Sequence assembly with MIRA3

The Definitive Guide

COLLABORATORS

	TITLE : Sequence assembly with MIRA3	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bastien Chevreux	November 14, 2012	
Extensive review of early reference manual	Jacqueline Weber	November 14, 2012	
Extensive review of early reference manual	Andrea Hörster	November 14, 2012	
Draft for section on preprocessing of ESTs in EST manual	Katrina Dlugosch	November 14, 2012	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Figures

List of Tables

Preface

'How much intelligence does one need to sneak upon lettuce? '

—Solomon Short

This "book" is actually the result of an exercise in self-defense. It contains texts from several years of help files, mails, postings, questions, anwers etc.pp concerning MIRA and assembly projects one can do with it.

I never really intended to push MIRA. It started out as a PhD thesis and I subsequently continued development when I needed something to be done which other programs couldn't do at the time. But MIRA has always been available as binary on the Internet since 1999 ... and as Open Source since 2007. Somehow, MIRA seems to have caught the attention of more than just a few specialised sequencing labs and over the years I've seen an ever growing number of mails in my inbox and on the MIRA mailing list. Both from people having been "since ever" in the sequencing business as well as from labs or people just getting their feet wet in the area.

The help files -- and through them this book -- sort of reflect this development. Most of the chapters[1] contain both very specialised topics as well as step-by-step walk-throughs intended to help people to get their assembly projects going. Some parts of the documentation are written in a decidedly non-scientific way. Please excuse, time for rewriting mails somewhat lacking, some texts were re-used almost verbatim.

Nothing is perfect, and both MIRA and this documentation are far from it. If you spot an error either in MIRA or the docs, feel free to report it. Or, even better, correct it if you can. At least with the help files it should be easy, they're just text files.

I hope that MIRA will be as useful to you as it has been to me. Have a lot of fun with it.

Rheinfelden, Summer 2011

Bastien Chevreux

[1] Avid readers of David Gerrold will certainly recognise the quotes from his books at the beginning of each chapter

Chapter 1

Introduction to MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'Half of being smart is to know what you're dumb at. '

—Solomon Short

1.1 What is MIRA?

MIRA is a multi-pass DNA sequence data assembler/mapper for whole genome and EST projects. MIRA assembles reads gained by

- electrophoresis sequencing (aka Sanger sequencing)
- 454 pyrosequencing (GS20, FLX or Titanium)
- Ion Torrent
- Solexa (Illumina) sequencing
- (in development) Pacific Biosciences sequencing

into contiguous sequences (called *contigs*). One can use the sequences of different sequencing technologies either in a single assembly run (a *true hybrid assembly*) or by mapping one type of data to an assembly of other sequencing type (a *semi-hybrid assembly (or mapping)*) or by mapping a data against consensus sequences of other assemblies (a *simple mapping*).

The MIRA acronym stands for **M**imicking **I**ntelligent **R**ead **A**ssembly and the program pretty well does what its acronym says (well, most of the time anyway). It is the Swiss army knife of sequence assembly that I've used and developed during the past 14 years to get assembly jobs I work on done efficiently - and especially accurately. That is, without me actually putting too much manual work into it.

Over time, other labs and sequencing providers have found MIRA useful for assembly of extremely 'unfriendly' projects containing lots of repetitive sequences. As always, your mileage may vary.

1.2 What to read in this manual and where to start reading?

At the last count, this manual had almost 200 pages and this might seem a little bit daunting. However, you very probably do not need to read everything.

You should read most of this introductionary chapter though: e.g.,

- the part with the MIRA quick tour
- the part which gives a quick overview for which data sets to use MIRA and for which not
- the part which showcases different features of MIRA (lots of screenshots!)
- where and how to get help if things don't work out as you expected

After that, reading should depend on the type of data you intend to work with: there are specific chapters for Sanger, 454, Solexa, IonTorrent and PacBio data, all of which containing an overview on how to prepare your data and how to launch MIRA for these data sets. There are also complete walkthroughs which exemplarily show from start to end one way of doing an assembly for a specific data set and what to do with the results of the assembly.

As the former named chapters are geared toward genome assemblies, there is also a chapter going into details on how to use MIRA for EST / RNASeq assemblies. Read that if you're into this kind of data.

As the previously cited chapters are more introductory in their nature, they do not go into the details of MIRA parametrisation. While MIRA has a comprehensive set of standard settings which should be suited for a majority of assembly tasks, the are more than 150 switches / parameters with which one can fine tune almost every aspect of an assembly. A complete description for each and every parameter and how to correctly set parameters for different use cases and sequencing technologies can be found in the reference chapter.

The chapter on working with results of MIRA should again be of general interest to everyone. It describes the structure of output directories and files and gives first pointers on what to find where. Also, converting results into different formats -- with and without filtering for specific needs -- is covered there.

As not every assembly project is simple, there is also a chapter with tipps on how to deal with projects which turn out to be "hard." It certainly helps if you at least skim through it even if you do not expect to have problems with your data ... it contains a couple of tricks on what one can see in result files as well as in temporary and log files which are not explained elsewhere.

As from time to time some general questions on sequencing are popping up on the MIRA talk mailing list, I have added a chapter with some general musings on what to consider when going into sequencing projects. This should be in no way a replacement for an exhaustive talk with a sequencing provider, but it can give a couple of hints on what to take care of.

There is also a FAQ chapter with some of the more frequently asked questions which popped up in the past few years.

Finally, there are also chapters covering some more technical aspects of MIRA: the MAF format and structure / content of the tmp directory have own chapters.

1.3 The MIRA quick tour

Input can be in various formats like Staden experiment (EXP), Sanger CAF, FASTA, FASTQ or PHD file. Ancillary data containing additional information helpful to the assembly as is contained in, e.g. NCBI traceinfo XML files or Staden EXP files, is also honoured. If present, base qualities in **phred** style and SCF signal electrophoresis trace files are used to adjudicate between or even correct contradictory stretches of bases in reads by either the integrated automatic EdIt editor (written by Thomas Pfisterer) or the assembler itself.

MIRA was conceived especially with the problem of repeats in genomic data and SNPs in transcript (EST / RNASeq) data in mind. Considerable effort was made to develop a number of strategies -- ranging from standard clone-pair size restrictions to discovery and marking of base positions discriminating the different repeats / SNPs -- to ensure that repetitive elements are correctly resolved and that misassemblies do not occur.

The resulting assembly can be written in different standard formats like CAF, Staden GAP4 directed assembly, ACE, HTML, FASTA, simple text or transposed contig summary (TCS) files. These can easily be imported into numerous finishing tools or further evaluated with simple scripts.

The aim of MIRA is to build the best possible assembly by

1. having a more or less full overview on the whole project at any time of the assembly, i.e. knowledge of almost all possible read-pairs in a project,

2. using high confidence regions (HCRs) of several aligned read-pairs to start contig building at a good anchor point of a contig, extending clipped regions of reads on a 'can be justified' basis.
3. using all available data present at the time of assembly, i.e., instead of relying on sequence and base confidence values only, the assembler will profit from trace files containing electrophoresis signals, tags marking possible special attributes of DNA, information on specific insert sizes of read-pairs etc.
4. having 'intelligent' contig objects accept or refuse reads based on the rate of unexplainable errors introduced into the consensus
5. learning from mistakes by discovering and analysing possible repeats differentiated only by single nucleotide polymorphisms. The important bases for discriminating different repetitive elements are tagged and used as new information.
6. using the possibility given by the integrated automatic editor to correct errors present in contigs (and subsequently) reads by generating and verifying complex error hypotheses through analysis of trace signals in several reads covering the same area of a consensus,
7. iteratively extending reads (and subsequently) contigs based on
   (a) additional information gained by overlapping read pairs in contigs and
   (b) corrections made by the automated editor.

MIRA was part of a bigger project that started at the DKFZ (Deutsches Krebsforschungszentrum, German Cancer Research Centre) Heidelberg in 1997: the "Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie" supported the PhD thesis of Thomas and myself by grant number *01 KW 9611*. Beside an assembler to tackle difficult repeats, the grant also supported the automated editor / finisher EdIt package -- written by Thomas Pfisterer. The strength of MIRA and EdIt is the automatic interaction of both packages which produces assemblies with less work for human finishers to be done.

I'd like to thank everybody who reported bugs to me, pointed out problems, sent ideas and suggestions they encountered while using the predecessors. Please continue to do so, the feedback made this third version possible.

1.4 For which data sets to use MIRA and for which not

As a general rule of thumb: if you have an organism with more than 100 to 150 megabases or more than 20 to 40 million reads, you might want to try other assemblers first.

1.4.1 Genome de-novo

For genome assembly, the version 3 series of MIRA have been reported to work on projects with something like a million Sanger reads (~80 to 100 megabases at 10x coverage), five to ten million 454 Titanium reads (~100 megabases at 20x coverage) and 20 to 40 million Solexa reads (enough for de-novo of a bacterium or a small eukaryote with 76mers or 100mers).

Provided you have the memory, MIRA is expected to work in de-novo mode with

- Sanger reads: 5 to 10 million
- 454 reads: 5 to 15 million
- Ion Torrent reads: 5 to 15 million
- Solexa reads: 15 to 20 million

and "normal" coverages, whereas "normal" would be at no more than 50x to 70x for genome projects. Higher coverages will also work, but may create somewhat larger temporary files without heavy parametrisation. Lower coverages (<4x for Sanger, <10x for 454, < 10x for IonTorrent) also need special attention in the parameter settings.

1.4.2 Genome mapping

As the complexity of mapping is a lot lower than de-novo, one can basically double (perhaps even triple) the number of reads compared to 'de-novo'. The limiting factor will be the amount of RAM though, and MIRA will also need lots of it if you go into eukaryotes.

The main limiting factor regarding time will be the number of reference sequences (backbones) you are using. MIRA being pedantic during the mapping process, it might be a rather long wait if you have more than 500 to 1000 reference sequences.

1.4.3 ESTs / RNASeq

The default values for MIRA should allow it to work with many EST and RNASeq data sets, sometimes even from non-normalised libraries. For extreme coverage cases however (like, something with a lot of cases at and above 10k coverage), one would perhaps want to resort to data reduction routines before feeding the sequences to MIRA.

On the other hand, recent developments of MIRA were targeted at making de-novo RNASeq assembly of non-normalised libraries liveable, and indeed I now regularly use MIRA for data sets with up to 50 million Illumina 100bp reads.

1.5 Any special features I might be interested in?

A few perhaps.

Note
The screenshots in this section show data from assemblies produced with MIRA, but the visualisation itself is done in a finishing program named **gap4**.
Some of the screenshots were edited for showing a special feature of MIRA. E.g., in the screenshots with Solexa data, quite some reads were left out of the view pane as else -- due to the amount of data -- these screenshots would need several pages for a complete printout.

1.5.1 MIRA learns to discern non-perfect repeats, leading to better assemblies

MIRA is an iterative assembler (it works in several passes) and acts a bit like a child when exploring the world: it explores the assembly space and is specifically parametrised to allow a couple of assembly errors during the first passes. But after each pass some routines (the "parents", if you like) check the result, searching for assembly errors and deduce knowledge about specific assemblies MIRA should not have ventured into. MIRA will then prevent these errors to re-occur in subsequent passes.

As an example, consider the following multiple alignment:

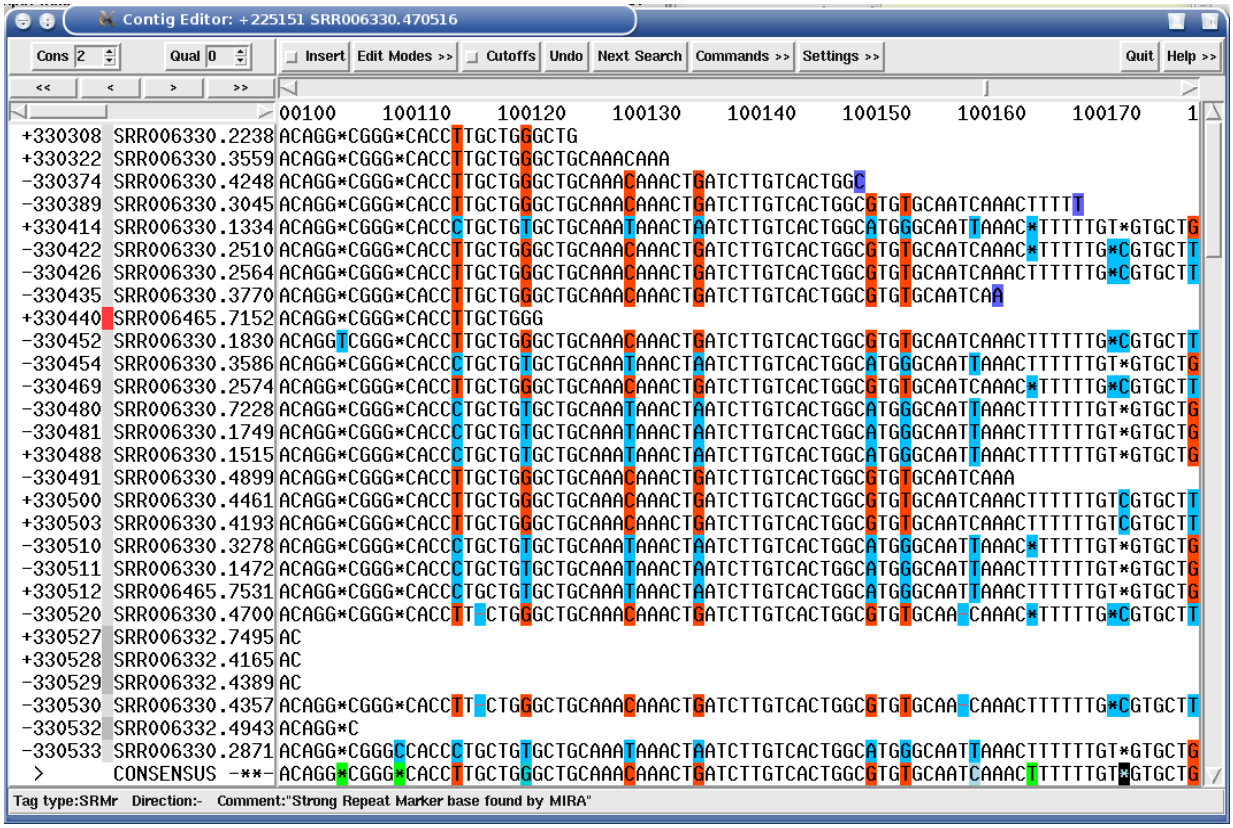

Figure 1.1: How MIRA learns from misassemblies (1). Multiple alignment after 1st pass with an obvious assembly error, notice the clustered columns discrepancies. Two slightly different repeats were assembled together.

These kind of errors can be easily spotted by a human, but are hard to prevent by normal alignment algorithms as sometimes there's only one single base column difference between repeats (and not several as in this example).

MIRA spots these things (even if it's only a single column), tags the base positions in the reads with additional information and then will use that information in subsequent passes. The net effect is shown in the next two figures:

Figure 1.2: Multiple alignment after last pass where assembly errors from previous passes have been resolved (1st repeat site)

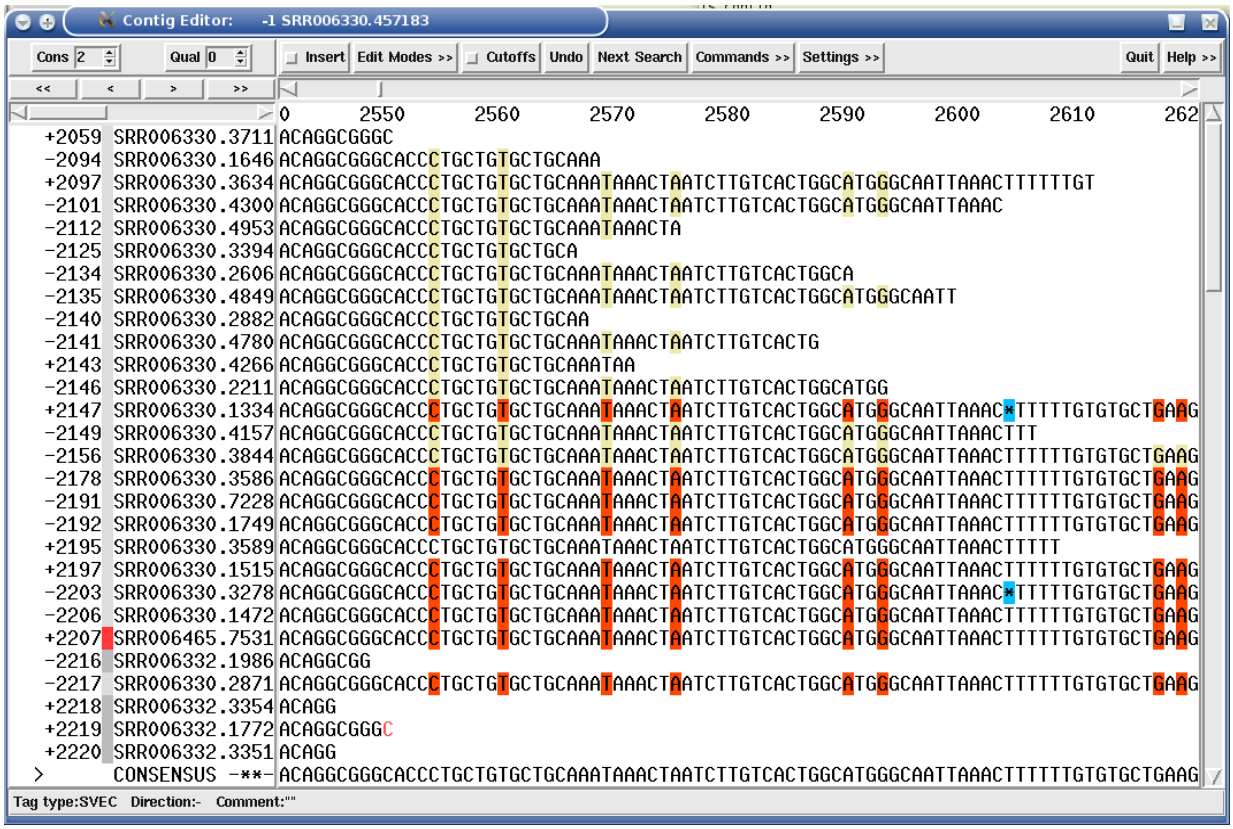

Figure 1.3: Multiple alignment after last pass where assembly errors from previous passes have been resolved (2nd repeat site)

The ability of MIRA to learn and discern non-identical repeats from each other through column discrepancies is nothing new. Here's the link to a paper from a talk I had at the German Conference on Bioinformatics in 1999: http://www.bioinfo.de/isb/gcb99/talks/chevreux/

I'm sure you'll recognise the basic principle in figures 8 and 9. The slides from the corresponding talk also look very similar to the screenshots above:

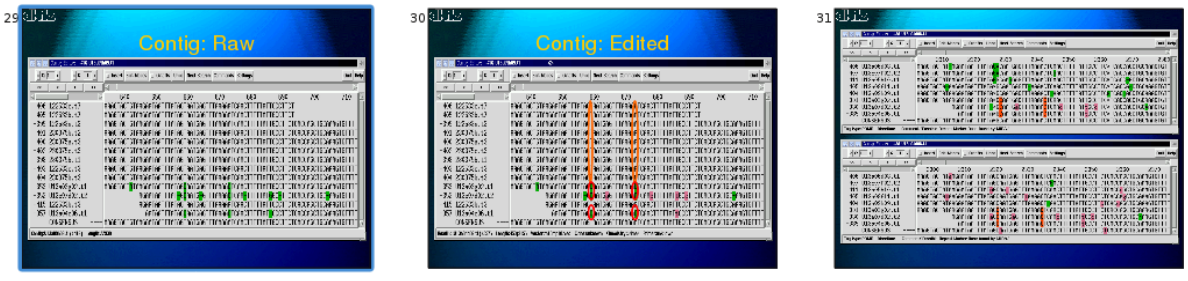

Figure 1.4: Slides presenting the repeat locator at the GCB 99

You can get the talk with these slides here: http://chevreux.org/dkfzold/gcb99/bachvortrag_gcb99.ppt

1.5.2 MIRA has integrated editors for data from Sanger, 454, IonTorrent sequencing

Since the first versions in 1999, the *EdIt* automatic Sanger sequence editor from Thomas Pfisterer has been integrated into MIRA.

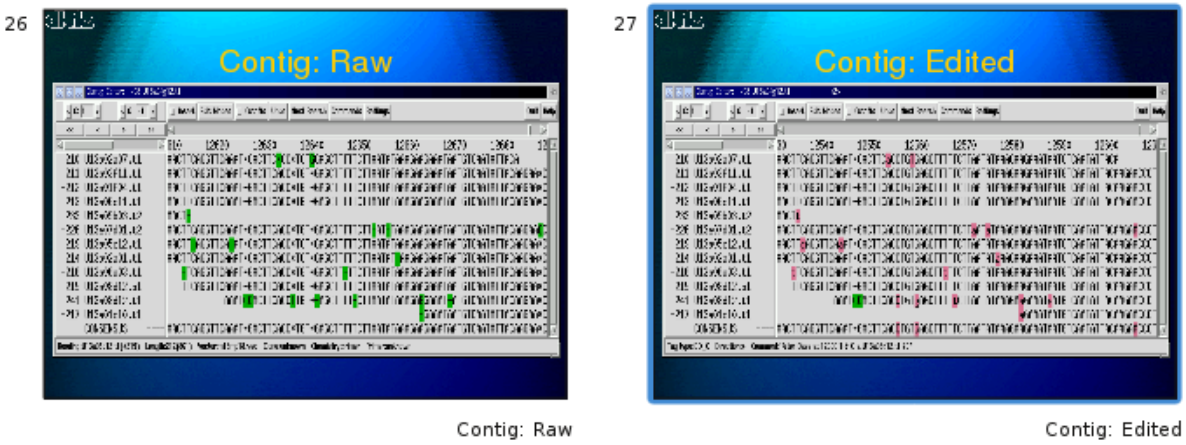

Figure 1.5: Slides presenting the Edit automatic Sanger editor at the GCB 99

The routines use a combination of hypothesis generation/testing together with neural networks (trained on ABI and ALF traces) for signal recognition to discern between base calling errors and true multiple alignment differences. They go back to the trace data to resolve potential conflicts and eventually recall bases using the additional information gained in a multiple alignment of reads.

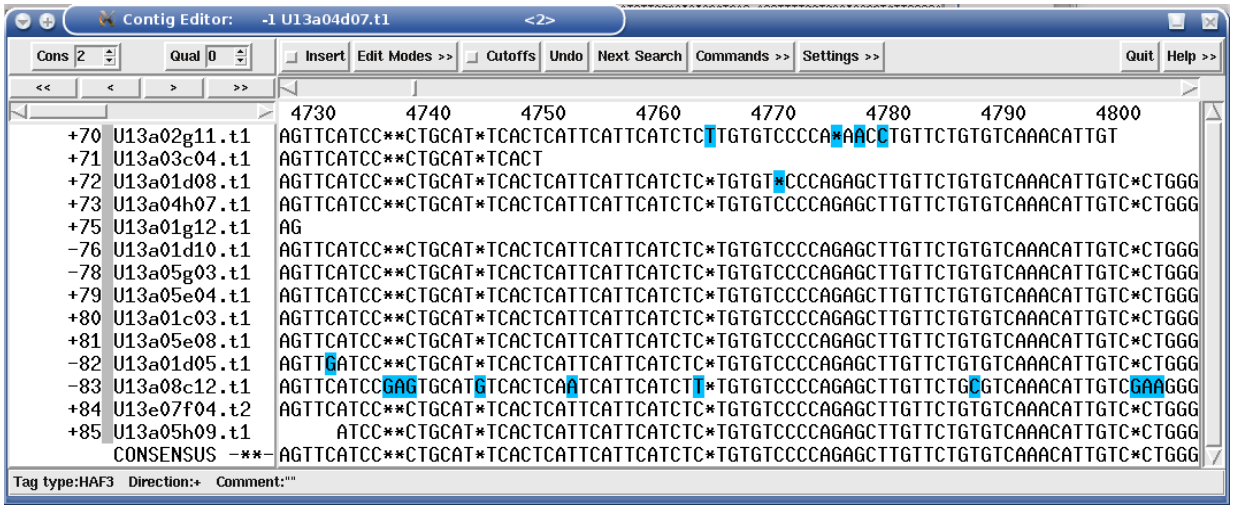

Figure 1.6: Sanger assembly without EdIt automatic editing routines. The bases with blue background are base calling errors.

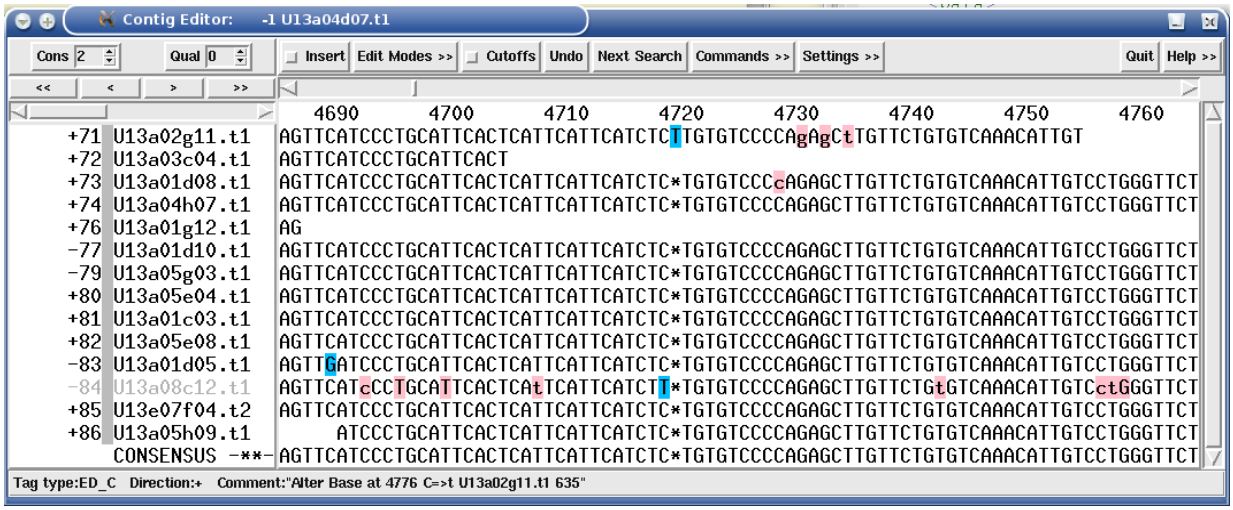

Figure 1.7: Sanger assembly with EdIt automatic editing routines. Bases with pink background are corrections made by EdIt after assessing the underlying trace files (SCF files in this case). Bases with blue background are base calling errors where the evidence in the trace files did not show enough evidence to allow an editing correction.

With the introduction of 454 reads, MIRA also got in 2007 specialised editors to search and correct for typical 454 sequencing problems like the homopolymer run over-/undercalls. These editors are now integrated into MIRA itself and are not part of EdIt anymore.

While not being paramount to the assembly quality, both editors provide additional layers of safety for the MIRA learning algorithm to discern non-perfect repeats even on a single base discrepancy. Furthermore, the multiple alignments generated by these two editors are way more pleasant to look at (or automatically analyse) than the ones containing all kind of gaps, insertions, deletions etc.pp.

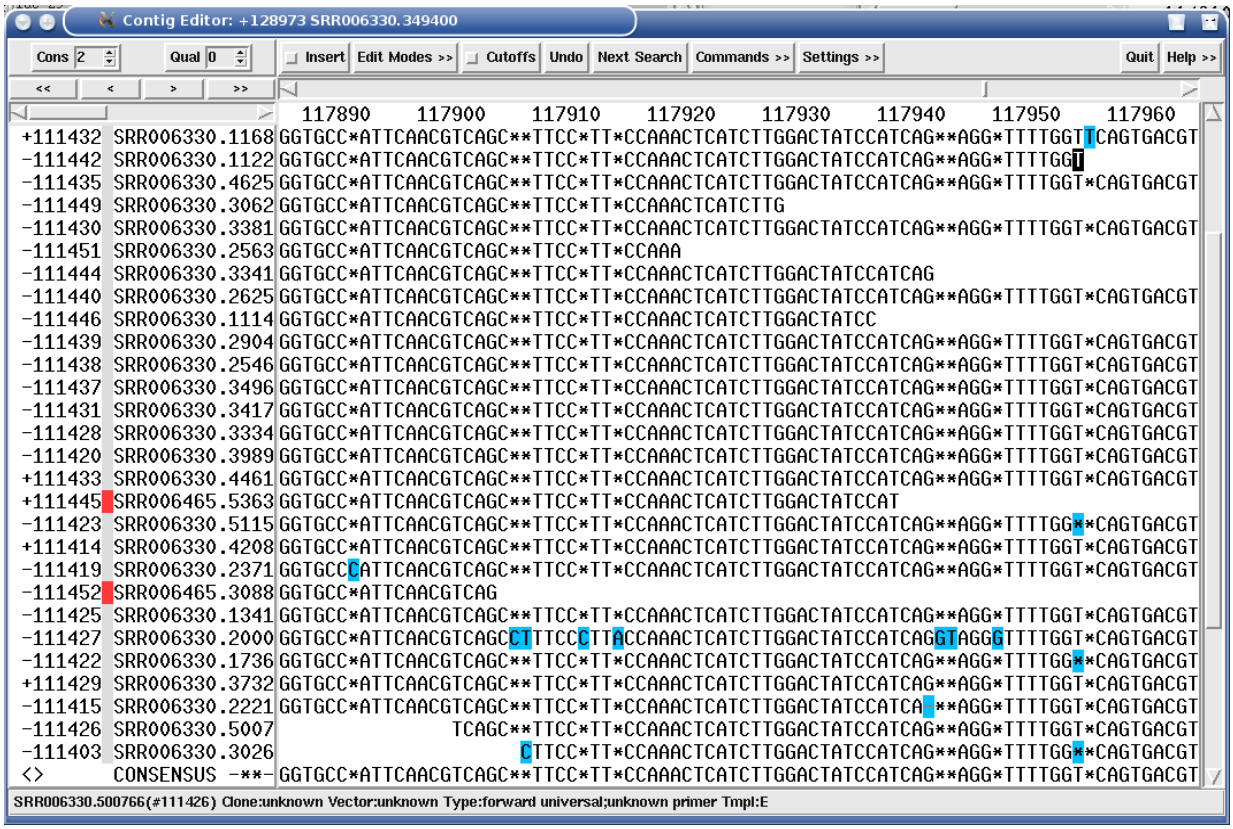

Figure 1.8: 454 assembly without 454 automatic editing routines

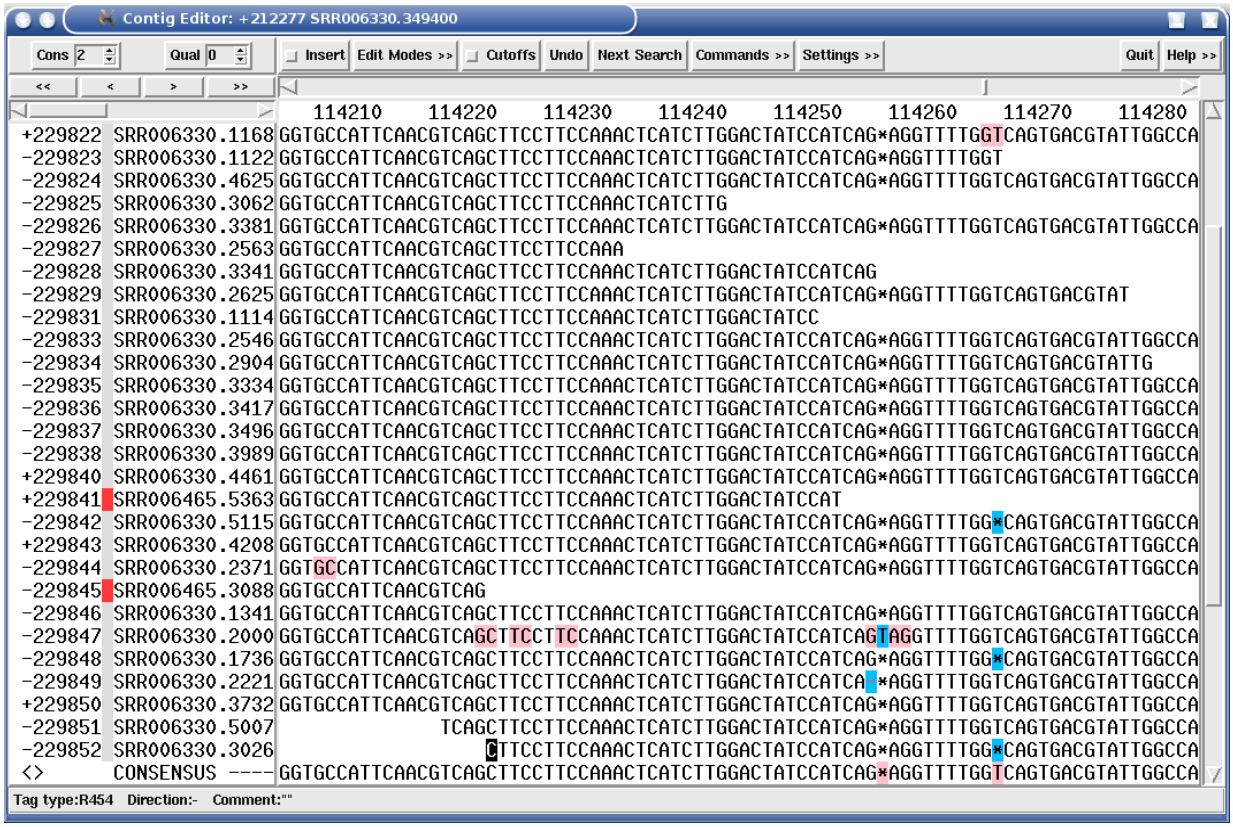

Figure 1.9: 454 assembly with 454 automatic editing routines

With introduction of PacBio strobed reads, MIRA also got an editor to handle "elastic dark inserts" (stretches of unread bases where the length is known only approximately). How this editor works is explained in the chapter on PacBio data, but in essence it allows to transform this:

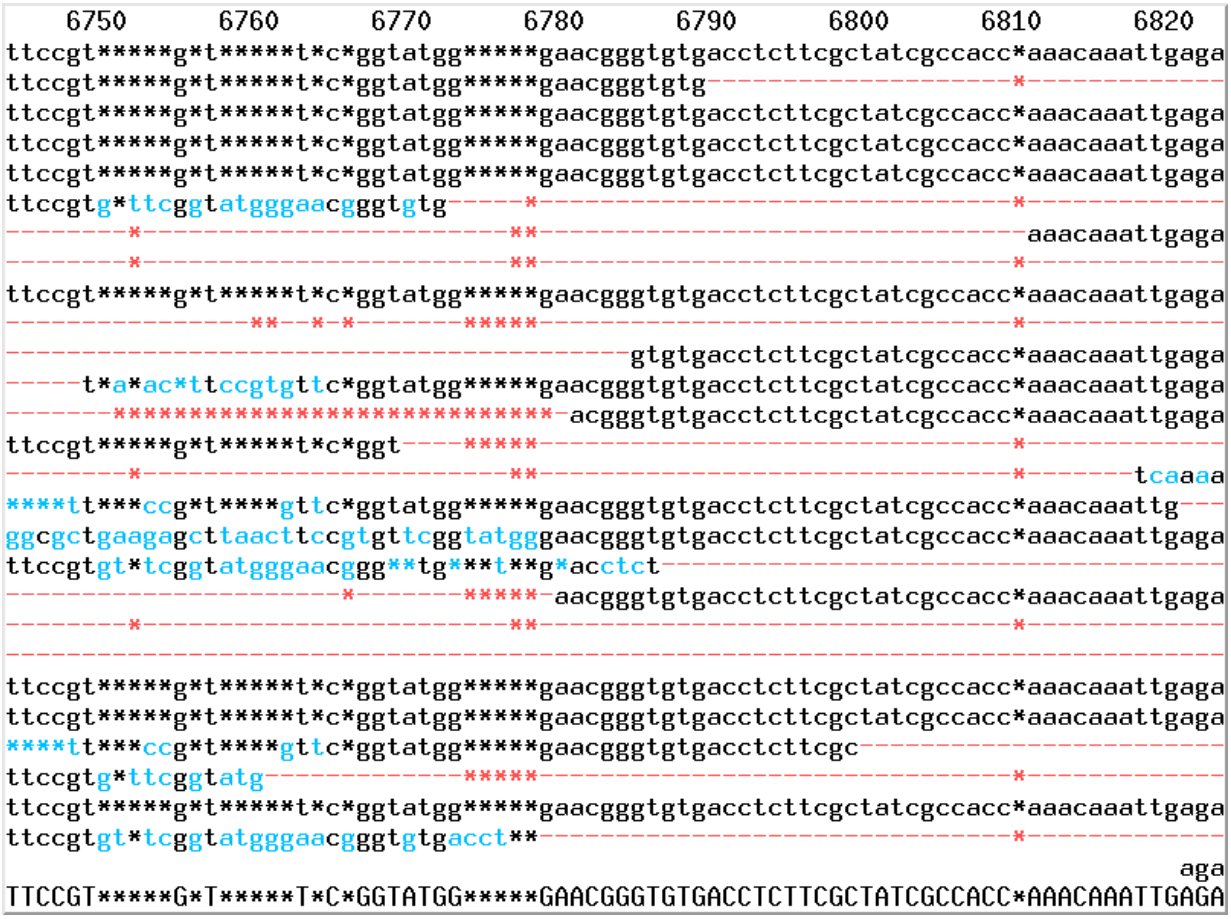

Figure 1.10: Multiple alignmen of PacBio elastic dark inserts without automatic editing

into this:

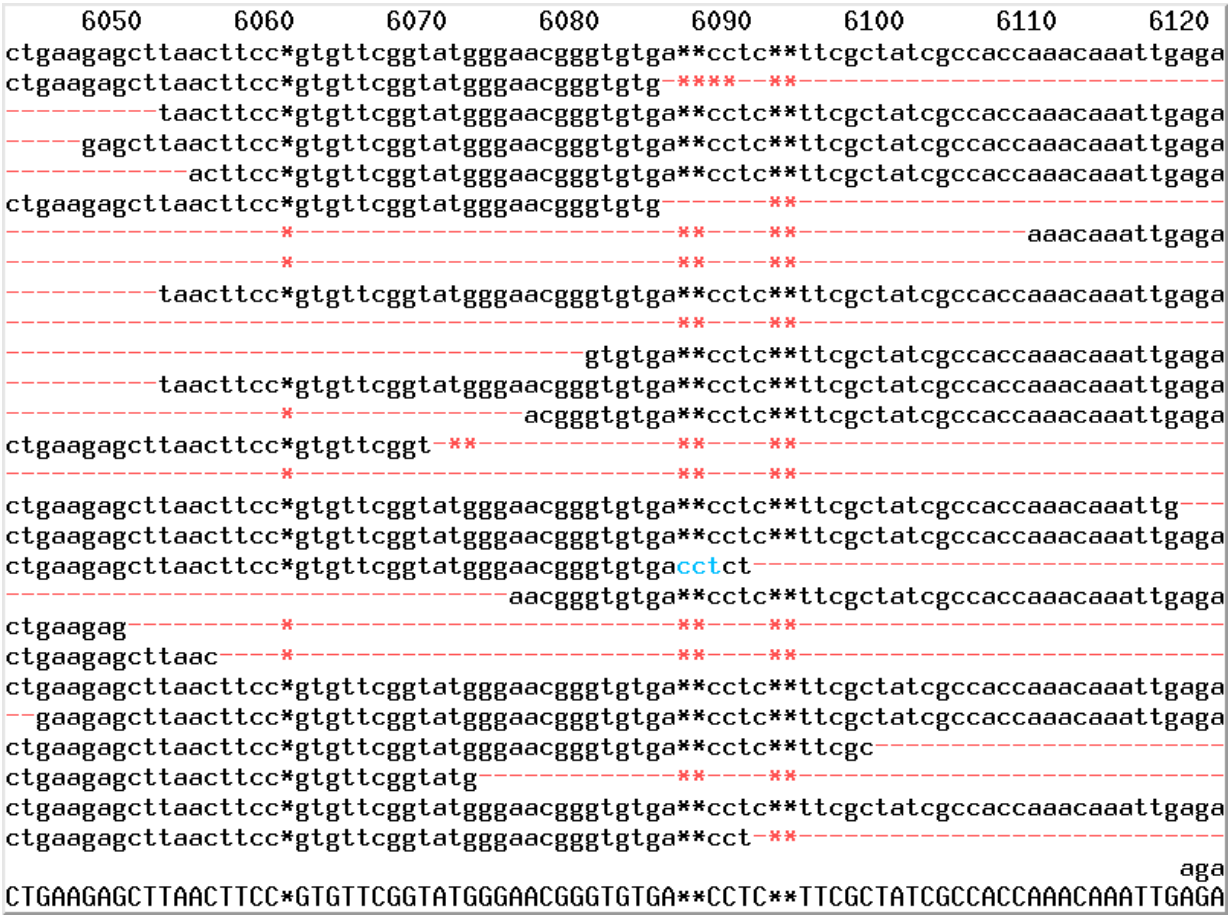

Figure 1.11: Multiple alignment of PacBio elastic dark inserts with automatic editing

1.5.3 MIRA lets you see why contigs end where they end

A very useful feature for finishing are hash frequency (HAF) tags which MIRA sets in the assembly. Provided your finishing editor understands those tags (**gap4**, **gap5** and **consed** are fine but there may be others), they'll give you precious insight where you might want to be cautious when joining to contigs or where you would need to perform some primer walking. MIRA colourises the assembly with the HAF tags to show repetitiveness.

You will need to read about the HAF tags in the reference manual, but in a nutshell: the HAF5, HAF6 and HAF7 tags tell you potentially have repetitive to very repetitive read areas in the genome, while HAF2 tags will tell you that these areas in the genome have not been covered as well as they should have been.

As an example, the following figure shows the coverage of a contig.

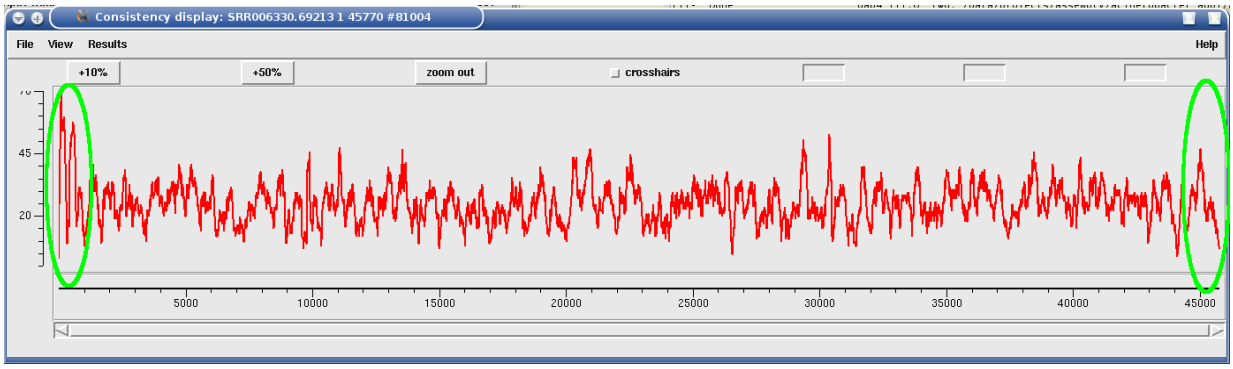

Figure 1.12: Coverage of a contig.

The question is now: why did MIRA stop building this contig on the left end (left oval) and why on the right end (right oval).

Looking at the HAF tags in the contig, the answer becomes quickly clear: the left contig end has HAF5 tags in the reads (shown in bright red in the following figure). This tells you that MIRA stopped because it probably could not unambiguously continue building this contig. Indeed, if you BLAST the sequence at the NCBI, you will find out that this is an rRNA area of a bacterium, of which bacteria normally have several copies in the genome:

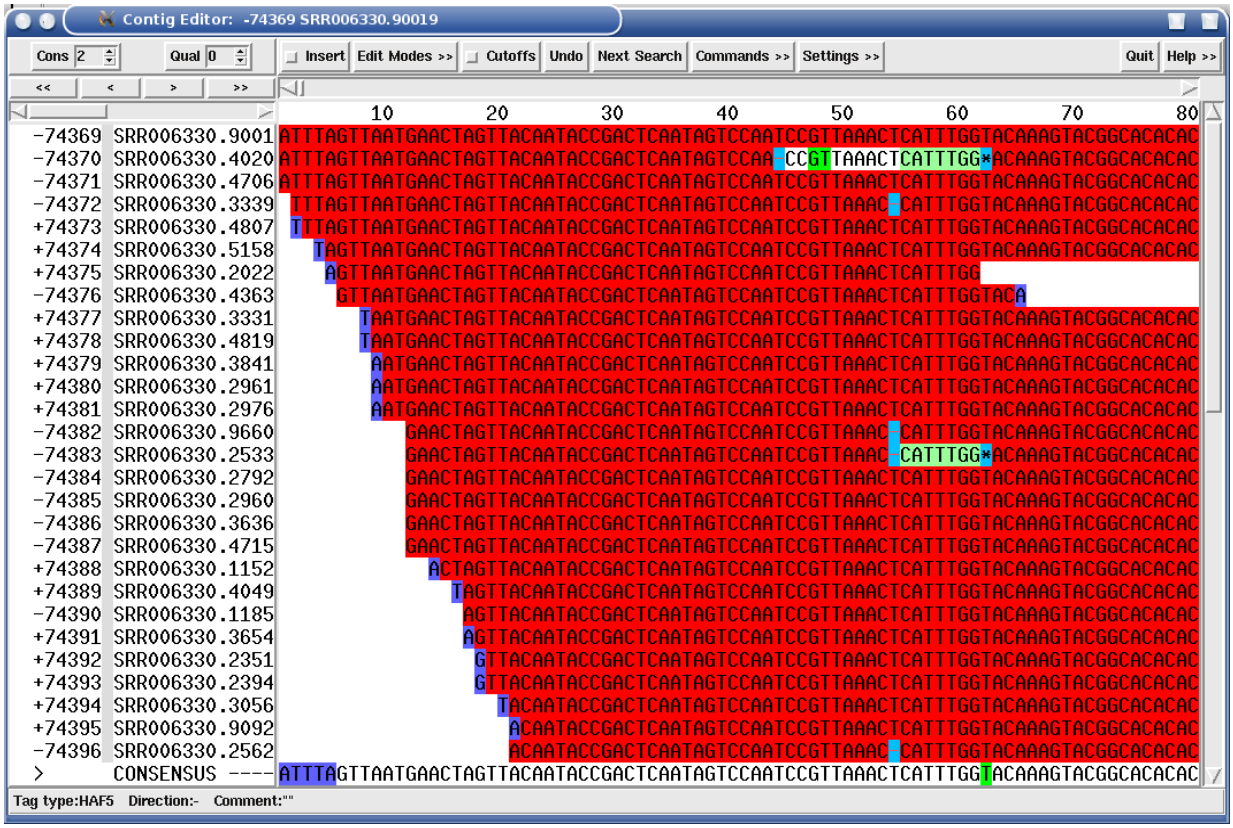

Figure 1.13: HAF5 tags (reads shown with red background) covering a contig end show repetitiveness as reason for stopping a contig build.

The right end of the same contig however ends in HAF3 tags (normal coverage, bright green in the next figure) and even HAF2 tags (below average coverage, pale green in the next image). This tells you MIRA stopped building the contig at this place simply because there were no more reads to continue. This is a perfect target for primer walking if you want to finish a genome.

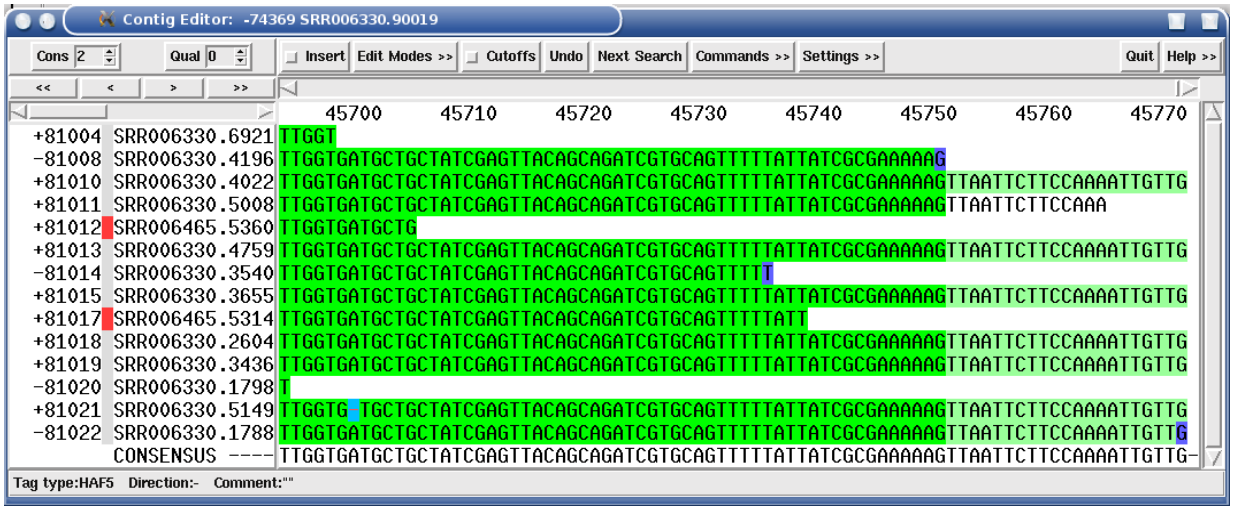

Figure 1.14: HAF2 tags covering a contig end show that no more reads were available for assembly at this position.

1.5.4 MIRA tags problematic decisions in hybrid assemblies

Many people combine Sanger & 454 -- or nowadays more 454 & Solexa -- to improve the sequencing quality of their project through two (or more) sequencing technologies. To reduce time spent in finishing, MIRA automatically tags those bases in a consensus of a hybrid assembly where reads from different sequencing technologies severely contradict each other.

The following example shows a hybrid 454 / Solexa assembly where reads from 454 (highlighted read names in following figure) were not sure whether to have one or two "G" at a certain position. The consensus algorithm would have chosen "two Gs" for 454, obviously a wrong decision as all Solexa reads at the same spot (the reads which are not highlighted) show only one "G" for the given position. While MIRA chose to believe Solexa in this case, it tagged the position anyway in case someone choses to check these kind of things.

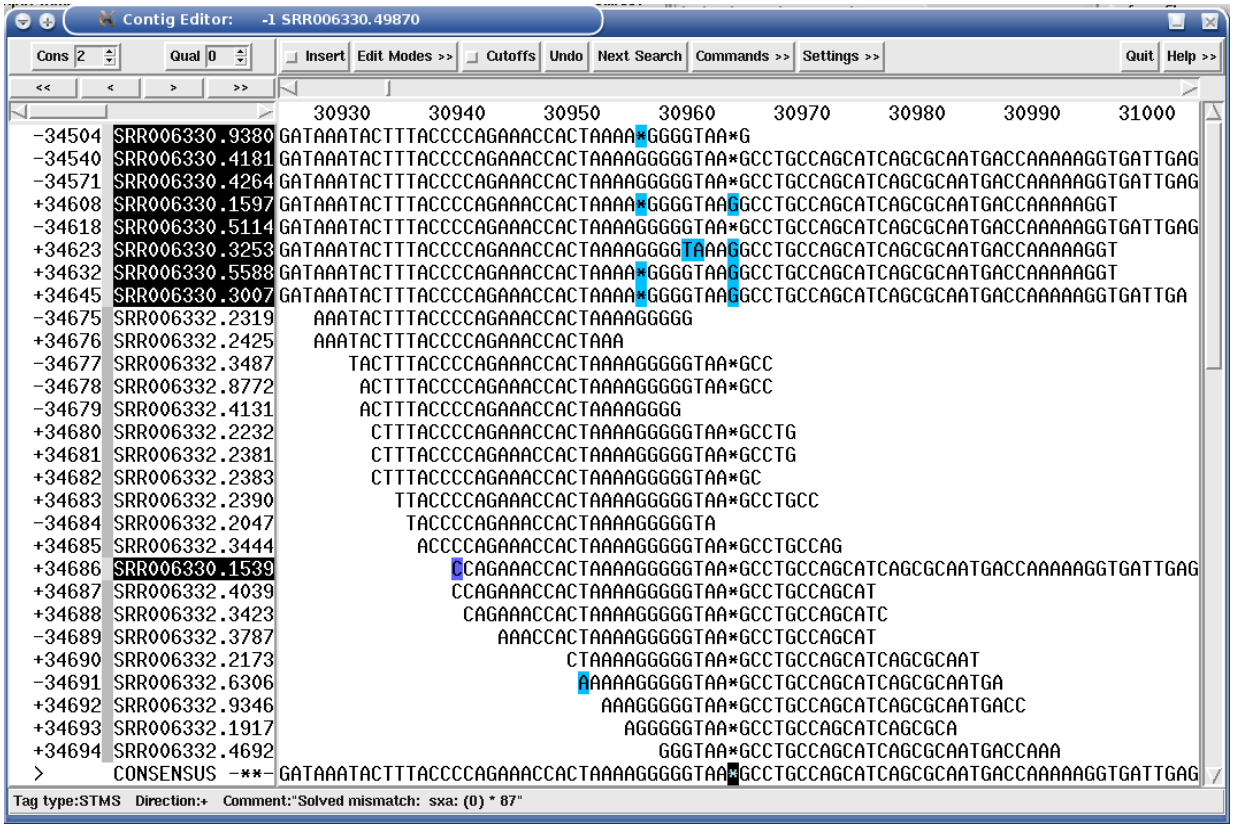

Figure 1.15: A "STMS" tag (Sequencing Technology Mismatch Solved, the black square base in the consensus) showing a potentially difficult decision in a hybrid 454 / Solexa de-novo assembly.

This works also for other sequencing technology combinations or in mapping assemblies. The following is an example in a hybrid Sanger / 454 project where by pure misfortune, all Sanger reads have a base calling error at a given position while the 454 reads show the true sequence.

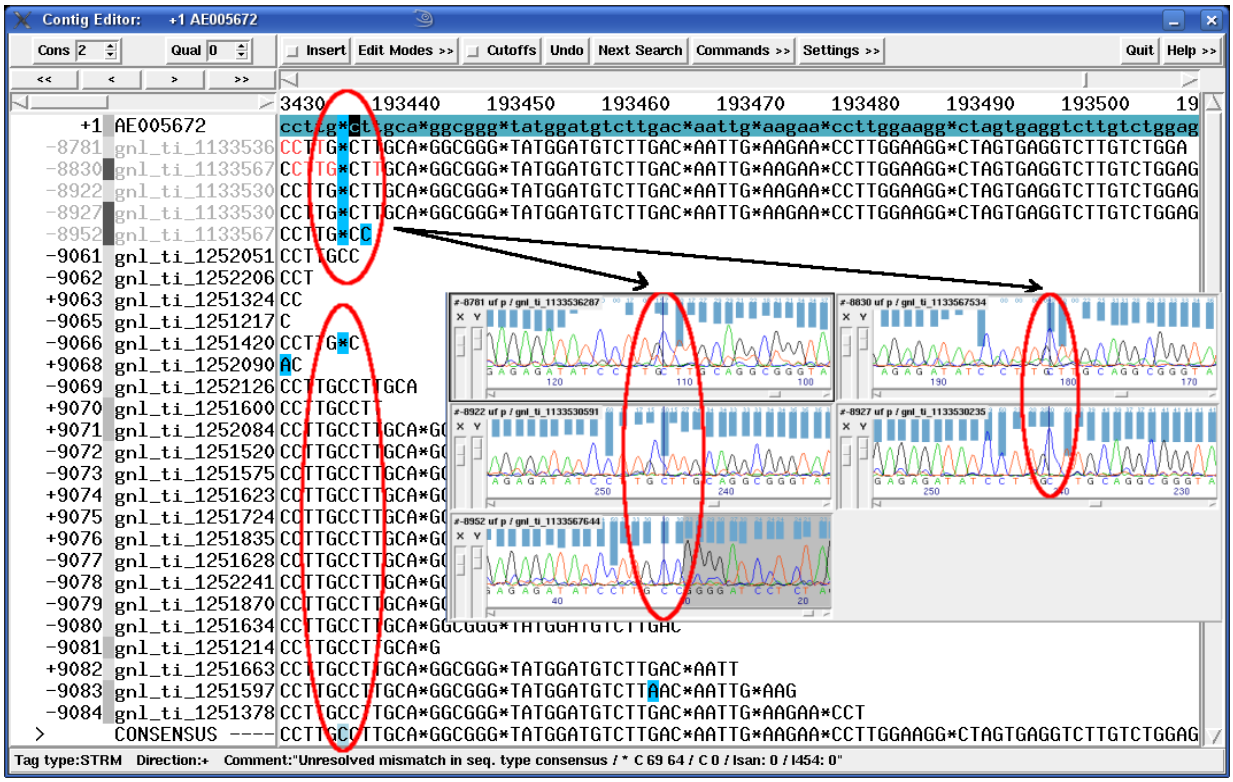

Figure 1.16: A "STMU" tag (Sequencing Technology Mismatch Unresolved, light blue square in the consensus at lower end of large oval) showing a potentially difficult decision in a hybrid Sanger / 454 mapping assembly.

