a Subset of Read Sequences

Using the ‘-q’ option, you can make a mapping file with only the reads from one of the read files. The read file is specified by its number in the input mapping file. If reads are interleaved, the output file 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 although the full mapping contains reads from several experiments.

7.5.6 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.

7.5.7 Output Reference File

By default, the output mapping refers to one or all of the reference files in the input cas file. 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 mapping. The new mapping automatically refers to this reference sequence file. This is typically useful when selecting only a single reference sequence and the input alignment contains many reference sequences in the same file. That way the output only contains the relevant reference sequence instead of many references with no matches. It makes the output easier and faster to work with.

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

7.5.8 Output Read File

By default, the output refers to one or all of the read files in the input. 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 automatically refers to this new read file instead of the originals.

This is very useful when making a sub mapping 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 mapping, making subsequent analyses more efficient.

When the reads are from a paired experiment, the read mapper expects 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 `clc_submapping` program, the paired read order is disrupted. Because of this, the ‘-p’ option should be used when the reads are from a paired experiment. It works by retaining reads that do not match the `clc_submapping` 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.

7.5.9 Handling of non-specific matches

If an assembly contains non-specific match reads and a sub mapping 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 `clc_submapping` program is meant to make it simpler to study a small region of a large mapping, so the original characteristics of the larger mapping are kept.

7.6 The `clc_unmapped_reads` Program

This program extracts the unmapped read sequences from a mapping. They are output in fasta format. By default the only output sequences are the ones that do not match at all. Using the options it is also possible to output the unaligned ends of reads. A minimum length of unmapped sequences 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 mapping. Sometimes, performing de novo assembly on these unmapped reads may be useful to determine their source. It could, for example, be mitochondrial DNA or vector sequence contamination. See Appendix B for further details.

7.7 The `clc_unpaired_reads` Program

Create a file containing the reads that mapped, but where the read pair did not meet the requirements to be considered an intact pair within the mapping. Reasons for failing to be considered an intact pair include:

- reads of the pair mapped with incorrect relative orientations
- reads of the pair mapped at a distance outside the expected range
- reads of the pair mapped to different reference sequences
- one of the two reads of the pair did not map to any of the references

Further details can be found in Appendix B.

7.8 The `clc_agp_join` Program

When using the `-f` option in the de novo assembler for outputting scaffold annotations in AGP format, scaffolded contigs are output as individual contigs and not as a single scaffold with N's inserted in between contigs. The AGP file which is generated contain information on scaffolded contigs and the size of gaps that separate contigs in a scaffold. This program takes a list of contigs and an AGP file as input and output a list of contigs where each contig represents a scaffold where contigs are separated with a number of N's corresponding to the gap size. That is, the output of this program is identical with the default output of the de novo assembler.

Further details can be found in Appendix B.

Chapter 8

Sequence preparation tools

8.1 The `clc_adapter_trim` Program

Trims adapters from sequences.

Many sequencing technologies may leave whole or partial adapter or linker sequences in the reads for various reasons. The `clc_adapter_trim` program is used to find and remove such adapters from the reads.

The `clc_adapter_trim` tool identifies likely adapters in reads and removes them. To account for sequencing errors known adapter sequence are aligned with each read. Matching positions in these alignments score 1, while each mismatch costs 2 and each gap costs 3. By default, a region that aligns with a score of at least 10 is considered a possible adapter region. The `-c` option can be used to change the default score threshold of 10.

By default, the `clc_adapter_trim` tool trims bases towards the 3' end of reads using the following approach:

- For any read with only one region scoring equal to or higher than 10, that region is considered to be an adapter. The adapter region and all bases towards the 3' end are removed.
- For any read where there is no alignment to a known adapter sequence scoring 10 or greater, the 3' end of the read is checked for any possible sign of adapter. If found, such bases will be removed. For example if a single nucleotide at the end of a read is identical to the first nucleotide of the adaptor it will be removed since it may have come from an adaptor. The end match is defined as the longest match at the end of the read having a non-negative score when aligned to the adapter.
- For any read with more than one region aligning with a score equal to or higher than 10, the region closest to the 3' end is considered to be an adapter and removed along with any bases towards the 3' end.