1.5.5 MIRA allows older finshing programs to cope with amount data in Solexa mapping projects

Quality control is paramount when you do mutation analysis for biologists: I know they'll be on my doorstep the very next minute they found out one of the SNPs in the resequencing data wasn't a SNP, but a sequencing artefact. And I can understand them: why should they invest -- per SNP -- hours in the wet lab if I can invest a couple of minutes to get them data false negative rates (and false discovery rates) way below 1%? So, finishing and quality control for any mapping project is a must.

Both **gap4** and **consed** start to have a couple of problems when projects have millions of reads: you need lots of RAM and scrolling around the assembly gets a test to your patience. Still, these two assembly finishing programs are amongst the better ones out there, although **gap5** starts to quickly arrive in a state in which it allows itself to substitute to **gap4**.

So, MIRA reduces the number of reads in Solexa mapping projects without sacrificing information on coverage. The principle is pretty simple: for 100% matching reads, MIRA tracks coverage of every reference base and creates long synthetic, coverage equivalent reads (CERs) in exchange for the Solexa reads. Reads that do not match 100% are kept as own entities, so that no information gets lost. The following figure illustrates this:

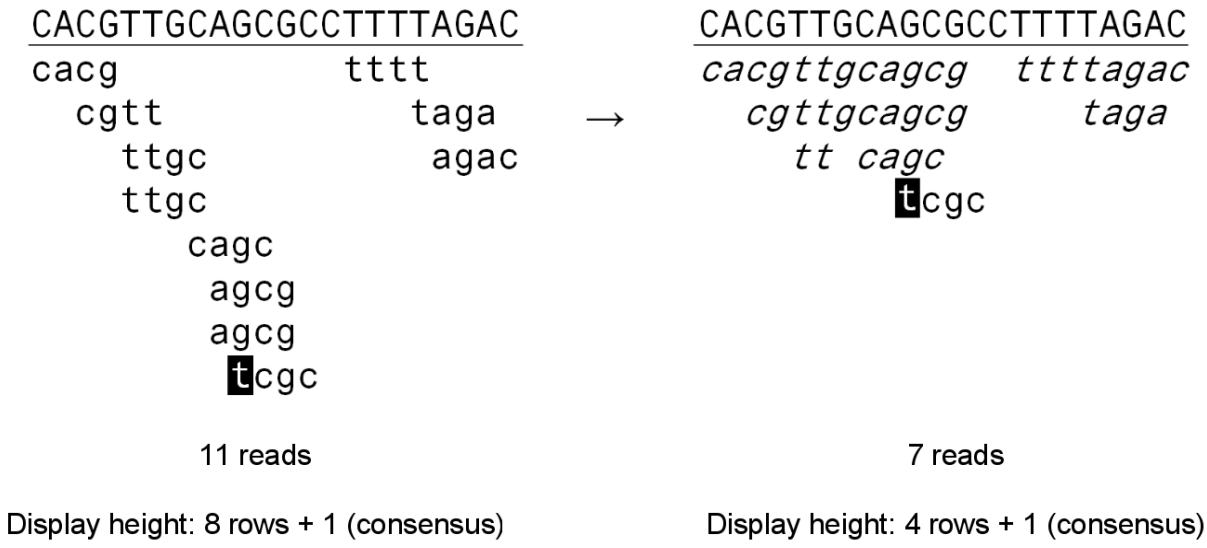

Figure 1.17: Coverage equivalent reads (CERs) explained.
Left side of the figure: a conventional mapping with eleven reads of size 4 against a consensus (in uppercase). The inversed base in the lowest read depicts a sequencing error.
Right side of the figure: the same situation, but with coverage equivalent reads (CERs). Note that there are less reads, but no information is lost: the coverage of each reference base is equivalent to the left side of the figure and reads with differences to the reference are stil present.

This strategy is very effective in reducing the size of a project. As an example, in a mapping project with 9 million Solexa 36mers, MIRA created a project with 1.7m reads: 700k CER reads representing ~8 million 100% matching Solexa reads, and it kept ~950k mapped reads as they had $\geq$ mismatch (be it sequencing error or true SNP) to the reference. A reduction of 80%, and numbers for mapping projects with Solexa 100bp reads are in a similar range.

Also, mutations of the resequenced strain now really stand out in the assembly viewer as the following figure shows:

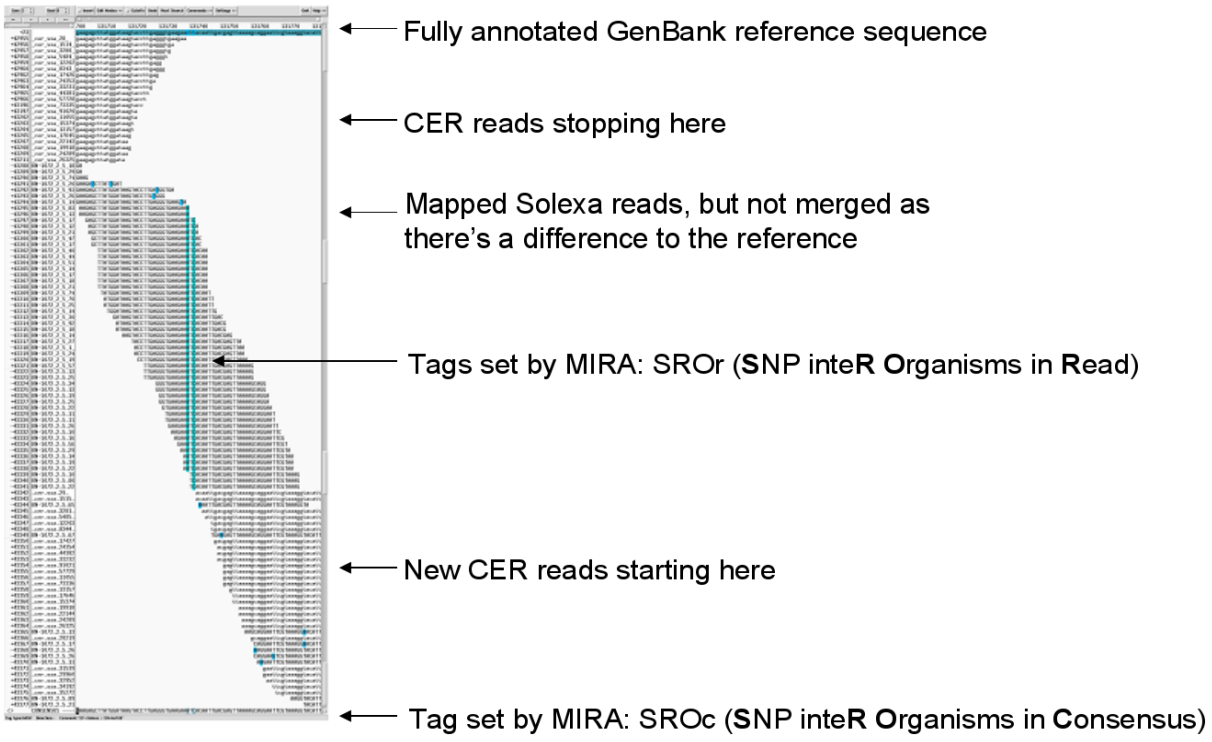

Figure 1.18: Coverage equivalent reads let SNPs become very visible in assembly viewers

1.5.6 MIRA tags SNPs and other features, outputs result files for biologists

Want to assemble two or several very closely related genomes without reference, but finding SNPs or differences between them?

Tired of looking at some text output from mapping programs and guessing whether a SNP is really a SNP or just some random junk?

MIRA tags all SNPs (and other features like missing coverage etc.) it finds so that -- when using a finishing viewer like gap4 or consed -- one can quickly jump from tag to tag and perform quality control. This works both in de-novo assembly and in mapping assembly, all MIRA needs is the information which read comes from which strain.

The following figure shows a mapping assembly of Solexa 36mers against a bacterial reference sequence, where a mutant has an indel position in an gene:

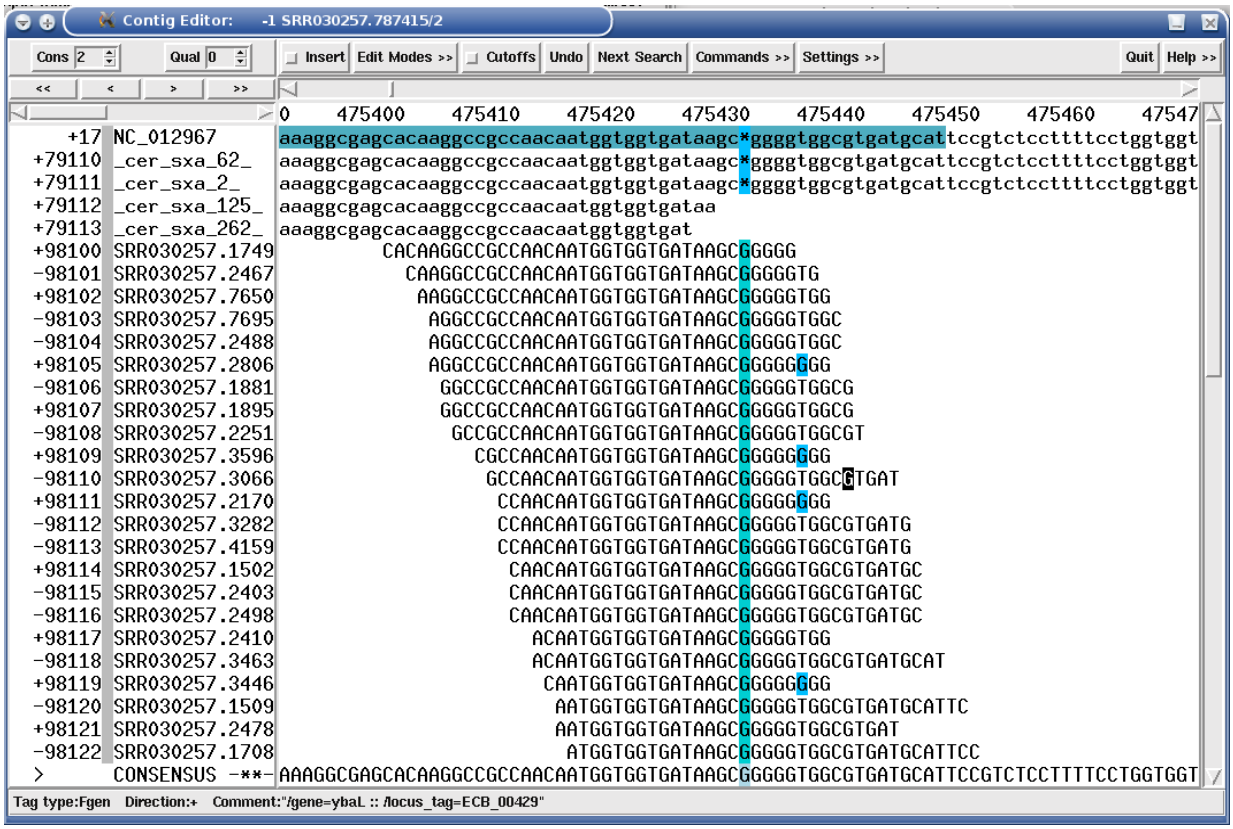

Figure 1.19: "SROc" tag (Snp inteR Organism on Consensus) showing a SNP position in a Solexa mapping assembly.

Other interesting places like deletions of whole genome parts are also directly tagged by MIRA and noted in diverse result files (and searchable in assembly viewers):

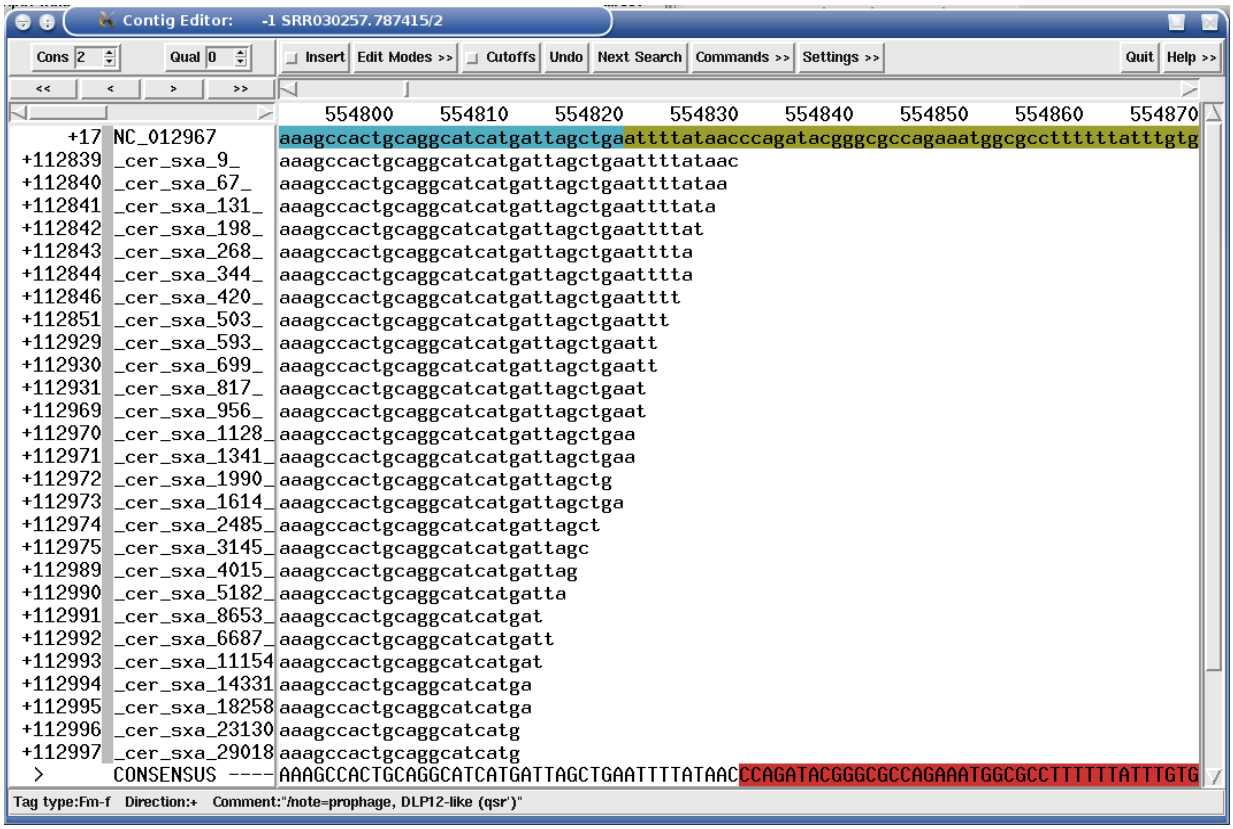

Figure 1.20: "MCVc" tag (Missing CoVerage in Consenus, dark red stretch in figure) showing a genome deletion in Solexa mapping assembly.

Note For bacteria -- and if you use annotated GenBank files as reference sequence -- MIRA will also output some nice lists directly usable (in Excel) by biologists, telling them which gene was affected by what kind of SNP, whether it changes the protein, the original and the mutated protein sequence etc.pp.

1.5.7 MIRA has ... much more

- Extensive possibilities to clip data if needed: by quality, by masked bases, by A/T stretches, by evidence from other reads, ...
- Routines to re-extend reads into clipped parts if multiple alignment allows for it.
- Read in ancillary data in different formats: EXP, NCBI TRACEINFO XML, SSAHA2, SMALT result files and text files.
- Detection of chimeric reads.
- Pipeline to discover SNPs in ESTs from different strains (miraSearchESTSNPs)
- Support for many different of input and output formats (FASTA, EXP, FASTQ, CAF, MAF, ...)
- Automatic memory management (when RAM is tight)
- Over 150 parameters to tune the assembly for a lot of use cases, many of these parameters being tunable individually depending on sequencing technology they apply to.

1.6 Versions, Licenses, Disclaimer and Copyright

1.6.1 Versions

There are two kind of versions for MIRA that can be compiled form source files: production and development.

Production versions are from the stable branch of the source code. These versions are available for download from SourceForge.

Development versions are from the development branch of the source tree. These are also made available to the public and should be compiled by users who want to test out new functionality or to track down bugs or errors that might arise at a given location. Release candidates (rc) also fall into the development versions: they are usually the last versions of a given development branch before being folded back into the production branch.

1.6.2 License

1.6.2.1 MIRA

MIRA has been put under the GPL version 2.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

You may also visit http://www.opensource.org/licenses/gpl-2.0.php at the Open Source Initiative for a copy of this licence.

1.6.2.2 Documentation

The documentation pertaining to MIRA is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

1.6.3 Copyright

1.6.4 External libraries

MIRA uses the excellent Expat library to parse XML files. Expat is Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper as well as Copyright © 2001, 2002 Expat maintainers.

See http://www.libexpat.org/ and http://sourceforge.net/projects/expat/ for more information on Expat.

1.7 Getting help / Mailing lists / Reporting bugs

Please try to find an answer to your question by first reading the documents provided with the MIRA package (FAQs, READMEs, usage guide, guides for specific sequencing technologies etc.). It's a lot, but then again, they hopefully should cover 90% of all questions.

If you have a tough nut to crack or simply could not find what you were searching for, you can subscribe to the MIRA talk mailing list and send in your question (or comment, or suggestion), see http://www.chevreux.org/mira_mailinglists.html for more information on that. Now that the number of subscribers has reached a good level, there's a fair chance that someone could answer your question before I have the opportunity or while I'm away from mail for a certain time.

Note
Please very seriously consider using the mailing list before mailing me directly. Every question which can be answered by participants of the list is time I can invest in development and documentation of MIRA. I have a day job as bioinformatician which has nothing to do with MIRA and after work hours are rare enough nowadays.
Furthermore, Google indexes the mailing list and every discussion / question asked on the mailing list helps future users as they show up in Google searches.
Only mail me directly (bach@chevreux.org) if you feel that there's some information you absolutely do not want to share publicly.

Note Subscribing to the list *before sending mails to it* is necessary as messages from non-subscribers will be stopped by the system to keep the spam level low.

To report bugs or ask for new features, please use the new ticketing system at: http://sourceforge.net/apps/trac/mira-assembler/. This ensures that requests do not get lost **and** you get the additional benefit to automatically know when a bug has been fixed (there won't be separate emails sent, that's what bug trackers are there for).

Finally, new or intermediate versions of MIRA will be announced on the separate MIRA announce mailing list. Traffic is very low there as the only one who can post there is me. Subscribe if you want to be informed automatically on new releases of MIRA.

1.8 Author

Bastien Chevreux (mira): bach@chevreux.org

WWW: http://www.chevreux.org/

MIRA can use automatic editing routines for Sanger sequences which were written by Thomas Pfisterer (EdIt): t.pfisterer@dkfz-heidelberg.de

1.9 Miscellaneous

1.9.1 Citing MIRA

Please use these citations:

For mira Chevreux, B., Wetter, T. and Suhai, S. (1999): *Genome Sequence Assembly Using Trace Signals and Additional Sequence Information*. Computer Science and Biology: Proceedings of the German Conference on Bioinformatics (GCB) 99, pp. 45-56.

For miraSearchESTSNPs (was named miraEST in earlier times) Chevreux, B., Pfisterer, T., Drescher, B., Driesel, A. J., Müller, W. E., Wetter, T. and Suhai, S. (2004): *Using the miraEST Assembler for Reliable and Automated mRNA Transcript Assembly and SNP Detection in Sequenced ESTs*. Genome Research, 14(6)

1.9.2 Postcards, gold and jewellery

If you find this software useful, please send the author a postcard. If postcards are not available, a treasure chest full of Spanish doubloons, gold and jewellery will do nicely, thank you.

Chapter 2

Installing MIRA

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'A problem can be found to almost every solution. '*
>
> —Solomon Short

2.1 Where to fetch MIRA

SourceForge: http://sourceforge.net/projects/mira-assembler/

There you will normally find a couple of precompiled binaries -- usually for Linux, sometimes also for Mac OSX -- or the source package for compiling yourself.

Precompiled binary packages are named in the following way:

`mira_miraversion_audience_OS-and-binarytype.tar.bz2`

where

- `miraversion` is usually a version number in three parts, like `3.0.5`, sometimes also followed by some postfix like in `3.2.0rc1` to denote release candidate 1 of the 3.2.0 version of MIRA.
- `audience` is either `prod` or `dev`; denoting either a *production* or a *development* version.

  The development version usually contains more checks and more debugging output to catch potential errors, hence it might run slower. Furthermore, development versions may contain some new code which did not get as extensive testing as usual.
- `OS-and-binarytype` finally define for which operating system and which processor class the package is destined. E.g., `linux-gnu_x86_64_static` contains static binaries for Linux running a 64 bit processor.

Source packages are usually named

`mira-miraversion.tar.bz2`

Examples for packages at SourceForge:

- `mira_3.0.5_prod_linux-gnu_x86_64_static.tar.bz2`
- `mira_3.0.5_prod_linux-gnu_i686_32_static.tar.bz2`
- `mira_3.0.5_prod_OSX_snowleopard_x86_64_static.tar.bz2`
- `mira-3.0.5.tar.bz2`

2.2 Installing from a precompiled binary package

Download the package, unpack it. Inside, there is -- beside other directories -- a `bin`. Copy or move the files and soft-links inside this directory to a directory in your $PATH variable.

Additional scripts for special purposes are in the `scripts` directory. You might or might not want to have them in your $PATH.

Scripts and programs for MIRA from other authors are in the `3rdparty` directory. Here too, you may or may not want to have (some of them) in your $PATH.

2.3 Integration with third party programs (gap4, consed)

MIRA sets tags in the assemblies that can be read and interpreted by the Staden **gap4** package or **consed**. These tags are extremely useful to efficiently find places of interest in an assembly (be it de-novo or mapping), but both **gap4** and **consed** need to be told about these tags.

Data files for a correct integration are delivered in the `support` directory of the distribution. Please consult the README in that directory for more information on how to integrate this information in either of these packages.

2.4 Compiling MIRA yourself

Please follow the instructions in the `INSTALL` file on the top level of the source package.

Over time, things will be transferred to this help file.

Chapter 3

MIRA3 reference

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'The manual only makes sense after you learn the program. '

—Solomon Short

3.1 Synopsis

mira [--project=<name>] [--cwd=<directory>] [--job=*arguments*]
[--fasta[=<filename>] | --fastq[=<filename>] | --caf[=<filename>] | --phd[=<filename>]] [--notraceinfo] [--noclipping[=...]] [--highlyrepetitive] [--lowqualitydata] [--highqualitydata] [--params=<filename>] [-GENERAL:*arguments*]
[-STRAIN/BACKBONE:*arguments*]
[-ASSEMBLY:*arguments*]
[-DATAPROCESSING:*arguments*]
[-CLIPPING:*arguments*]
[-SKIM:*arguments*]
[-ALIGN:*arguments*]
[-CONTIG:*arguments*]
[-EDIT:*arguments*]
[-MISC:*arguments*]
[-DIRECTORY:*arguments*]
[-FILENAME:*arguments*]
[-OUTPUT:*arguments*]
[COMMON_SETTINGS | SANGER_SETTINGS | 454_SETTINGS | IONTOR_SETTINGS | PACBIO_SETTINGS | SOLEXA_SETTI

3.2 Important notes

For an easy introduction on how to use mira, a number of tutorials with step-by-step instructions are available:

1. `mira_usage` for basic Sanger assembly
2. `mira_454` for basic 454 assembly
3. `mira_iontor` for basic Ion Torrent assembly
4. `mira_pacbio` for basic assembly of sequences from Pacific Bioscience
5. `mira_solexadev` for basic mapping assembly of Solexa data

6. `mira_est` for some advice concerning assembly of EST sequence (and miraSearchESTSNPs)
7. `mira_hard` some notes on how to assemble 'hard' data sets: EST data sets or genome projects for eukaryotes, but some prokaryotes also qualify for this
8. `mira_faq` with some frequently asked question

3.3 Requirements

To use mira itself, one doesn't need very much:

- Sequence data in EXP, CAF, PHD, FASTA or FASTQ format (ideally preprocessed)
- Optionally: ancillary information in NCBI traceinfo XML format; ancillary information about strains in tab delimited format, vector screen information generated with **ssaha2** or **smalt**.
- Some memory and disk space. Actually lots of both if you are venturing into 454 or Solexa.

3.4 Working modes

mira has three basic working modes: genome, EST or EST-reconstruction-and-SNP-detection. From version 2.4 on, there is only executable which supports all modes. The name with which this executable is called defines the working mode:

1. **mira** for assembly of genomic data as well as assembly of EST data from one or multiple strains / organisms

   and

2. **miraSearchESTSNPs** for assembly of EST data from different strains (or organisms) and SNP detection within this assembly. This is the former **miraEST** program which was renamed as many people got confused regarding whether to use mira in est mode or miraEST.

Note that **miraSearchESTSNPs** is usually realised as a link to the **mira** executable, the executable decides by the name it was called with which module to start.

3.5 Parameters

Parameters can be given on the command line or loaded via parameter files.

3.5.1 Overview

mira knows two basic parameter types: quick switches and extensive switches.

1. **quick switches**, also dubbed DWIM switches (for '**D**o-**W**hat-**I**-**M**ean'), are easy to use-and-combine switches activating parameter collections for predefined tasks that will suit most people's needs.
2. **extensive switches** offer a way to set about any possible parameter to configure mira for any kind of special need. While the format of extensive switches might look a little bit strange, it is borrowed from the SGI C compiler options and allows both compact command lines as well as readable and / or script generated parameter files.

Due to the introduction of new sequencing technologies like 454, Solexa and ABI SOLiD, the extensive switches had to be split into two groups:

1. **technology independent switches** which control general behaviour of MIRA like, e.g., the number of assembly passes or file names etc.
2. **technology dependent switches** which control behaviour of algorithms where the sequencing technology plays a role. Example for this would be the minimum length of a read (like 200 for Sanger reads and 120 for 454 FLX reads).

More on this a bit further down in this documentation.

As example, a typical call of **mira** using quick switches and some tweaking with extended switches on the command line could look like this:

```
mira --job=denovo,genome,draft,sanger --fasta
  SANGER_SETTINGS
   -ALIGN:min_relative_score=70
   -GENERAL:use_template_information=yes
   -GENERAL:templateinsertsizeminimum=500:templateinsertsizemaximum=2500
```

or in short form

```
mira --job=denovo,genome,draft,sanger --fasta
  SANGER_SETTINGS
   -AL:mrs=70
   -GE:uti=yes:tismin=500:tismax=2500
```

Please note that it is also perfectly legal to decompose the switches so that they can be used more easily in scripted environments (notice the multiple -GE in the following example):

```
mira --job=denovo,genome,draft,sanger --fasta
  SANGER_SETTINGS
   -AL:mrs=70
   -GE:uti=yes
   -GE:tismin=500
   -GE:tismax=2500
```

3.5.2 Quick switches

These switches are '**D**o-**W**hat-**I**-**M**ean' parameter collections for predefined tasks which should suit most people's needs. You might still need a few of the extensive switches, but not too many anymore.

Important note 1: For de-novo assembly of genomes, these switches are optimised for 'decent' coverages that are commonly seen to get you something useful, i.e., $\geq$ 7x for Sanger, >=18x for 454 FLX or Titanium, $\geq$ 25x for 454 GS20 and $\geq$ 30x for Solexa. Should you venture into lower coverage or extremely high coverage (say, >=60x for 454), you will need to adapt a few parameters via extensive switches.

Important note 2: For some switches, the order of appearance in the command line (or parameter file) is important. This is because the quick switches are realised internally as a collection of extensive switches that will overwrite any previously manually set extensive switch. It is generally a good idea to place switches in the order as described in this documentation, that is: first the order dependent quick switches, then other quick switches, then all the other extensive switches.

Warning
E.g. always write `--job=...  --highlyrepetitive` and not `--highlyrepetitive --job=...`. In the same vein, always write `--job=...  -SK:mnr=yes` and not `-SK:mnr=yes --job=...`.

3.5.2.1 Order dependent quick switches

[--job=] The main one-stop-switches for most assemblies. You can choose between two different assembly methods (*denovo* or *mapping*), two different assembly types (*genome* or *est*), two different quality grades (*draft* or *accurate*) and mix different sequencing technologies (*sanger*, *454*, *iontor*, *solexa*). This switch is explained in more detail in the subsection "The --job= switch in detail".

[-highlyrepetitive] A modifier switch for genome data that is deemed to be highly repetitive. The assemblies will run slower due to more iterative cycles that give mira a chance to resolve nasty repeats.

[--noclipping=...] Switches off clipping options for given sequencing technologies. Technologies can be *sanger*, *454*, *iontor*, *solexa* or *solid*. Multiple entries separated by comma.

Note that [-CL:pec] and the chimera clipping [-CL:ascdc] are not switched off by this parameter and should be switched off separately.

Examples:

1. Switch off 454 and Solexa (but keep eventually keep Sanger clipping): `--noclipping=454,solexa`
2. Switch off all: `--noclipping` or `--noclipping=all`

[--notraceinfo] Switches off loading TRACEINFO ancillary data in XML files for **all** technologies. Place it after [--fasta] and/or [--job=] quick switches.

[--params|--parameterfile=<filename>] Loads parameters from the filename given. Allows a maximum of 10 levels of recursion, i.e. a --params option appearing within a file that loads other parameter files (though I cannot think of useful applications with more than 3 levels).

[--cwd=<directory>] When encountered during parameter parsing, MIRA will change the working directory immediately to the directory given and read and write files there.

Therefore, a call like `mira -DI:cwd=/somedir --params=myparameters.txt` will be enough to let MIRA change to the directory `/somedir` and then read further parameters from a text file `myparamaters.txt` (which should be present there) and at the same time have all the input and output of the assembly occuring in firectory `/somedir`.

[--fasta | --fasta=<filename>] Sets parameters suited for loading sequences from FASTA files. The version with =<filename> will also set the input file to the given filename.

[--phd | --phd=<filename>] Sets parameters suited for loading sequences from PHD files. The version with =<filename> will also set the input file to the given filename.

[--caf | --caf=<filename>] Sets parameters suited for loading sequences from CAF files. The version with =<filename> will also set the input file to the given filename.

3.5.2.2 Order independent quick switches

The following switches can be placed anywhere on the command line without interfering with other switches:

[--project=<name>] Default is mira. Defines the project name for this assembly. The project name automatically influences the name of input and output files / directories. E.g. in the default setting, the file names for the output of the assembly in FASTA format would be `mira_out.fasta` and `mira_out.fasta.qual`. Setting the project name to "MyProject" would generate `MyProject_out.fasta` and `MyProject_out.fasta.qual`. See also **-FILENAME:** and **-DIRECTORY:** for a list of names that are influenced.

[--projectin=<name>] Default is mira. Works like [--project=<name>], but takes only effect on input files.

[--projectout=<name>] Default is mira. Works like [--project=<name>], but takes only effect on output files.

Note: A double dash (e.g. --params) may also be used instead of a single one in front of the quick switches.

Examples for using these switches can be found in the documentation files describing mira usage.

3.5.2.3 The --job= switch in detail

This is the main one-stop-switches for most assemblies. You need to make your choice mainly in four steps and in the end concatenate your choices to the [--job=] switch:

1. are you building an assembly from scratch (choose: *denovo*) or are you mapping reads to an existing backbone sequence (choose: *mapping*)? Pick one. Leaving this out automatically chooses *denovo* as default.
2. are the data you are assembling forming a larger contiguous sequence (choose: *genome*) or are you assembling small fragments like in EST or mRNA libraries (choose: *est*)? Pick one. Leaving this out automatically chooses *genome* as default.
3. do you want a quick and dirty assembly for first insights (choose: *draft*) or an assembly that should be able to tackle even most nasty cases (choose: *accurate*)? Pick one. Leaving this out automatically chooses *accurate* as default.
4. finally, which sequencing technologies have created your reads: *sanger*, *454*, *iontor*, *solexa* or *solid*? You can pick multiple. Leaving this out automatically chooses only *sanger* as default.

Once you're done with your choices, concatenate everything with commas and you're done. E.g.: '`--job=denovo,genome,draft,sanger,iontor`' will give you a de-novo assembly of a genome in draft quality using a hybrid assembly method with Sanger and Ion Torrent reads.

3.5.3 Extensive switches

Extensive switches open up the full panoply of possibilities the MIRA assembler offers. This ranges from fine-tuning assemblies with the quick switches from above to setting parameters in a way so that mira is suited also for very special assembly cases.

Important note: As soon as you use a quick switch (especially --job), the 'default' settings given for extensive switches in the manual below probably do not apply anymore as the quick switch tweaks a lot of extensive switches internally.

3.5.3.1 Technology sections

With the introduction of new sequencing technologies, mira also had to be able to set values that allow technology specific behaviour of algorithms. One simple example for this could be the minimum length a read must have to be used in the assembly. For Sanger sequences, having this value to be 150 (meaning a read should have at least 150 unclipped bases) would be a very valid albeit conservative choice. For 454 reads and especially Solexa and ABI SOLiD reads however, this value would be ridiculously high.

To allow very fine grained behaviour, especially in hybrid assemblies, and to prevent the explosion of parameter names, mira uses *technology mode switching* in the parameter files or on the command line.

Example: assume the following basic command line

```
mira -fasta -job=denovo,genome,draft,454,solexa
```

Here is exemplary a part of the output of used parameters that mira will show:

```
...

Assembly options (-AS):
    Number of passes (nop)                      : 1
    Skim each pass (sep)                        : yes
    Maximum number of RMB break loops (rbl)     : 1
    Spoiler detection (sd)                      : no
    Last pass only (sdlpo)                      : yes

    Minimum read length (mrl)                   :  [san]  80
                                                   [454]  40
                    [sxa]  20
    Base default quality (bdq)                  :  [san]  10
```

```
                                                    [454]  10
                  [sxa]  10
...
```

You can see the two different kind of settings that mira uses: *common settings* (like [-AS:nop]) and *technology dependent settings* (like [-AS:mrl]), where for each sequencing technology used in the project, the setting can be different.

How would one set a minimum read length of 80 and a base default quality of 10 for 454 reads, but for Solexa reads a minimum read length of 30 with a base default quality of 15? The answer:

```
mira -job=denovo,genome,draft,454,solexa  -fasta
  454_SETTINGS -AS:mrl=80:bdq=10 SOLEXA_SETTINGS -AS:mrl=30:bdq=15
```

Notice the ..._SETTINGS section in the command line (or parameter file): these tell mira that all the following parameters until the advent of another switch are to be set specifically for the said technology.

Beside COMMON_SETTINGS there are currently 6 technology settings available:

1. SANGER_SETTINGS
2. 454_SETTINGS
3. IONTOR_SETTINGS
4. PACBIO_SETTINGS
5. SOLEXA_SETTINGS
6. SOLID_SETTINGS

Some settings of mira are influencing global behaviour and are not related to a specific sequencing technology, these must be set in the COMMON_SETTINGS environment. For example, it would not make sense to try and set different number of assembly passes for each technology like in

```
mira -job=denovo,genome,draft,454,solexa  -fasta
  454_SETTINGS -AS:nop=4 SOLEXA_SETTINGS -AS:nop=3
```

mira will complain about cases like these. Simply set those common settings in an area prefixed with the COMMON_SETTINGS switch like in

```
mira -job=denovo,genome,draft,454,solexa -fasta
  COMMON_SETTINGS -AS:nop=4 454_SETTINGS ... SOLEXA_SETTINGS ...
```

Since MIRA 3rc3, the parameter parser will help you by checking whether parameters are correctly defined as COMMON_SETTINGS or technology dependent setting.

3.5.3.2 -GENERAL (-GE)

General options control the type of assembly to be performed and other switches not belonging anywhere else.

[project(pro)=*string*] Same as the quick switch [-project]. Defines the name of your project and influences the naming of your input and output files.

[number_of_threads(not)=*1* $\leq$ *integer* $\leq$ *256*] Default is 2. Master switch to set the number of threads used in different parts of mira.

Note 1: currently only the SKIM algorithm uses multiple threads, other parts will follow.

Note 2: Although the main data structures are shared between the threads, there's some additional memory needed for each thread.

Note 3: when running the SKIM in parallel threads, MIRA can give different results when started with the same data and same arguments. While the effect could be averted for SKIM, the memory cost for doing so would be an additional 50% for one of the large tables, so this has not been implemented at the moment. Besides, at the latest when the Smith-Watermans run in parallel, this could not be easily avoided at all.

[automatic_memory_management(amm)=*on|yes|1, off|no|0*] Default is Yes. Whether mira tries to optimise run time of certain algorithms in a space/time trade-off memory usage, increasing or reducing some internal tables as memory permits.

Note 1: This functionality currently relies on the `/proc` file system giving information on the system memory ("MemTotal" in /proc/meminfo) and the memory usage of the current process ("VmSize" in `/proc/self/status`). If this is not available, the functionality is switched off.

Note 2: The automatic memory management can only work if there actually is unused system memory. It's not a wonder switch which reduces memory consumption. In tight memory situations, memory management has no effect and the algorithms fall back to minimum table sizes. This means that the effective size in memory can grow larger than given in the memory management parameters, but then MIRA will try to keep the additional memory requirements to a minimum.

[max_process_size(mps)=*0 ≤ integer*] Default is 0. If automatic memory management is used (see above), this number is the size in gigabytes that the MIRA process will use as maximum target size when looking for space/time trade-offs. A value of 0 means that MIRA does not try keep a fixed upper limit.

Note: when in competition to [-GE:kpmf] (see below), the smaller of both sizes is taken as target. Example: if your machine has 64 GiB but you limit the use to 32 GiB, then the MIRA process will try to stay within these 32 GiB.

[keep_percent_memory_free(kpmf)=*0 ≤ integer*] Default is 10. If automatic memory management is used (see above), this number works a bit like [-GE:mps] but the other way round: it tries to keep x percent of the memory free.

Note: when in competition to [-GE:mps] (see above), the argument leaving the most memory free is taken as target. Example: if your machine has 64 GiB and you limit the use to 42 GiB via [-GE:mps] but have a [-GE:kpmf] of 50, then the MIRA process will try to stay within 64-(64*50%)=32 GiB.

[est_snp_pipeline_step(esps)=*1 ≤ integer ≤ 4*] Default is 1. Controls the starting step of the SNP search in EST pipeline and is therefore only useful in miraSearchESTSNPs.

EST assembly is a three step process, each with different settings to the assembly engine, with the result of each step being saved to disk. If results of previous steps are present in a directory, one can easily "play around" with different setting for subsequent steps by reusing the results of the previous steps and directly starting with step two or three.

[use_template_information(uti)=*on|yes|1, off|no|0*] Default is Yes. Two reads sequenced from the same clone template form a read pair with a known minimum and maximum distance. This feature will definitively help for contigs containing lots of repeats. Set this to 'yes' if your data contains information on insert sizes (e.g. in paired-end sequencing).

Information on insert sizes can be given via the SI tag in EXP files (for each read pair individually), via insert_size and insert_stdev elements of NCBI TRACEINFO XML files or for the whole project using [-GE:tismin] and [-GE:tismax] (see below).

Additional information to set the orientation of the read-pairs can be given via [-GE:tpbd].

[templateinsertsizeminimum(tismin)=*integer*] Default is -1. The default value for the minimum template size for reads that have no template size in ancillary data. If -1 is used as value, then no default value is given and reads without ancillary data giving this number will behave as if they had no template.

[templateinsertsizemaximum(tismax)=*integer*] Default is -1. The default value for the maximum template size for reads that have no template size in ancillary data. If -1 is used as value, then no default value is given and reads without ancillary data giving this number will behave as if they had no template.

[templatepartnerbuilddirection(tpbd)=*-1 or 1*] Default is -1 for all sequencing technologies.

This value tells MIRA how read-pairs of a template must be oriented in a contig to be valid. A value of "-1" means the orientation must be 5'-3' to 3'-5', a value of "1" means 5'-3' to 5'-3'.

Set this to "1" if you assemble paired-end 454 data downloaded from the Short Read Archives (SRAs, at the NCBI and EMBL). Set this also to "1" for Solexa data where the paired-end sequencing protocol used creates 5'-3' to 5'-3' pairs.

Note Although with Solexa it is possible to build libraries in both directions, with MIRA it is currently not possible to mix within the same sequencing technology paired-end reads which need "-1" as direction with mate-pair reads which have "1" as direction. This will be worked on if the need arises.

[print_date(pd)=*on|yes|1, off|no|0*] Default is yes. Controls whether date and time are printed out during the assembly. Suppressing it is not useful in normal operation, only when debugging or benchmarking.

3.5.3.3 -LOADREADS options (-LR)

Here one defines what type of reads to load.

[load_sequence_data(lsd)=*on|yes|1, off|no|0*] Default is No. Defines whether to load data generated by a given technology.

[file_type(ft)=*fofnexp, fasta, fastq, caf, phd, fofnphd*] Default is fasta. Takes effect only when [-LR:lsd]) is 'yes'.

Defines whether to load for assembly from FASTA sequences (`<projectname>_in.fasta`) and their qualities (`<projectn_in.fasta.qual`), from a FASTQ file (`<projectname>_in.fastq`), from EXP files from a file of filenames (`<projectname>_in.fofn`), from a phd file (`<projectname>_in.phd`) or from a CAF file (`<projectname>_in.caf`) and assemble or eventually reassemble it.

Note 1: Only Sanger supports all file types. 454, Ion Torrent and Solexa support only FASTA and FASTQ.

Note 2: fofnphd currently not available.

[external_quality(eq)=*none, SCF*] Default is SCF. Takes effect only when [-LR:lsd]) is 'yes' and for Sanger reads.

Defines the source format for reading qualities from external sources. Normally takes effect **only** when these are not present in the format of the load_job project (EXP and FASTA can have them, CAF and PHD must have them).

[external_quality_override(eqo)=*on|yes|1, off|no|0*] Takes effect only when [-LR:lsd]) is 'yes' and for Sanger reads.

Default is no, only takes effect when load_job is fofnexp. Defines whether or not the qualities from the external source override the possibly loaded qualities from the load_job project. This might be of use in case some post-processing software fiddles around with the quality values of the input file but one wants to have the original ones.

[discard_read_on_eq_error(droeqe)=*on|yes|1, off|no|0*] Default is yes. Takes effect only when [-LR:lsd]) is 'yes' and for Sanger reads.

Should there be a major mismatch between the external quality source and the sequence (e.g.: the base sequence read from a SCF file does not match the originally read base sequence), should the read be excluded from assembly or not. If not, it will use the qualities it had before trying to load the external qualities (either default qualities or the ones loaded from the original source).

[wants_quality_file(wqf)=*on|yes|1, off|no|0*] Default is yes. When set to yes, MIRA will stop the assembly if there is no quality file for a given sequence file. E.g., if the FASTA quality file is missing when loading from FASTA.

[readnaming_scheme(rns)=*sanger, tigr, fr, stlouis, solexa*] Default is sanger for Sanger sequencing data, fr for 454 and Ion Torrent while solexa for Solexa. Defines the read naming scheme for read suffixes. These suffixes can be used by mira to deduce a template name if none is given in ancillary data.

Currently supported: Sanger centre, TIGR, simple forward / reverse naming, St. Louis schemes and Solexa/Illumina schemes are supported out of the box.

How to choose: please read the documentation available at the different centres or ask your sequence provider. In a nutshell (and probably over-simplified):

Sanger scheme "somename.*[pqsfrw][12][bckdeflmnpt][a|b|c|...*" (e.g. U13a08f10.p1ca), but the length of the postfix must be at least 4 characters, i.e., ".p" alone will not be recognised.

Usually, ".p" + 3 characters or "f" + 3 characters are used for forwards reads, while reverse complement reads take either ".q" or ".r" (+ 3 characters in both cases).

TIGR scheme "somename*TF*|TR*|TA**" (e.g. GCPBN02TF or GCPDL68TABRPT103A58B),

Forward reads take "TF*", reverse reads "TR*".

Forward/Reverse scheme "somename.*[fr]**" (e.g. E0K6C4E01DIGEW.f or E0K6C4E01BNDXN.r2nd),

".f*" for forward, ".r*" for reverse.

St. Louis scheme "somename.*[sfrxzyingtpedca]**"

Solexa scheme Even simpler than the forward/reverse scheme, it allows only for one two reads per template: "somename/*[12]*"

[solexa_scores_in_qual_file(ssiqf)=*on|yes|1, off|no|0*] Default is no. This switch applies only for sequences from older Illumina / Solexa sequencing technology when loading from FASTA! Defines whether the FASTA quality file contains Solexa scores (which also have negative values) instead of quality values. Solexa scores also have negative values. If set to yes, mira will automatically convert the Solexa scores to phred style quality values.

[fastq_qualoffset(fqqo)=*integer*] Default is 0. This switch applies only for sequences loaded from FASTQ format!

Defines the quality offset used to convert characters into quality values. Usually, 33 is used for FASTQ in Sanger style, Solexa 1.0 format uses 59 (I think) and newer Solexa 1.3 format uses 64.

The default value of 0 switches on routines that try to guess the correct value from the data present in the FASTQ (which they do when the data contains at least one read which at least one base with quality between 0 and 4).

[merge_xmltraceinfo(mxti)=*on|yes|1, off|no|0*] Default is no. Some file formats above (FASTA, PHD or even CAF and EXP) possibly do not contain all the info necessary or useful for each read of an assembly. Should additional information -- like clipping positions etc. -- be available in a XML trace info file in NCBI format (see **File formats**), then set this option to yes and it will be merged to all the data loaded, be it for Sanger, 454, Ion Torrent, Solexa or SOLiD technology. See also **-FILENAME:** for the name of the XML file to load.

Please note: quality clippings given here will override quality clippings loaded earlier (e.g. in EXP files) or performed by mira. Minimum clippings will still be made by the program, though.

[filecheck_only(fo)=*on|yes|1, off|no|0*] Default is no. If set to yes, the project will not be assembled and no assembly output files will be produced. Instead, the project files will only be loaded. This switch is useful for checking consistency of input files.

3.5.3.4 -ASSEMBLY (-AS)

General options for controlling the assembly.

[num_of_passes(nop)=*integer*] Default is dependent of the sequencing technology and assembly quality level. Defines how many iterations of the whole assembly process are done.

As a special use case, a value of 0 will let MIRA just run the following tasks: loading and clipping of reads as well as calculating hash frequencies and read repeat information. The resulting reads can then be found as MAF file in the checkpoint directory; the read repeat information in the info directory.

Early termination: if the number of passes was chosen too high, one can simply create a file `projectname_assembly/projectname_d_chkpt/terminate`. At the beginning of a new pass, MIRA checks for the existence of that file and, if it finds it, acknowledges by renaming it to `terminate_acknowledged` and then run 2 more passes (with special "last pass routines") before finishing the assembly.

[skim_each_pass(sep)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology and assembly quality level. Defines whether the skim algorithm (and with it also the recalculation of Smith-Waterman alignments) is called in-between each main pass. If set to no, skimming is done only when needed by the workflow: either when read extensions are searched for ([-DP:ure]) or when possible vector leftovers are to be clipped ([-CL:pvc]).

Setting this option to yes is highly recommended, setting it to no only for quick and dirty assemblies.

[rmb_break_loops(rbl)=*integer > 0*] Default is dependent of the sequencing technology and assembly quality level. Defines the maximum number of times a contig can be rebuilt during a main assembly passes ([-AS:nop]) if misassemblies due to possible repeats are found.

[max_contigs_per_pass(mcpp)=*integer*] Default is 0. Defines how many contigs are maximally built in each pass. A value of 0 stands for 'unlimited'. Values >0 can be used for special use cases like test assemblies etc.

If in doubt, do not touch this parameter.

[automatic_repeat_detection(ard)=*on|yes|1, off|no|0*] Default is is currently yes. Tells mira to use coverage information accumulated over time to more accurately pinpoint reads that are in repetitive regions.

[coverage_threshold(ardct)=*float > 1.0*] Default is 2.0 for all sequencing technologies in most assembly cases. This option says this: if mira a read has ever been aligned at positions where the total coverage of all reads of the same sequencing technology attained the average coverage times [-AS:ardct] (over a length of [-AS:ardml], see below), then this read is considered to be repetitive.

[min_length(ardml)=*integer > 1*] Default is dependent of the sequencing technology, currently 400 for Sanger and 200 for 454 and Ion Torrent.

A coverage must be at least this number of bases higher than [-AS:ardct] before being really treated as repeat.

[grace_length(ardgl)=*integer > 1*] Default is dependent of the sequencing technology.

[uniform_read_distribution(urd)=*on|yes|1, off|no|0*] Default is currently yes for genome assemblies and no for EST assemblies or assemblies with Solexa data.

Takes effect only if uniform read distribution ([-AS:urd]) is on.

When set to yes, mira will analyse coverage of contigs built at a certain stage of the assembly and estimate an average expected coverage of reads for contigs. This value will be used in subsequent passes of the assembly to ensure that no part of the contig gets significantly more read coverage of reads that were previously identified as repetitive than the estimated average coverage allows for.

This switch is useful to disentangle repeats that are otherwise 100% identical and generally allows to build larger contigs. It is expected to be useful for Sanger and 454 sequences. Usage of this switch with Solexa and Ion Torrent data is currently not recommended.

It is a real improvement to disentangle repeats, but has the side-effect of creating some "contig debris" (small and low coverage contigs, things you normally can safely throw away as they are representing sequence that already has enough coverage).

This switch must be set to no for EST assembly, assembly of transcripts etc. It is recommended to also switch this off for mapping assemblies.

[urd_startinpass(urdsip)=*integer > 0*] Default is dependent of the sequencing technology and assembly quality level. Recommended values are: 3 for an assembly with 3 to 4 passes ([-AS:nop]). Assemblies with 5 passes or more should set the value to the number of passes minus 2.

Takes effect only if uniform read distribution ([-AS:urd]) is on.

[urd_clipoffmultiplier(urdcm)=*float > 1.0*] Default is 1.5 for all sequencing technologies in most assembly cases. The [--highlyrepetitive] quick-switch sets this to 1.2.

This option says this: if mira determined that the average coverage is x, then in subsequent passes it will allow coverage for reads determined to be repetitive to be built into the contig only up to a total coverage of $x*urdcm$. Reads that bring the coverage above the threshold will be rejected from that specific place in the contig (and either be built into another copy of the repeat somewhere else or end up as contig debris).

Please note that the lower [-AS:urdcm] is, the more contig debris you will end up with (contigs with an average coverage less than half of the expected coverage, mostly short contigs with just a couple of reads).

Takes effect only if uniform read distribution ([-AS:urd]) is on.

[keep_long_repeats_separate(klrs)=*on|yes|1, off|no|0*] Default is is dependent on --job quality: currently no for draft and yes for accurate. Switched of for EST assembly.

Tells mira to use keep repeats longer that the length of reads in separate contigs.

[spoiler_detection(sd)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology and assembly quality level. A spoiler can be either a chimeric read or it is a read with long parts of unclipped vector sequence still included (that was too long for the [-CL:pvc] vector leftover clipping routines). A spoiler typically prevents contigs to be joined, MIRA will cut them back so that they represent no more harm to the assembly.

Recommended for assemblies of mid- to high-coverage genomic assemblies, not recommended for assemblies of ESTs as one might loose splice variants with that.

A minimum number of two assembly passes ([-AS:nop]) must be run for this option to take effect.

[sd_last_pass_only(sdlpo)=*on|yes|1, off|no|0*] Default is yes. Defines whether the spoiler detection algorithms are run only for the last pass or for all passes ([-AS:nop]).

Takes effect only if spoiler detection ([-AS:sd]) is on. If in doubt, leave it to 'yes'.

[minimum_read_length(mrl)=*integer* $\geq$ *20*] Default is dependent of the sequencing technology. Defines the minimum length that reads must have to be considered for the assembly. Shorter sequences will be filtered out at the beginning of the process and won't be present in the final project.

[minimum_reads_per_contig(mrpc)=*integer* ≥ *1*] Default is dependent of the sequencing technology and the [--job] parameter. For genome assemblies it's usually around 2 for Sanger, 5 for 454, 5 for Ion Torrent, 5 for PacBio and 10 for Solexa. In EST assemblies, it's currently 2 for all sequencing technologies.

Defines the minimum number of reads a contig must have before it is built or saved by MIRA. Overlap clusters with less reads than defined will not be assembled into contigs but reads in these clusters will be immediately transferred to debris.

This parameter is useful to considerably reduce assembly time in large projects with millions of reads (like in Solexa projects) where a lot of small "junk" contigs with contamination sequence or otherwise uninteresting data may be created otherwise.

Note Important: a value larger 1 of this parameter interferes with the functioning of [-OUT:sssip] and [-OUT:stsip].

[base_default_quality(bdq)=*integer* ≥ *0*] Default is currently 10 for all sequencing technologies. Defines the default base quality of reads that have no quality read from file.

[enforce_presence_of_qualities(epoq)=*on|yes|1, off|no|0*] Default is yes. When set to yes, MIRA will stop the assembly if any read has no quality values loaded.

[use_genomic_pathfinder(ugpf)=*on|yes|1, off|no|0*] Default is yes. MIRA has two different pathfinder algorithms it chooses from to find its way through the (more or less) complete set of possible sequence overlaps: a genomic and an EST pathfinder. The genomic looks a bit into the future of the assembly and tries to stay on safe grounds using a maximum of information already present in the contig that is being built. The EST version on the contrary will directly jump at the complex cases posed by very similar repetitive sequences and try to solve those first and is willing to fall back to first-come-first-served when really bad cases (like, e.g., coverage with thousands of sequences) are encountered.

Generally, the genomic pathfinder will also work quite well with EST sequences (but might get slowed down a lot in pathological cases), while the EST algorithm does not work so well on genomes. If in doubt, leave on yes for genome projects and set to no for EST projects.

[use_emergency_search_stop(uess)=*on|yes|1, off|no|0*] Default is yes. Another important switch if you plan to assemble non-normalised EST libraries, where some ESTs may reach coverages of several hundreds or thousands of reads. This switch lets MIRA save a lot of computational time when aligning those extremely high coverage areas (but only there), at the expense of some accuracy.

[ess_partnerdepth(esspd)=*integer > 0*] Default is 500. Defines the number of potential partners a read must have for MIRA switching into emergency search stop mode for that read.

[use_max_contig_buildtime(umcbt)=*on|yes|1,off|no|0*] Default is no. Defines whether there is an upper limit of time to be used to build one contig. Set this to yes in EST assemblies where you think that extremely high coverages occur. Less useful for assembly of genomic sequences.

[buildtime_in_seconds(bts)=*integer > 0*] Default is 10000. Depending on [-AS:umcbt] above, this number defines the time in seconds allocated to building one contig.

3.5.3.5 -STRAIN/BACKBONE (-SB)

General options for controlling backbone options for mapping assemblies as well as general strain information.

[load_straindata(lsd)=*on|yes|1, off|no|0*] Default is no. Straindata is a key value file, one read per line. First the name of the read, then the strain name of the organism the read comes from. It is used by the program to differentiate different types of SNPs appearing in organisms and classifying them.

[assign_default_strain(ads)=*on|yes|1, off|no|0*] Default is no for *de-novo* assemblies and yes for *mapping*.

Defines whether, after having loaded all data from all possible source, MIRA will assign a strain name to reads which didn't get strain information via said data files (either NCBI TRACEINFO XML files or the simple MIRA straindata files). The strain name to assign the is determined via [-SB:dsn] (see below).

[default_strain_name(dsn)=*string*] Default is StrainX. Defines the strain name to assign to reads which don't have a strain name after loading, works only if [-SB:ads=yes] (see above).

[load_backbone(lb)=*on|yes|1, off|no|0*] Default is no. A backbone is a sequence (or a previous assembly) that is used as template for a mapping assembly. The current assembly process will assemble reads first to those loaded backbone contigs before creating new contigs (if any).

This feature is helpful for assembling against previous (and already possibly edited) assembly iterations, or to make a comparative assembly of two very closely related organisms. Please read "very closely related" as in: only SNP mutations or short indels present.

[startbackboneusage_inpass(sbuip)=*0 < integer*] Default is dependent on assembly quality level chosen: 0 for 'draft' and [-AS:nop] divided by 2 for 'accurate'.

When assembling against backbones, this parameter defines the pass iteration (see [-AS:nop]) from which on the backbones will be really used. In the passes preceding this number, the non-backbone reads will be assembled together as if no backbones existed. This allows mira to correctly spot repetitive stretches that differ by single bases and tag them accordingly. Note that full assemblies are considerably slower than mapping assemblies, so be careful with this when assembling millions of reads.

Rule of thumb: if backbones belong to same strain as reads to assemble, set to 1. If backbones are a different strain, then set [-SB:sbuib] to 1 lower than [-AS:nop] (example: nop=4 and sbuip=3).

[backbone_strainname(bsn)=*string*] Default isReferenceStrain. Defines the name of the strain that the backbone sequences have.

[backbone_strainname_forceforall(bsnffa)=*on|yes|1, off|no|0*] Default is no. Useful when using CAF as input for backbone: forces all reads of the backbone contigs to get assigned the new backbone strain, even if they previously had other strains assigned.

Main usage is in multi-step hybrid assemblies.

[backbone_strain_name(bsn)=*string*] Default is Default is an empty string. Useful when using CAF as input for backbone: when set to a given strain name, mira will internally use only reads from the given strain to build the rails it will use to align reads.

Main usage is in multi-step hybrid assemblies.

[backbone_filetype(bft)=*fasta, caf, gbf*] Default is fasta. Defines the filetype of the backbone file given. Currently only FASTA, CAF and GBF files are supported.

When GBF (GenBank files, more commonly named '.gbk') files are loaded, the features within theses files are automatically transformed into Staden compatible tags and get passed through the assembly.

[backbone_raillength(brl)=*0 ≤ integer ≤ 10000*] Default is 0. Parameter for the internal sectioning size of the backbone to compute optimal alignments. Should be set to two times length of longest read in input data + 15%. When set to 0, MIRA will compute optimal values from the data loaded.

[backbone_railoverlap(bro)=*0 ≤ integer ≤ 2000*] Default is 0. Parameter for the internal sectioning size of the backbone to compute optimal alignments. Should be set to length of the longest read. When set to 0, MIRA will compute optimal values from the data loaded.

[backbone_basequality(bbq)=*-1 ≤ integer ≤ 100*] Default is -1. Defines the default quality that the backbone sequences have if they came without quality values in their files (like in GBF format or when FASTA is used without .qual files). A value of -1 mira to use the same default quality for backbones as for reads.

[also_build_new_contigs(abnc)=*on|yes|1, off|no|0*] Default is no. Standard mapping assembly mode of the assembler is to map available reads to a backbone and discard reads that do not fit. If set to 'yes', mira will use reads that did not map to the backbone(s) to make new contigs (if possible). Please note: while a simple mapping assembly is comparatively cheap in terms of memory and time consumed, setting this option to 'yes' means that behind the scenes data for a full blown de-novo assembly is generated in addition to the data needed for a mapping assembly, which makes it a bit more costly that a de-novo assembly per se.

3.5.3.6 -DATAPROCESSING (-DP)

Options for controlling some data processing during the assembly.

[use_read_extension(ure)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used: yes for Sanger, no for all others. mira expects the sequences it is given to be quality clipped. During the assembly though, it will try to extend reads into the clipped region and gain additional coverage by analysing Smith-Waterman alignments between reads that were found to be valid. Only the right clip is extended though, the left clip (most of the time containing sequencing vector) is never touched.

[read_extension_window_length(rewl)=*integer > 0*] Default is dependent of the sequencing technology used. Only takes effect when [-DP:ure] (see above) is set to yes. The read extension routines use a sliding window approach on Smith-Waterman alignments. This parameter defines the window length.

[read_extension_with_maxerrors(rewme)=*integer > 0*] Default is dependent of the sequencing technology used. Only takes effect when [-DP:ure] (see above) is set to yes. The read extension routines use a sliding window approach on Smith-Waterman alignments. This parameter defines the number maximum number of errors (=disagreements) between two alignment in the given window.

[first_extension_in_pass(feip)=*integer* $\geq$ *0*] Default is dependent of the sequencing technology used. Only takes effect when [-DP:ure] (see above) is set to yes. The read extension routines can be called before assembly and/or after each assembly pass (see [-AS:nop]). This parameter defines the first pass in which the read extension routines are called. The default of 0 tells mira to extend the reads the first time before the first assembly pass.

[last_extension_in_pass(leip)=*integer* $\geq$ *0*] Default is dependent of the sequencing technology used. Only takes effect when [-DP:ure] (see above) is set to yes. The read extension routines can be called before assembly and/or after each assembly pass (see [-AS:nop]). This parameter defines the last pass in which the read extension routines are called. The default of 0 tells mira to extend the reads the last time before the first assembly pass.

3.5.3.7 -CLIPPING (-CL)

Controls for clipping options: when and how sequences should be clipped.

Every option in this section can be set individually for every sequencing technology, giving a very fine grained control on how reads are clipped for each technology.

[merge_ssahasmaltvectorscreen(msvs)=*on|yes|1, off|no|0*] Default is no. Uses the parameters [-CL:msvsgs:msvsmfg:msvsme (see below).

Before running mira, the **ssaha2** or **smalt** programs from the Sanger centre can be used to detect possible vector sequence stretches in the input data for the assembly. This parameter - if set to yes - will let mira load the result file of a ssaha2 or smalt run and tag the possible vector sequences at the ends of reads.

ssaha2 must be called like this "`ssaha2 <ssaha2options> vector.fasta sequences.fasta`" to generate an output that can be parsed by mira. In the above example, replace `vector.fasta` by the name of the file with your vector sequences and `sequences.fasta` by the name of the file containing your sequencing data.

smalt must be called like this: "`smalt map -f ssaha <ssaha2options> hash_index sequences.fasta`"

This makes you basically independent from any other commercial or license-requiring vector screening software. For Sanger reads, a combination of **lucy** and **ssaha2** or **smalt** together with this parameter should do the trick. For reads coming from 454 pyro-sequencing, **ssaha2** or **smalt** and this parameter will also work very well. See the usage manual for a walkthrough example on how to use SSAHA2 / SMALT screening data.

Note 1: the output format of SSAHA2 must the native output format (`-output ssaha2`). For SMALT, the output option `-f ssaha` must be used. Other formats cannot be parsed by MIRA.

Note 2: when using SSAHA2 results, the input file must be named `<projectname>_ssaha2vectorscreen_in.txt`. When using SMALT results, the input file must be named `<projectname>_smaltvectorscreen_in.txt`.

Note 3: if both a ssah2 and smalt result file are present, both will be read.

Note 4: I currently use the following SSAHA2 options: `-kmer 8 -skip 1 -seeds 1 -score 12 -cmatch 9 -ckmer 6`

Note 5: Anyone contributing SMALT parameters?

Note 6: the sequence vector clippings generated from SSAHA2 / SMALT data do not replace sequence vector clippings loaded via the EXP, CAF or XML files, they rather extend them.

[msvs_gap_size(msvsgs)=*integer* $\geq$ *0*] Default is dependent of the sequencing technology used. Takes effect only if [-CL:msvs] is yes. While performing the clip of screened vector sequences, mira will look if it can merge larger chunks of sequencing vector bases that are a maximum of [-CL:msvgsgs] apart.

[msvs_max_front_gap(msvsmfg)=*integer* $\geq$ *0*] Default is dependent of the sequencing technology used. Takes effect only if [-CL:msvs] is yes. While performing the clip of screened vector sequences at the start of a sequence, mira will allow up to this number of non-vector bases in front of a vector stretch.

[msvs_max_end_gap(msvsmeg)=*integer* $\geq$ *0*] Default is dependent of the sequencing technology used. Takes effect only if [-CL:msvs] is yes. While performing the clip of screened vector sequences at the end of a sequence, mira will allow up to this number of non-vector bases behind a vector stretch.

[possible_vector_leftover_clip(pvlc)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used: yes for Sanger, no for any other. mira will try to identify possible sequencing vector relics present at the start of a sequence and clip them away. These relics are usually a few bases long and were not correctly removed from the sequence in data preprocessing steps of external programs.

You might want to turn off this option if you know (or think) that your data contains a lot of repeats and the option below to fine tune the clipping behaviour does not give the expected results.

You certainly want to turn off this option in EST assemblies as this will quite certainly cut back (and thus hide) different splice variants. But then make certain that you pre-processing of Sanger data (sequencing vector removal) is good, other sequencing technologies are not affected then.

[pvc_maxlenallowed(pvcmla)=*integer* $\geq$ *0*] Default is dependent of the sequencing technology used. The clipping of possible vector relics option works quite well. Unfortunately, especially the bounds of repeats or differences in EST splice variants sometimes show the same alignment behaviour than possible sequencing vector relics and could therefore also be clipped.

To refrain the vector clipping from mistakenly clip repetitive regions or EST splice variants, this option puts an upper bound to the number of bases a potential clip is allowed to have. If the number of bases is below or equal to this threshold, the bases are clipped. If the number of bases exceeds the threshold, the clip is **NOT** performed.

Setting the value to 0 turns off the threshold, i.e., clips are then always performed if a potential vector was found.

[quality_clip(qc)=*on|yes|1, off|no|0*] Default is no. This will let mira perform its own quality clipping before sequences are entered into the assembly. The clip function performed is a sequence end window quality clip with back iteration to get a maximum number of bases as useful sequence. Note that the bases clipped away here can still be used afterwards if there is enough evidence supporting their correctness when the option [-DP:ure] is turned on.

Warning: The windowing algorithm works pretty well for Sanger, but apparently does not like 454 type data. It's advisable to not switch it on for 454. Beside, the 454 quality clipping algorithm performs a pretty decent albeit not perfect job, so for genomic 454 data (not! ESTs), it is currently recommended to use a combination of [-CL:emrc] and [-DP:ure].

[qc_minimum_quality(qcmq)=*integer* $\geq$ *15 and* $\leq$ *35*] Default is dependent of the sequencing technology used. This is the minimum quality bases in a window require to be accepted. Please be cautious not to take too extreme values here, because then the clipping will be too lax or too harsh. Values below 15 and higher than 30-35 are not recommended.

[qc_window_length(qcwl)=*integer* $\geq$ *10*] Default is dependent of the sequencing technology used. This is the length of a window in bases for the quality clip.

[bad_stretch_quality_clip (bsqc)=*on|yes|1, off|no|0*] Default is no. This option allows to clip reads that were not correctly preprocess and have unclipped bad quality stretches that might prevent a good assembly.

mira will search the sequence in forward direction for a stretch of bases that have in average a quality less than a defined threshold and then set the right quality clip of this sequence to cover the given stretch.

[bsqc_minimum_quality (bsqcmq)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. Defines the minimum average quality a given window of bases must have. If this quality is not reached, the sequence will be clipped at this position.

[bsqc_window_length (bsqcwl)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. Defines the length of the window within which the average quality of the bases are computed.

[maskedbases_clip(mbc)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used. This will let mira perform a 'clipping' of bases that were masked out (replaced with the character X). It is generally not a good idea to use mask bases to remove unwanted portions of a sequence, the EXP file format and the NCBI traceinfo format have excellent possibilities to circumvent this. But because a lot of preprocessing software are built around cross_match, scylla- and phrap-style of base masking, the need arose for mira to be able to handle this, too. mira will look at the start and end of each sequence to see whether there are masked bases that should be 'clipped'.

[mbc_gap_size(mbcgs)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. While performing the clip of masked bases, mira will look if it can merge larger chunks of masked bases that are a maximum of [-CL:mbcgs] apart.

[mbc_max_front_gap(mbcmfg)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. While performing the clip of masked bases at the start of a sequence, mira will allow up to this number of unmasked bases in front of a masked stretch.

[mbc_max_end_gap(mbcmeg)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. While performing the clip of masked bases at the end of a sequence, mira will allow up to this number of unmasked bases behind a masked stretch.

[lowercase_clip(lcc)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used: on for 454 data, off for all others. This will let mira perform a 'clipping' of bases that are in lowercase at both ends of a sequence, leaving only the uppercase sequence. Useful when handling 454 data that does not have ancillary data in XML format.

[clip_polyat(cpat)=*on|yes|1, off|no|0*] Default is no. This option is useful in EST assembly. Poly-A stretches in forward reads and poly-T stretches in reverse reads that were not correctly masked or clipped in preprocessing steps from external programs get clipped or tagged here. The assembler will not use these stretches for critical operations.

[cp_keep_poly_signal (cpkps)=*on|yes|1, off|no|0*] Default is no. This option is currently **not active** (as of version 2.9.22).

In future, this will allow to keep the poly-A signal in the reads and tag them. The tags provide a good visual anchor when looking at the assembly with different programs.

[cp_min_signal_len(cpmsl)=*integer > 0*] Default is 10. Only takes effect when [-CP:cpat] (see above) is set to yes. Defines the number of ``A" (in forward direction) or ``T" (in reverse direction" must be present to be considered a poly-A signal stretch.

[cp_max_errors_allowed(cpmea)=*integer > 0*] Default is 1. Only takes effect when [-CL:cpat] (see above) is set to yes. Defines the maximum number of errors allowed in the potential poly-A signal stretch. The distribution of these errors is not important.

[cp_max_gap_from_end(cpmgfe)=*integer > 0*] Default is 9. Only takes effect when [-CL:cpat] (see above) is set to yes.Defines the number of bases from the end of a sequence (if masked: from the end of the masked area) within which a poly-A signal stretch is looked for.

[ensure_minimum_left_clip(emlc)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used. If on, ensures a minimum left clip on each read according to the parameters in [-CL:mlcr:smlc]

[minimum_left_clip_required(mlcr)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. If [-CL:emlc] is on, checks whether there is a left clip which length is at least the one specified here.

[set_minimum_left_clip(smlc)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. If [-CL:emlc] is on and actual left clip is < [-CL:mlcr], set left clip of read to the value given here.

[ensure_minimum_right_clip(emrc)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used. If on, ensures a minimum right clip on each read according to the parameters in [-CL:mrcr:smrc]

[minimum_right_clip_required(mrcr)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. If [-CL:emrc] is on, checks whether there is a right clip which length is at least the one specified here.

[set_minimum_right_clip(smrc)=*integer* ≥ *0*] Default is dependent of the sequencing technology used. If [-CL:emrc] is on and actual right clip is < [-CL:mrcr], set the length of the right clip of read to the value given here.

[apply_skim_chimeradetectionclip(ascdc)=*on|yes|1, off|no|0*] Default is yes for [--job=genome] assemblies and no for [--job=est] assemblies.

The SKIM routines of MIRA can be also used without much time overhead to find chimeric reads. When this parameter is set, MIRA will use that info to cut back chimeras to their longest non-chimeric length.

Warning When working on low coverage data (e.g. < 5 to 6x Sanger and < 10x 454 or 10x Ion Torrent, you may want to switch off this option if you try to go for the longest contigs. Reason: single reads joining otherwise disjunct contigs will probably be categorised as chimeras.

[apply_skim_junkdetectionclip(asjdc)=*on|yes|1, off|no|0*] Default is currently no.

The SKIM routines of MIRA can be also used without much time overhead to find junk sequence at end of reads. When this parameter is set, MIRA will use that info to cut back junk in reads.

It is currently suggested to leave this parameter switched off as the routines seem to be a bit too "trigger happy" and also cut back perfectly valid sequences.

[propose_end_clips(pec)=*on|yes|1, off|no|0*] Default is is dependent on --job quality: currently yes for all genome assemblies. Switched off for EST assemblies (but one wmight want to switch it on sometimes).

This implements a pretty powerful strategy to ensure a good "high confidence region" (HCR) in reads, basically eliminating 99.9% of all junk at the 5' and 3' ends of reads. Note that one still must ensure that sequencing vectors (Sanger) or adaptor sequences (454, Solexa ion Torrent) are "more or less" clipped prior to assembly.

Warning Extremely effective, but should NOT be used for very low coverage genomic data, for EST projects or if one wants to retain rare transcripts.

[handle_solexa_ggcxg_problem(pechsgp)=*on|yes|1, off|no|0*] Default is is dependent yes.

Solexa data has a pretty awful problem with in some reads when a `GGCxG` motif occurs (read more about it in the chapter on Solexa data). In short: the sequencing errors produced by this problem lead to many false positive SNP discoveries in mapping assemblies or problems in contig building in de-novo assembly.

MIRA knows about this problem and can look for it in Solexa reads during the proposed end clipping and further clip back the reads, greatly minimising the impact of this problem.

[pec_bases_per_hash(pecbph)=*integer* ≥ *10*] Default is is dependent on technology and quality in the --job switch: usually between 17 and 21 for Sanger, higher for 454 (up to 27) and highest for Solexa (31). Ion Toorent has at the moment 17, but this may change in the future to somewhat higher values.

This parameter defines the minimum number of bases at each end of a read that should be free of any sequencing errors. Note that the algorithm is based on SKIM hashing (see below), and compares hashes of all reads with each other. Therefore, using values less than 12 will lead to false negative hits.

3.5.3.8 -SKIM (-SK)

Options that control the behaviour of the initial fast all-against-all read comparison algorithm. Matches found here will be confirmed later in the alignment phase. The new SKIM3 algorithm that is in place since version 2.7.4 uses a hash based algorithm that works similarly to SSAHA (see Ning Z, Cox AJ, Mullikin JC; "SSAHA: a fast search method for large DNA databases."; Genome Res. 2001;11;1725-9).

The major differences of SKIM3 and SSAHA are:

1. the word length *n* of a hash can be up to 31 bases (in 64 bit versions of MIRA)
2. SKIM3 uses a maximum fixed amount of RAM that is independent of the word size. E.g., SSAHA would need 4 exabyte to work with word length of 30 bases ... SKIM3 just takes a couple of hundred MB.

The parameters for SKIM3:

[number_of_threads(not)=*integer* $\geq$ *1*] Number of threads used in SKIM, default is 2. A few parts of SKIM are non-threaded, so the speedup is not exactly linear, but it should be very close. E.g., with 2 processors I get a speedup of 180-195%, with 4 between 350 and 395%.

Although the main data structures are shared between the threads, there's some additional memory needed for each thread.

[also_compute_reverse_complements(acrc)=*on|yes|1, off|no|0*] Default is on. Defines whether SKIM searches for matches only in forward/forward direction or whether it also looks for forward/reverse direction.

You usually will not want to touch the default, except for very special application cases where you do not want MIRA to use reverse complement sequences at all.

[bases_per_hash(bph)=*10 < integer* $\leq$ *32*] Controls the number of consecutive bases n which are used as a word hash. The higher the value, the faster the search. The lower the value, the more weak matches are found. Values below 10 are not recommended. Defaults are dependend on "--job" switch.

[hash_save_stepping(hss)=*integer* $\geq$ *1*] Default is 1. This is a parameter controlling the stepping increment s with which hashes are generated. This allows for more or less fine grained search as matches are found with at least $n+s$ (see [-SK:bph]) equal bases. The higher the value, the faster the search. The lower the value, the more weak matches are found.

[percent_required(pr)=*integer* $\geq$ *1*] Default is dependent of the sequencing technology used and assembly quality wished. Controls the relative percentage of exact word matches in an approximate overlap that has to be reached to accept this overlap as possible match. Increasing this number will decrease the number of possible alignments that have to be checked by Smith-Waterman later on in the assembly, but it also might lead to the rejection of weaker overlaps (i.e. overlaps that contain a higher number of mismatches).

Note: most of the time it makes sense to keep this parameter in sync with [-AL:mrs].

[maxhits_perread(mhpr)=*integer* $\geq$ *1*] Default is 2000. Controls the maximum number of possible hits one read can maximally transport to the graph edge reduction phase. If more potential hits are found, only the best ones are taken.

In the pre-2.9.x series, this was an important option for tackling projects which contain *extreme* assembly conditions. It still is if you run out of memory in the graph edge reduction phase. Try then to lower it to 1000, 500 or even 100.

As the assembly increases in passes ([-AS:nop]), different combinations of possible hits will be checked, always the probably best ones first. So the accuracy of the assembly should only suffer when lowering this number too much.

[sw_check_on_backbones(swcob)=*on|yes|1, off|no|0*] Default is currently (3.4.0) yes for accurate mapping jobs. Takes effect only in mapping assemblies. Defines whether SKIM hits against a backbone (reference) sequence with less than 100% identity are double checked with Smith-Waterman to improve mapping accuracy.

You will want to set this option to yes whenever your reference sequence contains more complex or numerous repeats and your data has SNPs in those areas.

[freq_est_minnormal(fenn)=*float < 0*] During SKIM analysism, MIRA will estimate how repetitive parts of reads are. Parts which are occuring less than [-SK:fenn] times the average occurence will be tagged with a HAF2 (less than average) tag.

[freq_est_maxnormal(fexn)=*float < 0*] During SKIM analysism, MIRA will estimate how repetitive parts of reads are. Parts which are occuring more than [-SK:fenn] but less than [-SK:fexn] times the average occurence will be tagged with a HAF3 (normal) tag.

[freq_est_repeat(fer)=*float < 0*] During SKIM analysism, MIRA will estimate how repetitive parts of reads are. Parts which are occuring more than [-SK:fexn] but less than [-SK:fer] times the average occurence will be tagged with a HAF4 (above average) tag.

[freq_est_heavyrepeat(fehr)=*float < 0*] During SKIM analysism, MIRA will estimate how repetitive parts of reads are. Parts which are occuring more than [-SK:fer] but less than [-SK:fehr] times the average occurence will be tagged with a HAF5 (repeat) tag.

[freq_est_crazyrepeat(fecr)=*float < 0*] During SKIM analysism, MIRA will estimate how repetitive parts of reads are. Parts which are occuring more than [-SK:fehr] but less than [-SK:fecr] times the average occurence will be tagged with a HAF6 (heavy repeat) tag. Parts which are occuring more than [-SK:fecr] but less than [-SK:nrr] times the average occurence will be tagged with a HAF7 (crazy repeat) tag.

[mask_nasty_repeats(mnr)=*on|yes|1, off|no|0*] Default is dependent on --job type: yes for de-novo, no for mapping. Tells mira to mask during the SKIM phase subsequences of size [-SK:nph] nucleotides that appear more often than the median occurrence of subsequences would otherwise suggest. The threshold from which on subsequences are considered nasty is set by [-SK:nrr] (see below).

There's one drawback though: the smaller the reads are that you try to assemble with this option turned on, the higher the probability that your reads will not span nasty repeats completely, therefore leading to a abortion of contig building at this site.

The masked parts are tagged with "MNRr" in the reads.

This option is extremely useful for assembly of larger projects (fungi-size) with a high percentage of repeats. Or in non-normalised EST projects, to get at least something assenbled.

Although it is expected that bacteria will not really need this, leaving it turned on will probably not harm except in unusual cases like several copies of (pro-)phages integrated in a genome.

[nasty_repeat_ratio(nrr)=*integer* $\geq$ *2*] Default is depending on the [--job=...] parameters. Normally it's high (around 100) for genome assemblies, but much lower (20 or less) for EST assemblies.

Sets the ratio from which on subsequences are considered nasty and hidden from the SKIM overlapper with a *MNRr* tag. The value of 10 means: mask all k-mers of [-SK:bph] length which are occurring more than 10 times more often than the average of the project.

[repeatlevel_in_infofile(rliif)=*integer; 0, 5-8*] Default is 6. Sets the minimum level of the HAF tags from which on MIRA will report tentatively repetitive sequence in the `*_info_readrepeats.lst` file of the info directory.

A value of 0 means "switched off". The default value of , 6 means all subsequences tagged with *HAF6*, *HAF7* and *MNRr* will be logged. If you, e.g., only wanted MNRr logged, you'd use 8 as parameter value.

See also [-SK:fenn:fexn:fer:fehr:mnr:nrr] to set the different levels for the *HAF* and *MNRr* tags.

[max_megahub_ratio(mmhr)=*integer* $\geq$ *0*] Default is 0. If the number of reads identified as megahubs exceeds the allowed ratio, mira will abort.

This is a fail-safe parameter to avoid assemblies where things look fishy. In case you see this, you might want to ask for advice on the mira_talk mailing list. In short: bacteria should never have megahubs (90% of all cases reported were contamination of some sort and the 10% were due to incredibly high coverage numbers). Eukaryotes are likely to contain megahubs if filtering is [-SK:mnr] not on.

EST project however, especially from non-normalised libraries, will very probably contain megahubs. In this case, you might want to think about masking, see [-SK:mnr].

[max_hashes_in_memory(mhim)=*integer* $\geq$ *100000*] Default is 15000000. Has no influence on the quality of the assembly, only on the maximum memory size needed during the skimming. The default value is equivalent to approximately 500MB.

Note: reducing the number will increase the run time, the more drastically the bigger the reduction. On the other hand, increasing the default value chosen will not result in speed improvements that are really noticeable. In short: leave this number alone if you are not desperate to save a few MB.

[memcap_hitreduction(mchr)=*integer* $\geq$ *10*] Default is 1024, 2048 when Solexa sequences are used. Maximum memory used (in MiB) during the reduction of skim hits.

Note: has no influence on the quality of the assembly, reducing the number will increase the runtime, the more drastically the bigger the reduction as hits then must be streamed multiple times from disk.

The default is good enough for assembly of bacterial genomes or small eukaryotes (using Sanger and/or 454 sequences). As soon as assembling something bigger than 20 megabases, you should increase it to 2048 or 4096 (equivalent to 2 or 4 GiB of memory).

3.5.3.9 -ALIGN (-AL)

The align options control the behaviour of the Smith-Waterman alignment routines. Only read pairs which are confirmed here may be included into contigs. Affects both the checking of possible alignments found by SKIM as well as the phase when reads are integrated into a contig.

Every option in this section can be set individually for every sequencing technology, giving a very fine grained control on how reads are aligned for each technology.

[bandwidth_in_percent(bip)=*integer > 0 and ≤100*] Default is dependent of the sequencing technology used. The banded Smith-Waterman alignment uses this percentage number to compute the bandwidth it has to use when computing the alignment matrix. E.g., expected overlap is 150 bases, bip=10 -> the banded SW will compute a band of 15 bases to each side of the expected alignment diagonal, thus allowing up to 15 unbalanced inserts / deletes in the alignment. INCREASING AND DECREASING THIS NUMBER: *increase*: will find more non-optimal alignments, but will also increase SW runtime between linear and \Circum2. *decrease*: the other way round, might miss a few bad alignments but gaining speed.

[bandwidth_min(bmin)=*integer > 0*] Default is dependent of the sequencing technology used. Minimum bandwidth in bases to each side.

[bandwidth_max(bmax)=*integer > 0*] Default is dependent of the sequencing technology used. Maximum bandwidth in bases to each side.

[min_overlap(mo)=*integer > 0*] Default is dependent of the sequencing technology used. Minimum number of overlapping bases needed in an alignment of two sequences to be accepted.

[min_score(ms)=*integer > 0*] Default is dependent of the sequencing technology used. Describes the minimum score of an overlap to be taken into account for assembly. mira uses a default scoring scheme for SW align: each match counts 1, a match with an N counts 0, each mismatch with a non-N base -1 and each gap -2. Take a bigger score to weed out a number of chance matches, a lower score to perhaps find the single (short) alignment that might join two contigs together (at the expense of computing time and memory).

[min_relative_score(mrs)=*integer > 0 and ≤100*] Default is dependent of the sequencing technology used. Describes the min % of matching between two reads to be considered for assembly. Increasing this number will save memory, but one might loose possible alignments. I propose a maximum of 80 here. Decreasing below 55% will make memory and time consumption probably explode.

Note: most of the time it makes sense to keep this parameter in sync with [-SK:pr].

[extra_gap_penalty(egp)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used. Defines whether or not to increase penalties applied to alignments containing long gaps. Setting this to 'yes' might help in projects with frequent repeats. On the other hand, it is definitively disturbing when assembling very long reads containing multiple long indels in the called base sequence ... although this should not happen in the first place and is a sure sign for problems lying ahead.

When in doubt, set it to yes for EST projects and de-novo genome assembly, set it to no for assembly of closely related strains (assembly against a backbone).

When set to no, it is recommended to have [-CO:amgb] and [-CO:amgbemc] both set to yes.

[egp_level(egpl)=*low|0, medium|1, high|2, split_on_codongaps|10*] Default is dependent of the sequencing technology used. Has no effect if extra_gap_penalty is off. Defines an extra penalty applied to 'long' gaps. There are these are predefined levels: low - use this if you expect your base caller frequently misses 2 or more bases. medium - use this if your base caller is expected to frequently miss 1 to 2 bases. high - use this if your base caller does not frequently miss more than 1 base.

For some stages of the EST assembly process, a special value *split_on_codongaps* is used. It's even a tick harsher that the 'high' level.

Also, usage of this parameter is probably a good thing if the repeat marker of the contig is set to not mark on gap bases ([-CO:amgb] equals to no). This is generally the case for 454 data.

[egp_level(megpp)=*0 ≤ integer ≤ 100*] Default is 100. Has no effect if extra_gap_penalty is off. Defines the maximum extra penalty in percent applied to 'long' gaps.

3.5.3.10 -CONTIG (-CO)

The contig options control the behaviour of the contig objects.

[name_prefix(np)=*string*] Default is <projectname>. Contigs will have this string prepended to their names. The [-project=] quick-switch will also change this option.

[reject_on_drop_in_relscore(rodirs)=*integer > 0 and ≤100*] Default is dependent of the sequencing technology used. When adding reads to a contig, reject the reads if the drop in the quality of the consensus is > the given value in %. Lower values mean stricter checking. This value is doubled should a read be entered that has a template partner (a read pair) at the right distance.

[mark_repeats(mr)=*on|yes|1, off|no|0*] Default is yes. One of the most important switches in MIRA: if set to yes, MIRA will try to resolve misassemblies due to repeats by identifying single base stretch differences and tag those critical bases as RMB (Repeat Marker Base, weak or strong). This switch is also needed when MIRA is run in EST mode to identify possible inter-, intra- and intra-and-interorganism SNPs.

[only_in_result(mroir)=*on|yes|1, off|no|0*] Default is no. Only takes effect when [-CO:mr] (see above) is set to yes. If set to yes, MIRA will not use the repeat resolving algorithm during build time (and therefore will not be able to take advantage of this), but only before saving results to disk.

This switch is useful in some (rare) cases of mapping assembly.

[assume_snp_instead_repeat(asir)=*on|yes|1, off|no|0*] Default is no. Only takes effect when [-CO:mr] (see above) is set to yes, effect is also dependent on the fact whether strain data (see - [-SB:lsd]) is present or not. Usually, mira will mark bases that differentiate between repeats when a conflict occurs between reads that belong to one strain. If the conflict occurs between reads belonging to different strains, they are marked as SNP. However, if this switch is set to yes, conflict within a strain are also marked as SNP.

This switch is mainly used in assemblies of ESTs, it should not be set for genomic assembly.

[min_reads_per_group(mrpg)=*integer ≥ 2*] Default is dependent of the sequencing technology used. Only takes effect when [-CO:mr] (see above) is set to yes. This defines the minimum number of reads in a group that are needed for the RMB (Repeat Marker Bases) or SNP detection routines to be triggered. A group is defined by the reads carrying the same nucleotide for a given position, i.e., an assembly with mrpg=2 will need at least two times two reads with the same nucleotide (having at least a quality as defined in [-CO:mgqrt]) to be recognised as repeat marker or a SNP. Setting this to a low number increases sensitivity, but might produce a few false positives, resulting in reads being thrown out of contigs because of falsely identified possible repeat markers (or wrongly recognised as SNP).

[min_neighbour_qual (mnq)=*integer ≥ 10*] Default is dependent of the sequencing technology used. Takes only effect when [-CO:mr] is set to yes. This defines the minimum quality of neighbouring bases that a base must have for being taken into consideration during the decision whether column base mismatches are relevant or not.

[min_groupqual_for_rmb_tagging(mgqrt)=*integer ≥ 25*] Default is dependent of the sequencing technology used. Takes only effect when [-CO:mr] is set to yes. This defines the minimum quality of a group of bases to be taken into account as potential repeat marker. The lower the number, the more sensitive you get, but lowering below 25 is not recommended as a lot of wrongly called bases can have a quality approaching this value and you'd end up with a lot of false positives. The higher the overall coverage of your project, the better, and the higher you can set this number. A value of 35 will probably remove most false positives, a value of 40 will probably never show false positives ... but will generate a sizable number of false negatives.

[endread_mark_exclusion_area(emea)=*integer ≥ 0*] Default is dependent of the sequencing technology used. Takes only effect when [-CO:mr] is set to yes. Using the end of sequences of Sanger type shotgun sequencing is always a bit risky, as wrongly called bases tend to crowd there or some sequencing vector relics hang around. It is even more risky to use these stretches for detecting possible repeats, so one can define an exclusion area where the bases are not used when determining whether a mismatch is due to repeats or not.

[emea_set1_on_clipping_pec(emeas1clpec)=*on|yes|1, off|no|0*] Default is yes. When [-CL:pec] is set, the end-read exclusion area can be considerably reduced. Setting this parameter will automatically do this.

Note Although the parameter is named "set to 1", it may be that the exclusion area is actually a bit larger (2 to 4), depending on what users will report back as "best" option.

[also_mark_gap_bases(amgb)=*on|yes|1, off|no|0*] Default is dependent of the sequencing technology used. Determines whether columns containing gap bases (indels) are also tagged.

Note: it is strongly recommended to not set this to 'yes' for 454 type data.

[also_mark_gap_bases_even_multicolumn(amgbemc)=*on|yes|1, off|no|0*] Default is yes. Takes effect only when [-CO:amgb] is set to yes. Determines whether multiple columns containing gap bases (indels) are also tagged.

[also_mark_gap_bases_need_both_strands(amgbnbs)=*on|yes|1, off|no|0*] Default is yes. Takes effect only when [-CO:amgb] is set to yes. Determines whether both for tagging columns containing gap bases, both strands.need to have a gap. Setting this to no is not recommended except when working in desperately low coverage situations.

[force_nonIUPACconsensus_perseqtype(fnicpst)=*on|yes|1, off|no|0*] Default is no for all sequencing types. If set to yes, mira will be forced to make a choice for a consensus base (A,C,G,T or gap) even in unclear cases where it would normally put a IUPAC base. All other things being equal (like quality of the possible consensus base and other things), mira will choose a base by either looking for a majority vote or, if that also is not clear, by preferring gaps over T over G over C over finally A.

mira makes a considerable effort to deduce the right base at each position of an assembly. Only when cases begin to be borderline it will use a IUPAC code to make you aware of potential problems. It is **suggested** to leave this option to no as IUPAC bases in the consensus are a sign that - if you need 100% reliability - you really should have a look at this particular place to resolve potential problems. You might want to set this parameter to yes in the following cases: 1) when your tools that use assembly result cannot handle IUPAC bases and you don't care about being absolutely perfect in your data (by looking over them manually). 2) when you assemble data without any quality values (which you should not do anyway), then this method will allow you to get a result without IUPAC bases that is "good enough" with respect to the fact that you did not have quality values.

Important note: in case you are working with a hybrid assembly, mira will still use IUPAC bases at places where reads from different sequencing types contradict each other. In fact, when not forcing non-IUPAC bases for hybrid assemblies, the overall consensus will be better and probably have less IUPAC bases as mira can make a better use of available information.

[merge_short_reads(msr)=*on|yes|1, off|no|0*] Default is yes for all Solexas when in a mapping assembly, else it's no. Can only be used in mapping assemblies. If set to yes, MIRA will merge all perfectly mapping Solexa reads into longer reads (Coverage Equivalent Reads, CERs) while keeping quality and coverage information intact.

This feature hugely reduces the number of Solexa reads and makes assembly results with Solexa data small enough to be handled by current finishing programs (gap4, consed, others) on normal workstations.

[msr_keepcontigendsunmerged(msrkceu)=*-1, integer > 0*] Default is -1 for all Solexas when in a mapping assembly. Takes only effect in mapping assemblies if [-CO:msr=yes] and for reads which have a paired-end / mate-pair partner actively used in the assembly.

If set to a value > 0, MIRA will not merge paired-end / mate-pair reads if they map within the given distance of a contig end of the original reference sequence (backbone). Instead of a fixed value, one can also use -1. MIRA will then automatically not merge reads if the distance from the contig end is within the maximum size of the template insert size of the sequencing library for that read (either given via [-GE:tismax] or via XML TRACEINFO for the given read).

This feature allows to use the data reduction from [-CO:msr] while enabling the result of such a mapping to be useful in subsequent scaffolding programs to order contigs.

3.5.3.11 -EDIT (-ED)

General options for controlling the integrated automatic editor. The editors generally make a good job cleaning up alignments from typical sequencing errors like (like base overcalls etc.). However, they may prove tricky in certain situations:

- in EST assemblies, they may edit rare transcripts toward almost identical, more abundant transcripts. Usage must be carefully weighed.

- the editors will not only change bases, but also sometimes delete or insert non-gap bases as needed to improve an alignment when facts (trace signals or other) show that this is what should have been the sequence. However, this can make post processing of assembly results pretty difficult with some formats like ACE, where the format itself contains no way to specify certain edits like deletion. There's nothing one can do about it and the only way to get around this problem is to use file formats with more complete specifications like CAF, MAF (and BAF once supported by MIRA).

The following edit parameters are supported:

[automatic_contig_editing(ace)=*on|yes|1, off|no|0*] Default is no. Once contigs have been build, mira can call a built-in versions of the automatic contig editors. For Sanger reads this is EdIt, for 454 reads it is a specially crafted editor that knows about deficiencies of the 454 technology (homopolymers).

EdIt will try to resolve discrepancies in the contig by performing trace analysis and correct even hard to resolve errors. This option is always useful, but especially in conjunction with [-AS:nop] and [-DP:ure] (see above).

Notice 1: the current development version has a memory leak in the editor, therefore the option is not automatically turned on.

Notice 2: it is strongly suggested to turn this option on for 454 data as this greatly improves the quality.

[strict_editing_mode(sem)=*on|yes|1, off|no|0*] Default is yes. Only for Sanger data. If set to yes, the automatic editor will not take error hypotheses with a low probability into account, even if all the requirements to make an edit are fulfilled.

[confirmation_threshold(ct)=*integer, 0 < x ≤ 100*] Default is 50. Only for Sanger data. The higher this value, the more strict the automatic editor will apply its internal rule set. Going below 40 is not recommended.

3.5.3.12 -MISC (-MI)

Options which would not fit elsewhere.

[stop_on_nfs(sonfs)=*on|yes|1, off|no|0*] Default is yes. MIRA will check whether the tmp directory is running on an NFS mount. If it is and [-MI:sonfs] is active, MIRA will stop with a warning message.

Warning
You should never ever at all run MIRA on a NFS mounted directory ... or face the the fact that the assembly process may very well take 5 to 10 times longer (or more) than normal. You have been warned.
The reason for the slowdown is the same as why one should never run a BLAST search on a big database being located on a NFS volume: access via network is terribly slow when compared to local disks, at least if you have not invested a lot of money into specialised solutions.

[large_contig_size(lcs)=*integer < 0*] Default is 500. This parameter has absolutely no influence whatsoever on the assembly process of MIRA. But is used in the reporting within the `*_assembly_info.txt` file after the assembly where MIRA reports statistics on *large* and *all* contigs. [-MI:lcs] is the threshold value for categorising contigs.

[large_contig_size_for_stats(lcs4s)=*integer < 0*] Default is 5000 for [--job=genome] and 1000 for [--job=est].

This parameter is used for internal statistics calculations and has a subtle influence when being in a [--job=genome] assembly mode.

MIRA uses coverage information of an assembly project to find out about potentially repetitive areas in reads (and thus, a genome). To calculate statistics which are reflecting the approximate truth, the value of [-MI:lcs4s] is used as a cutoff threshold: contigs smaller than this value do not contribute to the calculation of average coverage while contigs larger or equal to this value do. Having this cutoff discards small contigs which tend to muddy the picture of average coverage of a project.

If in doubt, don't touch this parameter.

3.5.3.13 -DIRECTORY (-DIR, -DI)

General options for controlling where to find or where to write data.

[cwd=*<directoryname>*] Default is an empty string. When encountered during parameter parsing, MIRA will change the working directory immediately to the directory given and read and write files there.

Therefore, a call like `mira -DI:cwd=/somedir --params=myparameters.txt` will be enough to let MIRA change to the directory `/somedir` and then read further parameters from a text file `myparamaters.txt` (which should be present there) and at the same time have all the input and output of the assembly occuring in firectory `/somedir`.

[tmp_redirected_to(trt)=*<directoryname>*] Default is an empty string. When set to a non-empty string, MIRA will create the tmp directory at the given location instead of using the current working directory.

This option is particularly useful for systems which have solid state disks (SSDs) and some very fast disk subsystems which can be used for temporary files. Or in projects where the input and output files reside on a NFS mounted directory (current working dir), to put the tmp directory somewhere outside the NFS (see also: Things you should not do).

In both cases above, and for larger projects, MIRA then runs a lot faster.

[gap4da=*<directoryname>*] Default is gap4da. Defines the extension of the directory where mira will write the result of an assembly ready to import into the Staden package (GAP4) in Direct Assembly format. The name of the directory will then be `<projectname>_.<extension>`

[exp=*<directoryname>*] Default is .. Defines the directory where mira should search for experiment files (EXP).

[scf=*<directoryname>*] Default is .. Defines the directory where mira should search for SCF files.

3.5.3.14 -FILENAME (-FN)

The file options allows you to define your own input and output files.

[fastain(fai)=*string*] Default is <projectname>_in.<seqtype>.fasta. Defines the fasta file to load sequences of a project from.

[fastaqualin(fqui)=*string*] Default is <projectname>_in.<seqtype>.fasta.qual. Defines the file containing base qualities. Although the order of reads in the quality file does not need to be the same as in the fasta or fofn (although it saves a bit of time if they are).

[fastqin(fqi)=*string*] Default is <projectname>_in.<seqtype>.fastq. Defines the fastq file to load sequences of a project from.

[cafin(ci)=*string*] Default is <projectname>_in.<seqtype>.caf. Defines the file to load a CAF project from. Filename must end with '.caf'.

[fofnexpin(fei)=*string*] Default is <projectname>_in.fofn. Defines the file of filenames where the names of the EXP files of a project are located.

[fofnphdin(fpi)=*string*] Default is <projectname>_in.fofn. Defines the file of filenames where the names of the PHD files of a project are located. Note: this is currently not available.

[phdin(pi)=*string*] Default is <projectname>_in.phd. Defines the file of where all the sequences of a project are in PHD format.

[straindatain(sdi)=*string*] Default is <projectname>_straindata_in.txt. Defines the file to load straindata from..

[xmltraceinfoin(xtii)=*string*] Default is <projectname>_xmltraceinfo_in.<seqtype>.xml. Defines the file to load a trace info file in XML format from. This can be used both when merging XML data to loaded files or when loading a project from an XML trace info file.

[ssahavectorscreenin(ssvsi)=*string*] Default is <projectname>_ssaha2vectorscreen_in.txt. Defines the file to load a the info about possible vector sequence stretches.

[smaltvectorscreenin(stvsi)=*string*] Default is <projectname>_smaltvectorscreen_in.txt. Defines the file to load a the info about possible vector sequence stretches.

[backbonein(bbin)=*string*] Default is <projectname>_in.<seqtype>.<filetype>. Defines the file to load a backbone from. Note that you still must define the file type with [-SB:bft].

3.5.3.15 -OUTPUT (-OUT)

Options for controlling which results to write to which type of files. Additionally, a few options allow output customisation of textual alignments (in text and HTML files).

There are 3 types of results: result, temporary results and extra temporary results. One probably needs only the results. Temporary and extra temporary results are written while building different stages of a contig and are given as convenience for trying to find out why mira set some RMBs or disassembled some contigs.

Output can be generated in these formats: CAF, Gap4 Directed Assembly, FASTA, ACE, TCS, WIG, HTML and simple text.

Naming conventions of the files follow the rules described in section **Input / Output**, subsection **Filenames**.

[savesimplesingletsinproject(sssip)=*on|yes|1,off|no|0*] Default is no. Controls whether 'unimportant' singlets are written to the result files.

> **Note** Note that a value larger 1 of the [-AS:mrpc] parameter will disable the function of this parameter.

[savetaggedsingletsinproject(stsip)=*on|yes|1,off|no|0*] Default is yes. Controls whether singlets which have certain tags (see below) are written to the result files, even if [-OUT:sssip] (see above) is set.

If one of the (SRMr, CRMr, WRMr, SROr, SAOr, SIOr) tags appears in a singlet, MIRA will see that the singlets had been part of a larger alignment in earlier passes and even was part of a potentially 'important' decision. To give the possibility to human finishers to trace back the decision, these singlets can be written to result files.

> **Note** Note that a value larger 1 of the [-AS:mrpc] parameter will disable the function of this parameter.

[remove_rollover_tmps(rrot)=*on|yes|1, off|no|0*] Default is yes. Removes log and temporary files once they should not be needed anymore during the assembly process.

[remove_tmp_directory(rld)=*on|yes|1, off|no|0*] Default is no. Removes the complete tmp directory at the end of the assembly process. Some logs and temporary files contain useful information that you may want to analyse though, therefor the default of MIRA is not to delete it.

[output_result_caf(orc)=*on|yes|1, off|no|0*] Default is yes.

[output_result_maf(orm)=*on|yes|1, off|no|0*] Default is yes.

[output_result_gap4da(org)=*on|yes|1, off|no|0*] Default is yes for projects only with Sanger reads, 'no' as soon as there are 454, Solexa or SOLiD reads involved.

> **Note** MIRA will automatically switch to no (and cannot be forced to 'yes') when 454 or Solexa reads are present in the project as this ensure that the file system does not get flooded with millions of files.

[output_result_fasta(orf)=*on|yes|1, off|no|0*] Default is yes.

[output_result_ace(ora)=*on|yes|1, off|no|0*] Default is yes.

> **Note**
> The ACE output of MIRA is conforming to the file specification given in the consed documentation. However, due to a bug in consed, consed cannot correctly load tags set by MIRA.
> There is a workaround: the MIRA distribution comes with a small Tcl script **fixACE4consed.tcl** which implements a workaround to allow consed loading the ACE generated by MIRA. Use the script like this:
>
> ```
> $ fixACE4consed.tcl infile.ace >outfile.ace
> ```
>
> and then load the resulting outfile into consed.

[output_result_txt(ort)=*on|yes|1, off|no|0*] Default is no.

[output_result_tcs(ors)=*on|yes|1, off|no|0*] Default is yes.

[output_result_html(orh)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_caf(otc)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_maf(otm)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_gap4da(otg)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_fasta(otf)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_ace(ota)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_txt(ott)=*on|yes|1, off|no|0*] Default is no.

[output_result_tcs(ots)=*on|yes|1, off|no|0*] Default is no.

[output_tmpresult_html(oth)=*on|yes|1, off|no|0*] Default is no.

[output_exttmpresult_caf(oetc)=*on|yes|1, off|no|0*] Default is no.

[output_exttmpresult_gap4da(oetg)=*on|yes|1, off|no|0*] Default is no.

[output_exttmpresult_fasta(oetf)=*on|yes|1, off|no|0*] Default is no.

[output_exttmpresult_ace(oeta)=*on|yes|1, off|no|0*] Default is no.

[output_exttmpresult_txt(oett)=*on|yes|1, off|no|0*] Default is no.

[output_exttmpresult_html(oeth)=*on|yes|1, off|no|0*] Default is no.

[text_chars_per_line(tcpl)=*integer > 0*] Default is 60. When producing an output in text format ([-OUT:ort|ott|oett]), this parameter defines how many bases each line of an alignment should contain.

[html_chars_per_line(tcpl)=*integer > 0*] Default is 60. When producing an output in HTML format, ([-OUT:orh|oth|oeth]), this parameter defines how many bases each line of an alignment should contain.

[text_endgap_fillchar(tegfc)=*<single character>*] Default is _ (a blank). When producing an output in text format ([-OUT:ort|ott|oett]), endgaps are filled up with this character.

[html_endgap_fillchar(hegfc)=*<single character>*] Default is _ (a blank). When producing an output in HTML format ([-OUT:orh|oth|oeth]), end-gaps are filled up with this character.

3.6 Input / Output

3.6.1 Directories

Since version 3.0.0, mira now puts all files and directories it generates into one sub-directory which is named `projectname_assembly`. This directory contains up to four sub-directories:

- `projectname_d_results`: this directory contains all the output files of the assembly in different formats.
- `projectname_d_info`: this directory contains information files of the final assembly. They provide statistics as well as, e.g., information (easily parseable by scripts) on which read is found in which contig etc.
- `projectname_d_tmp`: this directory contains tmp files and temporary assembly files. It can be safely removed after an assembly as there may be easily a few GB of data in there that are not normally not needed anymore.

  In case of problems: please do not delete. I will get in touch with you for additional information that might possibly be present in the tmp directory.
- `projectname_d_chkpt`: this directory contains checkpoint files needed to resume assemblies that crashed or were stopped.

Note The checkpointing functionality has not been completely implemented yet and currently cannot be used.

3.6.2 Filenames

3.6.2.1 Input

The input files must be placed (or linked to) in the directory from which mira is called.

projectname_in.fofn File of filenames containing the names of the experiment or phd files to assemble when the [-LR:ft=FOFNEXP] option is used. One filename per line, blank lines accepted, lines starting with a hash (#) are treated as comment lines, nothing else. Use [-FN:fofnin] to change the default name.

projectname_in.phd File containing the sequences (and their qualities) to assemble in PHD format.

projectname_in.fasta File containing sequences and ...

projectname_in.fasta.qual ... file containing quality values of sequences for the assembly in FASTA format.

projectname_in.fastq FASTQ file containing sequences and qualities. MIRA automatically recognises Sanger FASTQ format (base quality offset = 33) and newer Illumina FASTQ format (base quality offset = 64). Old Illumina FASTQ format with negative base qualities (base offset < 64) is not supported anymore).

projectname_in.caf File containing the sequences (and their qualities) to assemble in CAF format. This format also may contain the result of an assembly (the contig consensus sequences).

3.6.2.2 Output

These result output files and sub-directories are placed in in the *projectname*_results directory after a run of mira.

projectname_out.<type> Assembled project written in type = (*gap4da* / *caf* / *ace* / *fasta* / *html* / *tcs* / *wig* / *text*) format by mira, final result.

Type *gap4da* is a directory containing experiment files and a file of filenames (called 'fofn'), all other types are files. *gap4da*, *caf*, *ace* contain the complete assembly information suitable for import into different post-processing tools (gap4, consed and others). *html* and *text* contain visual representations of the assembly suited for viewing in browsers or as simple text file. *tcs* is a summary of a contig suited for "quick" analyses from command-line tools or even visual inspection. *wig* is a file containing coverage information (useful for mapping assemblies) which can be loaded and shown by different genome browsers (IGB, GMOD, USCS and probably many more.

fasta contains the contig consensus sequences (and .fasta.qual the consensus qualities). Please note that they come in two flavours: padded and unpadded. The padded versions may contains stars (*) denoting gap base positions where there was some minor evidence for additional bases, but not strong enough to be considered as a real base. Unpadded versions have these gaps removed. Padded versions have an additional postfix *.padded*, while unpadded versions do not have a special postfix.

3.6.2.3 Assembly statistics and information files

These information files are placed in in the *projectname*_info directory after a run of mira.

projectname_info_assembly.txt This file contains basic information about the assembly. MIRA will split the information in two parts: information about *large* contigs and information about all contigs.

For more information on how to interpret this file, please consult the chapter on "Results" of the MIRA documentation manual.

Note In contrast to other information files, this file appears always in the "info" directory, even when just intermediate results are reported.

---

projectname_info_contigreadlist.txt This file contains information which reads have been assembled into which contigs (or singlets).

projectname_info_contigstats.txt This file contains statistics about the contigs themselves, their length, average consensus quality, number of reads, maximum and average coverage, average read length, number of A, C, G, T, N, X and gaps in consensus.

projectname_info_consensustaglist.txt This file contains information about the tags (and their position) that are present in the consensus of a contig.

projectname_info_readrepeats.lst Tab delimited file with three columns: read name, repeat level tag, sequence.

This file permits a quick analysis of the repetitiveness of different parts of reads in a project. See [-SK:rliif] to control from which repetitive level on subsequences of reads are written to this file,

---

Note Reads can have more than one entry in this file. E.g., with standard settings (`-SK:rliif=6`) if the start of a read is covered by MNRr, followed by a HAF3 region and finally the read ends with HAF6, then there will be two lines in the file: one for the subsequence covered by MNRr, one for HAF6.

---

projectname_info_readstooshort A list containing the names of those reads that have been sorted out of the assembly before any processing started only due to the fact that they were too short.

projectname_info_readtaglist.txt This file contains information about the tags and their position that are present in each read. The read positions are given relative to the forward direction of the sequence (i.e. as it was entered into the the assembly).

projectname_error_reads_invalid A list of sequences that have been found to be invalid due to various reasons (given in the output of the assembler).

3.6.3 File formats

MIRA can write almost all of the following formats and can read most of them.

EXP Standard experiment files used in genome sequencing. Correct EXP files are expected. Especially the ID record (containing the id of the reading) and the LN record (containing the name of the corresponding trace file) should be correctly set. See http://www.sourceforge.net/projects/staden/ for links to online format description.

SCF The Staden trace file format that has established itself as compact standard replacement for the much bigger ABI files. See http://www.sourceforge.net/projects/staden/ for links to online format description.

The SCF files should be V2-8bit, V2-16bit, V3-8bit or V3-16bit and can be packed with compress or gzip.

CAF Common Assembly Format (CAF) developed by the Sanger Centre. http://www.sanger.ac.uk/resources/software/caf.html provides a description of the format and some software documentation as well as the source for compiling caf2gap and gap2caf (thanks to Rob Davies for this).

ACE The assembly file format used mainly by phrap and consed. Support for .ace output is currently only in test status in mira as documentation on that format is ... sparse and I currently don' have access to consed to verify my assumptions.

Using consed, you will need to load projects with -nophd to view them. Tags /in reads and consensus) are fully supported. The only hitch: consed has a bug which prevents it to read consensus tags which are located throughout the whole file (as MIRA writes per default). The solution to that is easy: filter the CAF file through the fixACE4consed.tcl script which is provided in the MIRA distributions, then all should be well.

If you don't have consed, you might want to try clview (http://www.tigr.org/tdb/tgi/software/) from TIGR to look at .ace files.

MAF MIRA Assembly Format (MAF). A faster and more compact form than EXP, CAF or ACE. See documentation in separate file.

HTML Hypertext Markup Language. Projects written in HTML format can be viewed directly with any table capable browser. Display is even better if the browser knows style sheets (CSS).

FASTA A simple format for sequence data, see http://www.ncbi.nlm.nih.gov/BLAST/fasta.html. An often used extension of that format is used to also store quality values in a similar fashion, these files have a .fasta.qual ending.

Mira writes two kinds of FASTA files for results: *padded* and *unpadded*. The difference is that the padded version still contains the gap (pad) character (an asterisk) at positions in the consensus where some of the reads apparently had some more bases than others but where the consensus routines decided that to treat them as artifacts. The *unpadded* version has the gaps removed.

PHD This file type originates from the phred base caller and contains basically -- along with some other status information -- the base sequence, the base quality values and the peak indices, but not the sequence traces itself.

GBF, GBK GenBank file format as used at the NCBI to describe sequences. mira is able to read this format for using sequences as backbones in an assembly. Features of the GenBank format are also transferred automatically to Staden compatible tags.

traceinfo.XML XML based file with information relating to traces. Used at the NCBI and ENSEMBL trace archive to store additional information (like clippings, insert sizes etc.) for projects. See further down for for a description of the fields used and http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?cmd=show&f=rfc&m=main&s=rfc for a full description of all fields.

TCS Transpose Contig Summary. A text file as written by mira which gives a summary of a contig in tabular fashion, one line per base. Nicely suited for "quick" analyses from command line tools, scripts, or even visual inspection in file viewers or spreadsheet programs.

In the current file version (TCS 1.0), each column is separated by at least one space from the next. Vertical bars are inserted as visual delimiter to help inspection by eye. The following columns are written into the file:

1. contig name (width 20)
2. padded position in contigs (width 3)
3. unpadded position in contigs (width 3)
4. separator (a vertical bar)
5. called consensus base
6. quality of called consensus base (0-100), but MIRA itself caps at 90.
7. separator (a vertical bar)
8. total coverage in number of reads. This number can be higher than the sum of the next five columns if Ns or IUPAC bases are present in the sequence of reads.
9. coverage of reads having an "A"
10. coverage of reads having an "C"
11. coverage of reads having an "G"
12. coverage of reads having an "T"
13. coverage of reads having an "*" (a gap)
14. separator (a vertical bar)
15. quality of "A" or "--" if none
16. quality of "C" or "--" if none
17. quality of "G" or "--" if none
18. quality of "T" or "--" if none
19. quality of "*" (gap) or "--" if none
20. separator (a vertical bar)
21. Status. This field sums up the evaluation of MIRA whether you should have a look at this base or not. The content can be one of the following:
    - everything OK: a colon (:)
    - unclear base calling (IUPAC base): a "!M"
    - potentially problematic base calling involving a gap or low quality: a "!m"
    - consensus tag(s) of MIRA that hint to problems: a "!$". Currently, the following tags will lead to this marker: SRMc, WRMc, DGPc, UNSc, IUPc.
22. list of a consensus tags at that position, tags are delimited by a space. E.g.: "DGPc H454"

3.6.4 STDOUT/STDERR

The actual stage of the assembly is written to STDOUT, giving status messages on what mira is actually doing. Dumping to STDERR is almost not used anymore by MIRA, remnants will disappear over time.

Some debugging information might also be written to STDOUT if mira generates error messages.

On errors, MIRA will dump these also to STDOUT. Basically, three error classes exist:

1. WARNING: Messages in this error class do not stop the assembly but are meant as an information to the user. In some rare cases these errors are due to (an always possible) error in the I/O routines of mira, but nowadays they are mostly due to unexpected (read: wrong) input data and can be traced back to errors in the preprocessing stages. If these errors arise, you definitively **DO** want to check how and why these errors came into those files in the first place.

   Frequent cause for warnings include missing SCF files, SCF files containing known quirks, EXP files containing known quirks etc.

2. FATAL: Messages in this error class actually stop the assembly. These are mostly due to missing files that mira needs or to very garbled (wrong) input data.

   Frequent causes include naming an experiment file in the 'file of filenames' that could not be found on the disk, same experiment file twice in the project, suspected errors in the EXP files, etc.

3. INTERNAL: These are true programming errors that were caught by internal checks. Should this happen, please mail the output of STDOUT and STDERR to the author.