With the `-e` option it is possible to change the behavior so the reads are trimmed towards the 5' end of reads, rather than the 3' end. In this case, the conditions described above are the same, with the directionality of the actions reversed.

The `clc_adapter_trim` program allows fine control over the behavior of the tool. For example,

- Should read sequences before or after the adapter be kept. The default action is to keep the sequence before the adapter, but this can be altered using the `-e` option.
- Which reads should be kept. For example, reads where adapter was found are kept by using the `-t` or `-f` options, and reads where the adapter was not found are kept by using the `-u` or `-g` options.
- Which adapter sequences should be searched for. One or several adapter sequences can be used, using the `-a` and `-d` options, and for paired data, different adapters can be used for the first and second reads in the pairs by using the `-j` and `-k` options.

For adapter sequences given with the `-a`, `-j` or `-k` options, the reverse complement of the adapter sequences is automatically added to the list of adapters to search for.

Further details can be found in Appendix B.

8.2 Quality trimming

The `clc_quality_trim` program is used to trim sequencing reads for low quality. The idea is to trim the reads at one or both ends so that only a region of high-quality bases are left. This is done by specifying a threshold value (using the `-c` option) for low-quality base calls. The default value is 20, which means that quality scores below 20 are marked as low quality. Since it is often not desirable to discard a high-quality region because of one isolated low-quality base, you can specify the fraction of low-quality bases allowed in a region using the `-b` option. The default value is 0.1 meaning that up to 10 % low-quality bases are allowed. The trim algorithm will then, for each read, find the longest region that fulfills these thresholds. Note that in some situations the full read will be discarded if no good quality regions can be found.

For paired data, two separate files are specified as output: one for the intact pairs (use the `-p` option for this output file) and one for the single reads whose mate was discarded during trimming (use the `-o` option for this output file).

There are other options to refine the quality trimming even more (see Appendix B).

8.2.1 Fastq quality scoring

The `clc_quality_trim` program uses an offset value of 64 by default for Illumina data (fastq). You will need to know what version of the Illumina pipeline was used on your original data and set the appropriate offset accordingly, using the `-f` option. The offset values for standard formats, which are also used in the CLC Workbench, are:

- NCBI/Sanger & Illumina Pipeline 1.8 and later: 33
- Illumina Pipeline 1.2 and earlier: 55
- Illumina Pipeline 1.3 and 1.4: 64
- Illumina Pipeline 1.5 to 1.7: 66

Hence, for example, the following command would stipulate a minimum quality value of 10, with a maximum tolerance of 10% bad bases, and an offset of 33. The program will return the longest

region for each read that fulfills these criteria. Reads that do not have regions that make the criteria cutoffs will be discarded.

```
quality_trim -r smallfile.fastq -c 10 -f 33 -o smallfile_trimmed.fasta
```

8.3 The `clc_remove_duplicates`

The duplicate read removal tool is designed to filter out duplicate reads. This tool is specifically well-suited to handle duplicate reads coming from PCR amplification errors which can have a negative effect because a certain sequence is represented in artificially high numbers.

The purpose of the tool is to reduce the data set to include only one copy of the duplicate sequence. The challenge is to achieve this without removing identical or almost identical reads that would arise from high coverage of certain regions, e.g. repeat regions or highly expressed exons from transcriptome sequencing. The algorithm takes sequencing errors into account (see below).

The approach taken here is based on the raw sequencing data without any knowledge about how they map to a reference sequence. This means that this is well-suited for both de novo assembly and resequencing purposes.

8.3.1 Looking for neighbors

An example of a read duplication can be easily distinguished when mapping reads to a reference sequence as shown in figure 8.1.