3.6.5 XML TRACEINFO

MIRA extracts the following data from the TRACEINFO files:

- trace_name (required)
- trace_file (recommended)
- trace_type_code (recommended)
- trace_end (recommended)
- clip_quality_left (recommended)
- clip_quality_right (recommended)
- clip_vector_left (recommended)
- clip_vector_right (recommended)
- strain (recommended)
- template_id (recommended for paired end)
- insert_size (recommended for paired end)
- insert_stdev (recommended for paired end)
- machine_type (optional)
- program_id (optional)

Other data types are also read, but the info is not used.

Here's the example for a TRACEINFO file with ancillary info:

```
<?xml version="1.0"?>
<trace_volume>
<trace>
  <trace_name>GCJAA15TF</trace_name>
  <program_id>PHRED (0.990722.G) AND TTUNER (1.1)</program_id>
  <template_id>GCJAA15</template_id>
  <trace_direction>FORWARD</trace_direction>
  <trace_end>F</trace_end>
  <clip_quality_left>3</clip_quality_left>
  <clip_quality_right>622</clip_quality_right>
  <clip_vector_left>1</clip_vector_left>
  <clip_vector_right>944</clip_vector_right>
  <insert_stdev>600</insert_stdev>
  <insert_size>2000</insert_size>
</trace>
<trace>
  ...
</trace>
...
</trace_volume>
```

See http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?cmd=show&f=rfc&m=main&s=rfc for a full description of all fields and more info on the TRACEINFO XML format.

3.6.6 Contig naming

MIRA names contigs the following way: <*projectname*>_<*contigtype*><*number*>. While <*projectname*> is dictated by the [--project=] parameter and <*number*> should be clear, the <*contigtype*> might need additional explaining. There are currently three contig types existing:

1. _c: these are "normal" contigs
2. _rep_c: these are contigs containing only repetitive areas. These contigs had *_lrc* as type in previous version of MIRA, this was changed to the *_rep_c* to make things clearer.
3. _s: these are singlet-contigs. Technically: "contigs" with a single read.

Basically, for genome assemblies MIRA starts to build contigs in areas which seem "rock solid", i.e., not a repetitive region (main decision point) and nice coverage of good reads. Contigs which started like this get a *_c* name. If during the assembly MIRA reaches a point where it cannot start building a contig in a non-repetitive region, it will name the contig *_rep_c* instead of *_c*.

Note Although the distinction between *_c* and *_rep_c* makes sense only for genome assemblies, EST assemblies also use it (for no better reason than me not having an alternative or better naming scheme there).

Note Depending on the settings of [-AS:mrpc], your project may or may not contain *_s* singlet-contigs. Also note that reads landing in the debris file will not get assigned to singlet-contigs and hence not get *_s* names.

3.6.7 Recovering strain specific consensus as FASTA

In case you used strain information in an assembly, you can recover the consensus for just any given strain by using **convert_project** and convert from a full assembly format (e.g. MAF or CAF) which also carries strain information to FASTA. MIRA will automatically detect the strain information and create one FASTA file per strain encountered.

Note To be able to distinguish between consensus bases with a 'N' call and areas of a strain which were not covered at all by any read of that strain, MIRA introduces the '@' sign as additional "base". That is, if you see a '@' in the consensus of a given strain, this may be either due to too low coverage -- and therefore a hole -- or to a genuine deletion in your strain.

3.7 Tags used in the assembly by MIRA and EdIt

MIRA uses and sets a couple of tags during the assembly process. That is, if information is known before the assembly, it can be stored in tags (in the EXP and CAF formats) and will be used in the assembly.

3.7.1 Tags read (and used)

This section lists "foreign" tags, i.e., tags that whose definition was made by other software packages than MIRA.

- ALUS, REPT: Sequence stretches tagged as ALUS (ALU Sequence) or REPT (general repetitive sequence) will be handled with extreme care during the assembly process. The allowed error rate after automatic contig editing within these stretches is normally far below the general allowed error rate, leading to much higher stringency during the assembly process and subsequently to a better repeat resolving in many cases.
- FpAS: GenBank feature for a poly-A signal. Used in EST, cDNA or transcript assembly. Either read in the input files or set when using [-CL:cpat]. This allows to keep the poly-A signal in the reads during assembly without them interfering as massive repeats or as mismatches.
- FCDS, Fgen: GenBank features as described in GBF/GBK files or set in the Staden package are used to make some SNP impact analysis on genes.
- other. All other tags in reads will be read and passed through the assembly without being changed and they currently do not influence the assembly process.

3.7.2 Tags set (and used)

This section lists tags which MIRA sets (and reads of course), but that other software packages might not know about.

- UNSr, UNSc: **UNS**ure in **R**ead respectively **C**ontig. These tags denote positions in an assembly with conflicts that could not be resolved automatically by mira. These positions should be looked at during the finishing process.

  For assemblies using good sequences and enough coverage, something 0.01% of the consensus positions have such a tag. (e.g. ~300 UNSc tags for a genome of 3 megabases).
- SRMr, WRMc: **S**trong **R**epeat **M**arker and **W**eak **R**epeat **M**arker. These tags are set in two flavours: as SRM**r** and WRM**r** when set in reads, and as SRM**c** and WRM**c** when set in the consensus. These tags are used on an individual per base basis for each read. They denote bases that have been identified as crucial for resolving repeats, often denoting a single SNP within several hundreds or thousands of bases. While a SRM is quite certain, the WRM really is either weak (there wasn't enough comforting information in the vicinity to be really sure) or involves gap columns (which is always a bit tricky).

  mira will automatically set these tags when it encounters repeats and will tag exactly those bases that can be used to discern the differences.

  Seeing such a tag in the consensus means that mira was not able to finish the disentanglement of that special repeat stretch or that it found a new one in one of the last passes without having the opportunity to resolve the problem.
- DGPc: **D**ubious **G**ap **P**osition in **C**onsensus. Set whenever the gap to base ratio in a column of 454 reads is between 40% and 60%.

- SAO, SRO, SIO: **S**NP intr**A** **O**rganism, **S**NP **R** **O**rganism, **S**NP **I**ntra and inter **O**rganism. As for SRM and WRM, these tags have a **r** appended when set in reads and a **c** appended when set in the consensus. These tags denote SNP positions.

  mira will automatically set these tags when it encounters SNPs and will tag exactly those bases that can be used to discern the differences. They denote SNPs as they occur within an organism (SAO), between two or more organisms (SRO) or within and between organisms (SIO).

  Seeing such a tag in the consensus means that mira set this as a valid SNP in the assembly pass. Seeing such tags only in reads (but not in the consensus) shows that in a previous pass, mira thought these bases to be SNPs but that in later passes, this SNP does not appear anymore (perhaps due to resolved misassemblies).

- STMS: (only hybrid assemblies). The **S**equencing **T**ype **M**ismatch **S**olved is tagged to positions in the assembly where the consensus of different sequencing technologies (Sanger, 454, Ion Torrent, Solexa, PacBio, SOLiD) reads differ, but mira thinks it found out the correct solution. Often this is due to low coverage of one of the types and an additional base calling error.

  Sometimes this depicts real differences where possible explanation might include: slightly different bugs were sequenced or a mutation occurred during library preparation.

- STMU: (only hybrid assemblies). The **S**equencing **T**ype **M**ismatch **U**nresolved is tagged to positions in the assembly where the consensus of different sequencing technologies (Sanger, 454, Ion Torrent, Solexa, SOLiD) reads differ, but mira could not find a good resolution. Often this is due to low coverage of one of the types and an additional base calling error.

  Sometimes this depicts real differences where possible explanation might include: slightly different bugs were sequenced or a mutation occurred during library preparation.

- MCVc: The **M**issing **C**o{V}erage in **C**onsensus. Set in assemblies with more than one strain. If a strain has no coverage at a certain position, the consensus gets tagged with this tag (and the name of the strain which misses this position is put in the comment). Additionally, the sequence in the result files for this strain will have an @ character.

- MNRr: (only with [-SK:mnr] active). The **M**asked **N**asty **R**epeat tags are set over those parts of a read that have been detected as being many more times present than the average sub-sequence. mira will hide these parts during the initial all-against-all overlap finding routine (SKIM3) but will otherwise happily use these sequences for consensus generation during contig building.

- FpAS: See "Tags read (and used)" above.

- ED_C, ED_I, ED_D: EDit Change, EDit Insertion, EDit Deletion. These tags are set by the integrated automatic editor EdIt and show which edit actions have been performed.

- HAF2, HAF3, HAF4, HAF5, HAF6, HAF7. These are **HA**sh **F**requency tags which show the status of read parts in comparison to the whole project. Only set if [-AS:ard] is active (default for genome assemblies).

  More info on how to use the information conveyed by HAF tags in the section dealing with repeats and HAF tags in finishing programs further down in this manual.

  HAF2 coverage below average (standard setting at < 0.5 times average)

  HAF3 coverage is at average (standard setting at $\geq$ 0.5 times average and $\leq$ 1.5 times average)

  HAF4 coverage above average (standard setting at > 1.5 times average and < 2 times average)

  HAF5 probably repeat (standard setting at $\geq$ 2 times average and < 5 times average)

  HAF6 'heavy' repeat (standard setting at > 8 times average)

  HAF7 'crazy' repeat (standard setting at > 20 times average)

3.8 Where reads end up: contigs, singlets, debris

At the start, things are simple: a read either aligns with other reads or it does not. Reads which align with other reads form contigs, and these MIRA will save in the results with a contig name of *_c*.

However, not all reads can be placed in an assembly. This can have several reasons and these reads may end up at two different places in the result files: either in the *debris* file, then just as a name entry, or as singlet (a "contig" with just one read) in the regular results.

1. reads are too short and get filtered out (before or after the MIRA clipping stages). These invariably land in the debris file.
2. reads are real singlets: they contain genuine sequence but have no overlap with any other read. These get either caught by the [-CL:pec] clipping filter or during the SKIM phase
3. reads contain mostly or completely junk.
4. reads contain chimeric sequence (therefore: they're also junk)

MIRA filters out these reads in different stages: before and after read clipping, during the SKIM stage, during the Smith-Waterman overlap checking stage or during contig building.

The exact place where these single reads land is dependend on why they do not align with other reads.

3.9 Detection of bases distinguishing non-perfect repeats and SNP discovery

MIRA is able to find and tag SNPs in any kind of data -- be it genomic or EST -- in both de-novo and mapping assemblies ... provided it knows which read in an assembly is coming from which strain, cell line or organism.

The SNP detection routines are based on the same routines as the routines for detecting non-perfect repeats. In fact, MIRA can even distinguish between bases marking a misassembled repeat from bases marking a SNP within the same project.

All you need to do to enable this feature is to set [-CO:mr=yes] (which is standard in all `--job=...` incantations of **mira** and in some steps of **miraSearchESTSNPs**. Furthermore, you will need:

in de-novo assemblies: to provide a *straindata* file for the reads or have the strain information in ancillary NCBI TRACEINFO XML files.

in mapping assemblies to provide a *straindata* file for the reads and also give the reference sequence(s) (backbone(s)) a strain name via the [-SB:bsn] parameter.

The effect of using strain names attached to reads can be described briefly like this. Assume that you have 6 reads (called R1 to R6), three of them having an `A` at a given position, the other three a `C`.

```
R1   ......A......
R2   ......A......
R3   ......A......
R4   ......C......
R5   ......C......
R6   ......C......
```

Note This example is just that: an example. It uses just 6 reads, with two times three reads as read groups for demonstration purposes and without looking at qualities. For MIRA to recognise SNPs, a few things must come together (e.g. for many sequencing technologies it wants forward and backward reads when in de-novo assembly) and a couple of parameters can be set to adjust the sensitivity. Read more about the parameters: [-CO:mrpg:mnq:mgqrt:emea:amgb:amgbemc:amgbnbs]

Now, assume you did not give any strain information. MIRA will most probably recognise a problem and, having no strain information, assume it made an error by assembling two different repeats of the same organism. It will tag the bases in the reads with repeat marker tags (SRMr) and the base in the consensus with a SROc tag (to point at an unresolved problem). In a subsequent pass, MIRA will then not assemble these six reads together again, but create two contigs like this:

```
Contig1:
R1   ......A......
R2   ......A......
R3   ......A......

Contig2:
R4   ......C......
R5   ......C......
R6   ......C......
```

The bases in the repeats will keep their SROr tags, but the consensus base of each contig will not get SROc as there is no conflict anymore.

Now, assume you gave reads R1, R2 and R3 the strain information "human", and read R4, R5 and R6 "chimpanzee". MIRA will then create this:

```
R1 (hum)  ......A......
R2 (hum)  ......A......
R3 (hum)  ......A......
R4 (chi)  ......C......
R5 (chi)  ......C......
R6 (chi)  ......C......
```

Instead of creating two contigs, it will create again one contig ... but it will tag the bases in the reads with a SROr tag and the position in the contig with a SROc tag. The SRO tags (**S**NP inte**R** **O**rganisms) tell you: there's a SNP between those two (or multiple) strains/organisms/whatever.

Changing the above example a little, assume you have this assembly early on during the MIRA process:

```
R1 (hum)  ......A......
R2 (hum)  ......A......
R3 (hum)  ......A......
R4 (chi)  ......A......
R5 (chi)  ......A......
R6 (chi)  ......A......
R7 (chi)  ......C......
R8 (chi)  ......C......
R9 (chi)  ......C......
```

Because "chimp" has a SNP within itself (`A` versus `C`) and there's a SNP between "human" and "chimp" (also `A` versus `C`), MIRA will see a problem and set a tag, this time a SIOr tag: **S**NP **I**ntra- and inter **O**rganism.

MIRA does not like conflicts occurring within an organism and will try to resolve these cleanly. After setting the SIOr tags, MIRA will re-assemble in subsequent passes this:

```
Contig1:
R1 (hum)  ......A......
R2 (hum)  ......A......
R3 (hum)  ......A......
R4 (chi)  ......A......
R5 (chi)  ......A......
R6 (chi)  ......A......

Contig2:
R7 (chi)  ......C......
R8 (chi)  ......C......
R9 (chi)  ......C......
```

The reads in Contig1 (hum+chi) and Contig2 (chi) will keep their SIOr tags, the consensus will have no SIOc tag as the "problem" was resolved.

When presented to conflicting information regarding SNPs and possible repeat markers or SNPs within an organism, MIRA will always first try to resolve the repeats marker. Assume the following situation:

```
R1 (hum)  ......A...T......
R2 (hum)  ......A...G......
R3 (hum)  ......A...T......
R4 (chi)  ......C...G......
R5 (chi)  ......C...T......
R6 (chi)  ......C...G......
```

While the first discrepancy column can be "explained away" by a SNP between organisms (it will get a SROr/SROc tag), the second column cannot and will get a SIOr/SIOc tag. After that, MIRA opts to get the SIO conflict resolved:

```
Contig1:
R1 (hum)  ......A...T......
R3 (hum)  ......A...T......
R5 (chi)  ......C...T......

Contig2:
R2 (hum)  ......A...G......
R4 (chi)  ......C...G......
R6 (chi)  ......C...G......
```

3.10 Caveats

3.10.1 Using data not from sequencing instruments: artifical / syntethic reads

The default parameters for MIRA assemblies work best when given real sequencing data and they even expect the data to behave like real sequencing data. But some assembly strategies work in multiple rounds, using so called "artificial" or "synthetic" reads in later rounds, i.e., data which was not generated through sequencing machines but might be something like the consensus of previous assemblies.

If one doesn't take utter care to make these artificial reads at least behave a little bit like real sequencing data, a number of quality ensurance algorithms of MIRA might spot that they "look funny" and trim back these artificial reads ... sometimes even removing them completely. The following list gives a short overview on what these synthetic reads should look like or which MIRA algorithms to switch off in certain cases:

- Forward and reverse complement directions: most sequencing technologies and strategies yield a mixture of reads with both forward and reverse complement direction to the DNA sequenced. In fact, having both directions allows for a much better quality control of an alignment as sequencing technology dependent sequencing errors will often affect only one direction at a given place and not both (the exception being homopolymers and 454).

  The MIRA *proposed end clipping* algorithm [-CL:pec] uses this knowledge to initially trim back ends of reads to an area without sequencing errors. However, if reads covering a given area of DNA are present in only one direction, then these reads will be completely eliminated.

  If you use only artificial reads in an assembly, then switch off the *proposed end clipping* [-CL:pec=no].

  If you mix artificial reads with "normal" reads, make sure that every part of an artificial read is covered by some other read in reverse complement direction (be it a normal or artificial read). The easiest way to do that is to add a reverse complement for every artificial read yourself, though if you use an overlapping strategy with artificial reads, you can calculate the overlaps and reverse complements of reads so that every second artificial read is in reverse complement to save time and memory afterwards during the computation.

- Sequencing type/technology: MIRA currently knows Sangers, 454, Ion Torrent, Solexa and PacBio as sequencing technologies, every read entered in an assembly must be one of those.

  Artificial reads should be classified depending on the data they were created from, that is, Sanger for consensus of Sanger reads, 454 for consensus of 454 reads etc. However, Should reads created from Illumina consensus be much longer than, say, 200 or 300 bases, you should treat them as Sanger reads.

- Quality values: be careful to assign decent quality values to your artificial reads as several quality clipping or consensus calling algorithms make extensive use of qualities. Pay attention to values of [-CL:qc:bsqc] as well as to [-CO:mrpg:mnq:mgqrt].

- Read lengths: current maximum read length for MIRA is around ~30kb. However, to account for some safety, MIRA currently allows only 20kb reads as maximum length.

3.10.2 Ploidy and repeats

MIRA treats ploidy differences as repeats and will therefore build a separate contigs for the reads of a ploidy that has a difference to the other ploidy/ploidies.

There is simply no other way to handle ploidy while retaining the ability to separate repeats based on differences of only a single base. Everything else would be guesswork. I thought for some time about doing a coverage analysis around the potential repeat/ploidy site, but came to the conclusion that due to the stochastic nature of sequencing data, this would very probably take wrong decisions in too many cases to be acceptable.

If someone has a good idea, I'll be happy to hear it.

3.10.3 Handling of repeats

3.10.3.1 Uniform read distribution

Under the assumption that reads in a project are uniformly distributed across the genome, MIRA will enforce an average coverage and temporarily reject reads from a contig when this average coverage multiplied by a safety factor is reached at a given site. This strategy reduces overcompression of repeats during the contig building phase and keeps reads in reserve for other copies of that repeat.

It's generally a very useful tool disentangle repeats, but has some slight secondary effects: rejection of otherwise perfectly good reads. The assumption of read distribution uniformity is the big problem we have here: of course it's not really valid. You sometimes have less, and sometimes more than "the average" coverage. Furthermore, the new sequencing technologies - 454 perhaps but certainly the ones from Solexa - show that you also have a skew towards the site of replication origin.

Warning: Solexa data from late 2009 and 2010 show a high GC content bias. This bias can reach 200 or 300%, i.e., sequence part for with low GC

One example: let's assume the average coverage of a project is 8 and by chance at one place there 17 (non-repetitive) reads, then the following happens:

(Note: p is the parameter [-AS:urdsip])

Pass 1 to $p-1$: MIRA happily assembles everything together and calculates a number of different things, amongst them an average coverage of ~8. At the end of pass $p-1$, it will announce this average coverage as first estimate to the assembly process.

Pass p: MIRA has still assembled everything together, but at the end of each pass the contig self-checking algorithms now include an "average coverage check". They'll invariably find the 17 reads stacked and decide (looking at the [-AS:ardct] parameter which is assumed to be 2 for this example) that 17 is larger than 2*8 and that this very well may be a repeat. The reads get flagged as possible repeats.

Pass $p+1$ to end: the "possibly repetitive" reads get a much tougher treatment in MIRA. Amongst other things, when building the contig, the contig now looks that "possibly repetitive" reads do not overstack by an average coverage multiplied by a safety value ([-AS:urdcm]) which we'll assume now to be 1.5 in this example. So, at a certain point, say when read 14 or 15 of that possible repeat want to be aligned to the contig at this given place, the contig will just flatly refuse and tell the assembler to please find another place for them, be it in this contig that is built or any other that will follow. Of course, if the assembler cannot comply, the reads 14 to 17 will end up as contiglet (contig debris, if you want) or if it was only one read that got rejected like this, it will end up as singlet or in the debris file.

Tough luck. I do have ideas on how to reintegrate those reads at the and of an assembly, but I have deferred doing this as in every case I had looked up, adding those reads to the contigs wouldn't have changed anything ... there's already enough coverage.

What should be done in those cases is simply filter away the contiglets (defined as being of small size and having an average coverage below the average coverage of the project divided 3 (or 2.5)) from a project.

3.10.3.2 Keeping 'long' repetitive contigs separate

MIRA had since 2.9.36 a feature to keep long repeats in separate contigs ([-AS:klrs]). Due to algorithm changes, this feature is now standard (even if the command line parameter is still present). The effect of this is that contigs with non-repetitive sequence will stop at a 'long repeat' border, including only the first few bases of the repeat. Long repeats will be kept as separate contigs.

This has been implemented to get a clean overview on which parts of an assembly are 'safe' and which parts will be 'difficult'. For this, the naming of the contigs has been extended: contigs named with a '_c' at the end are contigs which contain mostly

'normal' coverage. Contigs with "rep_c" are contigs which contain mostly sequence classified as repetitive and which could not be assembled together with a 'c' contig.

The question remains: what are 'long' repeats. MIRA defines these as repeats that are not spanned by any read that has non-repetitive parts at the end. So, basically, the mean length of the reads that go into the assembly defines the length of 'long' repeats that have to be kept in separate contigs.

It has to be noted that when using paired-end (or template) sequencing, 'long' repeats which can be spanned by read-pairs (or templates) are mostly integrated into 'normal' contigs as MIRA can correctly place them most of the time.

3.10.3.3 Helping finishing by tagging reads with HAF tags

HAF tags (HAsh Frequency) are set by MIRA when the option to colour reads by hash frequency ([-GE:crhf], on by default in most --job combinations) is on. These tags show the status of k-mers (stretch of bases of given length k) in read sequences: whether MIRA recognised them as being present in sub-average, average, above average or repetitive numbers.

When using a finishing programs which can display tags in reads (and using the proposed tag colour schemes for gap4 or consed, the assembly will light up in colours ranging from light green to dark red, indicating whether a certain part of the assembly is deemed non-repetitive to extremely repetitive.

One of the biggest advantages of the HAF tags is the implicit information they convey on why the assembler stopped building a contig at an end.

- if the read parts composing a contig end are mostly covered with HAF2 tags (below average frequency, coloured light-green), then one very probably has a hole in the contig due to coverage problems which means there are no or not enough reads covering a part of the sequence.
- if the read parts composing a contig end are mostly covered with HAF3 tags (average frequency, coloured green), then you have an unusual situation as this should only very rarely occur. The reason is that MIRA saw that there are enough sequences which look the same as the one from your contig end, but that these could not be joined. Likely reasons for this scenario include non-random sequencing artifacts (seen in 454 data) or also non-random chimeric reads (seen in Sanger and 454 data).
- if the read parts composing a contig end are mostly covered with HAF4 tags (above average frequency, coloured yellow), then the assembler stopped at grey zone of the coverage not being normal anymore, but not quite repetitive yet. This can happen in cases where the read coverage is very unevenly distributed across the project. The contig end in question might be a repeat occurring two times in the sequence, but having less reads than expected. Or it may be non-repetitive coverage with an unusual excess of reads.
- if the read parts composing a contig end are mostly covered with HAF5 (repeat, coloured red), HAF6 (heavy repeat, coloured darker red) and HAF7 tags (crazy repeat, coloured very dark red), then there is a repetitive area in the sequence which could not be uniquely bridged by the reads present in the assembly.

This information can be especially helpful when joining reads by hand in a finishing program. The following list gives you a short guide to cases which are most likely to occur and what you should do.

- the proposed join involves contig ends mostly covered by HAF2 tags. Joining these contigs is probably a safe bet. The assembly may have missed this join because of too many errors in the read ends or because sequence having been clipped away which could be useful to join contigs. Just check whether the join seems sensible, then join.
- the proposed join involves contig ends mostly covered by HAF3 tags. Joining these contigs is probably a safe bet. The assembly may have missed this join because of several similar chimeric reads reads or reads with similar, severe sequencing errors covering the same spot. Just check whether the join seems sensible, then join.
- the proposed join involves contig ends mostly covered by HAF4 tags. Joining these contigs should be done with some caution, it may be a repeat occurring twice in the sequence. Check whether the contig ends in question align with ends of other contigs. If not, joining is probably the way to go. If potential joins exist with other contigs, then it's a repeat (see below).
- the proposed join involves contig ends mostly covered by HAF5, HAF6 or HAF7 tags. Joining these contigs should be done with utmost caution, you are almost certainly (HAF5) and very certainly (HAF6 and HAF7) in a repetitive area of your sequence. You will probably need additional information like paired-end or template info in order join your contigs.

3.10.4 Consensus in finishing programs (gap4, consed, ...)

MIRA goes a long way to calculate a consensus which is as correct as possible. Unfortunately, communication with finishing programs is a bit problematic as there currently is no standard way to say which reads are from which sequencing technology.

It is therefore often the case that finishing programs calculate an own consensus when loading a project assembled with MIRA. This is the case for at least, e.g., gap4. This consensus may then not be optimal.

The recommended way to deal with this problem is: import the results from MIRA into your finishing program like you always do. Then finish the genome there, export the project from the finishing program as CAF and finally use convert_project (from the MIRA package) with the "-r" option to recalculate the optimal consensus of your finished project.

E.g., assuming you have just finished editing the gap4 database `DEMO.3`, do the following. First, export the gap4 database back to CAF:

```
$ gap2caf -project DEMO -version 3 >demo3.caf
```

Then, use convert_project *with option '-r'* to convert it into any other format that you need. Example for converting to a CAF and a FASTA format with correct consensus:

```
$ convert_project -f caf -t caf -t fasta -r c demo3.caf final_result
```

3.10.5 Some other things to consider

- mira cannot work with EXP files resulting from GAP4 that already have been edited. If you want to reassemble an edited GAP4 project, convert it to CAF format and use the [-caf] option to load.
- As also explained earlier, mira relies on sequencing vector being recognised in preprocessing steps by other programs. Sometimes, when a whole stretch of bases is not correctly marked as sequencing vector, the reads might not be aligned into a contig although they might otherwise match quite perfectly. You can use [-CL:pvc] and [-CO:emea] to address problem with incomplete clipping of sequencing vectors. Also having the assembler work with less strict parameters may help out of this.
- mira has been developed to assemble shotgun sequencing or EST sequencing data. There are no explicit limitations concerning length or number of sequences. However, there are a few implicit assumptions that were made while writing portions of the code:

  1. Sequence data produced by electrophoresis rarely surpasses 1000 usable bases and I never heard of, let alone seen, more than 1100. The fast filtering SKIM relies on the fact that sequences will never exceed 10000 bases in length.
  2. The next problem that might arise with 'unnatural' long sequence reads will be my implementation of the Smith-Waterman alignment routines. I use a banded version with linear running time (linear to the bandwidth) but quadratic space usage. So, comparing two 'reads' of length 5000 will result in memory usage of 100MB. I know that this could be considered as a flaw. On the other hand - unless someone comes up with electrophoresis producing reads with more than 2000 usable bases - I see no real need to change this as long as there are more important things on the TODO list. Of course, if anyone is willing to contribute a fast banded SW alignment routine which runs in linear time and space, just feel free to contact the author.
  3. Current data structures allow for a worst case read coverage of maximally 16384 reads on top of the other.

     Note: this limit was more than enough for about any kind of genome sequencing, but since people started to do sequencing of non-normalised EST libraries with 454 and Solexa, this limit can be reached all too often. This will change in future releases.
  4. the 32-bit Linux version is limited by the memory made available by the Linux kernel (somewhere around 2.3 to 2.7GB).
  5. to reduce memory overhead, the following assumptions have been made:
  6. the 64-bit Linux version has no implicit memory limits, although the maximum number of bases of all reads may not surpass 2.147.483.648 bases. With that, even aliens with a genome size ~800 times bigger than humans could be tackled (if it were not for other limitations, mainly RAM and processing power).
  7. mira is not fully multi-threaded (yet), but even Sanger projects for bigger bacteria can be assembled in ~2-3 hours on a current hardware platform. Fungi may take two or three days.

     For 454 genome projects, bacteria should be done in about a day at most, Fungi could take about 10 days.

- a project does not contain sequences from more than 255 different:
  - sequencing machine types
  - primers
  - strains (in mapping mode: 7)
  - base callers
  - dyes
  - process status
- a project does not contain sequences from more than 65535 different
  - clone vectors
  - sequencing vectors

Note: Versions with uneven minor versions (e.g. 1.1.x, 1.3.x, ..., 2.1.x, ... etc.) are development versions which might be unstable in parts (although I don't think so). But to catch possible bugs, development versions of mira are distributed with tons of internal checks compiled into the code, making it somewhere between 10% and 50% slower than it could be.

3.11 Things you should not do

3.11.1 Do not run MIRA on NFS mounted directories without redirecting the tmp directory

Of course one can run MIRA atop a NFS mount (a "disk" mounted over a network using the NFS protocol), but the performance will go down the drain as the NFS server respectively the network will not be able to cope with the amount of data MIRA needs to shift to and from disk (writes/reads to the tmp directory). Slowdowns of a factor of 10 and more have been observed. In case you have no other possibility, you can force MIRA to run atop a NFS using [-MI:sonfs=no], but you have been warned.

In case you want to keep input and output files on NFS, you can use [-DI:trt] to redirect the tmp directory to a local filesystem. Then MIRA will run at almost full speed.

3.11.2 Do not assemble without quality values

Assembling sequences without quality values is like ... like ... like driving a car downhill a sinuous mountain road at 200 km/h without brakes, airbags and no steering wheel. With a ravine on one side and a rock face on the other. Did I mention the missing seat-belts? You *might* get down safely, but experience tells the result will rather be a bloody mess.

All MIRA routines internally are geared toward quality values guiding decisions. No one should ever assembly anything without quality values. Never. Ever. Even if quality values are sometimes inaccurate, they do help.

Now, there are **very rare occasions** where getting quality values is not possible. If you absolutely cannot get them, and I mean only in this case, use these switches:`--noqualities[=SEQUENCINGTECHNOLOGY]` *`SEQUENCINGTECHNOLOGY`*`_SETTINGS -AS:bdq=30`. E.g.:

```
--noqualities=454 454_SETTINGS -AS:bdq=30
```

or

```
--noqualities SANGER_SETTINGS -AS:bdq=30 454_SETTINGS -AS:bdq=30
```

This tells MIRA not to complain about missing quality values and to fake a quality value of 30 for all reads having no qualities, allowing some MIRA routines (in standard parameter settings) to start disentangling your repeats.

Warning Doing the above has some severe side-effects. You will be, e.g., at the mercy of non-random sequencing errors. I suggest combining the above with a [-CO:mrpg=4] or higher. You also may want to tune the [-AS:bdq] parameter together with [-CO:mnq] and [-CO:mgqrt] in cases where you mix sequences with and without quality values.

3.12 Useful third party programs

Viewing the results of a mira assembly or preprocessing the sequences for an assembly can be done with a number of different programs. The following ones are are just examples, there are a lot more packages available:

HTML browser If you have really nothing else as viewer, a browser who understands tables is needed to view the HTML output. A browser knowing style sheets (CSS) is recommended, as different tags will be highlighted. Konqueror, Opera, Mozilla, Netscape and Internet Explorer all do fine, lynx is not really ... optimal.

Assembly viewer / finishing / preprocessing You'll want GAP4 (generally speaking: the Staden package) to preprocess the sequences, visualise and eventually rework the results when using gap4da output. The Staden package comes with a fully featured sequence preparing and annotating engine (pregap4) that is very useful to preprocess your data (conversion between file types, quality clipping, tagging etc.).

See http://www.sourceforge.net/projects/staden/ for further information and also a possibility to download precompiled binaries for different platforms.

Vector screening Reading result files from **ssaha2** or **smalt** from the Sanger centre is supported directly by mira to perform a fast and efficient tagging of sequencing vector stretches. This makes you basically independent from any other commercial or license-requiring vector screening software. For Sanger reads, a combination of **lucy** (see below), **ssaha2** or **smalt** together with the mira parameters for SSAHA2 / SMALT support ([-CL:msvs]) and quality clipping ([-CL:qc]) should do the trick. For reads coming from 454 pyro-sequencing, **ssaha2** or **smalt** and the SSAHA2 / SMALT support also work pretty well.

See http://www.sanger.ac.uk/resources/software/ssaha2/ and / or http://www.sanger.ac.uk/resources/software/smalt/ for further information and also a possibility to download the source or precompiled binaries for different platforms.

Preprocessing **lucy** from TIGR (now JCVI) is another useful sequence preprocessing program. Lucy is a utility that prepares raw DNA sequence fragments for sequence assembly. The cleanup process includes quality assessment, confidence reassurance, vector trimming and vector removal.

There's a small script in the MIRA 3rd party package which converts the clipping data from the lucy format into something mira can understand (NCBI Traceinfo).

See ftp://ftp.tigr.org/pub/software/Lucy/ to download the source code of lucy.

Assembly viewer Viewing `.ace` file output without consed can be done with clview from TIGR. See http://www.tigr.org/tdb/tgi/software/.

Tablet http://bioinf.scri.ac.uk/tablet/ may also be used for this.

Assembly coverage analysis The Integrated Genome Browser (IGB) of the GenoViz project at SourceForge (http://sourceforg net/projects/genoviz/) is just perfect for loading a genome and looking at mapping coverage (provided by the wiggle result files of MIRA).

Preprocessing (base calling) TraceTuner (http://sourceforge.net/projects/tracetuner/) is a tool for base and quality calling of trace files from DNA sequencing instruments. Originally developed by Paracel, this code base was released as open source in 2006 by Celera.

Preprocessing / viewing phred (basecaller) - cross_match (sequence comparison and filtering) - phrap (assembler) - consed (assembly viewer and editor). This is another package that can be used for this type of job, but requires more programming work. The fact that sequence stretches are masked out (overwritten with the character X) if they shouldn't be used in an assembly doesn't really help and is considered harmful (but it works).

Note the bug of consed when reading ACE files, see more about this in the section on file types (above) in the entry for ACE.

See http://www.phrap.org/ for further information.

text viewer A text viewer for the different textual output files.

As always, most of the time a combination of several different packages is possible. My currently preferred combo for genome projects is **ssaha2** or **smalt** and or **lucy** (vector screening), MIRA (assembly, of course) and gap4 (assembly viewing and finishing).

For re-assembling projects that were edited in gap4, one will also need the gap2caf converter. The source for this is available at http://www.sanger.ac.uk/resources/software/caf.html.

3.13 Speed and memory considerations

3.13.1 Estimating needed memory for an assembly project

Since the V2.9.24x3 version of mira, there is **miramem** as program call. When called from the command line, it will ask a number of questions and then print out an estimate of the amount of RAM needed to assemble the project. Take this estimate with a grain of salt, depending on the sequences properties, variations in the estimate can be +/- 30% for bacteria and 'simple' eukaryotes. The higher the number of repeats is, the more likely you will need to restrict memory usage in some way or another.

Here's the transcript of a session with miramem:

```
This is MIRA V3.2.0rc1 (development version).

Please cite: Chevreux, B., Wetter, T. and Suhai, S. (1999), Genome Sequence
Assembly Using Trace Signals and Additional Sequence Information.
Computer Science and Biology: Proceedings of the German Conference on
Bioinformatics (GCB) 99, pp. 45-56.

To (un-)subscribe the MIRA mailing lists, see:
        http://www.chevreux.org/mira_mailinglists.html

After subscribing, mail general questions to the MIRA talk mailing list:
        mira_talk@freelists.org

To report bugs or ask for features, please use the new ticketing system at:
        http://sourceforge.net/apps/trac/mira-assembler/
This ensures that requests don't get lost.

[...]

miraMEM helps you to estimate the memory needed to assemble a project.
Please answer the questions below.

Defaults are give in square brackets and chosen if you just press return.
Hint: you can add k/m/g modifiers to your numbers to say kilo, mega or giga.

Is it a genome or transcript (EST/tag/etc.) project? (g/e/) [g]
g
Size of genome? [4.5m] 9.8m
9800000
Size of largest chromosome? [9800000]
9800000
Is it a denovo or mapping assembly? (d/m/) [d]
d
Number of Sanger reads? [0]
0
Are there 454 reads? (y/n/) [n] y
y
Number of 454 GS20 reads? [0]
0
Number of 454 FLX reads? [0]
0
Number of 454 Titanium reads? [0] 750k
```

```
750000
Are there PacBio reads? (y/n/) [n]
n
Are there Solexa reads? (y/n/) [n]
n


************************ Estimates ************************

The contigs will have an average coverage of ~ 30.6 (+/- 10%)

RAM estimates:
           reads+contigs (unavoidable): 7.0 GiB
                large tables (tunable): 688. MiB
                                        ---------
                           total (peak): 7.7 GiB

            add if using -CL:pvlc=yes : 2.6 GiB

Estimates may be way off for pathological cases.

Note that some algorithms might try to grab more memory if
the need arises and the system has enough RAM. The options
for automatic memory management control this:
  -AS:amm, -AS:kpmf, -AS:mps
Further switches that might reduce RAM (at cost of run time
or accuracy):
  -SK:mhim, -SK:mchr (both runtime); -SK:mhpr (accuracy)
************************************************************
```

If your RAM is not large enough, you can still assemble projects by using disk swap. Up to 20% of the needed memory can be provided by swap without the speed penalty getting too large. Going above 20% is not recommended though, above 30% the machine will be almost permanently swapping at some point or another.

3.13.2 Some numbers on speed

NEW since 2.7.4: The new SKIM3 algorithm (initial all-against-all read comparison) is now approximately 60 times faster that the SKIM algorithms of earlier versions. E.g. SKIMming of 53,000 Sanger type shotguns reads now takes a bit more than a minute instead of 62 minutes.

The times given below are only approximate and were gathered on my home development box (Athlon 4800+) using a single core and minimal debug code compiled in, somewhat slowing down the whole process.

Example 1: a small genomic project with 720 reads forming 35k bases of contig sequences. Using `--job=denovo,genome,accurate,sanger` and resolving minor repeat misassemblies, full read extension and automatic contig editing takes 19 seconds.

Example 2: a bacterial genome project with two very closely related strains, 53000 Sanger reads forming a bit more than 3 megabases of contig sequences for each strain. Using the `--job=denovo,genome,accurate,sanger` (four main passes, read extension, clipping of vector remnants), resolving repeat misassemblies (mostly RNA stretches, but also some very closely related genes) takes 1hr and 48 minutes and uses a maximum of 1.2GB of RAM (miramem estimated the usage to be 1.5GB).

Example 3: Here are the times for miraSearchESTSNPs in a non-normalised (thus very repetitive) EST project, 9747 reads with a average length of 674 used bases,

- The fast filtering algorithm performs about 12 million sequence comparisons per second (8 seconds).
- Banded Smith-Waterman performs around 750 sequence alignments per second (with a 15% band to each side, which is quite generous), 4:07 for about 182000 alignment checks.

The three steps of miraSearchESTSNPs (each one again subdivided in a number of MIRA passes), including resolving very high coverage contigs (>500 sequences) in multiple passes and splitting them into different SNP and splice variants takes about 20 minutes.

3.14 Known Problems / Bugs

File Input / Output:

1. mira can only read unedited EXP files.
2. There sometimes is a (rather important) memory leak occurring while using the assembly integrated Sanger read editor. I have not been able to trace the reason yet.
3. There's an unexpected bug for MacOS which leads to rubbish assemblies on large data sets which assemble totally fine with Linux versions of MIRA. I have not been able to find out the reason for this yet.

Assembly process:

1. The routines for determining *Repeat Marker Bases* (SRMr) are sometimes too sensitive, which sometimes leads to excessive base tagging and preventing right assemblies in subsequent assembly processes. The parameters you should look at for this problem are [-CO:mrc:nrz:mgqrt:mgqwpc]. Also look at [-CL:pvc] and [-CO:emea] if you have a lot of sequencing vector relics at the end of the sequences.
2. The assignment of reads to debris or singlets and whether or not they are put into the final result is messy, the statistic numbers about this sometimes even wrong. Needs to be redone.

3.15 TODOs

These are some of the topics on my TODO list for the next revisions to come:

1. Making parts of the process multi-threaded (currently stopped due to other priorities like Solexa etc.)
2. Less disk usage when using EST assembly on 10 or more million Solexa reads
3. Others nifty ideas that I have not completely thought out yet.

3.16 Working principles

Note: description is old and needs to be adapted to the current 2.9.x / 3.x line.

To avoid the "garbage-in, garbage-out" problematic, mira uses a 'high quality alignments first' contig building strategy. This means that the assembler will start with those regions of sequences that have been marked as good quality (high confidence region - HCR) with low error probabilities (the clipping must have been done by the base caller or other preprocessing programs, e.g. pregap4) and then gradually extends the alignments as errors in different reads are resolved through error hypothesis verification and signal analysis.

This assembly approach relies on some of the automatic editing functionality provided by the EdIt package which has been integrated in parts within mira.

This is an approximate overview on the steps that are executed while assembling:

1. All the experiment / phd / fasta sequences that act as input are loaded (or the CAF project). Qualities for the bases are loaded from the FASTA or SCF if needed.

2. the ends of the reads are cleaned ensure they have a minimum stretch of bases without sequencing errors
3. The high confidence region (HCR) of each read is compared with a quick algorithm to the HCR of every other read to see if it could match and have overlapping parts (this is the 'SKIM' filter).
4. All the reads which could match are being checked with an adapted Smith-Waterman alignment algorithm (banded version). Obvious mismatches are rejected, the accepted alignments form one or several alignment graphs.
5. Optional pre-assembly read extension step: mira tries to extend HCR of reads by analysing the read pairs from the previous alignment. This is a bit shaky as reads in this step have not been edited yet, but it can help. Go back to step 2.
6. A contig gets made by building a preliminary partial path through the alignment graph (through in-depth analysis up to a given level) and then adding the most probable overlap candidates to a given contig. Contigs may reject reads if these introduce to many errors in the existing consensus. Errors in regions known as dangerous (for the time being only ALUS and REPT) get additional attention by performing simple signal analysis when alignment discrepancies occur.
7. Optional: the contig can be analysed and corrected by the automatic editor ("EdIt" for Sanger reads, or the new MIRA editor for 454 reads).
8. Long repeats are searched for, bases in reads of different repeats that have been assembled together but differ sufficiently (for EdIT so that they didn't get edited and by phred quality value) get tagged with special tags (SRMr and WRMr).
9. Go back to step 5 if there are reads present that have not been assembled into contigs.
10. Optional: Detection of spoiler reads that prevent joining of contigs. Remedy by shortening them.
11. Optional: Write out a checkpoint assembly file and go back to step 2.
12. The resulting project is written out to different output files and directories.

3.17 See Also

The other MIRA manuals and walkthroughs as well as **EdIt**, **gap4**, **pregap4**, **gap5**, **clview**, **caf2gap**, **gap2caf**, **ssaha2**, **smalt**, **compress** and **gzip**, **cap3**, **ttuner**, **phred**, **phrap**, **cross_match**, **consed**.

Chapter 4

Short usage introduction to MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'Just when you think it's finally settled, it isn't. '*
>
> —Solomon Short

This guide assumes that you have basic working knowledge of Unix systems, know the basic principles of sequencing (and sequence assembly) and what assemblers do. Furthermore, it is advised to read through the main documentation of the assembler as this is really just a getting started guide.

4.1 Important notes

For working parameter settings for assemblies involving 454 and / or Solexa data, please also read the MIRA help files dedicated to these platforms.

4.2 Quick start for the impatient

This example assumes that you have a few sequences in FASTA format that may or may not have been preprocessed - that is, where sequencing vector has been cut back or masked out. If quality values are also present in a fasta like format, so much the better.

We need to give a name to our project: throughout this example, we will assume that the sequences we are working with are from *Bacillus chocorafoliensis* (or short: *Bchoc*); a well known, chocolate-adoring bug from the *Bacillus* family which is able to make a couple of hundred grams of chocolate vanish in just a few minutes.

Our project will therefore be named 'bchoc'.

4.2.1 Estimating memory needs

"Do I have enough memory?" has been one of the most often asked question in former times. To answer this question, please use miramem which will give you an estimate. Basically, you just need to start the program and answer the questions, for more information please refer to the corresponding section in the main MIRA documentation.

Take this estimate with a grain of salt, depending on the sequences properties, variations in the estimate can be +/- 30%.

4.2.2 Preparing and starting an assembly from scratch with FASTA files

4.2.2.1 With data pre-clipped or pre-screened for vector sequence

The following steps will allow to quickly start a simple assembly if your sequencing provider gave you data which was pre-clipped or pre-screened for vector sequence:

```
$ mkdir bchoc_assembly1
$ cd bchoc_assembly1
bchoc_assembly1$ cp /your/path/sequences.fasta bchoc_in.sanger.fasta
bchoc_assembly1$ cp /your/path/qualities.someextension bchoc_in.sanger.fasta.qual
bchoc_assembly1$ mira --project=bchoc --job=denovo,genome,accurate,sanger --fasta
```

Explanation: we created a directory for the assembly, copied the sequences into it (to make things easier for us, we named the file directly in a format suitable for mira to load it automatically) and we also copied quality values for the sequences into the same directory. As last step, we started mira with options telling it that

- our project is named 'bchoc' and hence, input and output files will have this as prefix;
- the data is in a FASTA formatted file;
- the data should be assembled *de-novo* as a *genome* at an assembly quality level of *accurate* and that the reads we are assembling were generated with Sanger technology.

By giving mira the project name 'bchoc' (`--project=bchoc`) and naming sequence file with an appropriate extension `_in.sanger.fasta`, mira automatically loaded that file for assembly. When there are additional quality values available (`bchoc_in.sanger.fasta.qual`), these are also automatically loaded and used for the assembly.

Note If there is no file with quality values available, MIRA will stop immediately. You will need to provide parameters to the command line which explicitly switch off loading and using quality files.

Warning Not using quality values is **NOT** recommended. Read the corresponding section in the MIRA reference manual.

4.2.2.2 Using SSAHA2 / SMALT to screen for vector sequence

If your sequencing provider gave you data which was NOT pre-clipped for vector sequence, you can do this yourself in a pretty robust manner using SSAHA2 -- or the successor, SMALT -- from the Sanger Centre. You just need to know which sequencing vector the provider used and have its sequence in FASTA format (ask your provider).

Note that this screening is a valid method for any type of Sanger sequencing vectors, 454 adaptors, Solexa adaptors and paired-end adaptors etc.

For SSAHA2 follow these steps (most are the same as in the example above):

```
$ mkdir bchoc_assembly1
$ cd bchoc_assembly1
bchoc_assembly1$ cp /your/path/sequences.fasta bchoc_in.sanger.fasta
bchoc_assembly1$ cp /your/path/qualities.someextension bchoc_in.sanger.fasta.qual
bchoc_assembly1$ ssaha2 -output ssaha2
  -kmer 8 -skip 1 -seeds 1 -score 12 -cmatch 9 -ckmer 6
  /path/where/the/vector/data/resides/vector.fasta
  bchoc_in.sanger.fasta > bchoc_ssaha2vectorscreen_in.txt
bchoc_assembly1$ mira -project=bchoc -job=denovo,genome,accurate,sanger -fasta  ↩
    SANGER_SETTINGS -CL:msvs=yes
```

Explanation: there are just two differences to the example above:

- calling SSAHA2 to generate a file which contains information on the vector sequence hitting your sequences.
- telling mira with `SANGER_SETTINGS -CL:msvs=yes` to load this vector screening data for Sanger data

For SMALT, the only difference is that you use SMALT for generating the vector-screen file and ask SMALT to generate it in SSAHA2 format. As SMALT works in two steps (indexing and then mapping), you also need to perform it in two steps and then call MIRA. E.g.:

```
bchoc_assembly1$ smalt index -k 7 -s 1 smaltidxdb /path/where/the/vector/data/resides/ ↩
    vector.fasta
bchoc_assembly1$ smalt map -f ssaha -d -1 -m 7 smaltidxdb bchoc_in.sanger.fasta  >  ↩
    bchoc_smaltvectorscreen_in.txt
bchoc_assembly1$ mira -project=bchoc -job=denovo,genome,accurate,sanger -fasta  ↩
    SANGER_SETTINGS -CL:msvs=yes
```

Note Please note that, due to subtle differences between output of SSAHA2 (in ssaha2 format) and SMALT (in ssaha2 format), MIRA identifies the source of the screening (and the parsing method it needs) by the name of the screen file. Therefore, screens done with SSAHA2 need to have the postfix `*_ssaha2vectorscreen_in.txt` in the file name and screens done with SMALT need `*_smaltvectorscreen_in.txt`.

4.3 Calling mira from the command line

Mira can be used in many different ways: building assemblies from scratch, performing reassembly on existing projects, assembling sequences from closely related strains, assembling sequences against an existing backbone (mapping assembly), etc.pp. Mira comes with a number of **quick switches**, i.e., switches that turn on parameter combinations which should be suited for most needs.

E.g.: `mira --project=foobar --job=sanger --fasta -highlyrepetitive`

The line above will tell mira that our project will have the general name *foobar* and that the sequences are to be loaded from FASTA files, the sequence input file being named `foobar_in.sanger.fasta` (and sequence quality file, if available, `foobar_in.sanger.fasta.qual`. The reads come from Sanger technology and mira is prepared for the genome containing nasty repeats. The result files will be in a directory named `foobar_results`, statistics about the assembly will be available in the `foobar_info` directory like, e.g., a summary of contig statistics in `foobar_info/foobar_info_contigstats.txt`. Notice that the *--job=* switch is missing some specifications, mira will automatically fill in the remaining defaults (i.e., denovo,genome,accurate in the example above).

E.g.: `mira --project=foobar --job=mapping,accurate,sanger --fasta --highlyrepetitive`

This is the same as the previous example except mira will perform a mapping assembly in 'accurate' quality of the sequences against a backbone sequence(s). mira will therefore additionally load the backbone sequence(s) from the file `foobar_backbone_in.fasta` (FASTA being the default type of backbone sequence to be loaded) and, if existing, quality values for the backbone from `foobar_backbone_in.fasta.qual`.

E.g.: `mira --project=foobar --job=mapping,accurate,sanger --fasta --highlyrepetitive -SB:bft=gbf`

As above, except we have added an **extensive switch** ([-SB:bft]) to tell mira that the backbones are in a GenBank format file (GBF). MIRA will therefore load the backbone sequence(s) from the file `foobar_backbone_in.gbf`. Note that the GBF file can also contain multiple entries, i.e., it can be a GBFF file.

E.g.: `mira --project=foobar --job=mapping,accurate,sanger --fastq --highlyrepetitive -SB:bft=gbf`

As above, except we have changed the input type for all files from FASTA to FASTQ.

4.4 Using multiple processors

This feature is in its infancy, presently only the SKIM algorithm uses multiple threads. Setting the number of processes for this stage can be done via the [-GE:not] parameter. E.g. `-GE:not=4` to use 4 threads.

4.5 Usage examples

4.5.1 Assembly from scratch with GAP4 and EXP files

A simple GAP4 project will do nicely. Please take care of the following: You need already preprocessed experiment / fasta / phd files, i.e., at least the sequencing vector should have been tagged (in EXP files) or masked out (FASTA or PHD files). It would be nice if some kind of not too lazy quality clipping had also been done for the EXP files, pregap4 should do this for you.

1. Step 1: Create a file of filenames (named `mira_in.fofn`) for the project you wish to assemble. The file of filenames should contain the newline separated names of the EXP-files and nothing else.
2. Step 2: Execute the mira assembly, eventually using command line options or output redirection:

   ```
   $ /path/to/the/mira/package/mira ... other options ...
   ```

   or simply

   ```
   $ mira ... other options ...
   ```

   if MIRA is in a directory which is in your PATH. The result of the assembly will now be in directory named `mira_results` where you will find `mira_out.caf`, `mira_out.html` etc. or in gap4 direct assembly format in the `mira_out.gap4da` sub-directory.
3. Step 3a: *(This is not recommended anymore)* Change to the gap4da directory and start gap4:

   ```
   $ cd mira_results/mira_out.gap4da
   mira_results/mira_out.gap4da$ gap4
   ```

   choose the menu 'File->New' and enter a name for your new database (like 'demo'). Then choose the menu 'Assembly->Directed assembly'. Enter the text 'fofn' in the entry labelled *Input readings from List or file name* and enter the text 'failures' into the entry labelled *Save failures to List or file name*. Press "OK".

   That's it.
4. Step 3b: *(Recommended)* As an alternative to step 3a, one can use the caf2gap converter (see below)

   ```
   mira_results$ caf2gap -project demo -version 0 -ace mira_out.caf
   mira_results$ gap4 DEMO.0
   ```

Out-of-the box example MIRA comes with a few really small toy project to test usability on a given system. Go to the minidemo directory and follow the instructions given in the section for own projects above, but start with step 2. Eventually, you might want to start mira while redirecting the output to a file for later analysis.

4.5.2 Reassembly of GAP4 edited projects

It is sometimes wanted to reassemble a project that has already been edited, for example when hidden data in reads has been uncovered or when some repetitive bases have been tagged manually. The canonical way to do this is by using CAF files as data exchange format and the caf2gap and gap2caf converters available from the Sanger Centre (http://www.sanger.ac.uk/Software/formats/CAF/).

Warning The project will be completely reassembled, contig joins or breaks that have been made in the GAP4 database will be lost, you will get an entirely new assembly with what mira determines to be the best assembly.

- Step 1: Convert your GAP4 project with the gap2caf tool. Assuming that the assembly is in the GAP4 database `CURRENT.0`, convert it with the gap2caf tool:

```
$ gap2caf -project CURRENT -version 0 -ace > newstart_in.caf
```

  The name *"newstart"* will be the project name of the new assembly project.

- Step 2: Start mira with the -caf option and tell it the name of your new reassembly project:

```
$ mira -caf=newstart
```

  (and other options like --job etc. at will.)

- Step 3: Convert the resulting CAF file `newstart_assembly/newstart_d_results/newstart_out.caf` to a gap4 database format as explained above and start gap4 with the new database:

```
$ cd newstart_assembly/newstart_d_results
newstart_assembly/newstart_d_results$ caf2gap -project reassembled -version 0 -ace  ↩
    newstart_out.caf
newstart_assembly/newstart_d_results$ gap4 REASSEMBLED.0
```

4.5.3 Using backbones to perform a mapping assembly against a reference sequence

One useful features of mira is the ability to assemble against already existing reference sequences or contigs (also called a mapping assembly). The parameters that control the behaviour of the assembly in these cases are in the [-STRAIN/BACKBONE] section of the parameters.

Please have a look at the example in the `minidemo/bbdemo2` directory which maps sequences from *C.jejuni RM1221* against (parts of) the genome of *C.jejuni NCTC1168*.

There are a few things to consider when using backbone sequences:

1. Backbone sequences can be as long as needed! They are not subject to normal read length constraints of a maximum of 10k bases. That is, if one wants to load one or several entire chromosomes of a bacterium or lower eukaryote as backbone sequence(s), this is just fine.

2. Backbone sequences can be single sequences like provided by, e.g., FASTA, FASTQ or GenBank files. But backbone sequences also can be whole assemblies when they are provided as, e.g., CAF format. This opens the possibility to perform semi-hybrid assemblies by assembling first reads from one sequencing technology de-novo (e.g. 454) and then map reads from another sequencing technology (e.g. Solexa) to the whole 454 alignment instead of mapping it to the 454 consensus.

   A semi-hybrid assembly will therefore contain, like a hybrid assembly, the reads of both sequencing technologies.

3. Backbone sequences will not be reversed! They will always appear in forward direction in the output of the assembly. Please note: if the backbone sequence consists of a CAF file that contain contigs which contain reversed reads, then the contigs themselves will be in forward direction. But the reads they contain that are in reverse complement direction will of course also stay reverse complement direction.

4. Backbone sequences will not not be assembled together! That is, if a sequence of the backbones has a perfect overlap with another backbone sequence, they will still not be merged.

5. Reads are assembled to backbones in a first come, first served scattering strategy.

   Suppose you have two identical backbones and one read which would match both, then the read would be mapped to the first backbone. If you had two (almost) identical reads, the first read would go to the first backbone, the second read to the second backbone. With three almost identical reads, the first backbone would get two reads, the second backbone one read.

6. Only in backbones loaded from CAF files: contigs made out of single reads (singlets) loose their status as backbones and will be returned to the normal read pool for the assembly process. That is, these sequences will be assembled to other backbones or with each other.

Examples for using backbone sequences:

- Example 1: assume you have a genome of an existing organism. From that, a mutant has been made by mutagenesis and you are skimming the genome in shotgun mode for mutations. You would generate for this a *straindata* file that gives the name of the mutant strain to the newly sequenced reads and simply assemble those against your existing genome, using the following parameters:

  `-SB:lsd=yes:lb=yes:bsn=`*`nameOriginalStrain`*`:bft=`*`caf|fasta|gbf`*

  When loading backbones from CAF, the qualities of the consensus bases will be calculated by mira according normal consensus computing rules. When loading backbones from FASTA or GBF, one can set the expected overall quality of the sequences (e.g. 1 error in 1000 bases = quality of 30) with [-SB:bbq=30]. It is recommended to have the backbone quality at least as high as the [-CO:mgqrt] value, so that mira can automatically detect and report SNPs.

- Example 2: suppose that you are in the process of performing a shotgun sequencing and you want to determine the moment when you got enough reads. One could make a complete assembly each day when new sequences arrive. However, starting with genomes the size of a lower eukaryote, this may become prohibitive from the computational point of view. A quick and efficient way to resolve this problem is to use the CAF file of the previous assembly as backbone and simply add the new reads to the pool. The number of singlets remaining after the assembly versus the total number of reads of the project is a good measure for the coverage of the project.

- Example 3: in EST assembly with miraSearchESTSNPs, existing cDNA sequences can also be useful when added to the project during step 3 (in the file `step3_in.par`). They will provide a framework to which mRNA-contigs built in previous steps will be assembled against, allowing for a fast evaluation of the results. Additionally, they provide a direction for the assembled sequences so that one does not need to invert single contigs by hand afterwards.

4.6 Troubleshooting

(To be expanded)

4.6.1 caf2gap cannot convert the result of a large assembly?

This can have two causes:

1. if you work with a 32 bit executable of caf2gap, it might very well be that the converter needs more memory than can be handled by 32 bit. Only solution: switch to a 64 bit executable of caf2gap.

2. you compiled caf2gap with a caftools version prior to 2.0.1 and then caf2gap throws `segmentation errors`. Simply grab the newest version of the caftools (at least 2.0.2) at ftp://ftp.sanger.ac.uk/pub/PRODUCTION_SOFTWARE/src/ and compile the whole package. caf2gap will be contained therein.

4.6.2 Reverse GenBank features are in forward direction in a gap4 project

caf2gap has currently (as of version 2.0.2) a bug that turns around all features in reverse direction during the conversion from CAF to a gap4 project. There is a fix available, please contact me for further information (until I find time to describe it here).

Chapter 5

Assembly of 454 data with MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'Upset causes changes. Change causes upset.'*
>
> —Solomon Short

5.1 Introduction

MIRA can assemble 454 type data either on its own or together with Sanger or Solexa type sequencing data (true hybrid assembly). Paired-end sequences coming from genomic projects can also be used if you take care to prepare your data the way MIRA needs it.

MIRA goes a long way to assemble sequence in the best possible way: it uses multiple passes, learning in each pass from errors that occurred in the previous passes. There are routines specialised in handling oddities that occur in different sequencing technologies

5.1.1 Some reading requirements

This guide assumes that you have basic working knowledge of Unix systems, know the basic principles of sequencing (and sequence assembly) and what assemblers do.

While there are step by step walkthroughs on how to setup your 454 data and then perform an assembly, this guide expects you to read at some point in time

- the *"Caveats when using 454 data"* section of this document (just below). **This. Is. Important. Read. It!**
- the *mira_usage* introductory help file so that you have a basic knowledge on how to set up projects in mira for Sanger sequencing projects.
- the *GS FLX Data Processing Software Manual* from Roche Diagnostics (or the corresponding manual for the GS20 or Titanium instruments).
- and last but not least the *mira_reference* help file to look up some command line options.

5.1.2 Playing around with some demo data

If you want to jump into action, I suggest you walk through the "Walkthrough: combined unpaired and paired-end assembly of Brucella ceti" section of this document to get a feeling on how things work. That particular walkthrough is with paired and unpaired 454 data from the NCBI short read archive, so be prepared to download a couple of hundred MiBs.

But please do not forget to come back to the "Caveats" section just below later, it contains a pointers to common traps lurking in the depths of high throughput sequencing.

5.2 Caveats when using 454 data

Please take some time to read this section. If you're really eager to jump into action, then feel free to skip forward to the walkthrough, but make sure to come back later.

5.2.1 Screen. Your. Sequences! (part 1)

Or at least use the vector clipping info provided in the SFF file and have them put into a standard NCBI TRACEINFO XML format. Yes, that's right: vector clipping info.

Here's the short story: 454 reads can contain a kind of vector sequence. To be more precise, they can - and very often do - contain the sequence of the (A or B)-adaptors that were used for sequencing.

To quote a competent bioinformatician who thankfully dug through quite some data and patent filings to find out what is going on: "These adaptors consist of a PCR primer, a sequencing primer and a key. The B-adaptor is always in because it's needed for the emPCR and sequencing. If the fragments are long enough, then one usually does not reach the adaptor at all. But if the fragments are too short - tough luck."

Basically it's tough luck for a lot of 454 sequencing project I have seen so far, both for public data (sequences available at the NCBI trace archive) and non-public data.

Tip Use the sff_extract script from Jose Blanca at the University of Valencia. The home of sff_extract is: `http://bioinf.comav.upv.es/sff_extract/index.html` but I am thankful to Jose for giving permission to distribute the script in the MIRA 3rd party package (separate download).

5.2.2 Screen. Your. Sequences! (optional part 2)

Some labs use specially designed tags for their sequencing (I've heard of cases with up to 20 bases). The tag sequences always being very identical, they will behave like vector sequences in an assembly. Like for any other assembler: if you happen to get such a project, then you must take care that those tags are filtered out, respectively masked from your sequences before going into an assembly. If you don't, the results will be messy at best.

Tip Put your FASTAs through SSAHA2 or better, SMALT with the sequence of your tags as masking target. MIRA can read the SSAHA2 output (or SMALT when using "-f ssaha" output) and mask internally using the MIRA [-CL:msvs] parameter and the options pertaining to it.

5.2.3 To right clip or not to right clip?

Sequences coming from the GS20, FLX or Titanium have usually pretty good clip points set by the Roche/454 preprocessing software. There is, however, a tendency to overestimate the quality towards the end of the sequences and declare sequence parts as 'good' which really shouldn't be.

Sometimes, these bad parts toward the end of sequences are so annoyingly bad that they prevent MIRA from correctly building contigs, that is, instead of one contig you might get two.

MIRA has the [-CL:pec] clipping option to deal with these annoyances (standard for all `--job=genome` assemblies). This algorithm performs *proposed end clipping* which will guarantee that the ends of reads are clean when the coverage of a project is high enough.

For genomic sequences: the term 'enough' being somewhat fuzzy ... everything above a coverage of 15x should be no problem at all, coverages above 10x should also be fine. Things start to get tricky below 10x, but give it a try. Below 6x however, switch off the [-CL:pec] option.

5.2.4 Left clipping wrongly preprocessed data

Short intro, to be expanded. (see example in B:ceti walkthrough)

5.3 A 454 assembly walkthrough

5.3.1 Estimating memory needs

"Do I have enough memory?" has been one of the most often asked question in former times. To answer this question, please use miramem which will give you an estimate. Basically, you just need to start the program and answer the questions, for more information please refer to the corresponding section in the main MIRA documentation.

Take this estimate with a grain of salt, depending on the sequences properties, variations in the estimate can be +/- 30%.

Take these estimates even with a larger grain of salt for eukaryotes. Some of them are incredibly repetitive and this leads currently to the explosion of some secondary tables in MIRA. I'm working on it.

5.3.2 Preparing the 454 data for MIRA

The basic data type you will get from the sequencing instruments will be SFF files. Those files contain almost all information needed for an assembly, but they need to be converted into more standard files before mira can use this information.

Let's assume we just sequenced a bug (*Bacillus chocorafoliensis*) and internally our department uses the short *bchoc* mnemonic for your project/organism/whatever. So, whenever you see *bchoc* in the following text, you can replace it by whatever name suits you.

For this example, we will assume that you have created a directory `myProject` for the data of your project and that the SFF files are in there. Doing a `ls -lR` should give you something like this:

```
arcadia:/path/to/myProject$ ls -lR
-rw-rw-rw- 1 bach users 475849664 2007-09-23 10:10 EV10YMP01.sff
-rw-rw-rw- 1 bach users 452630172 2007-09-25 08:59 EV5RTWS01.sff
-rw-rw-rw- 1 bach users 436489612 2007-09-21 08:39 EVX95GF02.sff
```

As you can see, this sequencing project has 3 `SFF` files.

5.3.3 Extracting sequence, quality and clipping info from SFF

The basic data type you will get from the 454 sequencing instruments will be SFF files. Those files contain almost all information needed for an assembly, but SFFs need to be converted into more standard files before MIRA can use this information.

In former times this was done using 3 files (FASTA, FASTA quality and XML), but nowadays the FASTQ format is used almost everywhere, so we will need only two files: FASTQ for sequence + quality and XML for clipping information.

We'll use the sff_extract script to do that. We'll name the output files in a way that makes them immediately suitable for MIRA input.

Note 1: make sure you have Python installed on your system

Note 2: make sure you have the sff_extract script in your path (or use absolute path names)

```
arcadia:/path/to/myProject$ sff_extract
  -Q
  -s bchoc_in.454.fastq
  -x bchoc_traceinfo_in.454.xml
  EV10YMP01.sff EV5RTWS01.sff EVX95GF02.sff
```

Note The above command has been split in multiple lines for better overview but should be entered in one line.

The parameters to **sff_extract** tell it to extract to FASTQ (via -Q), give the FASTQ file a name we chose (via -s), give the XML file with clipping information a name we chose (via -x) and convert the SFFs named `EV10YMP01.sff`, `EV5RTWS01.sff` and `EVX95GF02.sff`.

This can take some time, the 1.2 million FLX reads from this example need approximately 9 minutes for conversion. Your directory should now look something like this:

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users  231698898 2007-10-21 15:16 bchoc_in.454.fastq
-rw-r--r-- 1 bach users  193962260 2007-10-21 15:16 bchoc_traceinfo_in.454.xml
-rw-rw-rw- 1 bach users  475849664 2007-09-23 10:10 EV10YMP01.sff
-rw-rw-rw- 1 bach users  452630172 2007-09-25 08:59 EV5RTWS01.sff
-rw-rw-rw- 1 bach users  436489612 2007-09-21 08:39 EVX95GF02.sff
```

By this time, the SFFs are not needed anymore. You can remove them from this directory if you want.

5.3.4 Starting the assembly

Starting the assembly is now just a matter of one line with some parameters set correctly:

```
arcadia:/path/to/myProject$ mira
--project=bchoc
--job=denovo,genome,accurate,454
>&log_assembly.txt
```

Note The above command has been split in multiple lines for better overview but should be entered in one line.

Now, that was easy, wasn't it? In the above example - for assemblies having only 454 data and if you followed the walkthrough on how to prepare the data - everything you might want to adapt in the first time are the following options:

- --project (for naming your assembly project)
- --job (perhaps to change the quality of the assembly to 'draft'

Of course, you are free to change any option via the extended parameters, but this is covered in the MIRA main reference manual.

5.4 A Sanger / 454 hybrid assembly walkthrough

Preparing the data for a Sanger / 454 hybrid assembly takes some more steps but is not really more complicated than a normal Sanger-only or 454-only assembly.

In the following sections, the example project is named *bchoc_hyb*, simply for us to remember that we did a hybrid assembly there.

Files with 454 input data will have `.454.` in the name, files with Sanger have `.sanger.`.

5.4.1 Preparing the 454 data

Please proceed exactly in the same way as described for the assembly of 454-only data in the section above, that is, without starting the actual assembly.

In the end you should have two files (FASTQ and TRACEINFO) for the 454 data ready.

5.4.2 Preparing the Sanger data

There are quite a number of sequencing providers out there, all with different pre-processing pipelines and different output file-types. MIRA supports quite a number of them, the three most important would probably be

1. (preferred option) FASTQ files and ancillary data in NCBI TRACEINFO XML format.
2. (preferred option) FASTA files which are coupled with FASTA quality files and ancillary data in NCBI TRACEINFO XML format.
3. (preferred option) CAF (from the Sanger Institute) files that contain the sequence, quality values and ancillary data like clippings etc.
4. (secondary option, not recommended) EXP files as the Staden pregap4 package writes.

Your sequencing provider MUST have performed at least a sequencing vector clip on this data. A quality clip might also be good to do by the provider as they usually know best what quality they can expect from their instruments (although MIRA can do this also if you want).

You can either perform clipping the hard way by removing physically all bases from the input (this is called *trimming*), or you can keep the clipped bases in the input file and provided clipping information in ancillary data files. These clipping information then MUST be present in the ancillary data (either the TRACEINFO XML, or in the combined CAF, or in the EXP files), together with other standard data like, e.g., mate-pair information when using a paired-ends approach.

This example assumes that the data is provided as FASTA together with a quality file and ancillary data in NCBI TRACEINFO XML format.

Put these files (appropriately renamed) into the directory with the 454 data.

Here's how the directory with the preprocessed data should now look like (note that we changed the *bchoc* mnemonic to *bchoc_hyb* just for fun ... and to make a distinction to the 454 only assembly above):

```
arcadia:/path/to/myProject$ ls -l
-rwxrwxrwx 1 bach  2007-10-13 22:44 bchoc_hyb_in.454.fastq
-rwxrwxrwx 1 bach  2007-10-13 22:44 bchoc_hyb_traceinfo_in.454.xml

-rwxrwxrwx 1 bach  2007-10-13 22:44 bchoc_hyb_in.sanger.fasta
-rwxrwxrwx 1 bach  2007-10-13 22:44 bchoc_hyb_in.sanger.fasta.qual
-rwxrwxrwx 1 bach  2007-10-13 22:44 bchoc_hyb_traceinfo_in.sanger.xml
```

5.4.3 Starting the hybrid assembly

The following command line starts a basic, but normally quite respectable hybrid 454 and Sanger assembly of a genome where the 454 data has FASTQ + XML TRACEINFO and the Sanger data has FASTA, FASTA quality + XML traceinfo is input type:

```
arcadia:/path/to/myProject$ mira
  --project=bchoc_hyb
  --job=denovo,genome,accurate,sanger,454
  SANGER_SETTINGS -LR:ft=fasta:mxti=yes
  >& log_assembly.txt
```

The only change to starting an assembly with only 454 data was adding "sanger" to the "-job=" command and telling MIRA that the Sanger data needs to be loaded from FASTA (+ quality) and merge ancillary information form the TRACEINFO file.

5.5 Walkthrough: combined unpaired and paired-end assembly of Brucella ceti

Here's a walkthrough which should help you in setting up own assemblies. You do not need to set up your directory structures as I do, but for this walkthrough it could help.

Note This walkthrough was written at times when the NCBI still offered SFFs for 454 data, which now it does not anymore. However, the approach is still valid for your data where you should get SFFs.

Note This walkthrough was written at times when the primary input for 454 data in MIRA was using FASTA + FASTA quality files. This has shifted nowadays to FASTQ as input (it's more compact and faster to parse). I'm sure you will be able to make the necessary changes to the command line of **sff_extract** yourself :-)

5.5.1 Preliminaries

Please make sure that sff_extract is working properly and that you have at least version 0.2.1 (use `sff_extract -v`). Please also make sure that SSAHA2 can be run correctly (test this by running `ssaha2 -v`).

5.5.2 Preparing your file system

Note: this is how I set up a project, feel free to implement whatever structure suits your needs.

```
$ mkdir bceti
$ cd bceti
bceti_assembly$ mkdir origdata data assemblies
```

Your directory should now look like this:

```
arcadia:bceti$ ls -l
drwxr-xr-x 2 bach users 48 2008-11-08 16:51 assemblies
drwxr-xr-x 2 bach users 48 2008-11-08 16:51 data
drwxr-xr-x 2 bach users 48 2008-11-08 16:51 origdata
```

Explanation of the structure:

- the `origdata` directory will contain the 'raw' result files that one might get from sequencing. Basically,.
- the `data` directory will contain the preprocessed sequences for the assembly, ready to be used by MIRA
- the `assemblies` directory will contain assemblies we make with our data (we might want to make more than one).

5.5.3 Getting the data

Note Since early summer 2009, the NCBI does not offer SFF files anymore, which is a pity. This guide will nevertheless allow you to perform similar assemblies on own data.

Please browse to http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?run=SRR005481&cmd=viewer&m=data&s=viewer and http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?run=SRR005482&cmd=viewer&m=data&s=viewer and download the SFF files to the `origdata` directory (press the download button on those pages).

En passant, note the following: SRR005481 is described to be a 454 FLX data set where the library contains unpaired data ("Library Layout: SINGLE"). SRR005482 has also 454 FLX data, but this time it's paired-end data ("Library Layout: PAIRED (ORIENTATION=forward)"). Knowing this will be important later on in the process.

```
arcadia:bceti$ cd origdata
arcadia:origdata$ ls -l
-rw-r--r-- 1 bach users 240204619 2008-11-08 16:49 SRR005481.sff.gz
-rw-r--r-- 1 bach users 211333635 2008-11-08 16:55 SRR005482.sff.gz
```

We need to unzip those files:

```
arcadia:bceti_assembly/origdata$ gunzip *.gz
```

And now this directory should look like this

```
arcadia:bceti_assembly/origdata$ ls -l
-rw-r--r-- 1 bach users 544623256 2008-11-08 16:49 SRR005481.sff
-rw-r--r-- 1 bach users 476632488 2008-11-08 16:55 SRR005482.sff
```

Now move into the (still empty) `data` directory

```
arcadia:origdata$ cd ../data
```

5.5.4 Data preprocessing with sff_extract

5.5.4.1 Extracting unpaired data from SFF

We will first extract the data from the unpaired experiment (SRR005481), the generated file names should all start with *bceti*:

```
arcadia:bceti_assembly/data$ sff_extract -o bceti ../origdata/SRR005481.sff
Working on '../origdata/SRR005481.sff':
Converting '../origdata/SRR005481.sff' ...  done.
Converted 311201 reads into 311201 sequences.

********************************************************************************
WARNING: weird sequences in file ../origdata/SRR005481.sff

After applying left clips, 307639 sequences (=99%) start with these bases:
TCTCCGTC

This does not look sane.

Countermeasures you *probably* must take:
1) Make your sequence provider aware of that problem and ask whether this can be
corrected in the SFF.
2) If you decide that this is not normal and your sequence provider does not
react, use the --min_left_clip of sff_extract.
(Probably '--min_left_clip=13' but you should cross-check that)
********************************************************************************
```

(Note: I got this on the SRR005481 data set downloaded in October 2008. In the mean time, the sequencing center or NCBI may have corrected the error)

Wait a minute ... what happened here?

We launched a pretty standard extraction of reads where the whole sequence were extracted and saved in the FASTA files and FASTA quality files, and clipping information will be given in the XML. Additionally, the clipped parts of every read will be shown in lower case in the FASTA file.

After two or three minutes, the directory looked like this:

```
arcadia:bceti_assembly/data$ ls -l
-rw-r--r-- 1 bach users  91863124 2008-11-08 17:15 bceti.fasta
-rw-r--r-- 1 bach users 264238484 2008-11-08 17:15 bceti.fasta.qual
-rw-r--r-- 1 bach users  52197816 2008-11-08 17:15 bceti.xml
```

5.5.4.2 Dealing with wrong clip-offs in the SFF

In the example above, sff_extract discovered an unusual pattern sequence and gave a (stern) warning: almost all the sequences created for the FASTA file had a skew in the distribution of bases.

Let's have a look at the first 30 bases of the first 20 sequences of the FASTA that was created:

```
arcadia:bceti_assembly/data$ head -40 bceti_in.454.fasta | grep -v ">" | cut -c 0-30
tcagTCTCCGTCGCAATCGCCGCCCCCACA
tcagTCTCCGTCGGCGCTGCCCGCCCGATA
tcagTCTCCGTCGTGGAGGATTACTGGGCG
tcagTCTCCGTCGGCTGTCTGGATCATGAT
tcagTCTCCGTCCTCGCGTTCGATGGTGAC
tcagTCTCCGTCCATCTGTCGGGAACGGAT
tcagTCTCCGTCCGAGCTTCCGATGGCACA
tcagTCTCCGTCAGCCTTTAATGCCGCCGA
tcagTCTCCGTCCTCGAAACCAAGAGCGTG
tcagTCTCCGTCGCAGGCGTTGGCGCGGCG
tcagTCTCCGTCTCAAACAAAGGATTAGAG
tcagTCTCCGTCCTCACCCTGACGGTCGGC
tcagTCTCCGTCTTGTGCGGTTCGATCCGG
tcagTCTCCGTCTGCGGACGGGTATCGCGG
tcagTCTCCGTCTCGTTATGCGCTCGCCAG
tcagTCTCCGTCTCGCATTTTCCAACGCAA
tcagTCTCCGTCCGCTCATATCCTTGTTGA
tcagTCTCCGTCCTGTGCTGGGAAAGCGAA
tcagTCTCCGTCTCGAGCCGGGACAGGCGA
tcagTCTCCGTCGTCGTATCGGGTACGAAC
```

What you see is the following: the leftmost 4 characters `tcag` of every read are the last bases of the standard 454 sequencing adaptor A. The fact that they are given in lower case means that they are clipped away in the SFF (which is good).

However, if you look closely, you will see that there is something peculiar: after the adaptor sequence, all reads seem to start with exactly the same sequence `TCTCCGTC`. This is *not* sane.

This means that the left clip of the reads in the SFF has not been set correctly. The reason for this is probably a wrong value which was used in the 454 data processing pipeline. This seems to be a problem especially when custom sequencing adaptors are used.

In this case, the result is pretty catastrophic: out of the 311201 reads in the SFF, 307639 (98.85%) show this behaviour. We will certainly need to get rid of these first 12 bases.

Now, in cases like these, there are three steps that you really should follow:

1. Is this something that you expect from the experimental setup? If yes, then all is OK and you don't need to take further action. But I suppose that for 99% of all people, these abnormal sequences are not expected.
2. Contact. Your. Sequence. Provider! The underlying problem is something that *MUST* be resolved on their side, not on yours. It might be a simple human mistake, but it it might very well be a symptom of a deeper problem in their quality assurance. Notify. Them. Now!
3. In the mean time (or if the sequencing provider does not react), you can use the [--min_left_clip] command line option from sff_extract as suggested in the warning message.

So, to correct for this error, we will redo the extraction of the sequence from the SFF, this time telling sff_extract to set the left clip starting at base 13 at the lowest:

```
arcadia:bceti_assembly/data$ sff_extract -o bceti --min_left_clip=13 ../origdata/SRR005481. ↩
    sff
Working on '../origdata/SRR005481.sff':
Converting '../origdata/SRR005481.sff' ...  done.
Converted 311201 reads into 311201 sequences.
```

```
arcadia:sff_from_ncbi/bceti_assembly/data$ ls -l
-rw-r--r-- 1 bach users  91863124 2008-11-08 17:31 bceti.fasta
-rw-r--r-- 1 bach users 264238484 2008-11-08 17:31 bceti.fasta.qual
-rw-r--r-- 1 bach users  52509017 2008-11-08 17:31 bceti.xml
```

This concludes the small intermezzo on how to deal with wrong left clips.

5.5.4.3 Extracting paired-end data from SFF

Let's move on to the paired-end data. While I would recommend that, when working on your own data, you should do some kind of data checking, I'll spare you that step for this walkthrough, just believe me that I did it and I found nothing really too suspicious.

The paired-end protocol of 454 will generate reads which contain the forward and reverse direction in one read, separated by a linker. You have to know the linker sequence! Ask your sequencing provider to give it to you. If standard protocols were used, then the linker sequence for GS20 and FLX will be

```
>flxlinker
GTTGGAACCGAAAGGGTTTGAATTCAAACCCTTTCGGTTCCAAC
```

while for Titanium data, you need to use two linker sequences

```
>titlinker1
TCGTATAACTTCGTATAATGTATGCTATACGAAGTTATTACG
>titlinker2
CGTAATAACTTCGTATAGCATACATTATACGAAGTTATACGA
```

In this case, the center apparently used the standard unmodified 454 FLX linker. Put that linker sequence into a FASTA file and copy to wherever you like ... in this walkthrough I put it into the `origdata` directory (not the `data` directory where we currently are.

```
arcadia:bceti_assembly/data$ cp /from/whereever/your/file/is/linker.fasta ../origdata
arcadia:bceti_assembly/data$ ls -l ../origdata
-rw-r--r-- 1 bach users        53 2008-11-08 17:32 linker.fasta
-rw-r--r-- 1 bach users 544623256 2008-11-08 16:49 SRR005481.sff
-rw-r--r-- 1 bach users 476632488 2008-11-08 16:55 SRR005482.sff
arcadia:bceti_assembly/data$ cat ../origdata/linker.fasta
>flxlinker
GTTGGAACCGAAAGGGTTTGAATTCAAACCCTTTCGGTTCCAAC
```

There's one thing that must be found out yet: what was the size of the paired-end library which was constructed, and what is the estimated standard deviation of the sizes? Normally, you will get this information from your sequence provider (if you didn't decide it for yourself). As we're working from a data set deposited at the NCBI, this information should also be available in the accompanying documentation there. But it isn't.

For this walkthrough, we'll simply take a library size of 4500 and an estimated standard deviation of 900.

Now let's extract the paired end sequences, and this may take eight to ten minutes.

```
arcadia:bceti_assembly/data$ sff_extract -o bceti
  -a -l ../origdata/linker.fasta
  -i "insert_size:3000,insert_stdev:900"
  ../origdata/SRR005482.sff

Testing whether SSAHA2 is installed and can be launched ...  ok.
Working on '../origdata/SRR005482.sff':
Creating temporary file from sequences in '../origdata/SRR005482.sff' ... done.
Searching linker sequences with SSAHA2 (this may take a while) ...  ok.
Parsing SSAHA2 result file ...  done.
Converting '../origdata/SRR005482.sff' ...  done.
Converted 268084 reads into 415327 sequences.
```

The above text tells you that the conversion process saw 268084 reads in the SFF. Searching for the paired-end linker and removing it, 415327 sequences were created. Obviously, some sequences had either no linker or the linker was on the far edges of the read so that the 'split' resulted into just one sequences.

The directory will now look like this:

```
arcadia:bceti_assembly/data$ ls -l
-rw-r--r-- 1 bach users 170346423 2008-11-08 17:55 bceti.fasta
-rw-r--r-- 1 bach users 483048864 2008-11-08 17:55 bceti.fasta.qual
-rw-r--r-- 1 bach users 165413112 2008-11-08 17:55 bceti.xml
```

We're almost done. As last step, we will rename the files into a scheme that suits MIRA (we could have used the -s, -q and -x options of sff_extract directly, but I wanted to keep the example straightforward.

```
arcadia:bceti_assembly/data$ mv bceti.fasta bceti_in.454.fasta
arcadia:bceti_assembly/data$ mv bceti.fasta.qual bceti_in.454.fasta.qual
arcadia:bceti_assembly/data$ mv bceti.xml bceti_traceinfo_in.454.xml
arcadia:bceti_assembly/data$ ls -l
-rw-r--r-- 1 bach users 170346423 2008-11-08 17:55 bceti_in.454.fasta
-rw-r--r-- 1 bach users 483048864 2008-11-08 17:55 bceti_in.454.fasta.qual
-rw-r--r-- 1 bach users 165413112 2008-11-08 17:55 bceti_traceinfo_in.454.xml
```

That's it.

5.5.5 Preparing an assembly

Preparing an assembly is now just a matter of setting up a directory and linking the input files into that directory.

```
arcadia:bceti_assembly/data$ cd ../assemblies/
arcadia:bceti_assembly/assemblies$ mkdir arun_08112008
arcadia:bceti_assembly/assemblies$ cd arun_08112008
arcadia:assemblies/arun_08112008$ ln -s ../../data/* .
arcadia:bceti_assembly/assemblies/arun_08112008$ ls -l
lrwxrwxrwx 1 bach users 29 2008-11-08 18:17 bceti_in.454.fasta -> ../../data/bceti_in.454. ↩
    fasta
lrwxrwxrwx 1 bach users 34 2008-11-08 18:17 bceti_in.454.fasta.qual -> ../../data/bceti_in ↩
    .454.fasta.qual
lrwxrwxrwx 1 bach users 33 2008-11-08 18:17 bceti_traceinfo_in.454.xml -> ../../data/ ↩
    bceti_traceinfo_in.454.xml
```

5.5.6 Starting the assembly 2

Start an assembly with the options you like, for example like this:

```
$ mira --project=bceti --job=denovo,genome,accurate,454 >&log_assembly
```

5.6 What to do with the MIRA result files?

Note Please consult the corresponding section in the *mirausage* document, it contains much more information than this stub.

But basically, after the assembly has finished, you will find four directories. The `tmp` directory can be deleted without remorse as it contains logs and some tremendous amount of temporary data (dozens of gigabytes for bigger projects). The `info` directory

has some text files with basic statistics and other informative files. Start by having a look at the `*_info_assembly.txt`, it'll give you a first idea on how the assembly went.

The `results` directory finally contains the assembly files in different formats, ready to be used for further processing with other tools.

If you used the uniform read distribution option, you will inevitably need to filter your results as this option produces larger and better alignments, but also more ``debris contigs''. For this, use the convert_project which is distributed together with the MIRA package.

Also very important when analysing 454 assemblies: screen the small contigs (< 1000 bases) for abnormal behaviour. You wouldn't be the first to have some human DNA contamination in a bacterial sequencing. Or some herpes virus sequence in a bacterial project. Or some bacterial DNA in a human data set. Look whether these small contigs

- have a different GC content than the large contigs
- whether a BLAST of these sequences against some selected databases brings up hits in other organisms that you certainly were not sequencing.

Chapter 6

Assembly of Ion Torrent data with MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'A baby is Life's way of insisting that the universe give it another chance. '

—Solomon Short (modified)

6.1 Introduction

MIRA can assemble Ion Torrent type data either on its own or together with Sanger, 454 or Solexa type sequencing data (true hybrid assembly). Paired-end sequences coming from genomic projects can also be used if you take care to prepare your data the way MIRA needs it.

MIRA goes a long way to assemble sequence in the best possible way: it uses multiple passes, learning in each pass from errors that occurred in the previous passes. There are routines specialised in handling oddities that occur in different sequencing technologies

Warning
Ion Torrent is pretty new and I did not have as much data to analyse as I had with Sanger, 454 or Solexa. MIRA has been configured to automatically work well with data currently available on the market and with data which is to be expected during the course of 2011 / 2012.
However, IonTorrent is - at the moment - a moving target: there are new protocols every few months and it might be that you need to fetch the latest MIRA version available to get the best possible assembly.

6.1.1 Some reading requirements

This guide assumes that you have basic working knowledge of Unix systems, know the basic principles of sequencing (and sequence assembly) and what assemblers do.

While there are step by step walkthroughs on how to setup your Ion Torrent data and then perform an assembly, this guide expects you to read at some point in time

- the *mira_usage* introductory help file so that you have a basic knowledge on how to set up projects in mira for Sanger sequencing projects.
- and last but not least the *mira_reference* help file to look up some command line options.

6.2 Characteristics of Ion Torrent data

What I can say at the moment is that Ion Torrent reads behave very much like the early data from the 454 technology (454 GS20): reads are mostly between 90 and 110 bases long, with Ion Torrent having a showcase with reads of ~220 to 230 bases. The following figure shows what you can get out of 100bp reads if you're lucky:

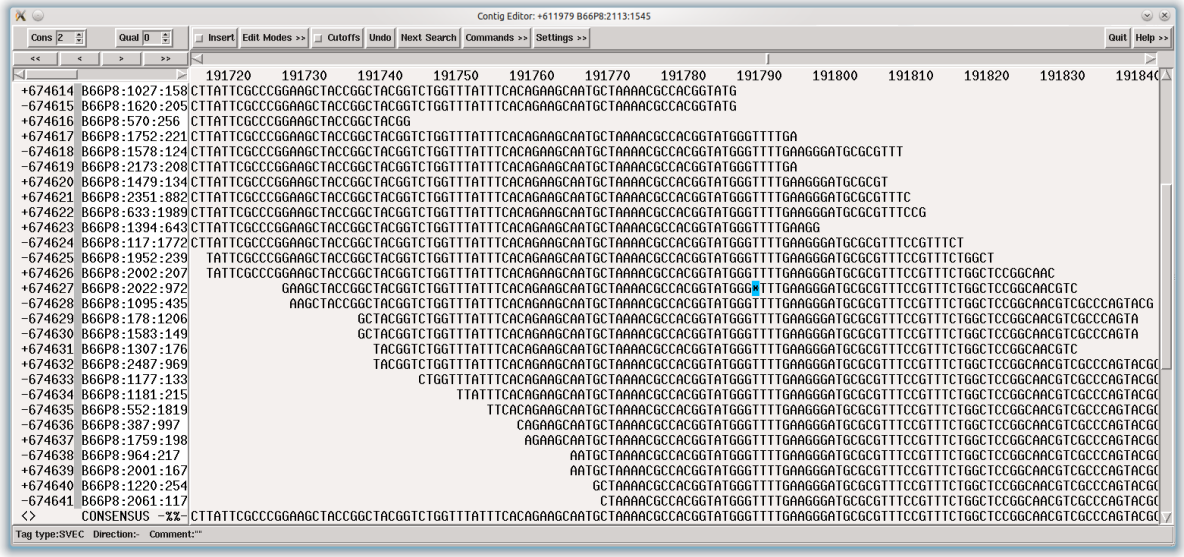

Figure 6.1: Example for good IonTorrent data (100bp reads). Note that only a single sequencing error - shown by blue background - can be seen. Except this, all homopolymers of size 3 and 4 in the area shown are good.

The "if you're lucky" part in the preceding sentence is not there by accident: having so many clean reads is more of an exception rather a rule. On the other hand, most sequencing errors in current IonTorrent data are unproblematic ... if it were not for indels, which is going to be explained on the next sections.

6.2.1 Homopolymer insertions / deletions

The main source of error in your data will be insertions / deletions (indels) especially in homopolymer regions (but not only there, see also next section). Starting with a base run of 4 to 6 bases, there is a distinct tendency to have an increased occurence of indel errors.

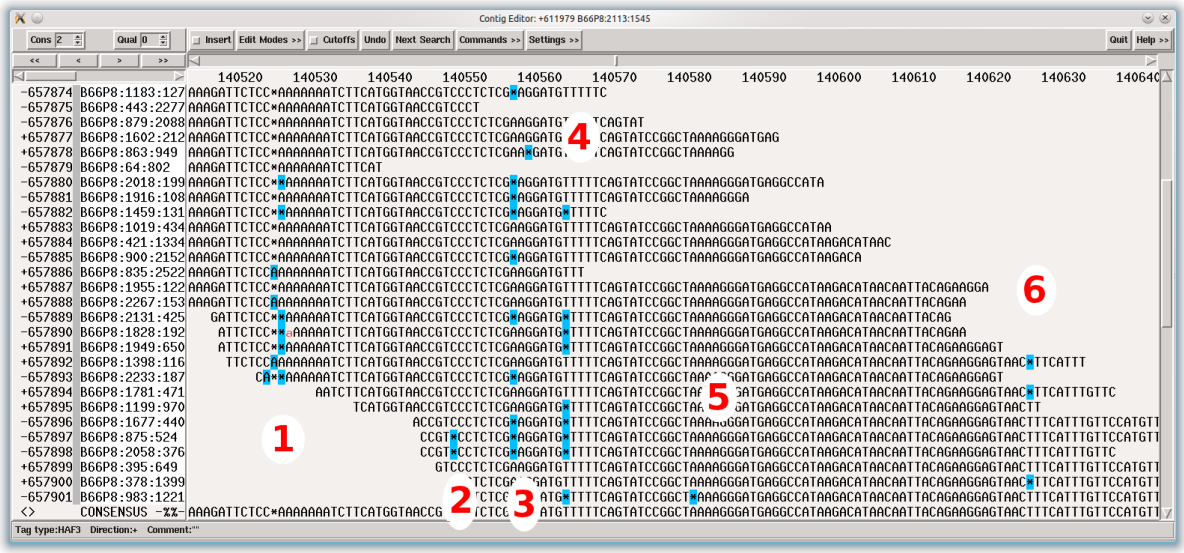

Figure 6.2: Example for problematic IonTorrent data (100bp reads).

The above figure contains a couple of particularly nasty indel problems. While areas 2 (C-homopolymer length 3), 5 (A-homopolymer length 4) and 6 (T-homopolymer length 3) are not a big problem as most of the reads got the length right, the areas 1, 3 and 4 are nasty.

Area 1 is an A-homopolymer of length 7 and while many reads geth that length right (enough to tell MIRA what the true length is), it also contains reads with a length of 6 and and others with a lengh of 8.

Area 2 is a "A-homopolymer" of length 2 where approximately half of the reads get the length right, the other half not. See also the following section.

Area 4 is a T-homopolymer of length 5 which also has approximately half the reads with a wrong length of 4.

6.2.2 Sequencing direction dependend insertions / deletions

In the previous section, the screenshot showing indels had an indel at a homopolymer of 2, which is something quite curious. Upon closer investigation, one might notice a pattern in the gap/nogap distribution: it is almost identical to the orientation of build direction of reads!

I looked for other examples of this behaviour and found quite a number of them, the following figure shows a very clear case of that error behaviour:

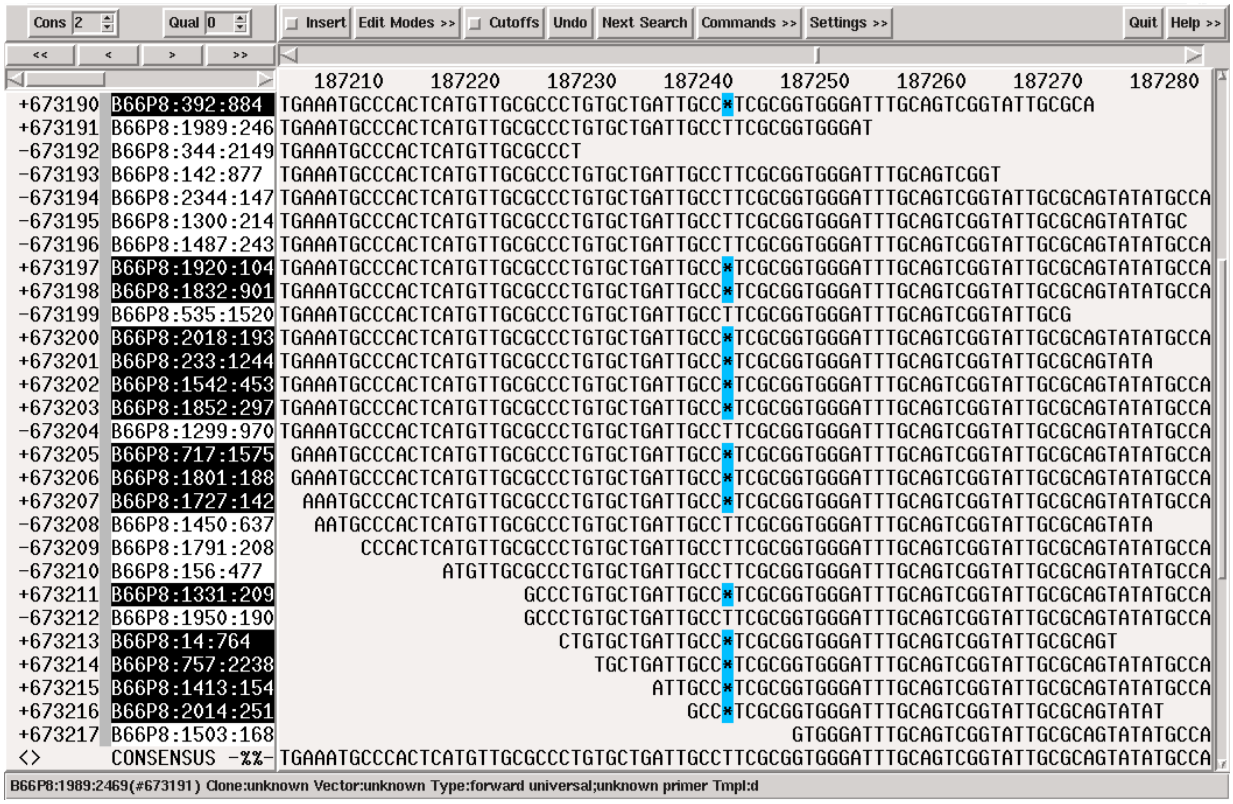

Figure 6.3: Example for a sequencing direction dependend indel. Note how all but one of the reads in '+' direction miss a base while all reads built in in '-' direction have the correct number of bases.

This is quite astonishing: the problem occurs at a site without real homopolymer (calling a 2-bases run a 'homopolymer' starts stretching the definition a bit) and there are no major problematic homopolymer sites near. In fact, this was more or less the case for all sites I had a look at.

Neither did the cases which were investigated show common base patterns, so unlike the Solexa GGCxG motif it does not look like that error of IonTorrent is bound to a particular motif.

While I cannot prove the following statement, I somehow suspect that there must be some kind of secondary structure forming which leads to that kind of sequencing error. If anyone has a good explanation I'd be happy to hear it: feel free to contact me at bach@chevreux.org.

6.2.3 Coverage variance

The coverage variance with the current ~100bp reads is a bit on the bad side for low coverage projects (10x to 15x): it varies wildly, sometimes dropping to nearly zero, sometimes reaching approximately double the coverage.

While showing the same up and down, the effect on an assembly will be less pronounced with higher coverages (25x and more) as the chance increases that some reads are sequenced that span a gap. The following two figures show typical coverage plots for the *E. coli* data (100bp reads, ~33x coverage) published by Ion Torrent.

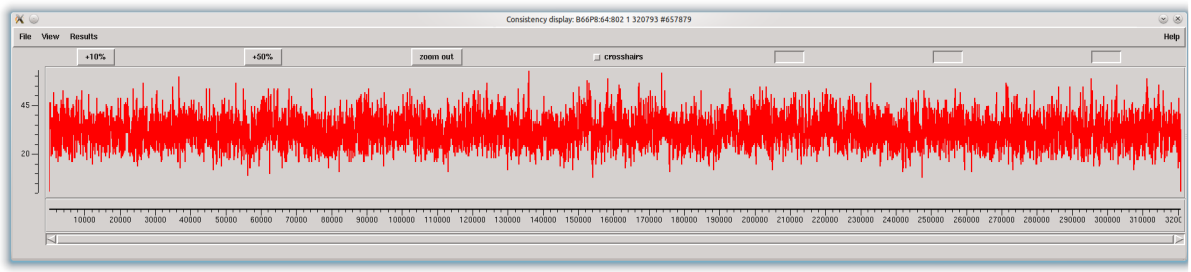

Figure 6.4: IonTorrent coverage (1) 320kb contig

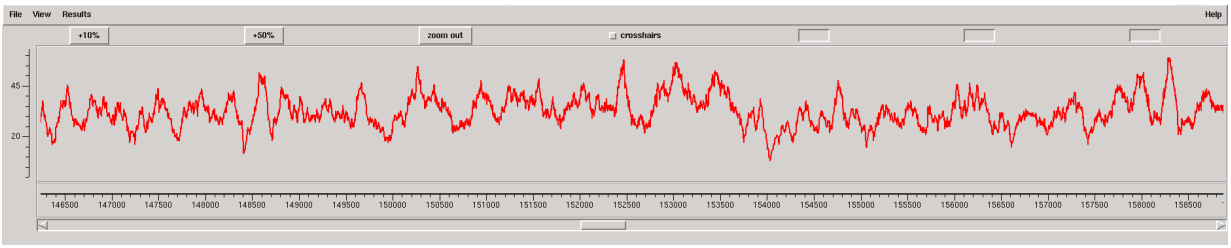

Figure 6.5: IonTorrent coverage (1) zoom of a 12kb stretch

From these figures (and some other data I have) I expect that one would need a coverage of

$\leq$1x for rough bug identification, i.e. answering what it is.

~ 5x for rough pathway exploration, i.e., answering the question which pathways are more or less present (even if one misses a gene or two in different pathways).

~ 12x to 15x for gene fishing expeditions, i.e., get enough sequence to have almost all genes of an organism somehow present, even if some are fragmented into different contigs or contain sequencing errors.

$\geq$ 25x for assemblies which are not too bad

$\geq$ 40x for assemblies which represent the best possible thing you can get with IonTorrent nowadays.

6.2.4 GC bias

The GC bias seems to be small to non-existent, at least I could not immediately make a correlation between GC content and coverage. However, the only data sets I've seen so far are for *E. coli* which has a GC content of rough 50% ... I'd like to check GC bias with a couple of other organisms before giving a final statement.

6.2.5 Other sources of error

You will want to keep an eye on the clipping of the data in the SFF files from IonTorrent: while it is generally good enough, some data sets of IonTorrent show that - for some error patterns - the clipping is too lax and strange artefacts appear. MIRA will take care of these - or at least of those it knows - but you should be aware of this potential problem.

6.2.6 Where to find further information

IonTorrent being pretty new, getting as much information on that technology is quite important. So here are a couple of links I found to be helpful:

- There is, of course, the TorrentDev site (http://lifetech-it.hosted.jivesoftware.com/community/torrent_dev) at Life Technologies which will be helpful to get a couple of questions answered.

  Just be aware that some of the documents over there are sometimes painting an - how should I say it diplomatically? - overly optimistic view on the performance of the technology. On the other hand, so do documents released by the main competitors like 454/Roche, Illumina, PacBio etc. ... so no harm done there.

- I found Nick Loman's blog Pathogens: Genes and Genomes to be my currently most valuable source of information on IonTorrent. While the group he works for won a sequencer from IonTorrent, he makes that fact very clear and still unsparingly dissects the data he gets from that machine.

  His posts got me going in getting MIRA grok IonTorrent.

- The blog of Lex Nederbragt In between lines of code is playing in the same league: very down to earth and he knows a bluff when he sees it ... and is not afraid to call it (be it from IonTorrent, PacBio or 454).

  The analysis he did on a couple of Ion data sets have saved me quite some time.

- Last, but not least, the board with IonTorrent-related-stuff over at SeqAnswers, the first and foremost one-stop-shop ... erm ... discussion board for everything related to sequencing nowadays.

6.3 An Ion Torrent assembly walkthrough

This walkthrough will use two data sets for *E. coli* strain DH10B made available by IonTorrent and will show you the main steps you need to perform to get assemblies going.

6.3.1 Preparing your file system

Note: this is how I set up a project, feel free to implement whatever structure suits your needs.

```
arcadia:$ mkdir dh10b
arcadia:$ cd dh10b
arcadia:dh10b$ mkdir origdata data assemblies
```

Your directory should now look like this:

```
arcadia:dh10b$ ls -l
drwxr-xr-x 2 bach users 48 2011-08-12 22:43 assemblies
drwxr-xr-x 2 bach users 48 2011-08-12 22:43 data
drwxr-xr-x 2 bach users 48 2011-08-12 22:43 origdata
```

Explanation of the structure:

- the `origdata` directory will contain the 'raw' result files that one might get from sequencing. In our case it will be the ZIP files from the IonTorrent site.
- the `data` directory will contain the preprocessed sequences for the assembly, ready to be used by MIRA
- the `assemblies` directory will contain assemblies we make with our data (we might want to make more than one).

6.3.2 Getting the data for this walkthrough

The data sets in question are

1. E. coli DH10B, PGM run B13-328 which you can download from http://lifetech-it.hosted.jivesoftware.com/docs/DOC-1651 (download the SFF). This data set, subsequently nicknamed B13, contains data from the 316 chip with reads of an average size of ~100bp.
2. E. coli DH10B, PGM run B14-387 which you can download from http://lifetech-it.hosted.jivesoftware.com/docs/DOC-1848. This data set, subsequently nicknamed B14, contains data from the 314 chip which IonTorrent uses to show off the "longer reads" capability of its sequencer. "Longer" meaning in this case an average of ~220 bp, which is not bad at all.

Save the two ZIP files into the `origdata` directory should now look like this:

```
arcadia:dh10b$ ls -l origdata
-rw-r--r-- 1 bach bach 824002890 2011-08-15 21:43 B13_328.sff.zip
-rw-r--r-- 1 bach bach 327926296 2011-08-14 20:32 B14_387_CR_0.05.sff.zip
```

Our data is still in ZIP files, let's get them out and put them into the `data` directory:

```
arcadia:dh10b$ cd data
arcadia:data$ unzip ../origdata/B13_328.sff.zip
Archive:  ../origdata/B13_328.sff.zip
  inflating: B13_328.sff
   creating: __MACOSX/
  inflating: __MACOSX/._B13_328.sff
arcadia:data$ unzip ../origdata/B14_387_CR_0.05.sff.zip
Archive:  ../origdata/B14_387_CR_0.05.sff.zip
  inflating: R_2011_07_19_20_05_38_user_B14-387-r121336-314_pool30-ms_B14-387_cafie_0.05. ↩
      sff
arcadia:data$ ls -l
-rw-r--r-- 1 bach bach 1721658336 2011-06-17 01:29 B13_328.sff
drwxrwxr-x 2 bach bach       4096 2011-06-21 16:16 __MACOSX
-rw-rw-r-- 1 bach bach  688207032 2011-07-28 23:31 R_2011_07_19_20_05_38_user_B14-387- ↩
    r121336-314_pool30-ms_B14-387_cafie_0.05.sff
```

Oooops, quite some chaos ... IonTorrent included some unnecessary things (the `__MACOSX` directory) and gave their data files wildly different names. Let's clean up a bit here:

```
arcadia:data$ rm -rf __MACOSX
arcadia:data$ mv B13_328.sff B13.sff
arcadia:data$ mv R_2011_07_19_20_05_38_user_B14-387-r121336-314_pool30-ms_B14-387_cafie_0 ↩
    .05.sff B14.sff
arcadia:data$ ls -l
-rw-r--r-- 1 bach bach 1721658336 2011-06-17 01:29 B13.sff
-rw-rw-r-- 1 bach bach  688207032 2011-07-28 23:31 B14.sff
```

There, much nicer.

6.3.3 Preparing the Ion Torrent data for MIRA

MIRA will need the base sequences, quality values attached to those bases and - if already present - clipping points for quality clips and sequencing adaptor clips.

The basic data type you will get from the sequencing instruments will be SFF files. Those files contain almost all information needed for an assembly, but SFFs need to be converted into more standard files before MIRA can use this information.

In former times this was done using 3 files (FASTA, FASTA quality and XML), but nowadays the FASTQ format is used almost everywhere, so we will need only two files: FASTQ for sequence + quality and XML for clipping information.

Tip Use the sff_extract script from Jose Blanca at the University of Valencia. The home of sff_extract is: http://bioinf.comav.upv.es/sff_extract/index.html but I am thankful to Jose for giving permission to distribute the script in the MIRA 3rd party package (separate download on SourceForge).

The data sets B13 and B14 have short and long IonTorrent reads and we will want to assemble them together, so let's put them together into the input files MIRA needs. For the sake of clarity, I want to name that assembly project *dh10b_b13b14*.

```
arcadia:data$ sff_extract
  -Q
  -s dh10b_b13b14_in.iontor.fastq
  -x dh10b_b13b14_traceinfo_in.iontor.xml
  B13.sff B14.sff
Working on 'B13.sff':
Converting 'B13.sff' ...  done.
Converted 1687490 reads into 1687490 sequences.
Working on 'B14.sff':
Converting 'B14.sff' ...  done.
Converted 350109 reads into 350109 sequences.
```

Note The above command has been split in multiple lines for better overview but should be entered in one line.

The parameters to **sff_extract** tell it to extract to FASTQ (via -Q), give the FASTQ file a name we chose (via -s), give the XML file with clipping information a name we chose (via -x) and convert the SFFs named `B13.sff` and `B14.sff`.

Warning
People "in the know" might want to get rid of the XML TRACEINFO file and tell **sff_extract** to simply dump hard-clip sequences into the FASTQ file via the [-c] argument. Hard-clipped means: the clipped sequence parts of a read are physically trimmed away, never to be seen again.
This is ***DISCOURAGED!***
Reason: unlike 454, IonTorrent actually uses actively the SFF feature to set different clipping points for quality and adaptor clips. This is useful information for MIRA. Furthermore, some of the quality control algorithms of MIRA use also the clipped part of a read to improve assembly quality with measurable effect. If a hard clip was performed on the sequences, these algorithms are not as effective anymore.
For the die hards out there who really do not want the XML TRACEINFO files: if MIRA gets only the sequence, it will use the usual 454/Roche convention to treat left and right lower case part of sequences as clipped and retain the uppercase middle part of a sequence. **sff_extract** adheres to this convention, and while the resulting assemblies are not quite as good as with the TRACEINFO XML, they're still better than with hard clipped sequences.

The conversion can take some time, the ~2 million IonTorrent reads from this example need approximately 2.5 minutes for conversion. Go grab a coffee, or tea, or whatever.

Welcome back. Your directory should now look something like this:

```
arcadia:data$ ls -l
-rw-r--r-- 1 bach bach  771462745 2011-08-19 22:24 dh10b_b13b14_in.iontor.fastq
-rw-r--r-- 1 bach bach  441331004 2011-08-19 22:24 dh10b_b13b14_traceinfo_in.iontor.xml
-rw-r--r-- 1 bach bach 1721658336 2011-06-17 01:29 B13.sff
-rw-rw-r-- 1 bach bach  688207032 2011-07-28 23:31 B14.sff
```

Cool. Last step: we do not need the SFF files anymore, let's get rid of them:

```
arcadia:data$ rm *sff
arcadia:data$ ls -l
-rw-r--r-- 1 bach bach  771462745 2011-08-19 22:24 dh10b_b13b14_in.iontor.fastq
-rw-r--r-- 1 bach bach  441331004 2011-08-19 22:24 dh10b_b13b14_traceinfo_in.iontor.xml
```

6.3.4 Starting the assembly

Good, we're almost there. Let's switch to the `assembly` directory and create a subdirectory for our first assembly test.