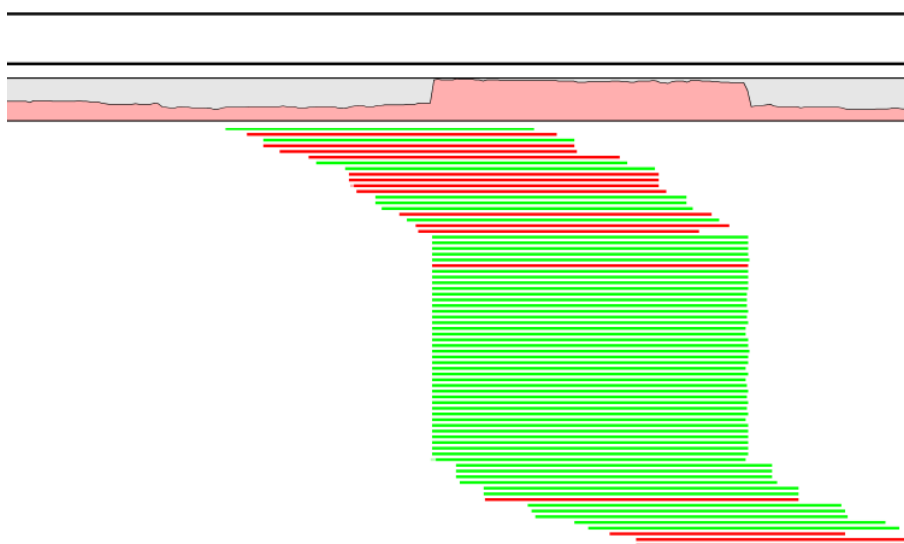


Figure 8.1: *Mapped reads with a set of duplicate reads, the colors denote the strand (green is forward and red is reverse).*

The typical signature is a lot of reads starting at the same position, resulting in a sudden rise in coverage, and all reads have the same orientation (denoted by the color). In a normal data set, you will also see fluctuations in coverage as shown in figure 8.2, but they lack the two important

features of duplicate reads: they do not all start at exactly the same position, and they are from different strands.

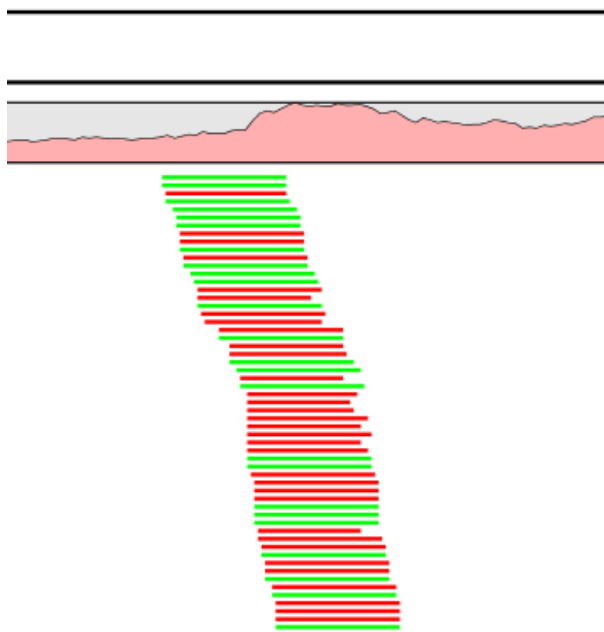


Figure 8.2: *Rise in coverage.*

The duplicate reads tool works directly on the sequencing reads, so there is no need to map the data to a reference genome first (figures 8.2 and 8.2 show the reads mapped for illustration purposes). In short, the algorithm will look for "neighboring" reads (i.e. reads that share most of the read sequence but with a small offset) and use these to determine whether there is generally high coverage for this sequence. If this is not the case, the read in question will be marked as a duplicate.