```
arcadia:data$ cd ../assemblies/
arcadia:assemblies$ mkdir 1sttest
arcadia:assemblies$ cd 1sttest
```

This directory is quite empty and the IonTorrent data is not present. Let's put some links to the files created in the previous step:

```
arcadia:1sttest$ ln -s ../../data/* .
arcadia:1sttest$ ls -l
lrwxrwxrwx 1 bach bach 42 2011-08-19 22:44 dh10b_b13b14_in.iontor.fastq -> ../../data/ ↩
    dh10b_b13b14_in.iontor.fastq
lrwxrwxrwx 1 bach bach 40 2011-08-19 22:44 dh10b_b13b14_in.iontor.xml -> ../../data/ ↩
    dh10b_b13b14_traceinfo_in.iontor.xml
```

Starting the assembly is now just a matter of one line with some parameters set correctly:

```
arcadia:1sttest$ mira
--project=dh10b_b13b14
--job=denovo,genome,accurate,iontor
>&log_assembly.txt
```

Note The above command has been split in multiple lines for better overview but should be entered in one line.

Now, that was easy, wasn't it? In the above example - for assemblies having only Ion Torrent data and if you followed the walkthrough on how to prepare the data - everything you might want to adapt in the first time are the following options:

- --project (for naming your assembly project)
- --job (perhaps to change the quality of the assembly to 'draft'

Of course, you are free to change any option via the extended parameters, perhaps change the default number of processors to use from 2 to 4 via [-GE:not=4] or any other of the > 150 parameters MIRA has ... but this is covered in the MIRA main reference manual.

6.4 What to do with the MIRA result files?

Note Please consult the corresponding section in the *mirausage* document, it contains much more information than this stub.

But basically, after the assembly has finished, you will find four directories. The `tmp` directory can be deleted without remorse as it contains logs and some tremendous amount of temporary data (dozens of gigabytes for bigger projects). The `info` directory has some text files with basic statistics and other informative files. Start by having a look at the `*_info_assembly.txt`, it'll give you a first idea on how the assembly went.

The `results` directory finally contains the assembly files in different formats, ready to be used for further processing with other tools.

If you used the uniform read distribution option, you will inevitably need to filter your results as this option produces larger and better alignments, but also more "debris contigs". For this, use the convert_project which is distributed together with the MIRA package.

Also very important when analysing Ion Torrent assemblies: screen the small contigs (< 1000 bases) for abnormal behaviour. You wouldn't be the first to have some human DNA contamination in a bacterial sequencing. Or some herpes virus sequence in a bacterial project. Or some bacterial DNA in a human data set. Or ...

Look whether these small contigs

- have a different GC content than the large contigs
- whether a BLAST of these sequences against some selected databases brings up hits in other organisms that you certainly were not sequencing.

Chapter 7

Solexa sequence assembly with MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'There is no such thing like overkill. '*
>
> —Solomon Short

Notes of caution:

1. this guide is still not finished (and may contain old information regarding read lengths in parts), but is should cover most basic use cases.
2. you need lots of memory ... ~ 1 to 1.5 GiB per million Solexa reads. Using mira for anything more than 50 to 100 million Solexa reads is probably not a good idea.

7.1 Introduction

This guide assumes that you have basic working knowledge of Unix systems, know the basic principles of sequencing (and sequence assembly) and what assemblers do.

While there are step by step instructions on how to setup your Solexa data and then perform an assembly, this guide expects you to read at some point in time

- the *MIRA reference manual* file to look up some command line options as well as general information on what tags MIRA uses in assemblies, files it generates etc.pp
- the *short usage introduction* to MIRA3 so that you have a basic knowledge on how to set up projects in mira for Sanger sequencing projects.

7.2 Caveats when assembling Solexa data

Even very short Solexa reads (< 50bp) are great for mapping assemblies. I simply love them as you can easily spot differences in mutant organisms ... or boost the quality of a newly sequenced genome to Q60.

Regarding de-novo assemblies ... well, from an assembler's point of view, very short reads are a catastrophe, regardless of the sequencing technology.

1. Repeats. The problem of repetitive sequences (e.g. rRNA stretches in bacteria) gets worse the shorter the read lengths get.

2. Amount of data. As mira is by heart an assembler to resolve difficult repetitive problems as they occur in Sanger and 454 reads, it drags along quite a lot of ancillary information which is useless in Solexa assemblies ... but still eats away memory

Things look better for the now available 'longer' Solexa reads. Starting with a length of 75bp and paired-end data, de-novo for bacteria is not that bad at all. The first Solexas with a length of ~110 bases are appearing in public, and from a contig building these are about as good for de-novo as the first 454 GS20 reads were.

Here's the rule of thumb I use: the longer, the better. If you have to pay a bit more to get longer reads (e.g. Solexa 100mers instead of 75mers), go get the longer reads. With these, the results you generate are way(!) better than with 36, 50 or even 75mers ... both in mapping and de-novo. Don't try to save a couple of hundred bucks in sequencing, you'll pay dearly afterwards in assembly.

7.3 Typical highlights and lowlights of Solexa sequencing data

Note: This section contains things I've seen in the past and simply jotted down. You may have different observations.

7.3.1 Highlights

7.3.1.1 Quality

For 36mers and the MIRA proposed-end-clipping, even in the old pipeline I get about 90 to 95% reads matching to a reference without a single error. For 72mers, the number is approximately 5% lower, 100mers another 5% less. Still, these are great numbers.

7.3.1.2 Improved base calling pipeline of Illumina

The new base calling pipeline (1.4 or 2.4?) rolled out by Illumina in Q1/Q2 2009 typically yields 20-50% more data from the very same images. Furthermore, the base calling is way better than in the old pipeline. For Solexa 76 mers, after trimming I get only 1% real junk, between 85 and 90% of the reads are matching to a reference without a single error. Of the remaining reads, roughly 50% have one error, 25% have two errors, 12.5% have three errors etc.

It is worthwhile to re-analyse your old data if the images are still around.

7.3.2 Lowlights

7.3.2.1 Long homopolymers

Long homopolymers (stretches of identical bases in reads) can be a slight problem for Solexa. However, it must be noted that this is a problem of all sequencing technologies on the market so far (Sanger, Solexa, 454). Furthermore, the problem in much less pronounced in Solexa than in 454 data: in Solexa, first problem appear may appear in stretches of 9 to 10 bases, in 454 a stretch of 3 to 4 bases may already start being problematic in some reads.

7.3.2.2 The GGCxG and GGC motifs

`GGCxG` or even `GGC` motif in the 5' to 3' direction of reads. This one is particularly annoying and it took me quite a while to circumvent in MIRA the problems it causes.

Simply put: at some places in a genome, base calling after a `GGCxG` or `GGC` motif is particularly error prone, the number of reads without errors declines markedly. Repeated `GGC` motifs worsen the situation. The following screenshots of a mapping assembly illustrate this.

The first example is a the `GGCxG` motif (in form of a `GGCTG`) occuring in approximately one third of the reads at the shown position. Note that all but one read with this problem are in the same (plus) direction.

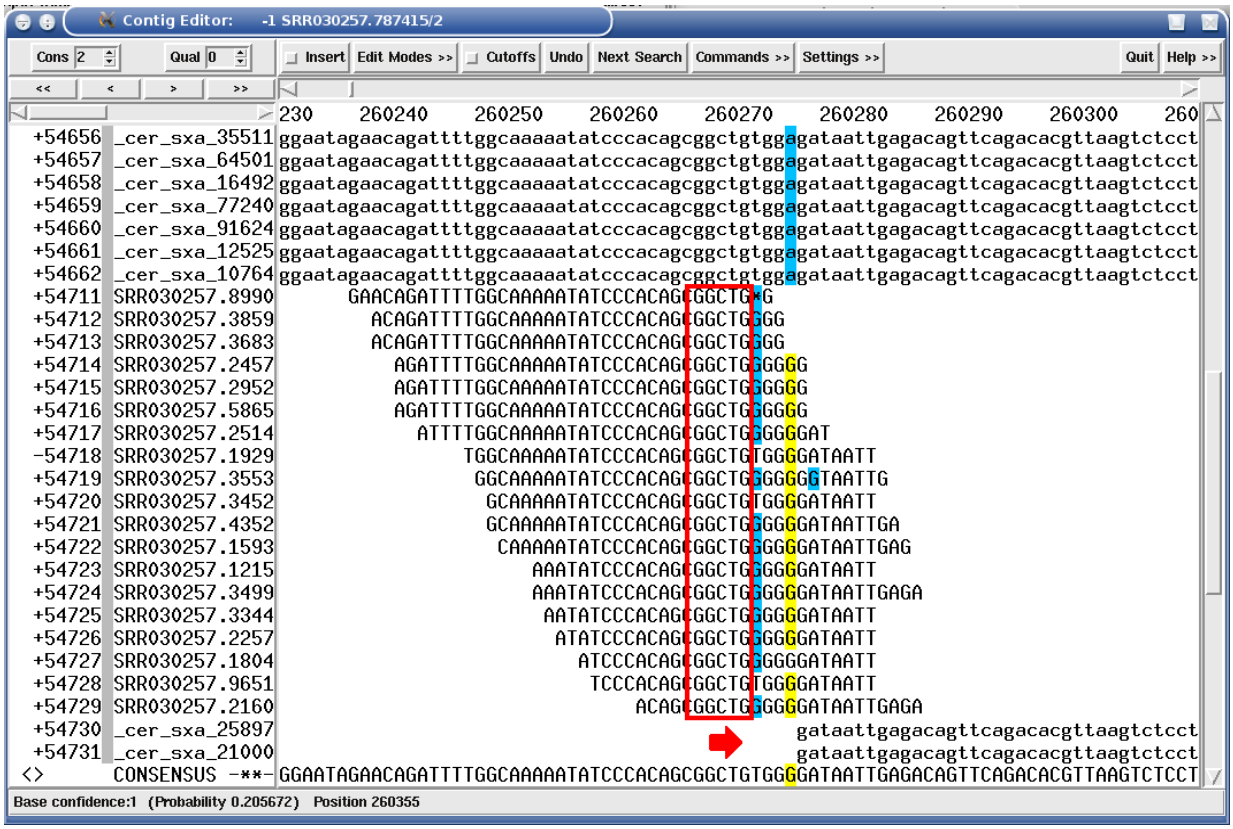

Figure 7.1: The Solexa GGCxG problem.

The next two screenshots show the `GGC`, once for forward direction and one with reverse direction reads:

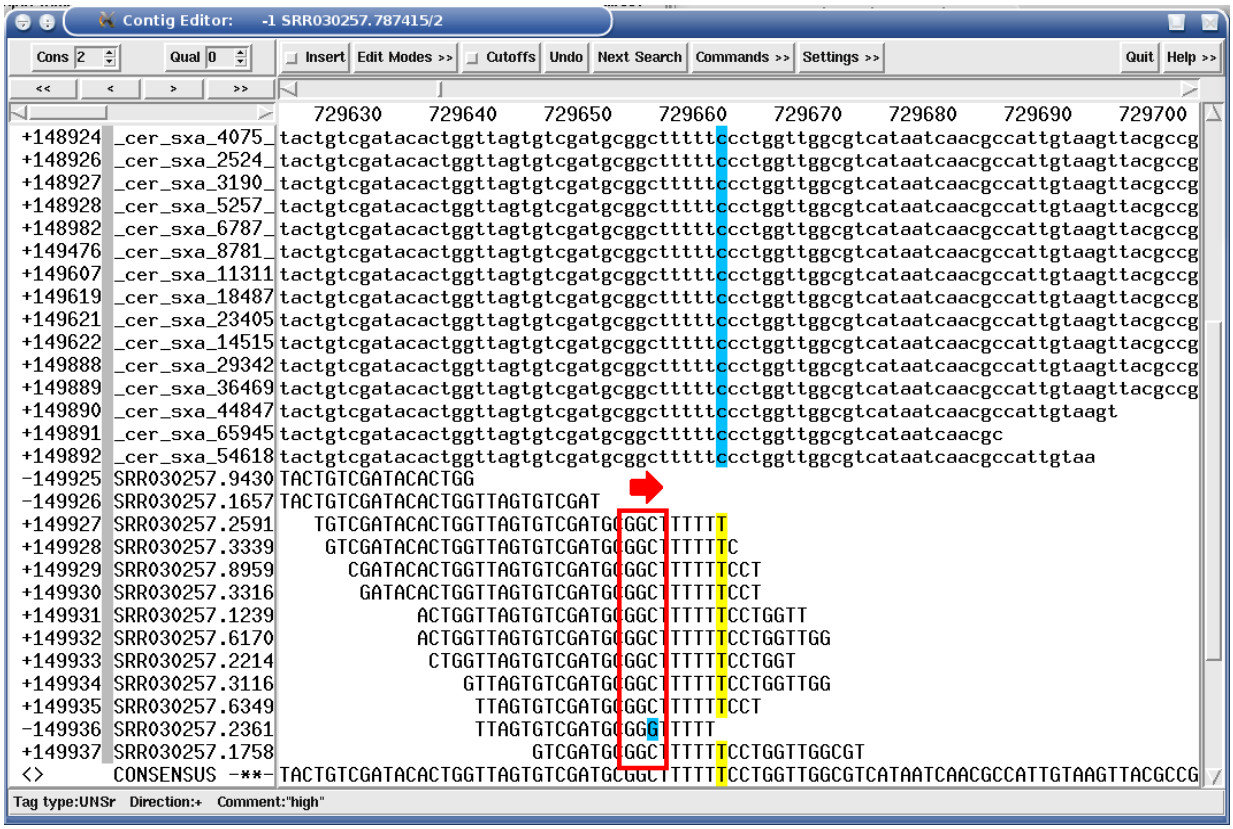

Figure 7.2: The Solexa GGC problem, forward example

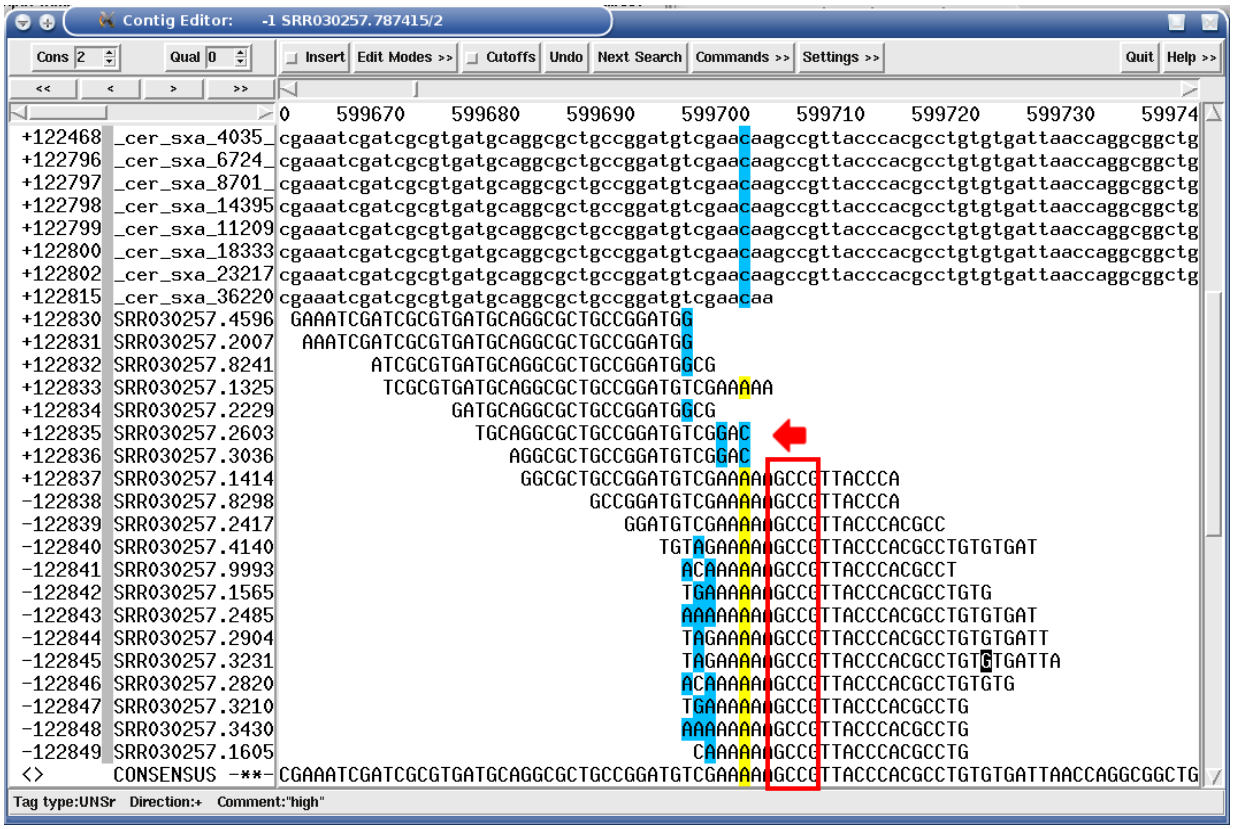

Figure 7.3: The Solexa GGC problem, reverse example

Places in the genome that have `GGCGGC.....GCCGCC` (a motif, perhaps even repeated, then some bases and then an inverted motif) almost always have very, very low number of good reads. Especially when the motif is `GGCxG`.

Things get especially difficult when these motifs occur at sites where users may have a genuine interest. The following example is a screenshot from the Lenski data (see walk-through below) where a simple mapping reveals an anomaly which -- in reality -- is an IS insertion (see http://www.nature.com/nature/journal/v461/n7268/fig_tab/nature08480_F1.html) but could also look like a `GGCxG` motif in forward direction (`GGCCG`) and at the same time a `GGC` motif in reverse direction:

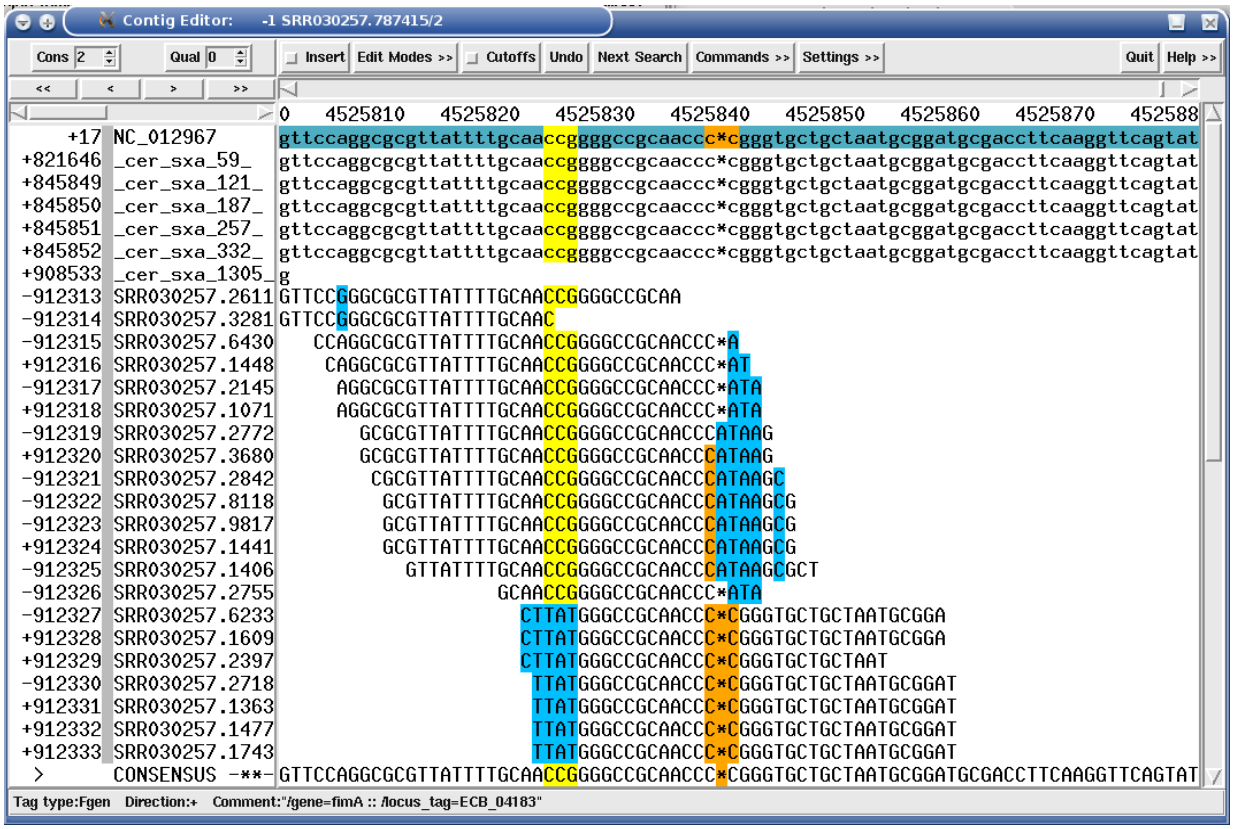

Figure 7.4: A genuine place of interest almost masked by the GGCxG problem.

7.3.2.3 Strong GC bias in some Solexa data (2nd half 2009 until advent of TruSeq kit at end of 2010)

Here I'm recycling a few slides from a couple of talks I held in 2010.

Things used to be so nice and easy with the early Solexa data I worked with (36 and 44mers) in late 2007 / early 2008. When sample taking was done right -- e.g. for bacteria: in stationary phase -- and the sequencing lab did a good job, the read coverage of the genome was almost even. I did see a few papers claiming to see non-trivial GC bias back then, but after having analysed the data I worked with I dismissed them as "not relevant for my use cases." Have a look at the following figure showing exemplarily the coverage of a 45% GC bacterium in 2008:

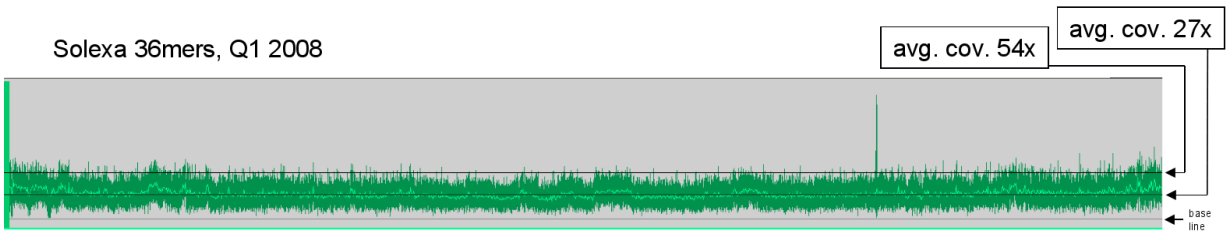

Figure 7.5: Example for no GC coverage bias in 2008 Solexa data. Apart from a slight *smile shape* of the coverage -- indicating the sample taking was not 100% in stationary phase of the bacterial culture -- everything looks pretty nice: the average coverage is at 27x, and when looking at potential genome duplications at twice the coverage (54x), there's nothing apart a single peak (which turned out to be a problem in a rRNA region).

Things changed starting somewhen in Q3 2009, at least that's when I got some data which made me notice a problem. Have a look at the following figure which shows exactly the same organism as in the figure above (bacterium, 45% GC):

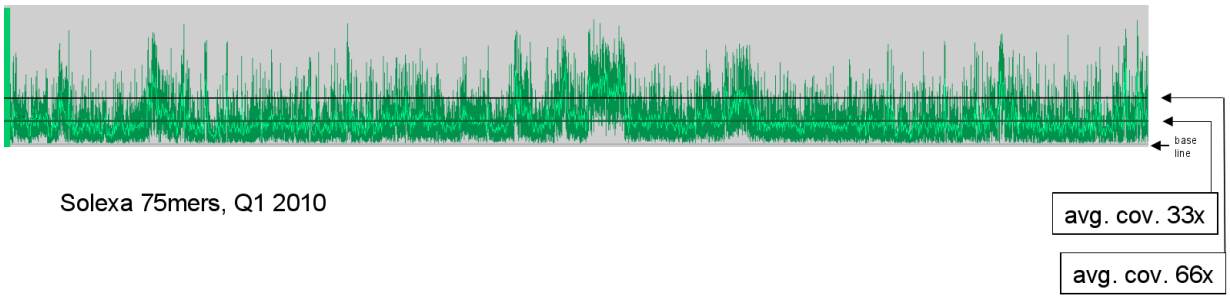

Figure 7.6: Example for GC coverage bias starting Q3 2009 in Solexa data. There's no *smile shape* anymore -- the people in the lab learned to pay attention to sample in 100% stationary phase -- but something else is extremely disconcerting: the average coverage is at 33x, and when looking at potential genome duplications at twice the coverage (66x), there are several dozen peaks crossing the 66x threshold over a several kilobases (in one case over 200 Kb) all over the genome. As if several small genome duplications happened.

By the way, the figures above are just examples: I saw over a dozen sequencing projects in 2008 without GC bias and several dozen in 2009 / 2010 with GC bias.

Checking the potential genome duplication sites, they all looked "clean", i.e., the typical genome insertion markers are missing. Poking around at possible explanations, I looked at GC content of those parts in the genome ... and there was the explanation:

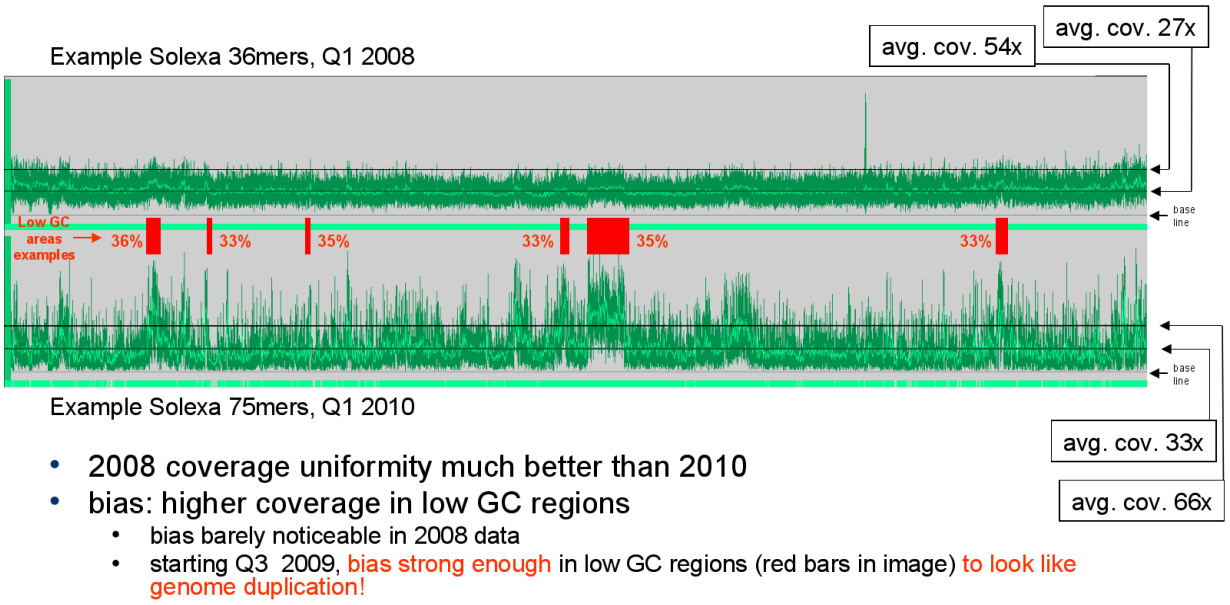

Figure 7.7: Example for GC coverage bias, direct comparison 2008 / 2010 data. The bug has 45% average GC, areas with above average read coverage in 2010 data turn out to be lower GC: around 33 to 36%. The effect is also noticeable in the 2008 data, but barely so.

Now as to actually *why* the GC bias suddenly became so strong is unknown to me. The people in the lab use the same protocol since several years to extract the DNA and the sequencing providers claim to always use the Illumina standard protocols.

But obviously something must have changed. Current ideas about possoble reasons include

- changed chemistries from Illumina leading perhaps to bias during DNA amplification
- changed "standard" protocols

- other ...

It took Illumina some 18 months to resolve that problem for the broader public: since data I work on were done with the TruSeq kit, this problem has vanished.

However, if you based some conclusions or wrote a paper with Illumina data which might be affected by the GC bias (Q3 2009 to Q4 2010), I suggest you rethink all the conclusion drawn. This should be especially the case for transcriptomics experiments where a difference in expression of 2x to 3x starts to get highly significant!

7.4 Mapping assemblies

This part will introduce you step by step how to get your data together for a simple mapping assembly.

I'll make up an example using an imaginary bacterium: *Bacillus chocorafoliensis* (or short: *Bchoc*).

In this example, we assume you have two strains: a wild type strain of *Bchoc_wt* and a mutant which you perhaps got from mutagenesis or other means. Let's imagine that this mutant needs more time to eliminate a given amount of chocolate, so we call the mutant *Bchoc_se* ... SE for **s**low **e**ater

You wanted to know which mutations might be responsible for the observed behaviour. Assume the genome of *Bchoc_wt* is available to you as it was published (or you previously sequenced it), so you resequenced *Bchoc_se* with Solexa to examine mutations.

7.4.1 Copying and naming the sequence data

You need to create (or get from your sequencing provider) the sequencing data in either FASTQ or FASTA + FASTA quality format. The following walkthrough uses what most people nowadays get: FASTQ.

Put the FASTQ data into an empty directory and rename the file so that it looks like this:

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users 263985896 2008-03-28 21:49 bchocse_in.solexa.fastq
```

7.4.2 Copying and naming the reference sequence

The reference sequence (the backbone) can be in a number of different formats: FASTA, GenBank, CAF. The later two have the advantage of being able to carry additional information like, e.g., annotation. In this example, we will use a GenBank file like the ones one can download from the NCBI. So, let's assume that our wild type strain is in the following file: `NC_someNCBInumber.gbk`. Copy this file to the directory (you may also set a link), renaming it as `bchocse_backbone_in.gbf`.

```
arcadia:/path/to/myProject$ cp /somewhere/NC_someNCBInumber.gbk bchocse_backbone_in.gbf
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users   6543511 2008-04-08 23:53 bchocse_backbone_in.gbf
-rw-r--r-- 1 bach users 263985896 2008-03-28 21:49 bchocse_in.solexa.fastq
```

7.4.3 Starting a mapping assembly: unpaired data

Starting the assembly is now just a matter of a simple command line with some parameters set correctly. The following is an example of what I use when mapping onto a reference sequence in GenBank format:

```
arcadia:/path/to/myProject$ mira
  --project=bchocse --job=mapping,genome,accurate,solexa
  -AS:nop=1
  -SB:bsn=bchoc_wt:bft=gbf:bbq=30
  SOLEXA_SETTINGS
  -SB:ads=yes:dsn=bchocse
  >&log_assembly.txt
```

Note 1
The above command has been split in multiple lines for better overview but should be entered in one line.

Note 2
Please look up the parameters used in the main manual. The ones above basically say: make an accurate mapping of Solexa reads against a genome; in one pass; the name of the backbone strain is 'bchoc_wt'; the file type containing backbone is a GenBank file; the base qualities for the backbone are to be assumed Q30; for Solexa data: assign default strain names for reads which have not loaded ancilarry data with strain info and that default strain name should be 'bchocse'.

Note 3
For a bacterial project having a backbone of ~4 megabases and with ~4.5 million Solexa 36mers, MIRA needs some ~21 minutes on my development machine.
A yeast project with a genome of ~20 megabases and ~20 million 36mers needs 3.5 hours and 28 GiB RAM.

For this example - if you followed the walk-through on how to prepare the data - everything you might want to adapt in the first time are the following options:

- -project (for naming your assembly project)
- -SB:bsn to give the backbone strain (your reference strain) another name
- -SB:bft to load the backbone sequence from another file type, say, a FASTA
- -SB:dsn to give a the Solexa reads another strain name

Of course, you are free to change any option via the extended parameters, but this will be the topic of another FAQ.

7.4.4 Assembling with multiple strains

MIRA will make use of ancillary information when present. The **strain name** is such an ancillary information. That is, we can tell MIRA the strain of each read we use in the assembly. In the example above, this information was given on the command line as all the reads to be mapped had the same strain information. But what to do if one wants to map reads from several strains?

We could generate a TRACEINFO XML file with all bells and whistles, but for strain data there's an easier way: the `straindata` file. It's a simple key-value file, one line per entry, with the name of the read as key (first entry in line) and, separated by a blank the name of the strain as value (second entry in line). E.g.:

```
1_1_207_113 strain1
1_1_61_711  strain1
1_1_182_374 strain2
...
2_1_13_654 strain2
...
```

Etcetera. You will obviously replace 'strain1' and 'strain2' with your strain names.

This file can be quickly generated automatically, using the extracted names from FASTQ files and rewritten a little bit. Here's how:

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users 494282343 2008-03-28 22:11 bchocse_in.solexa.fastq

arcadia:/path/to/myProject$ grep "^@" bchocse_in.solexa.fastq
  | sed -e 's/@//'
  | cut -f 1
```

```
  | cut -f 1 -d ' '
  |  sed -e 's/$/ bchocse/'
  > bchocse_straindata_in.txt

arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users 494282343 2008-03-28 22:11 bchocse_in.solexa.fastq
-rw-r--r-- 1 bach users 134822451 2008-03-28 22:13 bchocse_straindata_in.txt
```

Note 1
The above command has been split in multiple lines for better overview but should be entered in one line.

Note 2
for larger files, this can run a minute or two.

Note 3
As you can also assemble sequences from more that one strain, the read names in `bchocse_straindata_in.txt` can have different strain names attached to them. You will then need to generate one straindata file from multiple FASTQ files.

This creates the needed data in the file `bchocse_straindata_in.txt` (well, it's one way to do it, feel free to use whatever suits you best).

7.4.5 Starting a mapping assembly: paired-end data

When using paired-end data, you must decide whether you want

1. use the MIRA feature to create long 'coverage equivalent reads' (CERs) which saves a lot of memory (both in the assembler and later on in an assembly editor). However, you then *loose paired-end information!*
2. or whether you want to *keep paired-end information!* at the expense of larger memory requirements both in MIRA and in assembly editors afterwards.

The Illumina pipeline generally gives you two files for paired-end data: a `project-1.fastq` and `project-2.fastq`. The first file containing the first read of a read-pair, the second file the second read. Depending on the preprocessing pipeline of your sequencing provider, the names of the reads can be either the very same in both files or already have a `/1` or `/2` appended.

Note For running MIRA, you *must* concatenate all sequence input files into one file.

If the read names do not follow the `/1/2` scheme, you must obviously rename them in the process. A ltlle **sed** command can do this automatically for you. Assuming your reads all have the prefix `SRR_something_` the following line appends `/1` to all lines which begin with `@SRR_something_`

```
arcadia:/path/to/myProject$ sed -e 's/^@SRR_something_/&\/1/' input.fastq >output.fastq
```

If you don't care about the paired-end information, you can start the mapping assembly exactly like an assembly for data without paired-end info (see section above).

In case you want to keep the paired-end information, here's the command line (again an example when mapping against a GenBank reference file, assuming that the library insert size is ~500 bases):

```
arcadia:/path/to/myProject$ mira
  --project=bchocse --job=mapping,genome,accurate,solexa
  -AS:nop=1
  -SB:lsd=yes:bsn=bchoc_wt:bft=gbf:bbq=30
  SOLEXA_SETTINGS
  -CO:msr=no -GE:uti=no:tismin=250:tismax=750
  -SB:ads=yes:dsn=bchocse
  >&log_assembly.txt
```

Note 1
For this example to work, make sure that the read pairs are named using the Solexa standard, i.e., having '/1' as postfix to the name of one read and '/2' for the other read. If yours have a different naming scheme, look up the -LR:rns parameter in the main documentation.

Note 2
Please look up the parameters used in the main manual. The ones above basically say: make an accurate mapping of Solexa reads against a genome, in one pass, load additional strain data, the name of the backbone is 'bchoc_wt', the file type containing backbone is a GenBank file, the base qualities for the backbone are to assumed Q30. Additionally, only for Solexa reads, do not merge short reads to the contig, use template size information and set minimum and maximum expected distance to 250 and 750 respectively.

Note 3
You will want to use other values than 250 and 750 if your Solexa paired-end library was not with insert sizes of approximately 500 bases.

Comparing this command line with a command line for unpaired-data, two parameters were added in the section for Solexa data:

1. `-CO:msr=no` tells MIRA not to merge reads that are 100% identical to the backbone. This also allows to keep the template information for the reads.
2. `-GE:uti=no` actually switches *off* checking of template sizes when inserting reads into the backbone. At first glance this might seem counter-intuitive, but it's absolutely necessary to spot, e.g., genome re-arrangements or indels in data analysis after the assembly.

   The reason is that if template size checking were on, the following would happen at, e.g. sites of re-arrangement: MIRA would map the first read of a read-pair without problem. However, it would very probably reject the second read because it would not map at the specified distance from its partner. Therefore, in mapping assemblies with paired-end data, checking of the template size must be switched off.
3. `-GE:tismin:tismax` were set to give the maximum and minimum distance paired-end reads may be away from each other. Though the information is not used by MIRA in the assembly itself, the information is stored in result files and can be used afterwards by analysis programs which search for genome re-arrangements.

Note: for other influencing factors you might want to change depending on size of Solexa reads, see section above on mapping of unpaired data.

7.4.6 Places of interest in a mapping assembly

This section just give a short overview on the tags you might find interesting. For more information, especially on how to configure gap4 or consed, please consult the *mira usage* document and the *mira* manual.

In file types that allow tags (CAF, MAF, ACE), SNPs and other interesting features will be marked by MIRA with a number of tags. The following sections give a brief overview. For a description of what the tags are (SROc, WRMc etc.), please read up the section "Tags used in the assembly by MIRA and EdIt" in the main manual.

Note Screenshots in this section are taken from the walk-through with Lenski data (see below).

7.4.6.1 Where are SNPs?

- the **SROc** tag will point to most SNPs. Should you assemble sequences of more than one strain (I cannot really recommend such a strategy), you also might encounter **SIOc** and **SAOc** tags.

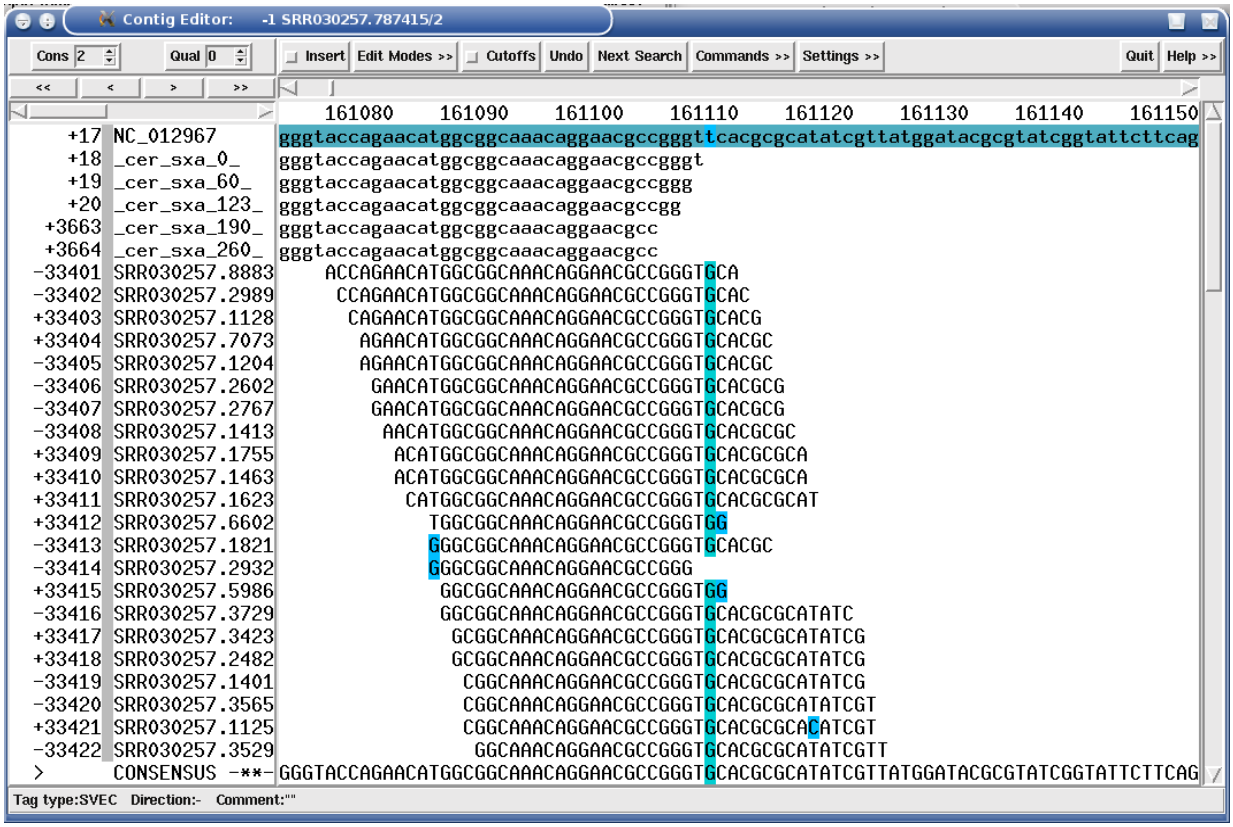

Figure 7.8: "SROc" tag showing a SNP position in a Solexa mapping assembly.

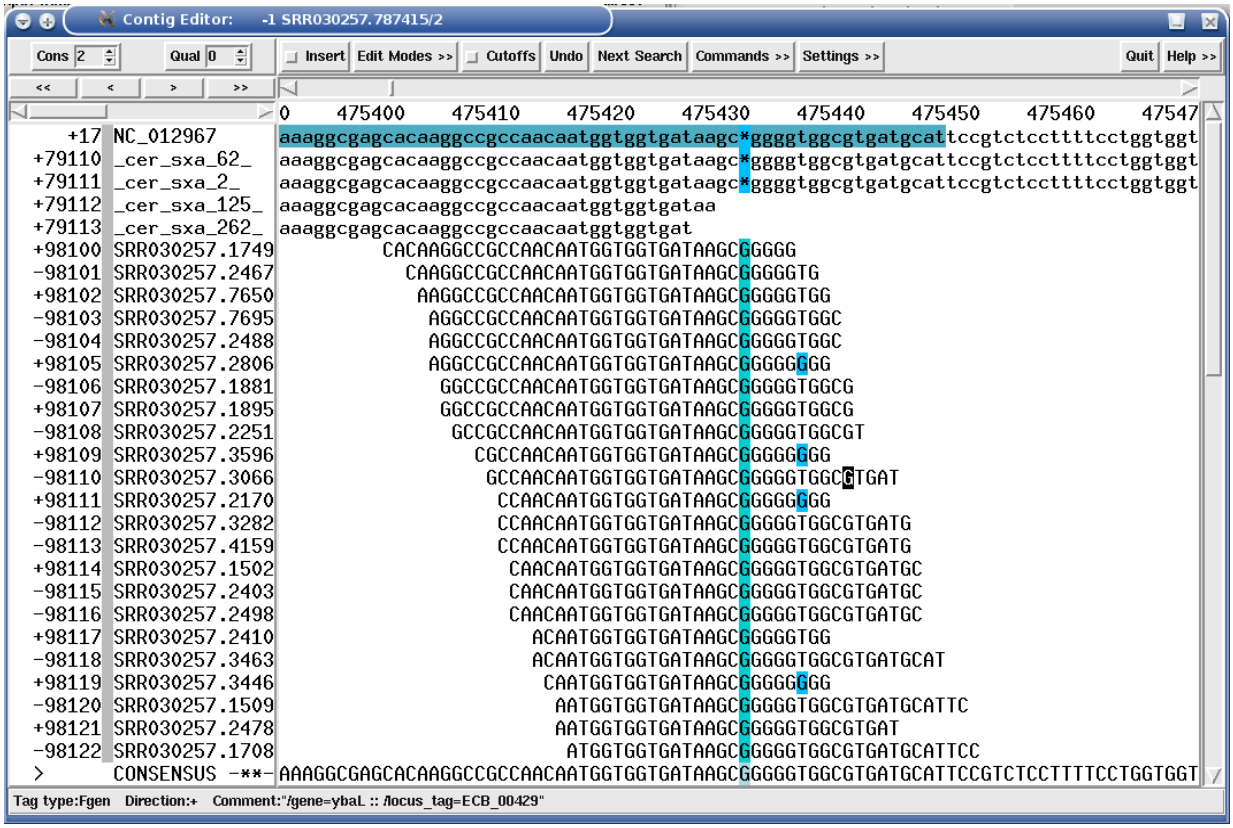

Figure 7.9: "SROc" tag showing a SNP/indel position in a Solexa mapping assembly.

- the **WRMc** tags might sometimes point SNPs to indels of one or two bases.

7.4.6.2 Where are insertions, deletions or genome re-arrangements?

- Large deletions: the **MCVc** tags point to deletions in the resequenced data, where no read is covering the reference genome.

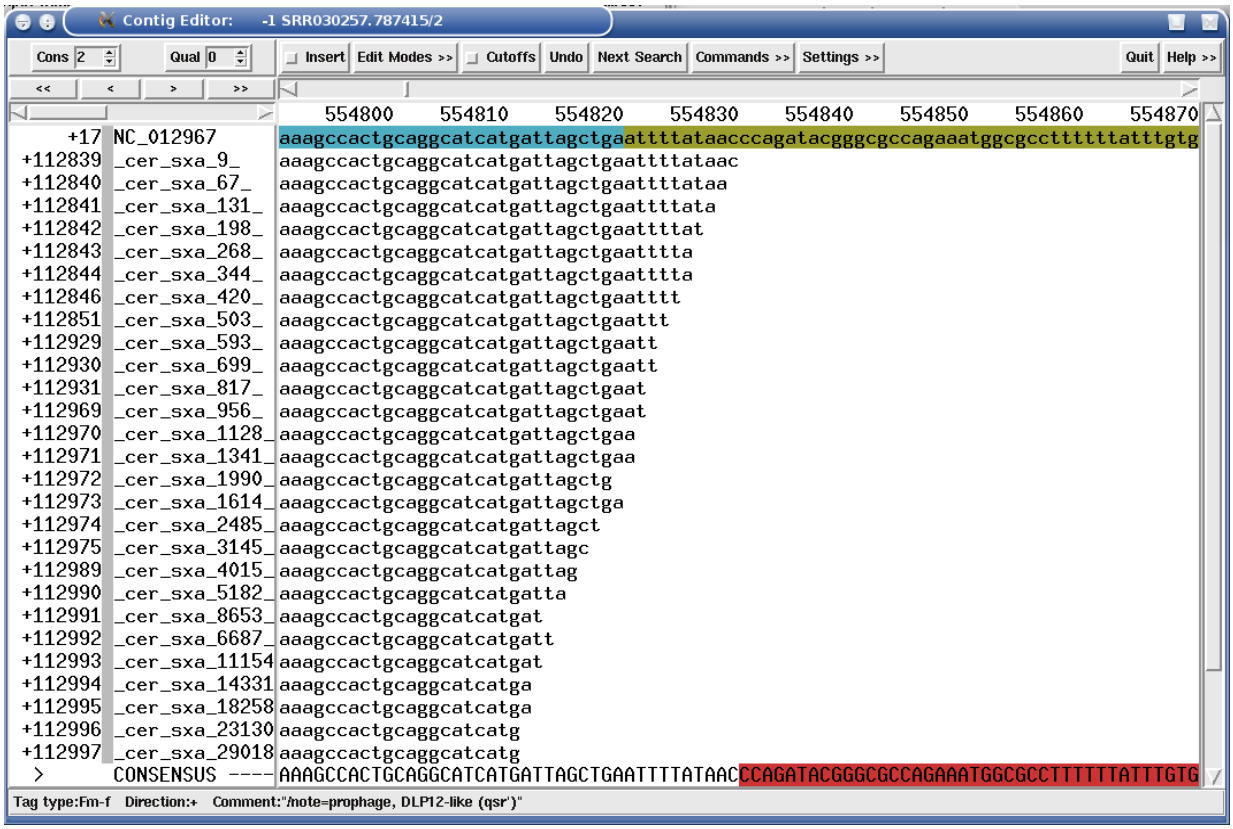

Figure 7.10: "MCVc" tag (dark red stretch in figure) showing a genome deletion in Solexa mapping assembly.

- Insertions, small deletions and re-arrangements: these are harder to spot. In unpaired data sets they can be found looking at clusters of **SROc**, **SRMc**, **WRMc**, and / or **UNSc** tags.

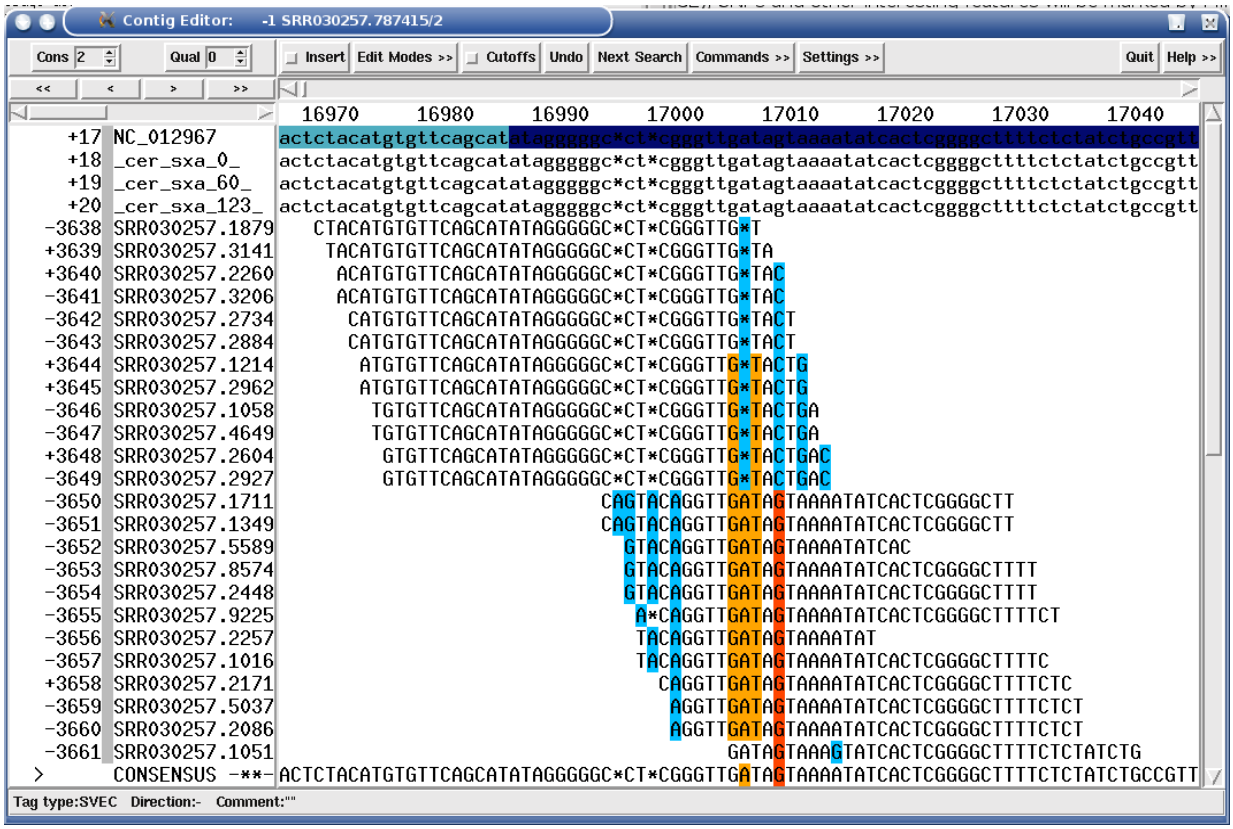

Figure 7.11: An IS150 insertion hiding behind a WRMc and a SRMc tags

more massive occurences of these tags lead to a rather colourful display in finishing programs, which is why these clusters are also sometimes called Xmas-trees.

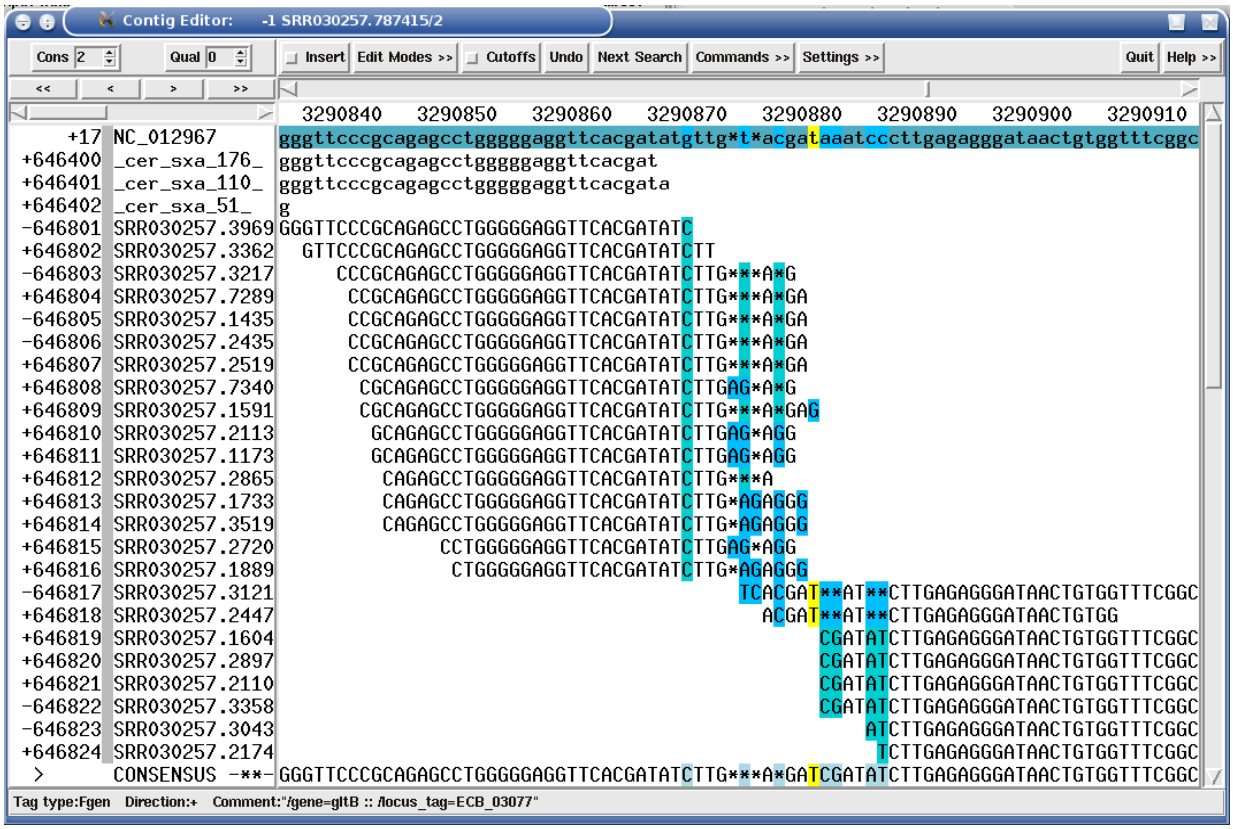

Figure 7.12: A 16 base pair deletion leading to a SROc/UNsC xmas-tree

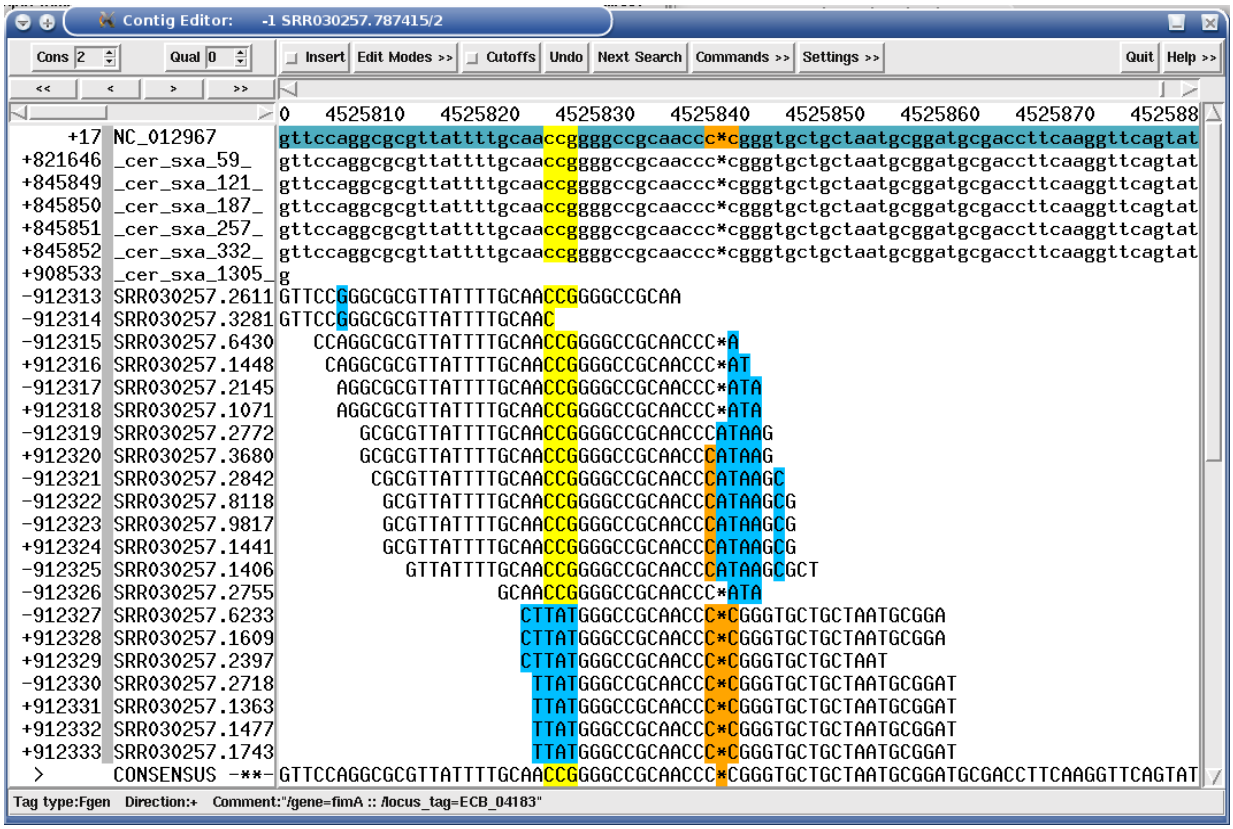

Figure 7.13: An IS186 insertion leading to a SROc/UNsC xmas-tree

In sets with paired-end data, post-processing software (or alignment viewers) can use the read-pair information to guide you to these sites (MIRA doesn't set tags at the moment).

7.4.6.3 Other tags of interest

- the **UNSc** tag points to areas where the consensus algorithm had troubles choosing a base. This happens in low coverage areas, at places of insertions (compared to the reference genome) or sometimes also in places where repeats with a few bases difference are present. Often enough, these tags are in areas with problematic sequences for the Solexa sequencing technology like, e.g., a `GGCxG` or even `GGC` motif in the reads.
- the **SRMc** tag points to places where repeats with a few bases difference are present. Here too, sequence problematic for the Solexa technology are likely to have cause base calling errors and subsequently setting of this tag.

7.4.6.4 Comprehensive spreadsheet tables (for Excel or OOcalc)

Biologists are not really interested in SNPs coordinates, and why should they? They're more interested where SNPs are, how good they are, which genes or other elements they hit, whether they have an effect on a protein sequence, whether they may be important etc. For organisms without intron/exon structure or splice variants, MIRA can generate pretty comprehensive tables and files if an annotated GenBank file was used as reference and strain information was given to MIRA during the assembly.

Well, MIRA does all that automatically for you if the reference sequence you gave was annotated.

For this, **convert_project** should be used with the *asnp* format as target and a CAF file as input:

```
$ convert_project -f caf -t asnp input.caf output
```

Note that it is strongly suggested to perform a quick manual cleanup of the assembly prior to this: for rare cases (mainly at site of small indels of one or two bases), mira will not tag SNPs with a SNP tag (SROc, SAOc or SIOc) but will be fooled into a tag

denoting unsure positions (UNSc). This can be quickly corrected manually. See further down in this manual in the section on post-processing.

After conversion, you will have four files in the directory which you can all drag-and-drop into spreadsheet applications like OpenOffice Calc or Excel.

The files should be pretty self-explanatory, here's just a short overview:

1. `output_info_snplist.txt` is a simple list of the SNPs, with their positions compared to the reference sequence (in bases and map degrees on the genome) as well as the GenBank features they hit.
2. `output_info_featureanalysis.txt` is a much extended version of the list above. It puts the SNPs into context of the features (proteins, genes, RNAs etc.) and gives a nice list, SNP by SNP, what might cause bigger changes in proteins.
3. `output_info_featuresummary.txt` looks at the changes (SNPs, indels) from the other way round. It gives an excellent overview which features (genes, proteins, RNAs, intergenic regions) you should investigate.

   There's one column (named 'interesting') which pretty much summarises up everything you need into three categories: yes, no, and perhaps. 'Yes' is set if indels were detected, an amino acid changed, start or stop codon changed or for SNPs in intergenic regions and RNAs. 'Perhaps' is set for SNPs in proteins that change a codon, but not an amino acid (silent SNPs). 'No' is set if no SNP is hitting a feature.
4. `output_info_featuresequences.txt` simply gives the sequences of each feature of the reference sequence and the resequenced strain.

7.4.6.5 HTML files depicting SNP positions and deletions

I've come to realise that people who don't handle data from NextGen sequencing technologies on a regular basis (e.g., many biologists) don't want to be bothered with learning to handle specialised programs to have a look at their resequenced strains. Be it because they don't have time to learn how to use a new program or because their desktop is not strong enough (CPU, memory) to handle the data sets.

Something even biologist know to operate are browsers. Therefore, convert_project has the option to load a CAF file of a mapping assembly at output to HTML those areas which are interesting to biologists. It uses the tags SROc, SAOc, SIOc and MCVc and outputs the surrounding alignment of these areas together with a nice overview and links to jump from one position to the previous or next.

This is done with the '-t hsnp' option of convert_project:

```
$ convert_project -f caf -t hsnp input.caf output
```

Note: I recommend doing this only if the resequenced strain is a very close relative to the reference genome, else the HTML gets pretty big. But for a couple of hundred SNPs it works great.

7.4.6.6 WIG files depicting contig coverage

convert_project can also dump a coverage file in WIG format (using '-t wig'). This comes pretty handy for searching genome deletions or duplications in programs like the Affymetrix Integrated Genome Browser (IGB, see http://igb.bioviz.org/).

7.4.7 Walkthrough: mapping of E.coli from Lenski lab against E.coli B REL606

We're going to use data published by Richard Lenski in his great paper *"Genome evolution and adaptation in a long-term experiment with Escherichia coli"*. This shows how MIRA finds all mutations between two strains and how one would need just a few minutes to know which genes are affected.

Note All steps described in this walkthrough are present in ready-to-be-run scripts in the `solexa3_lenski` demo directory of the MIRA package.

Note This walkthrough takes a few detours which are not really necessary, but show how things can be done: it reduces the number of reads, it creates a strain data file etc. Actually, the whole demo could be reduced to two steps: downloading the data (naming it correctly) and starting the assembly with a couple of parameters.

7.4.7.1 Getting the data

We'll use the reference genome E.coli B REL606 to map one of the strains from the paper. For mapping, I picked strain REL8593A more or less at random. All the data needed is fortunately at the NCBI, let's go and grab it:

1. the NCBI has REL606 named *NC_012967*. We'll use the RefSeq version and the GenBank formatted file you can download from `ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_B_REL606/NC_012967.gbk`
2. the Solexa re-sequencing data you can get from `ftp://ftp.ncbi.nlm.nih.gov/sra/static/SRX012/SRX012992/` Download both FASTQ files, `SRR030257_1.fastq.gz` and `SRR030257_2.fastq.gz`.

   If you want more info regarding these data sets, have a look at `http://www.ncbi.nlm.nih.gov/sra/?db=sra&term=SRX012992&report=full`

7.4.7.2 Preparing the data for an assembly

In this section we will setup the directory structure for the assembly and pre-process the data so that MIRA can start right away.

Let's start with setting up a directory structure. Remember: you can setup the data almost any way you like, this is just how I do things.

I normally create a project directory with three sub-directories: `origdata`, `data`, and `assemblies`. In `origdata` I put the files exactly as I got them from the sequencing or data provider, without touching them and even removing write permissions to these files so that they cannot be tampered with. After that, I pre-process them and put the pre-processed files into `data`. Pre-processing can be a lot of things, starting from having to re-format the sequences, or renaming them, perhaps also doing clips etc. Finally, I use these pre-processed data in one or more assembly runs in the `assemblies` directory, perhaps trying out different assembly options.