For certain sequencing platforms such as 454, the reads will have varying lengths - and this is taken into account by the algorithm as well.

SOLiD data is also supported, taking into account the dual base coding features of SOLiD data (color space).

8.3.2 Sequencing errors in duplicates

It is important to take sequencing errors into account when filtering duplicate reads. Imagine an example with 100 duplicates of a read of 100 bp. If there is a random 0.1 % probability of a sequencing error, it means that 10 of these reads have an error. If the algorithm only removed the 90 identical reads, there will be 10 reads left with sequencing errors. This is a big problem since the correct sequence is only represented once.

To address this issue, the duplicate read removal program accounts for sequencing errors when it identifies duplicate reads. Specifically, reads are considered duplicates if:

- they share a common sequence of at least 20 bases¹ in the beginning, or at any of four other regions distributed evenly across the read, and

¹For paired reads, this is only 10 bases.

- the rest of the read has an alignment score above 80 % of the optimal score, where the optimal score is what a read would get if it aligned perfectly to the consensus for a group of duplicates.

Please note that these thresholds for similarity are not enough for reads to be marked as duplicates - they just define how different reads are allowed to be and still be considered duplicates. Rather, the duplicates are identified as explained in section 8.3.1.

8.3.3 Paired data

For paired data, the assumption is made that if both parts of the pair share the same sequence, they are duplicates, and only one copy of the pair is left in the output. Figure 8.3 shows an example of a paired read duplicate.

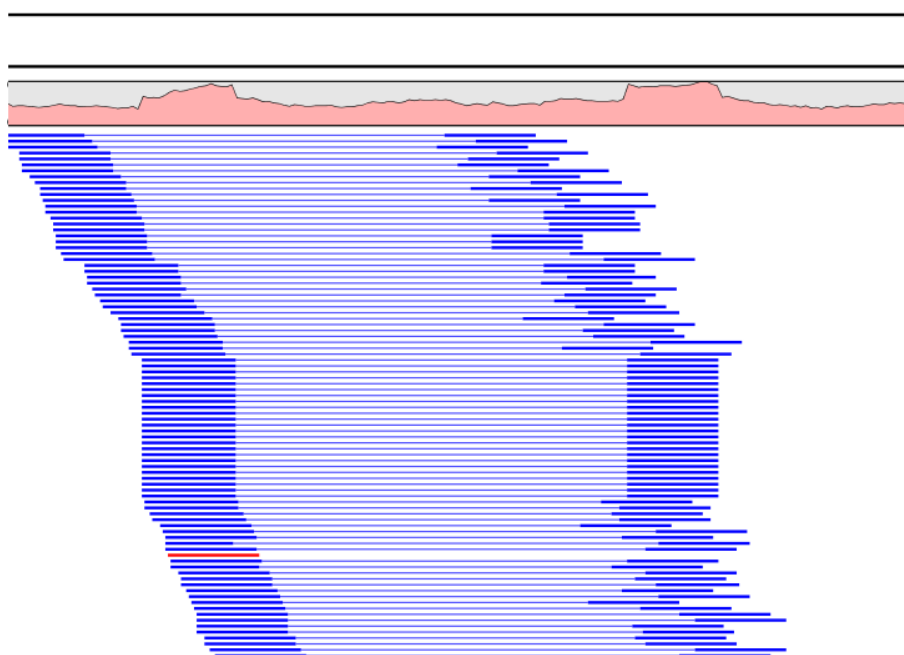


Figure 8.3: Paired reads with identical starting positions.

The algorithm also takes sequencing errors into account when filtering out paired data.

8.3.4 Known limitations

In its current version, the duplicate read removal has a limitation when there are duplicate reads that contain several alleles. The algorithm will identify that there are duplicate reads to be removed, but it is not able to distinguish between sequencing errors and true variation in the reads. So if you have a heterozygous SNP in such an area, you may risk that only one of the alleles are preserved. We are working on improving the algorithm to handle this.