```
arcadia:/some/path/$ mkdir lenskitest
arcadia:/some/path/$ cd lenskitest
arcadia:/some/path/lenskitest$ mkdir data origdata assemblies
arcadia:/some/path/lenskitest$ ls -l
drwxr-xr-x 2 bach bach 4096 2009-12-06 16:06 assemblies
drwxr-xr-x 2 bach bach 4096 2009-12-06 16:06 data
drwxr-xr-x 2 bach bach 4096 2009-12-06 16:06 origdata
```

Now copy the files you just downloaded into the directory `origdata`.

```
arcadia:/some/path/lenskitest$ cp /wherever/the/files/are/SRR030257_1.fastq.gz origdata
arcadia:/some/path/lenskitest$ cp /wherever/the/files/are/SRR030257_2.fastq.gz origdata
arcadia:/some/path/lenskitest$ cp /wherever/the/files/are/NC_012967.gbk origdata
arcadia:/some/path/lenskitest$ ls -l origdata
-rw-r--r-- 1 bach bach  10543139 2009-12-06 16:38 NC_012967.gbk
-rw-r--r-- 1 bach bach 158807975 2009-12-06 15:15 SRR030257_1.fastq.gz
-rw-r--r-- 1 bach bach 157595587 2009-12-06 15:21 SRR030257_2.fastq.gz
```

Great, let's preprocess the data. For this you must know a few things:

- the standard Illumina naming scheme for Solexa paired-end reads is to append forward read names with `/1` and reverse read names with `/2`. The reads are normally put into at least two different files (one for forward, one for reverse). Now, the Solexa data stored in the Short Read Archive at the NCBI also has forward and reverse files for paired-end Solexas. That's OK. What's a bit less good is that the read names there DO NOT have /1 appended to names of forward read, or /2 to names of reverse reads. The forward and reverse reads in both files are just named exactly the same. We'll need to fix that.

- while Sanger and 454 reads should be preprocessed (clipping sequencing vectors, perhaps quality clipping etc.), reads from Solexa present do not. Some people perform quality clipping or clipping of reads with too many 'N's in the sequence, but this is not needed when using MIRA. In fact, MIRA will perform everything needed for Solexa reads itself and will generally do a much better job as the clipping performed is independent of Solexa quality values (which are not always the most trustworthy ones).
- for a mapping assembly, it's good to give the strain name of the backbone and the strain name for the reads mapped against. The former can be done via command line, the later is done for each read individually in a key-value file (the *straindata* file).

So, to pre-process the data, we will need to

- put the reads of the NCBI forward and reverse pairs into one file
- append `/1` to the names of forward reads, and `/2` for reverse reads.
- create a straindata file for MIRA

To ease things for you, I've prepared a small script which will do everything for you: copy and rename the reads as well as creating strain names. Note that it's a small part of a more general script which I use to sometimes sample subsets of large data sets, but for the Lenski data set is small enough so that everything is taken.

Create a file `prepdata.sh` in directory `data` and copy paste the following into it:

```
######################################################################
#######
####### Prepare paired-end Solexa downloaded from NCBI
#######
######################################################################

# srrname:    is the SRR name as downloaded form NCBI SRA
# numreads:   maximum number of forward (and reverse) reads to take from
#              each file. Just to avoid bacterial projects with a coverage
#              of 200 or so.
# strainname: name of the strain which was re-sequenced

srrname="SRR030257"
numreads=5000000
strainname="REL8593A"

################################

numlines=$((4*${numreads}))

# put "/1" Solexa reads into file
echo "Copying ${numreads} reads from _1 (forward reads)"
zcat ../origdata/${srrname}_1.fastq.gz | head -${numlines} | sed -e 's/SRR[0-9.]*/&\/1/' >$ ↩
    {strainname}-${numreads}_in.solexa.fastq

# put "/2" Solexa reads into file
echo "Copying ${numreads} reads from _2 (reverse reads)"
zcat ../origdata/${srrname}_2.fastq.gz | head -${numlines} | sed -e 's/SRR[0-9.]*/&\/2/' >> ↩
    ${strainname}-${numreads}_in.solexa.fastq

# make file with strainnames
echo "Creating file with strain names for copied reads (this may take a while)."
grep "@SRR" ${strainname}-${numreads}_in.solexa.fastq | cut -f 1 -d ' ' | sed -e 's/@//' -e ↩
     "s/$/ ${strainname}/" >>${strainname}-${numreads}_straindata_in.txt
```

Now, let's create the needed data:

```
arcadia:/some/path/lenskitest$ cd data
arcadia:/some/path/lenskitest/data$ ls -l
-rw-r--r-- 1 bach bach       1349 2009-12-06 17:05 prepdata.sh
arcadia:/some/path/lenskitest/data$ sh prepdata.sh
Copying 5000000 reads from _1 (forward reads)
Copying 5000000 reads from _2 (reverse reads)
Creating file with strain names for copied reads (this may take a while).
arcadia:/some/path/lenskitest/data$ ls -l
-rw-r--r-- 1 bach bach       1349 2009-12-06 17:05 prepdata.sh
-rw-r--r-- 1 bach bach 1553532192 2009-12-06 15:36 REL8593A-5000000_in.solexa.fastq
-rw-r--r-- 1 bach bach  218188232 2009-12-06 15:36 REL8593A-5000000_straindata_in.txt
```

Last step, just for the sake of completeness, link in the GenBank formatted file of the reference strain, giving it the same base name so that everything is nicely set up for MIRA.

```
arcadia:/some/path/lenskitest/data$ ln -s ../origdata/NC_012967.gbk REL8593A-5000000 ↩
    _backbone_in.gbf
arcadia:/some/path/lenskitest/data$ ls -l
-rw-r--r-- 1 bach bach       1349 2009-12-06 17:05 prepdata.sh
lrwxrwxrwx 1 bach bach         25 2009-12-06 16:39 REL8593A-5000000_backbone_in.gbf -> ../ ↩
    origdata/NC_012967.gbk
-rw-r--r-- 1 bach bach 1553532192 2009-12-06 15:36 REL8593A-5000000_in.solexa.fastq
-rw-r--r-- 1 bach bach  218188232 2009-12-06 15:36 REL8593A-5000000_straindata_in.txt
arcadia:/some/path/lenskitest/data$ cd ..
arcadia:/some/path/lenskitest$
```

Perfect, we're ready to start assemblies.

7.4.7.3 Starting the mapping assembly

```
arcadia:/some/path/lenskitest$ cd assemblies
arcadia:/some/path/lenskitest/assemblies$ mkdir 1sttest
arcadia:/some/path/lenskitest/assemblies$ lndir ../../data
arcadia:/some/path/lenskitest/assemblies/1sttest$ ls -l
lrwxrwxrwx 1 bach bach         22 2009-12-06 17:18 prepdata.sh -> ../../data/prepdata.sh
lrwxrwxrwx 1 bach bach         43 2009-12-06 16:40 REL8593A-5000000_backbone_in.gbf ->  ↩
    ../../data/REL8593A-5000000_backbone_in.gbf
lrwxrwxrwx 1 bach bach         43 2009-12-06 15:39 REL8593A-5000000_in.solexa.fastq ->  ↩
    ../../data/REL8593A-5000000_in.solexa.fastq
lrwxrwxrwx 1 bach bach         45 2009-12-06 15:39 REL8593A-5000000_straindata_in.txt ->  ↩
    ../../data/REL8593A-5000000_straindata_in.txt
```

Oooops, we don't need the link `prepdata.sh` here, just delete it.

```
arcadia:/some/path/lenskitest/assemblies/1sttest$ rm prepdata.sh
```

Perfect. Now then, start a simple mapping assembly:

```
arcadia:/some/path/lenskitest/assemblies/1sttest$ mira
  --fastq
  --project=REL8593A-5000000
  --job=mapping,genome,accurate,solexa
  -SB:lsd=yes:bsn=ECO_B_REL606:bft=gbf
  >&log_assembly.txt
```

Note 1
The above command has been split in multiple lines for better overview but should be entered in one line. It basically says: load all data in FASTQ format; the project name is *REL8593A-5000000* (and therefore all input and output files will have this prefix by default if not chosen otherwise); we want an accurate mapping of Solexa reads against a genome; load strain data of a separate strain file ([-SB:lsd=yes]); the strain name of the reference sequence is 'ECO_B_REL606' ([-SB:bsn=ECO_B_REL606]) and the file type containing the reference sequence in a GenBank format ([-SB:bft=gbf]). Last but not least, redirect the progress output of the assembler to a file named `log_assembly.txt`.

Note 2
The above assembly takes approximately 35 minutes on my computer (i7 940 with 12 GB RAM) when using 4 threads (I have '-GE:not=4' additionally). It may be faster or slower on your computer.

Note 3
You will need some 10.5 GB RAM to get through this. You might get away with a bit less RAM and using swap, but less than 8 GB RAM is not recommended.

Let's have a look at the directory now:

```
arcadia:/some/path/lenskitest/assemblies/1sttest$ ls -l
-rw-r--r-- 1 bach bach 1463331186 2010-01-27 20:41 log_assembly.txt
drwxr-xr-x 6 bach bach       4096 2010-01-27 20:04 REL8593A-5000000_assembly
lrwxrwxrwx 1 bach bach         43 2009-12-06 16:40 REL8593A-5000000_backbone_in.gbf ->  ↩
    ../../data/REL8593A-5000000_backbone_in.gbf
lrwxrwxrwx 1 bach bach         43 2009-12-06 15:39 REL8593A-5000000_in.solexa.fastq ->  ↩
    ../../data/REL8593A-5000000_in.solexa.fastq
lrwxrwxrwx 1 bach bach         45 2009-12-06 15:39 REL8593A-5000000_straindata_in.txt ->  ↩
    ../../data/REL8593A-5000000_straindata_in.txt
```

Not much which changed. All files created by MIRA will be in the REL8593A-5000000_assembly directory. Going one level down, you'll see 4 sub-directories:

```
arcadia:/some/path/lenskitest/assemblies/1sttest$ cd REL8593A-5000000_assembly
arcadia:.../1sttest/REL8593A-5000000_assembly$ ls -l
drwxr-xr-x 2 bach bach 4096 2010-01-27 20:29 REL8593A-5000000_d_chkpt
drwxr-xr-x 2 bach bach 4096 2010-01-27 20:40 REL8593A-5000000_d_info
drwxr-xr-x 2 bach bach 4096 2010-01-27 20:30 REL8593A-5000000_d_tmp
drwxr-xr-x 2 bach bach 4096 2010-01-27 21:19 REL8593A-5000000_d_results
```

You can safely delete the tmp and the chkpt directories, in this walkthrough they are not needed anymore.

7.4.7.4 Looking at results

Results will be in a sub-directories created by MIRA. Let's go there and have a look.

```
arcadia:/some/path/lenskitest/assemblies/1sttest$ cd REL8593A-5000000_assembly
arcadia:.../1sttest/REL8593A-5000000_assembly$ cd REL8593A-5000000_d_results
arcadia:.../REL8593A-5000000_d_results$ ls -l
-rw-r--r-- 1 bach bach  455087340 2010-01-27 20:40 REL8593A-5000000_out.ace
-rw-r--r-- 1 bach bach  972479972 2010-01-27 20:38 REL8593A-5000000_out.caf
-rw-r--r-- 1 bach bach  569619434 2010-01-27 20:38 REL8593A-5000000_out.maf
-rw-r--r-- 1 bach bach    4708371 2010-01-27 20:39 REL8593A-5000000_out.padded.fasta
-rw-r--r-- 1 bach bach   14125036 2010-01-27 20:39 REL8593A-5000000_out.padded.fasta.qual
-rw-r--r-- 1 bach bach  472618709 2010-01-27 20:39 REL8593A-5000000_out.tcs
-rw-r--r-- 1 bach bach    4707025 2010-01-27 20:39 REL8593A-5000000_out.unpadded.fasta
-rw-r--r-- 1 bach bach   14120999 2010-01-27 20:39 REL8593A-5000000_out.unpadded.fasta.qual
-rw-r--r-- 1 bach bach   13862715 2010-01-27 20:39 REL8593A-5000000_out.wig
```

You can see that MIRA has created output in many different formats suited for a number of different applications. Most commonly known will be ACE and CAF for their use in finishing programs (e.g. gap4 and consed).

In a different directory (the info directory) there are also files containing all sorts of statistics and useful information.

```
arcadia:.../REL8593A-5000000_d_results$ cd ../REL8593A-5000000_d_info/
arcadia:.../REL8593A-5000000_d_info$ ls -l
-rw-r--r-- 1 bach bach     2256 2010-01-27 20:40 REL8593A-5000000_info_assembly.txt
-rw-r--r-- 1 bach bach      124 2010-01-27 20:04 REL8593A-5000000_info_callparameters.txt
-rw-r--r-- 1 bach bach    37513 2010-01-27 20:37 REL8593A-5000000_info_consensustaglist.txt
-rw-r--r-- 1 bach bach 28522692 2010-01-27 20:37 REL8593A-5000000_info_contigreadlist.txt
-rw-r--r-- 1 bach bach      176 2010-01-27 20:37 REL8593A-5000000_info_contigstats.txt
-rw-r--r-- 1 bach bach 15359354 2010-01-27 20:40 REL8593A-5000000_info_debrislist.txt
-rw-r--r-- 1 bach bach 45802751 2010-01-27 20:37 REL8593A-5000000_info_readtaglist.txt
```

Just have a look at them to get a feeling what they show. You'll find more information regarding these files in that main manual of MIRA. At the moment, let's just have a quick assessment of the differences between the Lenski reference strain and the REL8593A train by counting how many SNPs MIRA thinks there are (marked with *SROc* tags in the consensus):

```
arcadia:.../REL8593A-5000000_d_info$ grep -c SROc REL8593A-5000000_info_consensustaglist. ↩
    txt
102
```

102 bases are marked with such a tag. You will later see that this is an overestimation due to several insert sites and deletions, but it's a good first approximation.

Let's count how many potential deletion sites REL8593A has in comparison to the reference strain:

```
arcadia:.../REL8593A-5000000_d_info$ grep -c MCVc REL8593A-5000000_info_consensustaglist. ↩
    txt
48
```

This number too is a slight overestimation due to cross-contamination with sequenced strain which did not have these deletions, but it's also a first approximate.

7.4.7.5 Post-processing with gap4 and re-exporting to MIRA

To have a look at your project in gap4, use the caf2gap program (you can get it at the Sanger Centre), and then gap4:

```
arcadia:.../REL8593A-5000000_d_results$ ls -l
-rw-r--r-- 1 bach bach  455087340 2010-01-27 20:40 REL8593A-5000000_out.ace
-rw-r--r-- 1 bach bach  972479972 2010-01-27 20:38 REL8593A-5000000_out.caf
-rw-r--r-- 1 bach bach  569619434 2010-01-27 20:38 REL8593A-5000000_out.maf
-rw-r--r-- 1 bach bach    4708371 2010-01-27 20:39 REL8593A-5000000_out.padded.fasta
-rw-r--r-- 1 bach bach   14125036 2010-01-27 20:39 REL8593A-5000000_out.padded.fasta.qual
-rw-r--r-- 1 bach bach  472618709 2010-01-27 20:39 REL8593A-5000000_out.tcs
-rw-r--r-- 1 bach bach    4707025 2010-01-27 20:39 REL8593A-5000000_out.unpadded.fasta
-rw-r--r-- 1 bach bach   14120999 2010-01-27 20:39 REL8593A-5000000_out.unpadded.fasta.qual
-rw-r--r-- 1 bach bach   13862715 2010-01-27 20:39 REL8593A-5000000_out.wig
arcadia:.../REL8593A-5000000_d_results$ caf2gap -project REL8593A -ace REL8593A-5000000_out ↩
    .caf >&/dev/null
arcadia:.../REL8593A-5000000_d_results$ ls -l
-rw-r--r-- 1 bach bach 1233494048 2010-01-27 20:43 REL8593A.0
-rw-r--r-- 1 bach bach  233589448 2010-01-27 20:43 REL8593A.0.aux
-rw-r--r-- 1 bach bach  455087340 2010-01-27 20:40 REL8593A-5000000_out.ace
-rw-r--r-- 1 bach bach  972479972 2010-01-27 20:38 REL8593A-5000000_out.caf
-rw-r--r-- 1 bach bach  569619434 2010-01-27 20:38 REL8593A-5000000_out.maf
-rw-r--r-- 1 bach bach    4708371 2010-01-27 20:39 REL8593A-5000000_out.padded.fasta
-rw-r--r-- 1 bach bach   14125036 2010-01-27 20:39 REL8593A-5000000_out.padded.fasta.qual
-rw-r--r-- 1 bach bach  472618709 2010-01-27 20:39 REL8593A-5000000_out.tcs
-rw-r--r-- 1 bach bach    4707025 2010-01-27 20:39 REL8593A-5000000_out.unpadded.fasta
```

```
-rw-r--r-- 1 bach bach   14120999 2010-01-27 20:39 REL8593A-5000000_out.unpadded.fasta.qual
-rw-r--r-- 1 bach bach   13862715 2010-01-27 20:39 REL8593A-5000000_out.wig

arcadia:.../REL8593A-5000000_d_results$ gap4 REL8593A.0
```

Search for the tags set by MIRA which denoted features or problems (SROc, WRMc, MCVc, UNSc, IUPc. See main manual for full list) in the assembly, and edit accordingly. Save your gap4 database as a new version (e.g. REL8593A.1), then exit gap4.

Then use the gap2caf command (also from the Sanger Centre) to convert the gap4 database back to CAF.

```
arcadia:.../REL8593A-5000000_d_results$ gap2caf -project REL8593A.1 >rel8593a_edited.caf
```

As gap4 jumbled the consensus (it does not know different sequencing technologies), having convert_project recalculate the consensus (with the "`-r c`" option) is generally a good idea.

```
arcadia:.../REL8593A-5000000_d_results$ convert_project -f caf -t caf -r c rel8593a_edited. ↩
    caf rel8593a_edited_recalled
```

7.4.7.6 Converting mapping results into HTML and simple spreadsheet tables for biologists

You will have to use either CAF or MAF as input, either of which can be the direct result from the MIRA assembly or an already cleaned and edited file. For the sake of simplicity, we'll use the file created by MIRA in the steps above.

Let's start with a HTML file showing all positions of interest:

```
arcadia:.../REL8593A-5000000_d_results$ convert_project -f caf -t hsnp REL8593A-5000000_out ↩
    .caf rel8593a
arcadia:.../REL8593A-5000000_d_results$ ls -l *html
-rw-r--r-- 1 bach bach 5198791 2010-01-27 20:49 rel8593a_info_snpenvironment.html
```

But MIRA can do even better: create tables ready to be imported in spreadsheet programs.

```
arcadia:.../REL8593A-5000000_d_results$ convert_project -f caf -t asnp REL8593A-5000000_out ↩
    .caf rel8593a
arcadia:.../REL8593A-5000000_d_results$ ls -l rel8593a*
-rw-r--r-- 1 bach bach      25864 2010-01-27 20:48 rel8593a_info_featureanalysis.txt
-rw-r--r-- 1 bach bach   12402905 2010-01-27 20:48 rel8593a_info_featuresequences.txt
-rw-r--r-- 1 bach bach     954473 2010-01-27 20:48 rel8593a_info_featuresummary.txt
-rw-r--r-- 1 bach bach    5198791 2010-01-27 20:49 rel8593a_info_snpenvironment.html
-rw-r--r-- 1 bach bach      13810 2010-01-27 20:47 rel8593a_info_snplist.txt
```

Have a look at all file, perhaps starting with the SNP list, then the feature analysis, then the feature summary (your biologists will love that one, especially when combined with filters in the spreadsheet program) and then the feature sequences.

7.5 De-novo Solexa only assemblies

This is actually quite straightforward if you name your reads according to the MIRA standard for input files. Assume you have the following files (*bchocse* being an example for your mnemonic for the project):

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users 263985896 2008-03-28 21:49 bchocse_in.solexa.fastq
```

7.5.1 Without paired-end

Here's the simplest way to start the assembly:

```
arcadia:/path/to/myProject$ mira
  --project=bchocse
  --job=denovo,genome,accurate,solexa
  >&log_assembly.txt
```

Of course, you can add any other switch you want like, e.g., changing the number of processors used, adding default strain names etc.pp

7.5.2 With paired-end (only one library size)

If you have only one library with one insert size, you just need to tell MIRA this minimum and maximum distance the reads should be away from each other. In the following example I have a library size of 500 bp and have set the minimum and maximum distance to +/- 50% (you might want to use other modifiers):

```
arcadia:/path/to/myProject$ mira
  --project=bchocse
  --job=denovo,genome,accurate,solexa
  SOLEXA_SETTINGS -GE:tismin=250:tismax=750
  >&log_assembly.txt
```

Note
For this example to work, make sure that the read pairs are named using the Solexa standard, i.e., having /1 for one read and /2 for the other read. If yours have a different naming scheme, look up the -LR:rns parameter in the main documentation.

7.5.3 With paired-end (several library sizes)

To tell MIRA exactly which reads have which insert size, one must use an XML file containing ancillary data in NCBI TRACEINFO format. In case you don't have such a file, here's a very simple example containing only insert sizes for reads (lane 1 has a library size of 500 bases and lane 2 a library size of 2 Kb):

```
<?xml version="1.0"?>
<trace_volume>
<trace>
<trace_name>1_17_510_1281/1</trace_name>
<insert_size>500</insert_size>
<insert_stdev>100</insert_stdev>
</trace>
<trace>
<trace_name>1_17_510_1281/2</trace_name>
<insert_size>500</insert_size>
<insert_stdev>100</insert_stdev>
</trace>
...
<trace>
<trace_name>2_17_857_850/1</trace_name>
<insert_size>2000</insert_size>
<insert_stdev>300</insert_stdev>
</trace>
<trace>
<trace_name>2_17_857_850/2</trace_name>
<insert_size>2000</insert_size>
<insert_stdev>300</insert_stdev>
```

```
</trace>
...
</trace_volume>
```

So, if your directory looks like this:

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users 263985896 2008-03-28 21:49 bchocse_in.solexa.fastq
-rw-r--r-- 1 bach users 324987513 2008-04-01 13:24 bchocse_traceinfo_in.solexa.xml
```

then starting the assembly is done like this (note the additional [-LR:mxti] parameter in the section for Solexa setting):

```
arcadia:/path/to/myProject$ mira
  --project=bchocse
  --job=denovo,genome,accurate,solexa
  SOLEXA_SETTINGS -LR:mxti=yes
  >&log_assembly.txt
```

7.6 De-novo hybrid assemblies (Solexa + ...)

Two strategies can be thought of to assemble genomes using a combination of Solexa and other (longer) reads: either using all reads for a full de-novo assembly or first assembling the longer reads and use the resulting assembly as backbone to map Solexa reads. Both strategies have their pro and cons.

7.6.1 All reads de-novo

Throwing all reads into a de-novo assembly is the most straightforward way to get 'good' assemblies. This strategy is also the one which - in most cases - yields the longest contigs as, in many projects, parts of a genome not covered by one sequencing technology will probably be covered by another sequencing technology. Furthermore, having the consensus covered by more than one sequencing technology make base calling a pretty robust thing: if MIRA finds disagreements it cannot resolve easily, the assembler at least leaves a tag in the assembly to point human finishers to these positions of interest.

The downside of this approach however is the fact that the sheer amount of data in Solexa sequencing projects makes life difficult for de-novo assemblers, especially for MIRA which is keeping quite some additional information in memory in de-novo assemblies and tries to use algorithms as exact as possible during contig construction. Therefore, MIRA sometimes still runs into data sets which make it behave quite badly with respect to assembly time and memory consumption (but this is being constantly improved).

Full de-novo hybrid assemblies can be recommended only for bacteria at the moment, although lower eukaryotes should also be feasible on larger machines.

7.6.1.1 Starting the assembly

Starting the assembly is now just a matter of a simple command line with some parameters set correctly. The following is a de-novo hybrid assembly with 454 and Solexa reads.

```
arcadia:/path/to/myProject$ mira
  --project=bchocse --job=denovo,genome,normal,454,solexa
  >&log_assembly.txt
```

7.6.2 Long reads first, then Solexa

This strategy works in two steps: first assembling long reads, then mapping short reads to the full alignment (not just a consensus sequence). The result will be an assembly containing 454 (or Sanger) and Solexa reads.

7.6.2.1 Step 1: assemble the 'long' reads (454 or Sanger or both)

Assemble your data just as you would when assembling 454 or Sanger data.

7.6.2.2 Step 2: filter the results

This step fetches 'long' contigs from the assembly before. Idea is to get all contigs larger than 500 bases.

```
$ convert_project -f caf -t caf -x 500 assemblyresult.caf hybrid_backbone_in.caf
```

You might eventually want to add an additional filter for minimum average coverage. If your project has an average coverage of 24, you should filter for a minimum average coverage of 33% (coverage 8, you might want to try out higher coverages) like this:

```
$ convert_project -f caf -t caf -x 500 -y 8 assemblyresult.caf hybrid_backbone_in.caf
```

7.6.2.3 Step 3: map the Solexa data

Copy the hybrid backbone to a new empty directory, add in the Solexa data, start a mapping assembly using the CAF as input for the backbone. If you assembled the 454 / Sanger data with strain info, the Solexa data should also get those (as described above).

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach bach 1159280980 2009-10-31 19:46 hybrid_backbone_in.caf
-rw-r--r-- 1 bach bach  338430282 2009-10-31 20:31 hybrid_in.solexa.fastq
arcadia:/path/to/myProject$ mira
  --project=hybrid --job=mapping,genome,accurate,solexa
  -AS:nop=1
  -SB:bft=caf
  >&log_assembly.txt
```

7.7 Post-processing of assemblies

This section is a bit terse, you should also read the chapter on *working with results* of MIRA3.

7.7.1 Post-processing mapping assemblies

When working with resequencing data and a mapping assembly, I always load finished projects into an assembly editor and perform a quick cleanup of the results.

For close relatives of the reference strain this doesn't take long as MIRA will have set tags (see section earlier in this document) at all sites you should have a look at. For example, very close mutant bacteria with just SNPs or simple deletions and no genome reorganisation, I usually clean up in 10 to 15 minutes. That gives the last boost to data quality and your users (biologists etc.) will thank you for that as it reduces their work in analysing the data (be it looking at data or performing wet-lab experiments).

Assume you have the following result files in the result directory of a MIRA assembly:

```
arcadia:/path/to/myProject/newstrain_d_results$ ls -l
-rw-r--r-- 1 bach bach 312607561 2009-06-08 14:57 newstrain_out.ace
-rw-r--r-- 1 bach bach 655176303 2009-06-08 14:56 newstrain_out.caf
...
```

The general workflow I use is to convert the CAF file to a gap4 database and start the gap4 editor:

```
arcadia:newstrain_d_results$ caf2gap -project NEWSTRAIN -ace newstrain_out.caf >& /dev/null
arcadia:newstrain_d_results$ gap4 NEWSTRAIN.0
```

Then, in gap4, I

1. quickly search for the UNSc and WRMc tags and check whether they could be real SNPs that were overseen by MIRA. In that case, I manually set a SROc (or SIOc) tag in gap4 via hotkeys that were defined to set these tags.
2. sometimes also quickly clean up reads that are causing trouble in alignments and lead to wrong base calling. These can be found at sites with UNSc tags, most of the time they have the 5' to 3' `GGCxG` motif which can cause trouble to Solexa.
3. look at sites with deletions (tagged with MCVc) and look whether I should clean up the borders of the deletion.

After this, I convert the gap4 database back to CAF format:

```
$ gap2caf -project NEWSTRAIN >newstrain_edited.caf
```

But beware: gap4 does not have the same consensus calling routines as MIRA and will have saved it's own consensus in the new CAF. In fact, gap4 performs rather badly in projects with multiple sequencing technologies. So I use convert_project from the MIRA package to recall a good consensus (and save it in MAF as it's more compact and a lot faster in handling than CAF):

```
$ convert_project -f caf -t maf -r c newstrain_edited.caf newstrain_edited_recalled
```

And from this file I can then convert with convert_project to any other format I or my users need: CAF, FASTA, ACE, WIG (for coverage analysis) etc.pp.

I can also also generate tables and HTML files with SNP analysis results (with the "`-t asnp`" and "`-t hsnp`" options of **convert_project**)

7.7.2 Post-processing de-novo assemblies

As the result file of MIRA de-novo assemblies contains everything down to 'contigs' with just two reads, it is advised to first filter out all contigs which are smaller than a given size or have a coverage lower than 1/3 to 1/2 of the overall coverage.

Filtering is performed by convert_project using CAF file as input. Assume you have the following file:

```
arcadia:/path/to/myProject/newstrain_d_results$ ls -l
...
-rw-r--r-- 1 bach bach 655176303 2009-06-08 14:56 newstrain_out.caf
...
```

Let's say you have a hybrid assembly with an average coverage of 50x. I normally filter out all contigs which have an average coverage less than 1/3 and are smaller than 500 bases. These are mostly junk contiglets remaining from the assembly and can be more or less safely ignored. This is done the following way:

```
arcadia:newstrain_d_results$ convert_project
  -f caf -t caf -x 500 -y 17 newstrain_out.caf newstrain_filterx500y17
```

From there on, convert the filtered CAF file to anything you need to continue finishing of the genome (gap4 database, ACE, etc.pp).

7.8 Known bugs / problems

These are actual for version 3 of MIRA and might or might not have been addressed in later version.

Bugs:

1. mapping of paired-end reads with one read being in non-repetitive area and the other in a repeat is not as effective as it should be. The optimal strategy to use would be to map first the non-repetitive read and then the read in the repeat. Unfortunately, this is not yet implemented in MIRA.

Problems:

1. the textual output of results is really slow with such massive amounts of data as with Solexa projects. If Solexa data is present, it's turned off by default at the moment.

Chapter 8

Assembling sequences from Pacific Biosciences with MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'New problems demand new solutions. New solutions create new problems. '*
>
> —Solomon Short

8.1 MIRA 3.4.0: currently only CCS or error-corrected-CLR supported

This is here to set the stage: at the moment, MIRA can only make use of PacBio reads which have an error rate of roundabout 5%. This means you will have to use either:

1. CCS (Circular Consensus Sequence) reads with at least 3 to 4 passes.
2. CLR (Continuous Long Reads) which were error corrected either with PacBio CCS reads or some other high-quality sequencing technology (Illumina comes to mind)

Impatients can directly jump to Section 8.6 which contains walkthroughs using data made publicly available by PacBio.

8.2 WARNING

When I developed the routines for PacBio, I had no access to their data and it shows. Now that first numbers regarding their reads get published and the first data sets available publicly, I realise a couple of preconditions were not met. Especially the fact that raw PacBio CLR reads seem to have an error rate between 1-in-5 (80% correct) and 1-in-7 (85% correct) means that MIRA will probably not operate very well with those. One should have something between 1-in-12 to 1-in-20 as error rate (92% to 95% correct) to get MIRA working happily.

During the course of 2011, PacBio has made available on their DevNet site quite a number of documents and introductory videos. A must read for everyone interested in this sequencing technology.

Note As of MIRA 3.4.0, large parts of this documentation are still from a time where things like terminology, sequencing specifics etc.pp were not available publicly. It'll take same time for me to convert the guide.

8.3 Introduction

Pacific Biosciences looks like the new kid on the block of sequencing technologies. They seem to have, for the first time since Sanger sequencing, something which is able to produce sequences which are actually longer than Sanger. They also have something new: *strobed sequencing*. That technique alone was reason enough for me to see whether it could be of any use. After a couple of modifications to the MIRA assembly engine, I think I can say that "yes, it very well can be."

One could feed strobed PacBio sequences to MIRA 3.0.0 and the 2.9.x line before and get some results out of it by faking them to be Sanger, though the results were not always pretty.

The first version of MIRA to officially support sequences from Pacific Biosciences is MIRA 3.2. Versions in the 3.0.1 to 3.0.5 range and 3.1.x had different degrees of support, but were never advertised having it.

8.3.1 Disclaimer

I am not affiliated with Pacific Biosciences nor do I -- unfortunately -- have early access to their data. Due to extreme secrecy, almost no one outside the company has actually seen their sequencing data. So some of what this guide contains is a bit of guesswork, reading through dozens and dozens of conference reports, blogs, press releases, tweets and whatever not.

But maybe I got some things right.

8.3.2 Some reading requirements

This guide assumes that you have basic working knowledge of Unix systems, know the basic principles of sequencing (and sequence assembly) and what assemblers do.

While there are step by step walk-throughs on how to setup your data for Sanger, 454 and Solexa in other MIRA guides, this guide is (currently) a bit more terse. You are expected to read at some point in time

- the *mira_reference* help file to look up some command line options.
- for hybrid assemblies of PacBio data with Sanger, 454, Solexa the corresponding *mira_usage*, *mira_454* or *mira_solexa* help files to look up how to prepare the different data sets.

8.3.3 Terminology

8.3.4 What is strobed sequencing?

Let's first have a look at what sequencing (either paired or unpaired) meant until now. I won't go into the details of conventional sequencing as this is covered elsewhere in the MIRA manuals (and in the Web).

8.3.4.1 Conventional way of sequencing a DNA template with Sanger, 454, Solexa, ...

In conventional, unpaired sequencing, you have a piece of DNA (a DNA *template*) which a machine reads out and then gives you the sequence back. Assume your piece of DNA to be 10 kilo-bases long, but your machine can read only 1000 bases. Then what you get back (`DNA` below is the DNA template, `R1` is a read) is this:

```
DNA: actgttg...gtgcatgctgatgactgact.........gactgtgacgtactgcttga...actggatctg

R1 : actgttg...gtgcatgct
     \__________________/
              |
          ~1000 bases
```

In conventional paired-end sequencing, you still can read only 1000 bases, but you can do it at the beginning and at the end of a DNA template. This looks like that:

```
DNA: actgttg...gtgcatgctgatgactgact.........gactgtgacgtactgcttga...actggatctg

R1 : actgttg...gtgcatgct
     \___________________/                              R2 : gcttga...actggatctg
               |                                              \___________________/
          ~1000 bases                                                    |
                                                                    ~1000 bases
```

While you still have just two reads of approximately 1000 bases, you know one additional thing: these two reads are approximately 10000 bases apart. This additional information is very useful in assembly as it helps to resolve problematic areas.

8.3.4.2 Sequencing a DNA template with Pacific Biosciences

Enter Pacific Biosciences with their strobed sequencing. With this approach, you can sequence also a given number of bases (they claim between 1000 and 3000), but you can sort of "distribute" the bases you want to read across the DNA template.

Warning Overly simplified and probably totally inaccurate description ahead! Furthermore, the extremely short read and gap lengths in these examples serve only for demonstration purposes.

Here's a simple example: assume you could read around 40 bases with your machinery, but that the DNA template is some ~80 bases. And assume you could tell your machine to read between 6 and 8 bases at a time, then leave out the next 6 to 8 bases, then read again etc. Like so:

```
DNA: actgttggtgcatgctgatgactgactgactgtgacgtacttgactgactggatctgtgactgactgtgactgactg

R1a: actgttg
R1b:                 gatgactgac
R1c:                                  cgtacttga
R1d:                                                    atctgtgac
R1e:                                                                          gactgactg
```

While in the example above we still read only 44 bases, these 44 bases span 77 bases on the DNA template. Furthermore, we have the additional information that the sequence of reads is R1a, R1b, R1c, R1d and R1e and, because we asked the machine to read in such a pattern, we expect the gaps between the reads to be between 6 and 8 bases wide.

This is actually possible with the system of PacBio. It streams the DNA template through a detection system which reads out the bases only, and only if, a light source (a laser) is switched on. Therefore, while streaming the template through the system, you read the DNA while the laser is on and you don't read anything while it's off ... meanwhile the template is still streamed through.

Now, why would one want to turn the laser off?

It seems as if the light source is actually also the major limitation factor, as it has as nasty side-effect the degradation of DNA it should still read. A real bummer: after 1000 to 3000 bases (sometimes more, sometimes less), the DNA you read is probably so degraded and error ridden (eventually even physically broken) that it makes no sense to continue reading.

Here comes the trick: instead of reading, say, 1000 bases in a row, you can read them in *strobes*: you switch the light on and start reading a couple of bases (say: 100), switch the light off, wait a bit until some bases (again, let's say approximately 100) have passed by, switch the light back on and read again ~100 bases, then switch off ... etc.pp until you have read your 1000 bases, or, more likely, as long as you can. But, as shown in the example above, these 1000 bases will be distributed across a much larger span on the original DNA template: in a pattern of ~100 bases read and ~100 bases not read, the smaller read-fragments span ~1800 to ~2000 bases.

Cool ... this is actually something an assembler can make real use of.

A more conventional approach could be: you switch the light on and start reading a couple of bases (say: 500), switch the light off, wait a bit until some bases (again, let's say approximately 10000) have passed by, switch the light back on and read again ~500 bases. This would be equivalent to a "normal" paired-end read with an insert size of 11Kb. But assemblers also can make good use of that.

8.3.5 Probable highlights and lowlights of PacBio sequencing data

8.3.5.1 Lowlights

8.3.5.1.1 Indel problems

Although Pacific Biosciences keeps pretty quiet on this topic, missed bases seem to be quite a problematic point. A bit like the 454 homopolymer problem but without homopolymers. From http://scienceblogs.com/geneticfuture/2010/02/pacific_biosciences_session_at.php

> 'Turner [the presenter from PacBio] said nothing concrete about error rates during his presentation, but this issue dominated the questions from the audience. Turner skilfully equivocated, steering clear of providing any hard numbers on the raw error rates and focusing on the system's ability to generate accurate consensus sequences through circular reads. Still, it's clear that deletion errors due to missing bases will pose a non-trivial problem for the system: Turner referred to algorithms for assembling sequence dominated by insertion/deletion errors currently in development.'

Someone else made a nice comment on this (from http://omicsomics.blogspot.com/2010/02/pacbios-big-splash.html):

> 'Well, not much on error rates from PacBio (apparently in the Q&A their presenter executed a jig, tango, waltz & rumba when asked).'

8.3.5.1.2 Variation in "dark insert" lengths

Astute readers will have noted that in the section on sequencing a DNA template with PacBio, I wrote 'approximately' when defining the length of the stretch of non-read bases in strobed sequencing. According to conference reports, the length can only be estimated with a variance of 10-20%. From http://www.genomeweb.com/sequencing/pacbio-says-strobe-sequen

> 'There is uncertainty regarding the size of the "dark" inserts, owing to "subtle fluctuations" in the DNA synthesis speed, he said, but it becomes smaller with longer inserts. For example, with 400-base inserts, the coefficient of variation of its size is 20 percent, but it decreases to 10 percent with 1,600 bases.'

8.3.5.2 Highlights

8.3.5.2.1 Read lengths

The reports are a bit contradictory regarding achievable read lengths. While PacBio mentions they have attained read length of up to 20Kb in their labs and expect to be able to go to up to 50Kb, the first generation machines are marketed with much lower expectations. However, "much lower" in this context still means: at least 1 Kb and very good chances to have a good percentage of reads in the 3 to 5 Kb range.

8.3.5.2.2 Strobed sequencing

The strobed sequencing method should allow to do a couple of interesting things. First of, simulate conventional paired-end sequencing. Then, going into real strobe sequencing, extending the length reads span over a DNA template by perhaps doubling or tripling the length will be extremely useful to cross most but the most annoying repeats one would encounter in prokaryotes ... and probably also eukaryotes once PacBio regularly achieves lengths of 10000 bases.

8.4 How MIRA handles strobes with "elastic dark inserts"

MIRA currently knows two ways to handle strobed reads:

1. a more traditional approach by using two strobes at a time as read pair
2. the "elastic dark insert" approach where all strobes are put in one read and connected by stretches of `N` representing the dark inserts. "Elastic" means that -- the initial lengths of the dark inserts being a rough estimate -- the length of the inserts are then corrected in the iterative assembly passes of MIRA.

The elastic dark insert approach has an invaluable advantage: it keeps the small strobes connected in order in a read. This considerably reduces sources of errors when highly repetitive data is to be assembled where paired-end approaches also have limits.

Keeping the dark inserts as integral part of the reads however also poses a couple of problems. Traditional Smith-Waterman alignments are likely to give some of these alignments a bad score as there will invariably be a number of overlaps where the true length of the dark stretch is so different from the real length that an alignment algorithm needs to counterbalance this with a considerable number of inserted gaps. Which in turn can lower Smith-Waterman score to a level where needed identity thresholds are not met. The following example shows an excerpt of a case where a read with dark insert which length was estimated too low aligning against a read without dark insert:

```
...TGACTGA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
```

While MIRA algorithms have methods to counterbalance this kind of scoring malus (e.g., by simply not counting gap scores in dark strobe inserts), another effect then appears: multiple alignment disorders. Like in many other assemblers the construction of multiple alignments is done iteratively by aggregating new reads to an existing contig by aligning it against a temporary consensus. As the misestimation of dark insert lengths can reach comparatively high numbers like 20 to 100 bases or more, problems can arise if several misestimated dark inserts in reads come together at one place. A simple example: assume the following scenario, where reads 1, 2, 3 and 4 get to form a contig by being assembled in exactly this order (1,2,3,4):

```
Read1  ...TGACTGAnnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
```

then

```
Read1  ...TGACTGA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
```

then

```
Read1  ...TGACTGA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
```

then

```
Read1  ...TGACT*****GA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

Why multiple alignments sometimes look funny

You will note that the above alignment does not seem optimal as the following would certainly look better:

```
Read1  ...TGACTGA********************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACTGA*****TGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACTGA*****TGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

However, remember that assembler add reads sequentially to a contig without prior knowledge of the complete multiple alignment. This leads to the necessity of using temporary consensus sequences as target for a new read to align.

MIRA knows a lot of tricks to mitigate the detrimental effects of iteratively building multiple alignments, one of which is to use an extended genetic code with weights hinting at potential gaps. This however does not always suffice to resolve every problem.

Here's what happens step by step in for this example. First, read 1 forms the contig, then read 2 is aligned and both form a new contig with a new temporary consensus (named TCONS2 below) like this:

```
Read1  ...TGACTGA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...

TCONS2 ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
```

Read 3 is then added and as it is equal to read 3 (at least in this part of the sequence), things go smoothly as the alignment of TCONS2 and read 3 is without difference and the alignment of

```
TCONS2 ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
```

leads to the multiple alignment of the three reads like this:

```
Read1  ...TGACTGA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...

TCONS3 ...TGACTGATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
```

Note that a new temporary consensus TCONS3 is formed against which the next read (read 4) will have to align:

```
TCONS3 ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

which then leads to the sub-optimal multiple alignment seen in the beginning:

```
Read1  ...TGACT*****GA***************nnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

It would have been totally illogical for a Smith-Waterman alignment to align the temporary consensus TCONS3 with read 4 like this:

```
TCONS3 ...TGACTGA*****TGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

as from purely mathematical point of view, the score of this alignment is lower than for the previous alignment.

This can lead to severe misaligns in multiple alignments with several reads as the following screenshot shows exemplarily.

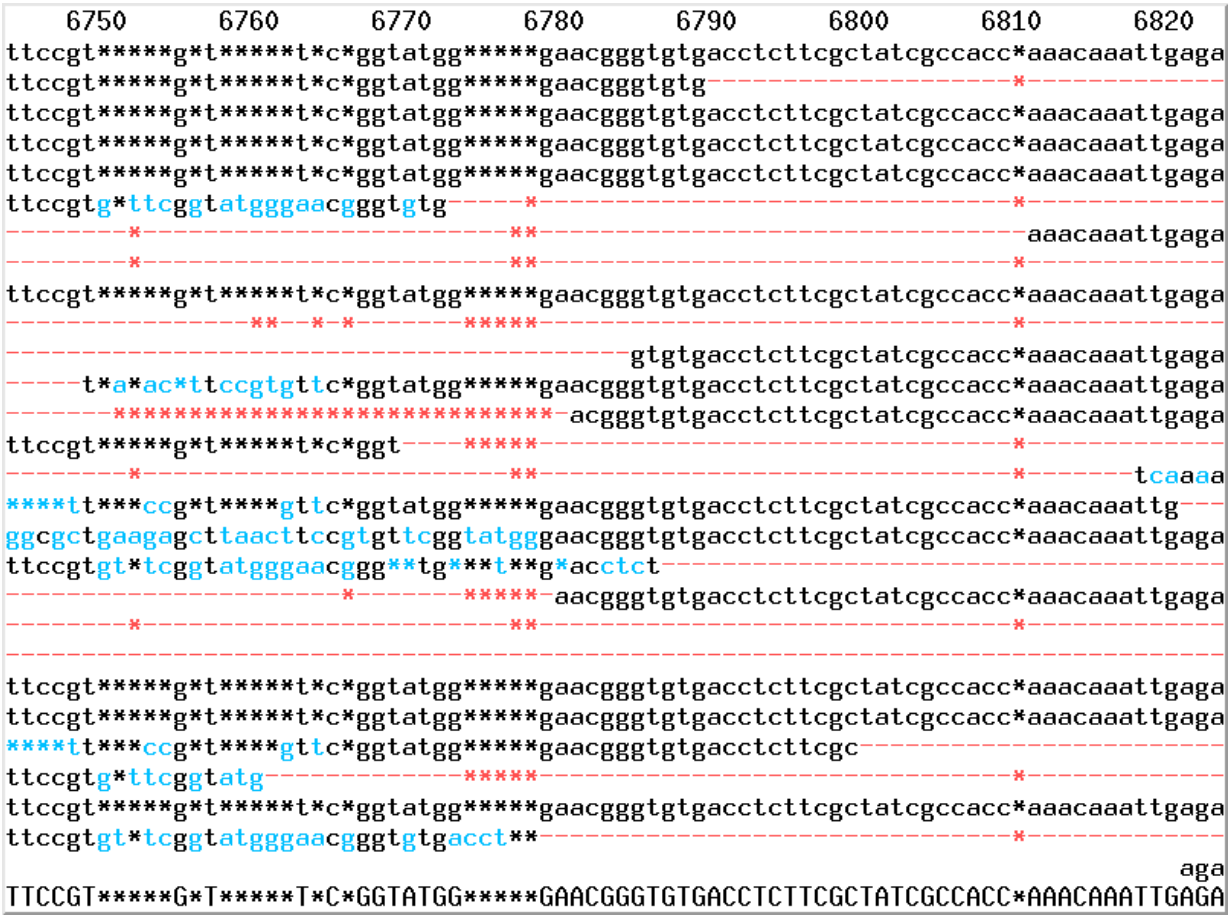

Figure 8.1: Multiple alignment with PacBio elastic dark inserts, initial status with severe misalignments

However, MIRA is an iterative assembler working in multiple passes and iterations within a pass. This allows for a strategy of iterative correction of the estimation of dark length inserts. Like with every sequencing technology it knows, MIRA analyses the multiple alignment of a contig in several ways and searches for, e.g., misassembled repeats (for more information on this, please refer to the MIRA manual). When having reads with the technology from Pacific Biosciences, MIRA also analyses the elastic dark inserts whether or not their length as measured in the multiple alignment fits the estimated length. If not, the length of the dark insert will be corrected up or down for the next pass, the correction factor being of two thirds of the estimated difference between true and measured length of the dark insert.

Coming back to the example used previously:

```
Read1   ...TGACT*****GA***************nnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2   ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3   ...TGACT*****GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4   ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

You will note that there are basically two elastic dark insert stretches. The first in read 1 has an underestimation of of the dark insert size of 16 bases, the second has an overestimation of five bases.

While it is not depicted in this simple example, the calculation of how many bases away the estimated length of a dark insert is from the real value is not always simple as multiple misaligns can make this task challenging. If taken without precautions, errors in this calculation can lead to even more false estimation of dark insert lengths and self-enhancing errors. During experimentation with simulated strobed projects, MIRA built the best contigs and needed the least iterations when applying a factor of two thirds to the calculated estimation of dark insert length error.

Accordingly, MIRA will add two thirds of 16 = 10 Ns to the estimated dark insert in read 1 and remove 3 Ns (two thirds of 5) from read 4:

```
Read1 old  ...TGACTGAnnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read1 new  ...TGACTGANNNNNNNNNNnnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...

Read4 old  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
Read4 new  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

These new reads will be used in the next (sub-)passes of MIRA. Continuing the example from above, the next multiple alignment of all four reads would look like this:

```
Read1  ...TGACT**GA******NNNNNNNNNNnnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACT**GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACT**GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

Again, the dark inserts would be corrected by MIRA, this time adding 4 Ns to read 1 and removing one N from read 4., so that the next multiple alignment is this:

```
Read1  ...TGACT*GA**NNNNNNNNNNNNNNnnnnnnnnnnnnnnnnnnnnnnTCAGTTGAT...
Read2  ...TGACT*GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read3  ...TGACT*GATGACTTTATCTATGGAGCTTATATGCGTCGAGCTTGGTCAGTTGAT...
Read4  ...TGACTnnnnnnnnnnnnnnnnnnnnnnnnnnnnATGCGTCGAGCTTGGTCAGTTGAT...
```

From there it is trivial to see one just needs two more iterations to replace the initial estimated length of the dark insert by the true length of it. The next screenshot continues the live example shown previously after the second pass of MIRA (remember that each pass can have multiple sub-passes):

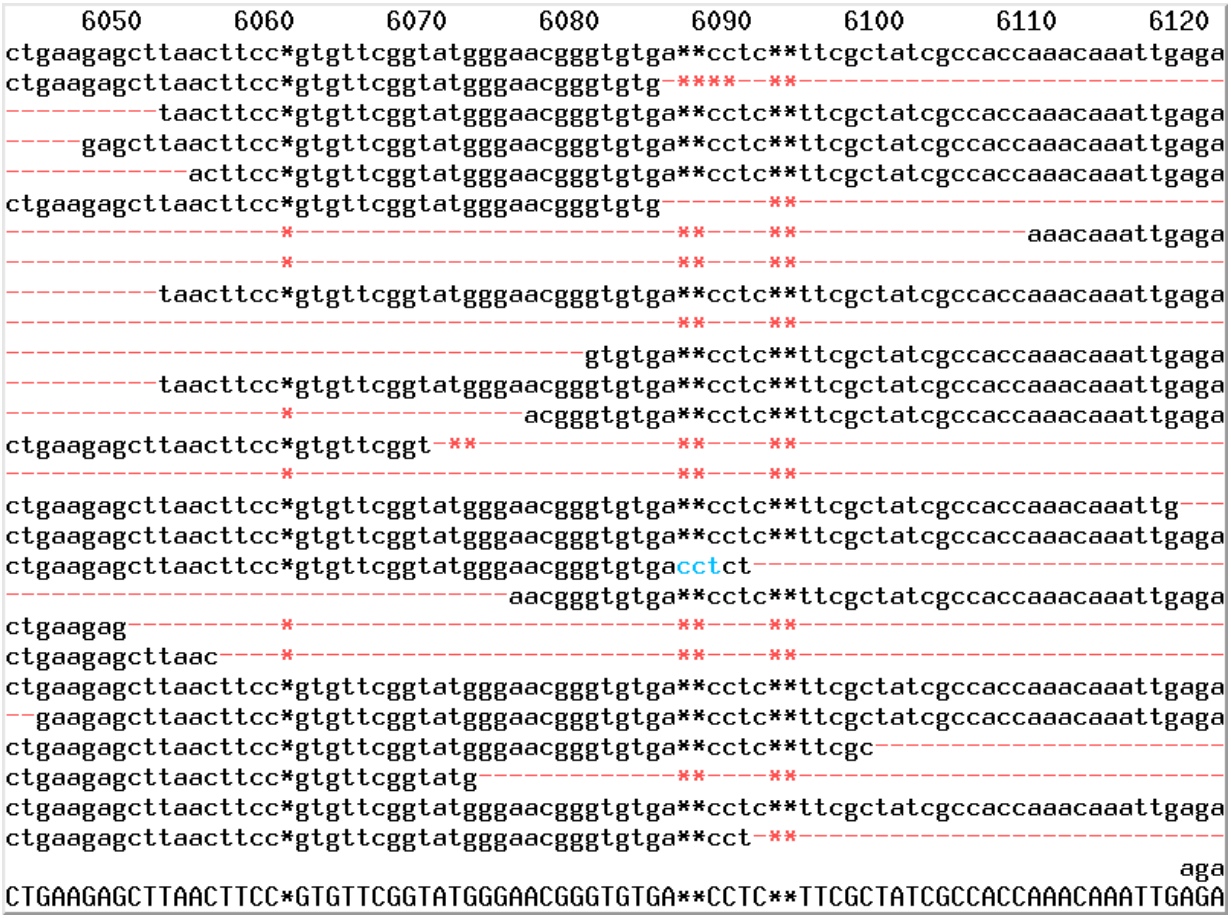

Figure 8.2: Assembly with PacBio dark inserts, status after second pass of MIRA

One pass (and multiple sub-passes) later, the elastic dark inserts in this example have reached their true lengths. The multiple alignment is as good as it can get as the following figure shows:

Figure 8.3: Assembly with PacBio dark inserts, status after third pass of MIRA

The elastic dark insert strategy is quite successful for resolving most problems but sometimes also fails to find a perfect solution. However, the remaining multiple alignment is -- in most cases -- good enough for a consensus algorithm to find the correct consensus as the next screenshot shows:

Figure 8.4: Assembly with PacBio dark inserts, example where elastic correction failed partially

8.5 Assembly of PacBio data with MIRA

8.5.1 Preparing data

MIRA will happily read data in several different formats (FASTA, FASTQ, etc.). For the sake of simplicity, this guide will use FASTQ as demonstration format, but most of the time not add the quality line.