8.3.5 Example of duplicate read removal

The following command outputs all reads to `coli_reads_nodup.fa` that are not identified as duplicates from the paired reads contained in `coli_reads_1/2.fa` and `coli_reads_2/2.fa`.

```
clc_remove_duplicates -p -r -i coli_reads_1/2.fa coli_reads_2/2.fa
                     -o coli_reads_nodup.fa
```

The program runs only in a single thread and for large data set it would be convenient to run multiple instances at the same time for each data file.

8.4 The `clc_sample_reads` Program

This tool extracts a subset of reads where the size of the subset is a percentage of the input size. Sampling is done in a pseudo-random way, which does not guarantee that the extracted subset comprises an exact percentage of the input reads. The input reads can be provided in both interleaved and non-interleaved format and reads marked as paired are kept together.

Read sampling can be useful for reducing coverage of datasets with a very high coverage (>500x coverage) in preparation for a de novo assembly. A reduction in coverage makes the assembly run faster and reduces the chance of having overlapping errors in the reads, thus increasing the assembly quality.

See Appendix B for full usage information.

8.5 The `clc_sort_pairs` Program

The `clc_sort_pairs` program takes two SOLiD read files, or two Ion Torrent read files, as input and generates as output a file containing paired reads and a file containing unpaired reads. Here the read names are used to sort the reads. This tool is necessary because pairing of the reads in these cases is based on the read names rather than just the position within the file, as would be the case for Illumina data. That is, paired reads for these data types need to be sorted, and paired reads separated from single reads.

To properly handle the input sequence data, the read names within the files must match certain patterns. These are:

Ion Torrent: [anytextinfo]:[number]:[number]

Solid: [number]_[number]_[number]_[R3/F3/F5/F5-P2/F5-BC]

In the case of Solid data, reads in one file of the pair should end in one of the patterns above, and reads in the other file of the pair should end with one of the other patterns. For example, one file might contain reads with names ending in R3, while reads in the other file have names ending in F5, or reads in one file might contain names ending in F3, while reads in the other file have names ending in F5, and so on.

Reads within a given file must be named consistently. That is, if a read has a name ending with a particular pattern, for example F5, then all reads in that file must have names ending in F5.

Please note that in the case of Solid data, the following combinations for the read names in a pair of files are **not allowed**:

F3 + F3

R5 + F5
R3 + R3
R3 + F5
F5 + R3

As mentioned in the Input Data section earlier in the manual, the full sequence of any read containing one or more . symbols, present in a .csfasta format file, will be converted to contain only N characters.

Further details are provided in Appendix B.

8.6 The `clc_split_reads` Program (for 454 paired data)

The 454 sequencing technology can produce paired read files where the two paired read fragments are contained within the same read, separated by a linker sequence. The linker may be placed anywhere in the read or even outside the read, so not all the reads will necessarily contain a pair. The `clc_split_reads` program finds the linker sequence and creates two new files, one with paired reads and one with unpaired reads.

Like adapter regions, linker regions may contain sequencing errors. With this in mind, the `clc_split_reads` tool identifies likely linker sequences by initially carrying out an alignment between the known linker sequence and each read. The alignment is global in terms of the linker and local in terms of the reads. That is, the whole linker must align to part of the read. Matching positions in these alignments score 1, while each mismatch costs 2 and each gap costs 3. For alignments found at the ends of reads, any non-matching linker bases that extend beyond the end of the read are not penalized.

By default, a region that aligns with a score of at least 10, or the length of the linker if less than 10, is considered a good enough match to identify a linker region.

If a match to the linker sequence with a score between 0 and 9 is found, the read will still be split, but in this situation the following will happen:

- The two parts of the read that have just been split are put into the singles list (not the paired list). The reasoning behind this is that since the linker did not match with a good enough score, the match location identified might not have been correct. If this was the case, marking such split sequences as a pair would mean that the paired distance information would be used in a de novo assembly, and this could end up being misleading within the assembly process.
- Any linker matches identified at the end of the read will also be trimmed. This extra trimming stage is carried out due to the possibility that the internal linker match identified might not have been correct.

The following situations are particularly detrimental to de novo assembly, and the `clc_split_reads` program tries to ensure they are avoided:

- Reads contain some remaining section of the linker sequence
- Reads are categorized as a pair when they should not be

In some cases, the start or end of a read is in the middle of the linker. In such cases, the linker sequence is still removed, and the read is put into the file with unpaired reads. If only very few nucleotides of the linker overlap with the read, they are also removed, even though they may not come from the linker. In the case where only a single nucleotide at the start and/or end of the read may come from a linker, it is removed. The rationale is that it is better to discard a few nucleotides and be sure there is no adapter sequence left, since remaining linker sequence is problematic for de novo assembly.

The '-m' option can be used to specify the minimum read length. Only reads this long or longer will be reported. The default value is 15. This becomes important when the linker is close to the start or end of the read, and only a small fragment is left on one side of the linker. If that small fragment is below the specified minimum length, it is discarded along with the linker. The remaining part of the read will be written to the unpaired file.

Further details of the options for this tool are provided in Appendix B.

8.7 The `clc_overlap_reads` Program

In cases where paired end library preparation methods use a relatively short fragment size, some read pairs will overlap. These overlapping reads can be handled as standard paired-end data.

However, in some situations it can be useful to merge the overlapping pairs into a single read. The benefit is that you get longer reads, and that the quality improves (normally the quality drops towards the end of a read, and by overlapping the ends of two reads, the consensus read now reflects two read ends instead of just one).

This joining of overlapping reads can be done using the `clc_overlap_reads` program. It aligns the ends of each read within pairs to see if there is evidence that they overlap. If the alignment of these read ends is relatively good, the reads are joined into one read and put in an output file for single (joint) reads. If there is no evidence of the reads overlapping, the original pair of reads is put in an output file for paired reads.

The nucleotides in the overlapping region of a joint read are assigned a quality score of 40 (very high quality) if the two reads agree on the nucleotide. Otherwise the nucleotide with the highest quality is chosen and its quality score is retained. Use the "-f 33" option if the quality offset value in the input fastq files is based on the ASCII character 33. This is the most common situation.

By default, the alignment between the ends of two reads must have a minimum length of 10 positions and a minimum similarity of 90% for the reads to be considered overlapping. These parameters can be adjusted using the various options for the program. The default is that the first read of each pair is a forward read and the other one is a backward read. This can also be adjusted.

Further details of the overlap reads options are provided in Appendix B.

Chapter 9

Format conversion tools

9.1 The `clc_cas_to_sam` Program

This tool converts a cas format file to sam or bam format format file.

The **`clc_cas_to_sam`** program takes a cas file as input and produces a corresponding SAM file or BAM file. The format generated depends on the filename you choose. If you choose an output file name with the suffix **`.sam`**, the output format will be SAM. If you choose an output file name with the suffix **`.bam`**, the output format will be BAM.

Please note that the read file(s) that you used in generating the cas file must be present in the same relative location to the cas file as they were when you ran the mapping. This is because, unlike cas format files, SAM and BAM files include all the read data. Thus the read data needs to be present in order to make a valid SAM or BAM file. This also means that the SAM or BAM files created will generally be substantially larger than the cas file they were generated from.

The SAM or BAM file created using the `clc_cas_to_sam` tool is not sorted or indexed. These steps can be necessary for some types of downstream processing and can be done using the samtools sort program (see <http://samtools.sourceforge.net/>).

Like cas format files, SAM and BAM files do not contain the reference sequence data.

Further details of the command line options for this tool are provided in Appendix B.

Further details about the SAM and BAM formats can be found at <http://samtools.sourceforge.net/>.

9.2 The `clc_sam_to_cas` Program

This tool converts a read mapping in sam or bam format to a cas format file.

The **`clc_sam_to_cas`** tool converts SAM or BAM files to cas format. Like cas format files, SAM and BAM files do not contain the reference sequence data, so the reference sequences need to be provided when running this command. This is so that the required information can be generated for the cas file.

You also need to provide a destination for the sequencing read data to be written to when running this program.

Further details of the command line options for this tool are provided in Appendix B.

9.3 The `clc_convert_sequences` Program

The primary purpose of this tool is to convert sequences to fasta or fastq format. However, it can currently be used for a variety of sequence conversion related functions. These include:

- Convert from one format to a fasta format file or to a fastq format file.
- Merge separate forward and reverse read files into a single, interleaved paired data file.
- Remove sequence names from within a file (to save space).
- Create a fastq format file from (separate) sequence and quality files.

Input formats supported for some or all functionality:

- fasta
- fastq
- genbank
- sff
- csfasta

Export formats are fasta or fastq.

See Appendix B for full usage information.

Appendix A

Updating program names from earlier versions

The names of the programs of CLC Assembly Cell have changed since version 3.2.2 and earlier.

A mapping of the new program names to the old program names is contained in the distribution in a file called **clc_name_changes.txt**.

For Linux and Mac users, a script has been included as an example of how you could update the Assembly Cell program names in existing scripts you have. The script is called **change_clc_names.sh**. If you plan to use the script without changes, please read through it first, to make sure that it will do what you need.

Below is a listing of the old names followed by the new names for the CLC Assembly Cell programs:

assembly_info	-->	clc_mapping_info
assembly_table	-->	clc_mapping_table
castosam	-->	clc_cas_to_sam
change_assembly_files	-->	clc_change_cas_paths
clc_assembly_viewer	-->	clc_mapping_viewer
clc_novo_assemble	-->	clc_assembler
clc_ref_assemble_long	-->	clc_mapper_legacy
clc_ref_assemble	-->	clc_mapper
filter_matches	-->	clc_filter_matches
find_variations	-->	clc_find_variations
host_info	-->	clc_host_info
join_assemblies	-->	clc_join_mappings
join_pairs	-->	clc_overlap_reads
quality_trim	-->	clc_quality_trim
remove_duplicates	-->	clc_remove_duplicates
samtocas	-->	clc_sam_to_cas
sequence_info	-->	clc_sequence_info
simulate_reads	-->	clc_simulate_reads
sort_pairs	-->	clc_sort_pairs
split_sequences	-->	clc_split_reads

sub_assembly	-->	clc_submapping
tofasta	-->	clc_convert_sequences
unassembled_reads	-->	clc_unmapped_reads
unpaired_reads	-->	clc_unpaired_reads

Appendix B

Options for All Programs

Please find the list of options for all programs in the online version of the user manual at http://clcsupport.com/clcassemblycell/current/index.php?manual=Options_All_Programs.html

Bibliography

- [Gnerre et al., 2011] Gnerre, S., Maccallum, I., Przybylski, D., Ribeiro, F. J., Burton, J. N., Walker, B. J., Sharpe, T., Hall, G., Shea, T. P., Sykes, S., Berlin, A. M., Aird, D., Costello, M., Daza, R., Williams, L., Nicol, R., Gnirke, A., Nusbaum, C., Lander, E. S., and Jaffe, D. B. (2011). High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences of the United States of America*, 108(4):1513–8.
- [Li et al., 2010] Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., and Wang, J. (2010). De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–72.
- [Zerbino and Birney, 2008] Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res*, 18(5):821–829.
- [Zerbino et al., 2009] 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.

Index

Bibliography, [79](#)

Cores, maximum limit, [12](#)

CPU architectures, [12](#)

CPU cores, maximum limit, [12](#)

References, [79](#)

System requirements, [12](#)