8.5.1.1 Simple, unstrobed, non-paired reads

This is actually quite simple. Just put your reads as FASTQ in a file and you are done. No need to bother about read naming conventions or similar things. Like so:

```
@readname_001
ACGTTGCAGGGTCATGCAGT...
@readname_002
...
```

8.5.1.2 Strobed reads with two strobes, simulating paired ends

You have two possibilities for that. If the "dark insert" is not too long and about the same size or shorter than the average sequenced length at the ends, you can put the data into one read and fill up the estimated dark insert length with the 'n' character. The following example shows this, using lengths of 10 for the sequenced parts and a dark insert size of 10:

```
@readname_001
ACGTTGCAGGnnnnnnnnnnGTCATGCAGT
@readname_002
...
```

In case you have long "dark inserts", it is preferable to keep both parts physically separated in different reads. In this case, read naming becomes important for an assembler. Pick any paired-end naming scheme you want and name your reads accordingly. The following example shows the same data as above, split in two reads and using the Solexa naming scheme to denote the first read of a pair by appending `/1` to the read name and the second part by appending `/2`:

```
@readname_001/1
ACGTTGCAGG
@readname_001/2
GTCATGCAGT
@readname_002/1
...
```

Note
The example above used Solexa naming scheme to denote paired-end partner reads. You can use any naming scheme you want as long as MIRA knows it. E.g.: the *forward/reverse* or *Sanger* or *TIGR* naming schemes, you will just need to tell MIRA about it with the [-LR:rns] parameter.
As soon as first data sets with PacBio are available, MIRA will also implement their naming scheme.

Should all your reads approximately have the same total length of first part (/1) + dark insert + second part (/2), then you don't need to create an additional file with information about expected distance between the parts, you can use [-GE:tismin:tismax] to tell MIRA about it. In case you have different sizes because, e.g. you have sequenced different libraries, then you will need to tell MIRA which reads have which distance from each other. You can do this in an XML file in TRACEFORMAT (as defined by the NCBI). There will be other means in the future, but these have not been implemented yet.

8.5.1.3 Strobed reads with multiple strobes

Like in the case with two strobes, you have the choice between putting all strobes in one read ... or to separate the strobes in multiple reads. The following example shows the case where all strobes are in one read:

```
@readname_001
ACGTTGCAGGnnnnnnnnnnGTCATGCAGTnnnnnnnnnnnnnnnnnnnnnnnnnnTATGCACTGACnnnnnTAGCTGA
@readname_002
...
```

Note that the "dark inserts" do not necessarily need to be of the same length, even within a read. Indeed, depending on your sequencing strategy they can have very varying lengths, although one should take care that these inserts are not much longer than the longest strobes (or longest unstrobed read) in your data set.

In case you have long dark inserts, you should split the parts separated by these long inserts into different reads. E.g., if your strobes are 500 bases long, but separated by dark inserts > 1Kb, split them. You are free to split them however you like, in sub-pairs, in single strobes or whatever. For the example given above, this could be done like this:

```
@readname_001/1
ACGTTGCAGGnnnnnnnnnnGTCATGCAGT
@readname_001/2
TATGCACTGACnnnnnTAGCTGA
@readname_002
...
```

Note the dark inserts remaining in each read of the "virtual" read-pair. The same sequences could also be split like this:

```
@readname_001a/1
ACGTTGCAGG
@readname_001a/2
GTCATGCAGT
@readname_001b/1
TATGCACTGAC
@readname_001b/2
TAGCTGA
@readname_002
...
```

which would then be two read-pairs: the first and second strobes are paired, as well as the third and fourth. Here too, you can use any combination strobes to pair to each other (or to use without pair information).

Combining first and fourth strobe as well as second and fourth would look like this:

```
@readname_001a/1
ACGTTGCAGG
@readname_001b/1
TAGCTGA
@readname_001b/2
GTCATGCAGT
@readname_001a/2
TATGCACTGAC
@readname_002
...
```

Note that in this case you probably need to provide paired-end information in a NCBI TRACEARCHIVE XML file to tell MIRA about the different insert sizes.

Finally, you can put the reads all in one template like this:

```
@readname_001.f1
ACGTTGCAGG
@readname_001.f2
GTCATGCAGT
@readname_001.f3
TATGCACTGAC
@readname_001.f4
TAGCTGA
@readname_002
...
```

Note the subtle change in the naming of reads where I changed to a different postfix naming. This is because the Solexa naming scheme currently does not (officially) allow for more than two reads per DNA template (well, /1 and /2). The forward/reverse naming scheme like implemented by MIRA however does allow this.

This has just one drawback: currently MIRA will not be able to store the distances between the strobes when they are all in one template. This is being worked on and will be possible in a future version.

8.5.2 Setting up files and directories

Create a directory where you copy your input data into (or where you set a soft-link where it really resides).

Currently (as of version 3.2.0) MIRA allows one input file per sequencing technology (one for Sanger, one for 454, one for Solexa and one for PacBio). This will change in the future, but for the moment it is how it is.

While you could name your input files whatever you like and pass these as parameters to MIRA, it is easier to follow a simple naming scheme that allows MIRA to find everything automatically. This scheme is `projectname_in.sequencingtechtype.filetypepostfix`

The `projectname` is a free string which you decide to give to your project. The `sequencingtechtype` can be one of "sanger", "454", "solexa" or "pacbio". Finally the `filetypepostfix` is either "fasta" and "fasta.qual", "fastq" or any other type supported by MIRA.

Note that MIRA supports loading a lot of other information files (XML TRACEINFO, strain data etc.), please consult the reference manual for more information.

8.5.3 Launching MIRA

In the most basic incantation, you will need to tell MIRA just five things:

1. the name of your project.
2. whether you want a "genome" or "EST" assembly
3. whether it is a denovo or mapping assembly
4. which quality level (draft, normal or accurate)
5. which sequencing technologies are involved (sanger, 454, solexa, pacbio)

Using the most basic *quick switches* of MIRA, the command line for an accurate denovo genome with PacBio data then looks like this:

```
mira --project=yourname --job=genome,denovo,accurate,pacbio
```

or for a hybrid PacBio and Solexa of the above:

```
mira --project=yourname --job=genome,denovo,accurate,pacbio,solexa
```

Note MIRA has -- at the last count -- more than 150 parameters one can use to fine tune almost every aspect of an assembly, from data loading options to results saving, from data preprocessing to results interpretation, from simple alignment parameters to parametrisation of internal misassembly decision rules ... and much more. Many of these parameters can be even set individually for each sequencing technology they apply to. Example given: in an assembly with Solexa, Sanger, 454 and PacBio data, the minimum read length for Solexa could be set to 30, while for 454 it could be 80, Sanger 100 and PacBio 150. Please refer to the reference manual for a full overview on how to use *quick switches* and *extended switches*.

8.6 Walkthroughs: real data sets from PacBio

We'll use some data provided by PacBio for the *E. coli* O104:H4 outbreak in 2011, see http://www.pacbiodevnet.com/Share/Datasets/E-coli-Outbreak for more info.

8.6.1 Error corrected CLR for E. coli C227-11$

That data set is quite interesting: PacBio took CLR reads (the reads with only ~85% accuracy) and mapped CCS reads (presumably >00% accuracy) to them to correct errors of the CLR reads. The resulting *error corrected CLR* data is of pretty good quality, not only from the quality values but when assembled, the number of sequencing errors in the reads which can be spotted in the aligments is obviously quite low.

8.6.1.1 Preparing a directory structure

Note: this is how I set up a project, feel free to implement whatever structure suits your needs.

```
$ mkdir c227-11-clrc
$ cd c227-11-clrc
arcadia:c227-11-clrc$ mkdir origdata data assemblies
```

Your directory should now look like this:

```
arcadia:c227-11-clrc$ ls -l
drwxr-xr-x 2 bach users 48 2011-08-19 20:21 assemblies
drwxr-xr-x 2 bach users 48 2011-08-19 20:21 data
drwxr-xr-x 2 bach users 48 2011-08-19 20:21 origdata
```

"c227-11-clrc" is an arbitrary name I just chose by concatenating the name of the bug and "-clrc" to indicate that this project has **CLR** sequences which were **C**orected. But you can name this whatever you want: foobar, blafurbsel, ...

Explanation of the structure:

- the `origdata` directory will contain the 'raw' result files that one might get from sequencing. In our case it will be the `.tar.gz` file with the data in FASTQ format from the Devnet site.
- the `data` directory will contain the preprocessed sequences for the assembly, ready to be used by MIRA
- the `assemblies` directory will contain assemblies we make with our data (we might want to make more than one).

8.6.1.2 Getting the data and preparing it

Head over to PacBio DevNet and fetch the data set for the E. coli C227-11 CLR corrected data set. Put it into the `origdata` directory created a few moments ago.

Now, let's extract the data to the `data` directory:

```
arcadia:c227-11-clrc$ cd data
arcadia:data$ tar xvzf ../origdata/e-coli-c227-11-corrected-fastq-1.2.2beta.tgz
e-coli-c227-11-corrected-fastq-1.2.2beta/
e-coli-c227-11-corrected-fastq-1.2.2beta/e-coli-c227-11-corrected.fastq
```

One thing you would quickly find out but which I tell now to save time: at the moment, PacBio seems to love ultra long read names. Here are the first 10 from the current data set:

```
arcadia:data$ grep ^@m e-coli-c227-11-corrected-fastq-1.2.2beta/e-coli-c227-11-corrected. ↩
    fastq | head -10
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/10040/0_5174/c0
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/10040/0_5174/c1
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/1017/0_1636/c0
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/1054/0_4073/c0
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/1054/0_4073/c1
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/1054/4121_4891/c0
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/10548/0_5766/c0
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/10548/0_5766/c1
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/10640/0_2393/c0
@m110618_035655_42142_c100158802555500000315044108071130_s1_p0/11000/0_3285/c0
```

How sweet! File names with 80 and more characters AND having the "/" character as component, the later being a recipe for desaster sooner or later in some post-processing pipelines.

Note MIRA has absolutely no problem with the above: neither with long read names nor with the "/" character in the name. However, long read names are a problem for example for **gap4** (an assembly viewer) and the "/" character might lead to confusion with the standard UNIX directory separator.

For the sake of simplicity and compatibility, let's rename all sequences. For this we'll use **convert_project**, which is a binary of the MIRA program package:

```
arcadia:data$ convert_project
  -f fastq -t fastq -R c227-11-clrc
  e-coli-c227-11-corrected-fastq-1.2.2beta/e-coli-c227-11-corrected.fastq
  c227-11-clrc_in.pacbio
Loading from fastq, saving to: fastq
Loading data from FASTQ ...
Counting sequences in FASTQ file: found 73496 sequences.
Localtime: Sat Aug 20 20:36:26 2011
Unusual offset of 34, guessing this file to be a Sanger-type FASTQ format.
Using calculated FASTQ quality offset: 33
Localtime: Sat Aug 20 20:36:26 2011
Loading data from FASTQ file:
 [0%] ....|.... [10%] ....|.... [20%] ....|.... [30%] ....|.... [40%] ....|....
[50%] ....|.... [60%] ....|.... [70%] ....|.... [80%] ....|.... [90%] ....|.... [100%]

Done.
Loaded 73496 reads, Localtime: Sat Aug 20 20:36:35 2011
 done.
Data conversion process finished, no obvious errors encountered.
```

Note The above command has been split in multiple lines for better overview but should be entered in one line.

The parameters to **convert_project** say

- **-f fastq**: the "from" type (type of the input file) is FASTQ
- **-t fastq**: the "to" type (type of the output file) should be FASTQ
- **-R c227-11-clrc**: sequences should be renamed, all starting with "c227-11-clrc" and to which **convert_project** will append an underscore and a counter.
- **e-coli-c227-11-corrected-fastq-1.2.2beta/e-coli-c227-11-corrected.fastq**: that's the full name of our input file.
- **c227-11-clrc_in.pacbio**: that's the partial name of our output file. Partial because **convert_project** will automatically add the postfix of the target format to the name, in this case `.fastq`.

  Just in case you wonder why this works like this: **convert_project** can convert to multiple formats at once, e.g., like this: convert_project -f fastq -t fasta -t fastq -t maf ... and then you will get nicely named files.

Your directory should now look like this ...

```
arcadia:data$ ls -l
-rw-r--r-- 1 bach bach 257844076 2011-08-20 21:04 c227-11-clrc_in.pacbio.fastq
drwxr-x--- 2 bach bach      4096 2011-07-22 04:49 e-coli-c227-11-corrected-fastq-1.2.2beta
```

... and as we do not need the subdirectory with the extracted data from PacBio anymore, let's get rid of it:

```
arcadia:data$ rm -rf e-coli-c227-11-corrected-fastq-1.2.2beta
arcadia:data$ ls -l
-rw-r--r-- 1 bach bach 257844076 2011-08-20 21:04 c227-11-clrc_in.pacbio.fastq
```

Perfect, we're done here.

8.6.1.3 Starting the assembly

Good, we're almost there. Let's switch to the `assembly` directory and create a subdirectory for our first assembly test.

```
arcadia:data$ cd ../assemblies/
arcadia:assemblies$ mkdir 1sttest
arcadia:assemblies$ cd 1sttest
```

This directory is quite empty and the PacBio data is not present. Let's link to the file we just created in the previous step:

```
arcadia:1sttest$ ln -s ../../data/* .
arcadia:1sttest$ ls -l
lrwxrwxrwx 1 bach bach      39 2011-08-20 16:56 c227-11-clrc_in.pacbio.fastq -> ../../data/ ↩
    c227-11-clrc_in.pacbio.fastq
```

Starting the assembly is now just a matter of one line with some parameters set correctly:

```
arcadia:1sttest$ mira
--project=c227-11-clrc
--job=denovo,genome,accurate,pacbio
>&log_assembly.txt
```

Note The above command has been split in multiple lines for better overview but should be entered in one line.

Some 3 to 4 hours later, you should have a nice and shiny assembly of your data.

Now, that was easy, wasn't it? In the above example - for assemblies having only PacBio data and if you followed the walkthrough on how to prepare the data - everything you might want to adapt in the first time are the following options:

- --project (for naming your assembly project)
- --job (perhaps to change the quality of the assembly to 'draft'

Of course, you are free to change any option via the extended parameters, perhaps change the default number of processors to use from 2 to 4 via [-GE:not=4] or any other of the > 150 parameters MIRA has ... but this is covered in the MIRA main reference manual.

8.6.1.4 Working with the results of the assembly

There is a whole chapter in the manual dedicated to this, you are expected to read it :-)

However, for the impatient, here's a quick rundown on what I am going to show as example in this section: filtering of results and loading results into an assembly viewer.

8.6.1.4.1 Filtering results

This assembly project has an average coverage of roundabout 23 to 24x. Due to sequencing errors, MIRA will have also created a few small contigs with just a few reads and lot less coverage. These contigs are, well, most of the time not interesting as they contain junk most of the time. I want to get rid of them.

Let's say that only contigs $\geq$ 500 base pairs and with an average coverage $\geq$ 8 (which is 1/3 of the average coverage of 24 of the whole project) are interesting. Let's filter them out of the MIRA results and create a CAF file which can then be imported into either **gap4** or **gap5** (assembly viewers and finishing tools from the Staden package).

Let's take a quick look at the main directories and files of the assembly:

```
arcadia:1sttest$ ls -l
drwxr-xr-x 6 bach bach    4096 2011-08-20 16:57 c227-11-clrc_assembly
lrwxrwxrwx 1 bach bach      39 2011-08-20 16:56 c227-11-clrc_in.pacbio.fastq -> ../../data/ ↩
    c227-11-clrc_in.pacbio.fastq
-rw-r--r-- 1 bach bach 2524406 2011-08-20 19:46 log_assembly.txt
arcadia:1sttest$ cd c227-11-clrc_assembly
arcadia:c227-11-clrc_assembly$ ls -l
drwxr-xr-x 2 bach bach  4096 2011-08-20 17:01 c227-11-clrc_d_chkpt
drwxr-xr-x 2 bach bach  4096 2011-08-20 19:46 c227-11-clrc_d_info
drwxr-xr-x 2 bach bach  4096 2011-08-20 20:10 c227-11-clrc_d_results
drwxr-xr-x 2 bach bach 36864 2011-08-20 19:46 c227-11-clrc_d_tmp
arcadia:c227-11-clrc_assembly$ ls -l c227-11-clrc_d_info
-rw-r--r-- 1 bach bach     2320 2011-08-20 19:46 c227-11-clrc_info_assembly.txt
-rw-r--r-- 1 bach bach       88 2011-08-20 16:57 c227-11-clrc_info_callparameters.txt
-rw-r--r-- 1 bach bach   168049 2011-08-20 19:46 c227-11-clrc_info_consensustaglist.txt
-rw-r--r-- 1 bach bach  1599427 2011-08-20 19:46 c227-11-clrc_info_contigreadlist.txt
-rw-r--r-- 1 bach bach     6718 2011-08-20 19:46 c227-11-clrc_info_contigstats.txt
-rw-r--r-- 1 bach bach     7401 2011-08-20 19:46 c227-11-clrc_info_debrislist.txt
-rw-r--r-- 1 bach bach    10572 2011-08-20 19:15 c227-11-clrc_info_readrepeats.lst
-rw-r--r-- 1 bach bach 42697892 2011-08-20 19:46 c227-11-clrc_info_readtaglist.txt
arcadia:c227-11-clrc_assembly$ cd c227-11-clrc_d_results
arcadia:c227-11-clrc_results$ ls -l
-rw-r--r-- 1 bach bach 192615652 2011-08-20 19:46 c227-11-clrc_out.ace
-rw-r--r-- 1 bach bach 597368574 2011-08-20 19:46 c227-11-clrc_out.caf
-rw-r--r-- 1 bach bach 306918692 2011-08-20 19:46 c227-11-clrc_out.maf
-rw-r--r-- 1 bach bach   6333500 2011-08-20 19:46 c227-11-clrc_out.padded.fasta
-rw-r--r-- 1 bach bach  18987968 2011-08-20 19:46 c227-11-clrc_out.padded.fasta.qual
-rw-r--r-- 1 bach bach      7240 2011-08-20 19:46 c227-11-clrc_out.tcs
-rw-r--r-- 1 bach bach   6297565 2011-08-20 19:46 c227-11-clrc_out.unpadded.fasta
-rw-r--r-- 1 bach bach  18881604 2011-08-20 19:46 c227-11-clrc_out.unpadded.fasta.qual
-rw-r--r-- 1 bach bach   4442567 2011-08-20 19:46 c227-11-clrc_out.wig
```

OK, we're at the right spot for filtering. While we are at it, tell **convert_project** to not only convert to CAF, but also to write a new file with tabular data on contig statistics of the filtered contigs:

```
arcadia:c227-11-clrc_results$ convert_project
  -f maf -t caf -t cstats -x 500 -y 8
  c227-11-clrc_out.maf
  c227-11-clrc_filteredx500y8
Loading from maf, saving to: caf cstats
First counting reads:
 [0%] ....|.... [10%] ....|.... [20%] ....|.... [30%] ....|.... [40%] ....|.... [50%]  ↩
     ....|.... [60%] ....|.... [70%] ....|.... [80%] ....|.... [90%] ....|.... [100%]
Now loading and processing data:
```

... lots of lines omitted ...

```
Data conversion process finished, no obvious errors encountered.
arcadia:c227-11-clrc_results$ ls -l *filtered*
-rw-r--r-- 1 bach bach 584042411 2011-08-20 22:12 c227-11-clrc_filteredx500y8.caf
-rw-r--r-- 1 bach bach      1518 2011-08-20 22:12 c227-11- ↩
    clrc_filteredx500y8_info_contigstats.txt
```

```
arcadia:c227-11-clrc_results$ cat c227-11-clrc_filteredx500y8_info_contigstats.txt
# name                  length  av.qual #-reads mx.cov. av.cov  GC%     CnIUPAC CnFunny CnN ↩
         CnX     CnGap   CnNoCov
c227-11-clrc_c1         335375  90      4081    35      21.27   49.73   2       0       0  ↩
          0       1975    0
c227-11-clrc_c2         651370  90      9224    41      23.88   51.23   3       0       0  ↩
          0       4535    0
```

```
c227-11-clrc_c3          356318  90      4962    40      24.18   50.81   0       0       0 ↩
          0       2208    0
c227-11-clrc_c4          386288  90      5178    39      23.58   51.49   3       0       0 ↩
          0       2367    0
c227-11-clrc_c5          908271  90      12277   40      23.41   50.73   3       0       0 ↩
          0       5912    0
...
```

Once filtered, how many contigs are there? Well, 22: it's the number of lines minus one of the file `c227-11-clrc_filteredx500y8_info_contigstats.txt`:

```
arcadia:c227-11-clrc_results$ wc -l c227-11-clrc_filteredx500y8_info_contigstats.txt
23
```

8.6.1.4.2 Looking at the assembly in gap4

I'm very fond of gap4, so I'll use it to show ho the assembly looks like:

```
arcadia:c227-11-clrc_results$ caf2gap -project c227-11 -ace c227-11-clrc_filteredx500y8.caf ↩
    >&/dev/null
arcadia:c227-11-clrc_results$ ls -l C*
-rw-r--r-- 1 bach bach 543539856 2011-08-20 22:35 C227-11.0
-rw-r--r-- 1 bach bach  39512896 2011-08-20 22:35 C227-11.0.aux
arcadia:c227-11-clrc_results$ gap4 C227-11.0
```

And while it's difficult to judge an assembly only from a screenshot, I made one: 22 contigs, none smaller than 4kb and pretty good certainty your bases are correct. Pray tell, isn't that beautiful?

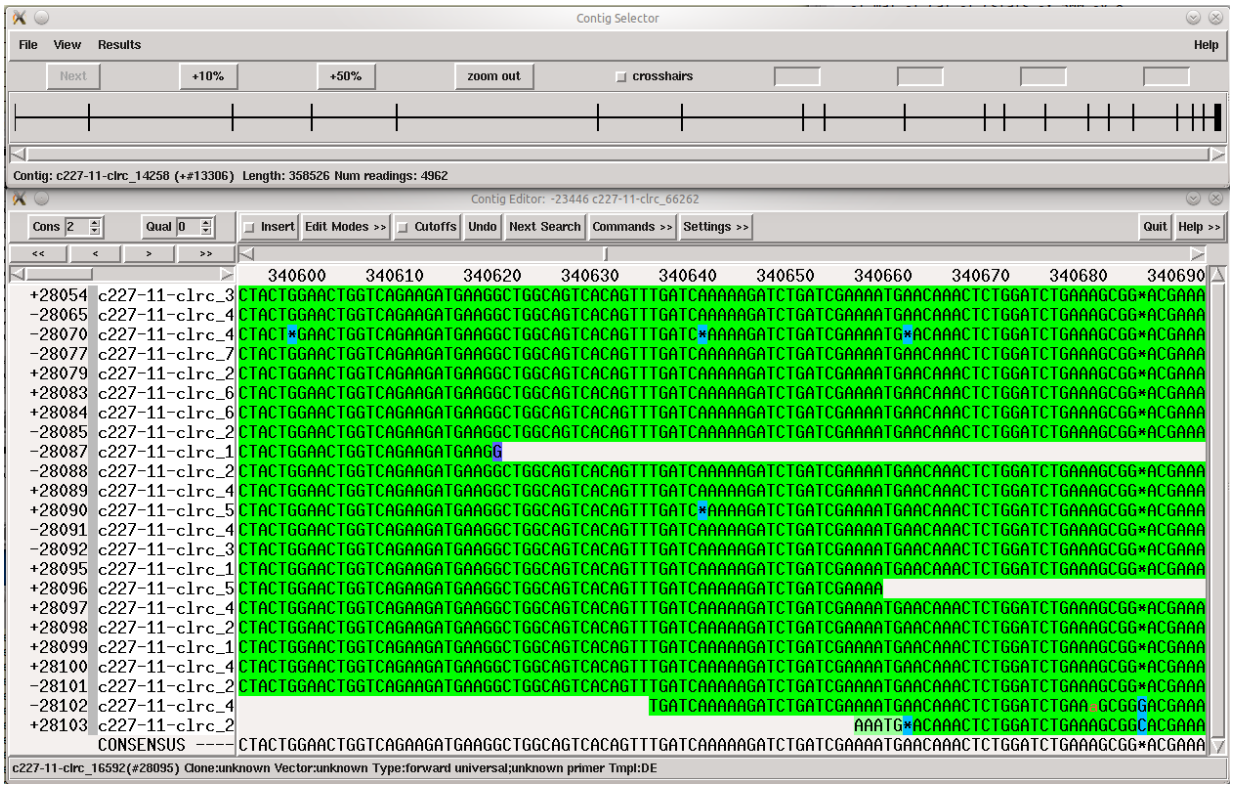

Figure 8.5: Result of the assembly of *E. coli* C227-11 with error corrected CLR reads.

8.6.2 CCS reads for E. coli C227-11

Head over to PacBio DevNet and fetch the data set for the E. coli C227-11 CCS data set.

For the rest ... well, it's pretty much the same as for the CLR data set. Just one little difference: in the `.tgz` you downloaded, PacBio has split the data set into multiple FASTQ files (for whatever reason). You will need to concatenate them into one file before starting to work with that. Yep, and that's it.

8.7 Walkthroughs: assembly examples with simulated PacBio data

Whole genome sequencing of bacteria will probably be amongst the first for which the long PacBio reads will have an impact. Simply put: the repeat structure -- like rRNA stretches, (pro)phages and or duplicated genes/operons -- of bacteria is such that most genomes known so far can be assembled and or scaffolded with paired-end libraries between 6Kb and 10Kb. **Cite paper ...!**

Well, using strobed reads where a DNA template is sequenced in several strobes and the dark inserts have approximately the same length as a strobe, the initial PacBio data should be capable to generate strobed data from DNA templates a total span between 2000 and 6000 bases.

Furthermore, strobed reads can be used to generate traditional paired-end sequence with large insert sizes like 10Kb or more.

In the first few examples showing assembly with only PacBio data, we will use the genome of the *Bacillus subtilis* 168, which is a long standing model organism for systems biology and also used in biotechnology. From a complexity point of view, the genome has some interesting things in. As example, there are 11 rRNA stretches, some of them clustered together, which probably comes from the fact that *Bsub* evolved under laboratory conditions to become a fast grower. The most awful multiple rRNA cluster is the one starting at ...Kb and is ... Kb long.

The examples afterwards we will work with *Escherichia coli* ... (Eco), another model organism in the bacterial community. That time we will mix simulated low coverage PacBio data with real data from Solexa deposited at the NCBI Short read Archive (SRA).

Note
Currently this section contains examples with real Solexa reads but only simulated PacBio reads as I do not have early access to real PacBio data. However, I think that these examples show the possibilities such a technology could have.

8.7.1 Some general notes on how directories and data are set-up in these examples

Everyone (or every sequencing group / center) has more or less an own standard on how to organise directories and data prior to an assembly. Here's how I normally do it and how the following examples will be -- more or less -- set up: one top directory with the name of the project containing three specific sub-directories; one for original data, one for eventually reformated data and one for assemblies. That looks a bit like this:

```
$ mkdir myproject
$ cd myproject
myproject$ mkdir origdata data assemblies
```

The *origdata* directory contains whatever data file (or links to those) I have for that project: sequencing files from the provider, reference genomes from databases etc.pp. The general rule: no other files, and these files are generally write protected and unchanged from the state of delivery.

The *data* directory contains the files as MIRA will want to use them, eventually reformatted or reworked or bundled together with other data. E.g.: if your provider delivered several data files with sequence data for PacBio, you currently need to combine them into one file as MIRA currently reads only one input file per sequencing technology.

The *assemblies* directory finally contains sub-directories with different assembly trials I make. Every sub-directory is quickly set-up by creating it, linking data files from the `data` directory to it and then start MIRA. Continuing the example from above:

```
myproject$ cd assemblies
myproject/assemblies$ mkdir firstassembly
myproject/assemblies$ cd firstassembly
myproject/assemblies/firstassembly$ lndir ../../data
myproject/assemblies/firstassembly$ mira --project=...
```

That strategy keeps things nice and tidy in place and allows for a maximum flexibility while testing out a couple of settings.

8.7.2 PacBio only: Bacterial whole genome, 1Kb "paired-end", 10Kb insert size

Set up directories and fetch genome of Bacillus subtilis 168 from GenBank

```
$ mkdir bsubdemo1
$ cd bsubdemo1
bsubdemo1$ mkdir origdata data assemblies
bsubdemo1$ cd origdata
bsubdemo1/origdata$ wget ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/Bacillus_subtilis/ ↩
    AL009126.fna
bsubdemo1/origdata$ ls -l
-rw-r--r-- 1 bach bach 4275918 2010-06-06 00:34 AL009126.fna
```

After that, we'll prepare the the simulated PacBio data by running a script which creates paired reads like we would expect from a sequencing with PacBio with the following properties: DNA templates have 10k bases or more, we sequence the first 1000 bases in a strobe, let approximately 8000 bases pass, then sequence another 1000 bases.

```
bsubdemo1/origdata$ cd ../data
bsubdemo1/data$ fasta2frag.tcl  -l 1000 -i 230 -p 1 -insert_size 10000 -pairednaming 454 -P ↩
     0 -r 2 -infile ../origdata/AL009126.fna -outfile bs168pe1k_10k_in.pacbio.fasta
no ../origdata/AL009126.fna.qual
fragging gi|225184640|emb|AL009126.3|
bsubdemo1/data$ ls -l
-rw-r--r-- 1 bach bach  38971051 2010-06-06 18:27 bs168pe1k_10k_in.pacbio.fasta
-rw-r--r-- 1 bach bach   1642208 2010-06-06 18:27 bs168pe1k_10k_in.pacbio.fasta.bambus
-rw-r--r-- 1 bach bach   1440988 2010-06-06 18:27 bs168pe1k_10k_in.pacbio.fasta.pairs
-rw-r--r-- 1 bach bach 111214374 2010-06-06 18:27 bs168pe1k_10k_in.pacbio.fasta.qual
```

The command line given above will create an artificial data set with equally distributed PacBio "paired-end" reads with an average coverage of 8.6 across the genome. Note that the `*.bambus` and `*.pairs` files are not needed for MIRA, but the Tcl script generates these for some other use cases.

Next, we move to the assembly directory, make a new one to run a first assembly and link all the needed input files for MIRA into this new directory:

```
bsubdemo1/data$ cd ../assemblies
bsubdemo1/assemblies$ mkdir firsttest
bsubdemo1/assemblies$ cd firsttest
bsubdemo1/assemblies/firsttest$ ln -s ../../data/bs168pe1k_10k_in.pacbio.fasta .
bsubdemo1/assemblies/firsttest$ ln -s ../../data/bs168pe1k_10k_in.pacbio.fasta.qual .
bsubdemo1/assemblies/firsttest$ ls -l
lrwxrwxrwx 1 bach bach 39 2010-06-06 01:01 bs168pe1k_10k_in.pacbio.fasta -> ../../data/ ↩
    bs168pe1k_10k_in.pacbio.fasta
lrwxrwxrwx 1 bach bach 44 2010-06-06 01:01 bs168pe1k_10k_in.pacbio.fasta.qual -> ../../data ↩
    /bs168pe1k_10k_in.pacbio.fasta.qual
```

We're all set up now, just need to start the assembly:

```
bsubdemo1/assemblies/firsttest$ mira
  --project=bs168pe1k_10k --job=genome,denovo,accurate,pacbio
  --notraceinfo -GE:not=4
  PACBIO_SETTINGS
```

```
  -GE:tpbd=1:tismin=9000:tismax=11000 -LR:rns=fr
  >&log_assembly.txt
```

The command above told MIRA

- the *name* (`bs168pe1k_10k`) you chose for your project. MIRA will search input files with this prefix as well as write output files and directories with that prefix.
- the *assembly job* MIRA should perform. In this case a de-novo genome assembly at accurate level with PacBio data.
- some additional information that MIRA should not search for additional ancillary information in NCBI TRACEINFO XML files
- the *number of threads* which MIRA should run at most in parallel.
- then tell MIRA that the following switches apply to reads in the assembly which are from Pacific Biosciences
- both reads of a PacBio read-pair should assemble in the same direction in a contig.. the minimum distance between the outer read ends should be at minimum 9000 bases and at maximum 11000 bases.
- the *read naming scheme* for a PacBio read-pair is "forward/reverse", i.e., the first read has ".f" appended to its name, the second read ".r".
- the standard output of MIRA should be *redirected* to a file name `log_assembly.txt`

Some 12 to 13 minutes later, the data set will be assembled. Though you should note that in real life projects with sequencing errors, MIRA will take perhaps 3 to 4 times longer. Have a look at the information files in directory `bs168pe1k_10k_assembly / bs168pe1k_10k_d_info/`, there especially to the files `bs168pe1k_10k_info_assembly.txt` and `bs168pe1k_10k_info_contigstats.txt` which give a first overview on how the assembly went.

In short: this assembly went -- unsurprisingly -- quite well: the complete chromosome of *Bacillus subtilis* 168 has been reconstructed into one complete contig. There are just two minor flaws disturbing just a little bit. First, a few (twelve) repetitive reads could not be placed and form a second small contig of 2Kb. Second, the reconstructed chromosome contains 4 single-base differences with respect to the original Bsub chromosome. It is an exercise left to the reader to find out that this is due to almost identical rRNA repeats where two almost adjacent elements lie within the expected template insert size of the simulated PacBio Reads and therefore troubled the assembler a bit.

Your next stop would then be the directory `bs168pe1k_10k_assembly / bs168pe1k_10k_d_results/` which contains the assembly results in all kind of formats. If a format you need is missing, have a look at **convert_project** from the MIRA package, it may be that the format you need can be generated with it.

8.7.3 PacBio only: Bacterial whole genome, 3Kb strobed bases (6Kb with elastic dark inserts)

Set up directories and fetch genome of Bacillus subtilis 168 from GenBank

```
$ mkdir bsubdemo2
$ cd bsubdemo2
bsubdemo2$ mkdir origdata data assemblies
bsubdemo2$ cd origdata
bsubdemo2/origdata$ wget ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/Bacillus_subtilis/ ↩
    AL009126.fna
bsubdemo2/origdata$ ls -l
-rw-r--r-- 1 bach bach 4275918 2010-06-06 00:34 AL009126.fna
```

After that, we'll prepare the the simulated PacBio data by running a script which creates strobed reads like we would expect from a sequencing with PacBio with the following properties: DNA templates are 6k bases or more, we sequence the first ~100 bases in a strobe, let approximately 100 bases pass, and repeat until we have 3000 bases in strobes.

```
bsubdemo2/origdata$ cd ../data
bsubdemo2/data$ fasta2frag.tcl -l 3000 -i 150 -r 2 -s 1 -strobeon 100 -strobeoff 100 - ↩
    infile ../origdata/AL009126.fna -outfile bs168_3ks_100_100_in.pacbio.fasta
no ../origdata/AL009126.fna.qual
fragging gi|225184640|emb|AL009126.3|
bsubdemo2/data$ ls -l
-rw-r--r-- 1 bach bach 166909136 2010-06-06 19:18 bs168_3ks_100_100_in.pacbio.fasta
-rw-r--r-- 1 bach bach 416614472 2010-06-06 19:18 bs168_3ks_100_100_in.pacbio.fasta.qual
```

The command line given above will create an artificial data set with equally distributed PacBio strobed reads with an average coverage of ~20 across the genome, of which only half is filled with sequence data so that the "real" coverage is ~10.

Next, we move to the assembly directory, make a new one to run a first assembly and link all the needed input files for MIRA into this new directory:

```
bsubdemo2/data$ cd ../assemblies
bsubdemo2/assemblies$ mkdir firsttest
bsubdemo2/assemblies$ cd firsttest
bsubdemo2/assemblies/firsttest$ ln -s ../../data/bs168_3ks_100_100_in.pacbio.fasta .
bsubdemo2/assemblies/firsttest$ ln -s ../../data/bs168_3ks_100_100_in.pacbio.fasta.qual .
bsubdemo2/assemblies/firsttest$ ls -l
lrwxrwxrwx 1 bach bach 39 2010-06-06 01:01 bs168_3ks_100_100_in.pacbio.fasta -> ../../data/ ↩
    bs168_3ks_100_100_in.pacbio.fasta
lrwxrwxrwx 1 bach bach 44 2010-06-06 01:01 bs168_3ks_100_100_in.pacbio.fasta -> ../../data/ ↩
    bs168_3ks_100_100_in.pacbio.fasta
```

We're all set up now, just need to start the assembly:

```
bsubdemo1/assemblies/firsttest$ mira
  --project=bs168_3ks_100_100 --job=genome,denovo,accurate,pacbio
  --notraceinfo --noclipping
  -GE:not=4
  -GO:mr=no
  PACBIO_SETTINGS
  -AL:egp=no
  >&log_assembly.txt
```

The command above told MIRA

- the *name* (`bs168_3ks_100_100`) you chose for your project. MIRA will search input files with this prefix as well as write output files and directories with that prefix.
- the *assembly job* MIRA should perform. In this case a de-novo genome assembly at accurate level with PacBio data.
- some additional information that MIRA should not search for additional ancillary information in NCBI TRACEINFO XML files
- the *number of threads* which MIRA should run at most in parallel.
- a MIRA parameter called *mark repeats* should be switched off for PacBio reads. This is absolutely necessary when you have strobed reads with elastic dark inserts as MIRA otherwise gets somewhat confused due to alignment problems shown earlier in this guide.
- then tell MIRA that the following switches apply to reads in the assembly which are from Pacific Biosciences
- a MIRA parameter called *extra gap penalty* should be switched off for PacBio reads. This is necessary when you have strobed reads with elastic dark inserts as otherwise alignment problems with larger gaps lead to unnecessary rejection of alignments.
- the standard output of MIRA should be *redirected* to a file name `log_assembly.txt`

Wait for approximately 4.5hrs for MIRA to complete. Using elastic dark inserts is a pretty expensive feature from a computation perspective: all the passes and sub-passes of MIRA to move from an estimated length to an actually correct value means to build and break apart all the contigs and start from anew.

Bad news first: looking at the results and info directories, you will see that one single contig with a length of 4199898 bases was created. The original B. subtilis genome we used for this walkthrough is 4215426 bases, so it looks like some 15.5Kb are "missing." But, and this is the good news, the contig which was created represents the B. subtilis genome pretty faithfully: a check with MUMMER confirms that no misassemblies respectively re-ordering event of genome elements occurred.

Note
The following will need MUMMER3 installed on your system. Fetch it here: http://mummer.sourceforge.net/

```
bsubdemo2/assemblies/firsttest$ cd bs168_3ks_100_100_assembly/bs168_3ks_100_100_d_results
../bs168_3ks_100_100_d_results$ ls -l
-rw-r--r-- 1 bach bach 280894715 2010-06-08 04:53 bs168_3ks_100_100_out.ace
-rw-r--r-- 1 bach bach 776536315 2010-06-08 04:52 bs168_3ks_100_100_out.caf
-rw-r--r-- 1 bach bach 461365272 2010-06-08 04:52 bs168_3ks_100_100_out.maf
-rw-r--r-- 1 bach bach   4347658 2010-06-08 04:52 bs168_3ks_100_100_out.padded.fasta
-rw-r--r-- 1 bach bach  13040564 2010-06-08 04:52 bs168_3ks_100_100_out.padded.fasta.qual
-rw-r--r-- 1 bach bach 436189259 2010-06-08 04:53 bs168_3ks_100_100_out.tcs
-rw-r--r-- 1 bach bach   4269919 2010-06-08 04:52 bs168_3ks_100_100_out.unpadded.fasta
-rw-r--r-- 1 bach bach  12808422 2010-06-08 04:52 bs168_3ks_100_100_out.unpadded.fasta.qual
-rw-r--r-- 1 bach bach   3203036 2010-06-08 04:53 bs168_3ks_100_100_out.wig
../bs168_3ks_100_100_d_results$ nucmer -maxmatch -c 100 -p nucmer ../../../../origdata/ ↩
    AL009126.fna bs168_3ks_100_100_out.unpadded.fasta
1: PREPARING DATA
2,3: RUNNING mummer AND CREATING CLUSTERS
[... some lines left out ...]
4: FINISHING DATA
../bs168_3ks_100_100_d_results$ delta-filter -q -l 1000 nucmer.delta > nucmer.delta.q
../bs168_3ks_100_100_d_results$ show-coords -r -c -l nucmer.delta.q > nucmer.coords
../bs168_3ks_100_100_d_results$ cat nucmer.coords
NUCMER
    [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  |  [LEN R]  [LEN Q ↩
       ]  |  [COV R]  [COV Q]  | [TAGS]
==========================================================================================

     181  4215606  |  4199698        1  |  4215426  4199698  |    99.62  |  4215606   ↩
        4199898  |   100.00   100.00  | gi|225184640|emb|AL009126.3|       ↩
        bs168_3ks_100_100_c1
```

As already said: not 100% perfect on a base by base basis, but good enough for a using as reference sequence in subsequent mapping assemblies to get all the bases right.

8.7.4 Hybrid assemblies: PacBio plus Sanger/454/Solexa

TO BE EXPANDED: no real walkthrough yest, just a few hints.

- Prepare your PacBio data like explained in this guide.
- Prepare your other data (Sanger, 454, Solexa, or any combination of it) like explained in the respective MIRA guides.
- Start MIRA with, e.g., `--job=denovo,genome,accurate,pacbio,solexa` (and any other parameter you need) for a denovo genome assembly at accurate level with PacBio and Solexa data.

8.8 Known bugs in 3.4.0 with PacBio data

For error-corrected CLR data, MIRA does not get the average coverage of a project correct: it underestimates it, sometimes by a factor of 10. This in turn leads to too many "large" contigs and subsequently to a N50 number which is way off the thruth.

Will be fixed asap.

Chapter 9

Assembly of EST data with MIRA3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'Expect the worst. You'll never get disappointed. '

—Solomon Short

9.1 Introduction

This document is not complete yet and some sections may be a bit unclear. I'd be happy to receive suggestions for improvements.

Some reading requirements
This guide assumes that you have basic working knowledge of Unix systems, know the basic principles of sequencing (and sequence assembly) and what assemblers do.
Basic knowledge on mRNA transcription and EST sequences should also be present.
While there are step by step walkthroughs on how to setup your EST data and then perform assemblies regarding different requirements, this guide expects you to read at some point in time

- the *mira_usage* introductory help file so that you have a basic knowledge on how to set up projects in mira for Sanger sequencing projects.
- the *mira_454* introductory help file so that you have a basic knowledge on how to set up projects in mira for 454 sequencing projects.
- and last but not least the *mira_reference* help file to look up some command line options.

9.2 Preliminaries: on the difficulties of assembling ESTs

Assembling ESTs can be, from an assemblers point of view, pure horror. E.g., it may be that some genes have thousands of transcripts while other genes have just one single transcript in the sequenced data. Furthermore, the presence of 5' and 3' UTR, transcription variants, splice variants, homologues, SNPs etc.pp complicates the assembly in some rather interesting ways.

9.2.1 Poly-A tails in EST data

Poly-A tails are part of the mRNA and therefore also part of sequenced data. They can occur as poly-A or poly-T, depending from which direction and which part of the mRNA was sequenced. Having poly-A/T tails in the data is a something of a

double edged sword. More specifically., if the 3' poly-A tail is kept unmasked in the data, transcripts having this tail will very probably not align with similar transcripts from different splice variants (which is basically good). On the other hand, homopolymers (multiple consecutive bases of the same type) like poly-As are features that are pretty difficult to get correct with today's sequencing technologies, be it Sanger, Solexa or, with even more problems problems, 454. So slight errors in the poly-A tail could lead to wrongly assigned splice sites ... and wrongly split contigs.

This is the reason why many people cut off the poly-A tails. Which in turn may lead to transcripts from different splice variants being assembled together.

Either way, it's not pretty.

9.2.2 Lowly expressed transcripts

Single transcripts (or very lowly expressed transcripts) containing SNPs, splice variants or similar differences to other, more highly expressed transcripts are a problem: it's basically impossible for an assembler to distinguish them from reads containing junky data (e.g. read with a high error rate or chimeras). The standard setting of many EST assemblers and clusterers is therefore to remove these reads from the assembly set. MIRA handles things a bit differently: depending on the settings, single transcripts with sufficiently large differences are either treated as debris or can be saved as *singlet*.

9.2.3 Chimeras

Chimeras are sequences containing adjacent base stretches which are not occurring in an organism as sequenced, neither as DNA nor as (m)RNA. Chimeras can be created through recombination effects during library construction or sequencing. Chimeras can, and often do, lead to misassemblies of sequence stretches into one contig although they do not belong together. Have a look at the following example where two stretches (denoted by `x` and `o` are joined by a chimeric read *r4* containing both stretches:

```
r1 xxxxxxxxxxxxxxx
r2 xxxxxxxxxxxxxxxx
r3 xxxxxxxxxxxxxxx
r4 xxxxxxxxxxxxxxxxx|ooooooooooooo
r5                       ooooooooooo
r6                       ooooooooooo
r7                         ooooooooo
```

The site of the recombination event is denoted by `x|o` in read *r4*.

MIRA does have a chimera detection -- which works very well in genome assemblies due to high enough coverage -- by searching for sequence stretches which are not covered by overlaps. In the above example, the chimera detection routine will almost certainly flag read *r4* as chimera and only use a part of it: either the `x` or `o` part, depending on which part is longer. There is always a chance that *r4* is a valid read though, but that's a risk to take.

Now, that strategy would also work totally fine in EST projects if one would not have to account for lowly expressed genes. Imagine the following situation:

```
s1 xxxxxxxxxxxxxxx
s2         xxxxxxxxxxxxxxxxxxxxxxxx
s3                         xxxxxxxxxxxxxxx
```

Look at read *s2*; from an overlap coverage perspective, *s2* could also very well be a chimera, leading to a break of an otherwise perfectly valid contig if *s2* were cut back accordingly. This is why chimera detection is switched off by default in MIRA.

Warning
When starting an EST assembly via the `--job=est,...` switch, chimera detection is switched off by default. It is absolutely possible to switch on the SKIM chimera detection afterwards via [-CL:ascdc]. However, this will have exactly the effects described above: chimeras in higher coverage contigs will be detected, but perfectly valid low coverage contigs will be torn apart.
It is up to you to decide what you want or need.

9.2.4 Missing library normalisation: very highly expressed transcripts

Another interesting problem for de-novo assemblers are non-normalised EST libraries. In each cell, the number of mRNA copies per gene may differ by several orders of magnitude, from a single transcripts to several tens of thousands. Pre-sequencing normalisation is a wet-lab procedure to approximately equalise those copy numbers. This can however, introduce other artifacts.

If an assembler is fed with non-normalised EST data, it may very well be that an overwhelming number of the reads comes only from a few genes (house-keeping genes). In Sanger sequencing projects this could mean a couple of thousand reads per gene. In 454 sequencing projects, this can mean several tens of thousands of reads per genes. With Solexa data, this number can grow to something close to a million.

Several effects then hit a de-novo assembler, the three most annoying being (in ascending order of annoyance): a) non-random sequencing errors then look like valid SNPs, b) sequencing and library construction artefacts start to look like valid sequences if the data set was not cleaned "enough" and more importantly, c) an explosion in time and memory requirements when attempting to deliver a "good" assembly. A sure sign of the latter are messages from MIRA about *megahubs* in the data set.

Note The guide on how to tackle *hard* projects with MIRA gives an overview on how to hunt down sequences which can lead to the assembler getting confused, be it sequencing artefacts or highly expressed genes.

9.3 Preprocessing of ESTs

With contributions from Katrina Dlugosch

EST sequences necessarily contain fragments of vectors or primers used to create cDNA libraries from RNA, and may additionally contain primer and adaptor sequences used during amplification-based library normalisation and/or high-throughput sequencing. These contaminant sequences need to be removed prior to assembly. MIRA can trim sequences by taking contaminant location information from a SSAHA2 or SMALT search output, or users can remove contaminants beforehand by trimming sequences themselves or masking unwanted bases with lowercase or other characters (e.g. 'x', as with **cross_match**). Many folks use preprocessing trimming/masking pipelines because it can be very important to try a variety of settings to verify that you've removed all of your contaminants (and fragments thereof) before sending them into an assembly program like MIRA. It can also be good to spend some time seeing what contaminants are in your data, so that you get to know what quality issues are present and how pervasive.

Two features of next generation sequencing can introduce errors into contaminant sequences that make them particularly difficult to remove, arguing for preprocessing: First, most next-generation sequence platforms seem to be sensitive to excess primers present during library preparation, and can produce a small percentage of sequences composed entirely of concatenated primer fragments. These are among the most difficult contaminants to remove, and the program TagDust (http://genome.gsc.riken.jp/osc/english/dataresource/) was recently developed specifically to address this problem. Second, 454 EST data sets can show high variability within primer sequences designed to anchor to polyA tails during cDNA synthesis, because 454 has trouble calling the length of the necessary A and T nucleotide repeats with accuracy.

A variety of programs exist for preprocessing. Popular ones include cross_match (http://www.phrap.org/phredphrapconse
html) for primer masking, and SeqClean (http://compbio.dfci.harvard.edu/tgi/software/), Lucy (http://lucy.sourceforge.net/), and SeqTrim (http://www.scbi.uma.es/cgi-bin/seqtrim/seqtrim_login.cgi) for both primer and polyA/T trimming. The pipeline SnoWhite (http://evopipes.net) combines Seqclean and TagDust with custom scripts for aggressive sequence and polyA/T trimming (and is tolerant of data already masked using cross_match). In all cases, the user must provide contaminant sequence information and adjust settings for how sensitive the programs should be to possible matches. To find the best settings, it is helpful to look directly at some of the sequences that are being trimmed and inspect them for remaining primer and/or polyA/T fragments after cleaning.

Warning When using **mira** or **miraSearchESTSNPs** with the the simplest parameter calls (using the "--job=..." quick switches), the default settings used include pretty heavy sequence pre-processing to cope with noisy data. Especially if you have your own pre-processing pipeline, you *must* then switch off different clip algorithms that you might have applied previously yourself. Especially poly-A clips should never be run twice (by your pipeline and by **mira**) as they invariably lead to too many bases being cut away in some sequences,

Note Here too: In some cases MIRA can get confused if something with the pre-processing went wrong because, e.g., unexpected sequencing artefacts like unknown sequencing vectors or adaptors remain in data. The guide on how to tackle *hard* projects with MIRA gives an overview on how to hunt down sequences which can lead to the assembler getting confused, be it sequencing artefacts or highly expressed genes.

9.4 The difference between *assembly* and *clustering*

MIRA in its base settings is an *assembler* and not a *clusterer*, although it can be configured as such. As assembler, it will split up read groups into different contigs if it thinks there is enough evidence that they come from different RNA transcripts.

9.4.1 Splitting transcripts into contigs based on SNPs

Imagine this simple case: a gene has two slightly different alleles and you've sequenced this:

```
A1-1  ..........T..........
A1-2  ..........T..........
A1-3  ..........T..........
A1-4  ..........T..........
A1-5  ..........T..........
B2-1  ..........G..........
B2-2  ..........G..........
B2-3  ..........G..........
B2-4  ..........G..........
```

Depending on base qualities and settings used during the assembly like, e.g., [-CO:mr:mrpg:mnq:mgqrt:emea:amgb] MIRA will recognise that there's enough evidence for a T and also enough evidence for a G at that position and create two contigs, one containing the "T" allele, one the "G". The consensus will be >99% identical, but not 100%.

Things become complicated if one has to account for errors in sequencing. Imagine you sequenced the following case:

```
A1-1  ..........T..........
A1-2  ..........T..........
A1-3  ..........T..........
A1-4  ..........T..........
A1-5  ..........T..........
B2-1  ..........G..........
```

It shows very much the same like the one from above, except that there's only one read with a "G" instead of 4 reads. MIRA will, when using standard settings, treat this as erroneous base and leave all these reads in a contig. It will likewise also not mark it as SNP in the results. However, this could also very well be a lowly expressed transcript with a single base mutation. It's virtually impossible to tell which of the possibilities is right.

Note You can of course force MIRA to mark situations like the one depicted above by, e.g., changing the parameters for [-CO:mrpg:mnq:mgqrt]. But this may have the side-effect that sequencing errors get an increased chance of getting flagged as SNP.

Further complications arise when SNPs and potential sequencing errors meet at the same place. consider the following case:

```
A1-1  ..........T..........
A1-2  ..........T..........
A1-3  ..........T..........
A1-4  ..........T..........
B1-5  ..........T..........
B2-1  ..........G..........
B2-2  ..........G..........
```

```
B2-3  ..........G..........
B2-4  ..........G..........
E1-1  ..........A..........
```

This example is exactly like the first one, except an additional read `E1-1` has made it's appearance and has an "A" instead of a "G" or "T". Again it is impossible to tell whether this is a sequencing error or a real SNP. MIRA handles these cases in the following way: it will recognise two valid read groups (one having a "T", the other a "G") and, in assembly mode, split these two groups into different contigs. It will also play safe and define that the single read `E1-1` will not be attributed to either one of the contigs but, if it cannot be assembled to other reads, form an own contig ... if need to be even only as single read (a *singlet*).

Note Depending on some settings, singlets may either appear in the regular results or end up in the debris file.

9.4.2 Splitting transcripts into contigs based on larger gaps

Gaps in alignments of transcripts are handled very cautiously by MIRA. The standard settings will lead to the creation of different contigs if three or more consecutive gaps are introduced in an alignment. Consider the following example:

```
A1-1  ..........CGA..........
A1-2  ..........*GA..........
A1-3  ..........**A..........
B2-1  ..........***..........
B2-2  ..........***..........
```

Under normal circumstances, MIRA will use the reads `A1-1`, `A1-2` and `A1-3` to form one contig and put `B2-1` and `B2-2` into a separate contig. MIRA would do this also if there were only one of the B2 reads.

The reason behind this is that the probability for having gaps of three or more bases only due to sequencing errors is pretty low. MIRA will therefore treat reads with such attributes as coming from different transcripts and not assemble them together, though this can be changed using the [-AL:egp:egpl] parameters of MIRA if wanted.

Problems with homopolymers, especially in 454 sequencing
As 454 sequencing has a general problem with homopolymers, this rule of MIRA will sometimes lead formation of more contigs than expected due to sequencing errors at "long" homopolymer sites ... where long starts at ~7 bases. Though MIRA does know about the problem in 454 homopolymers and has some routines which try to mitigate the problem. this is not always successful.

9.5 mira and miraSearchESTSNPs

The assembly of ESTS can be done in two ways when using the MIRA3 system: by using mira or miraSearchESTSNPs.

If one has data from only one strain, mira using the "--job=est" quickmode switch is probably the way to go as it's easier to handle.

For data from multiple strains where one wants to search SNPs, miraSearchESTSNPs is the tool of choice. It's an automated pipeline that is able to assemble transcripts cleanly according to given organism strains. Afterwards, an integrated SNP analysis highlights the exact nature of mutations within the transcripts of different strains.

9.5.1 Using mira for EST assembly

Using mira in EST projects is quite useful to get a first impression of a given data set or when used in projects that have no strain or only one strain.

It is recommended to use 'est' in the [-job=] quick switch to get a good initial settings default and then eventually adapt with own settings.

Note that by their nature, single transcripts end up in the debris file as they do not match any other reads and therefore cannot be aligned.

An interesting approach to find differences in multiploid genes is to use the result of an "mira --job=est ..." assembly as input for the third step of the miraSearchESTSNPs pipeline.

9.5.2 Using mira for EST clustering

Like for EST assembly, it is recommended to use 'est' in the [-job=] quick switch to get a good initial settings default. Then however, one should adapt a couple of switches to get a clustering like alignment:

-AL:egp=no switching off extra gap penalty in alignments allows assembly of transcripts having gap differences of more than 3 bases

-AL:egpl=... In case [-AL:egp] is not switched off, the extra gap penalty level can be fine tuned here.

-AL:megpp=... In case [-AL:egp] is not switched off, the maximum extra gap penalty in percentage can be fine tuned here. This allows, together with [-AL:egpl] (see below), to have MIRA accept alignments which are two or three bases longer than the 3 bases rejection criterion of the standard [-AL:egpl=split_on_codongaps] in EST assemblies.

-CO:asir=yes This forces MIRA to assume that valid base differences (occurring in several reads) in alignments are SNPs and not repeats/marker bases for different variants. Note that depending on whether you have only one or several strains in your assembly, you might want to enable or disable this feature to allow/disallow clustering of reads from different strains.

-CO:mrpg:mnq:mgqrt With these three parameters you can adjust the sensitivity of the repeat / SNP discovery algorithm.

-AL:mrs=... When [-CO:asir=no] and [-AL:egp=no], MIRA has lost two of its most potent tools to not align complete nonsense. In those cases, you should increase the minimum relative score allowed in Smith-Waterman alignments to levels which are higher than the usual MIRA standards. 90 or 95 might be a good start for testing.

-CO:rodirs=... Like [-AL:mrs] above, [-CO:rodirs] is a fallback mechanism to disallow building of completely non-sensical contigs when [-CO:asir=no] and [-AL:egp=no]. You should decrease [-CO:rodirs] to anywhere between 10 and 0.

Please look up the complete description of the above mentioned parameters in the MIRA reference manual, they're listed here just with the *why* one should change them for a clustering assembly.

Note Remember that some of the parameters above can be set independently for reads of different sequencing technologies. E.g., when assembling EST sequences from *Sanger* and *454* sequencing technologies, it is absolutely possible to allow the 454 sequences from having large gaps in alignments (to circumvent the homopolymer problem), but to disallow Sanger sequences from having them. The parameters would need be set like this:

```
$ mira [...] --job=est,... [...]
  SANGER_SETTINGS -AL:egp=yes:egpl=split_on_codongaps
  454_SETTINGS -AL:egp=no
```

or in shorter form (as `--job=est` already presets `-AL:egp=yes:egpl=split_on_codongaps` for all technologies):

```
$ mira [...] --job=est,... [...]
  454_SETTINGS -AL:egp=no
```

9.5.3 Using miraSearchESTSNPs for EST assembly

miraSearchESTSNPs is a pipeline that reconstructs the pristine mRNA transcript sequences gathered in EST sequencing projects of more than one strain, which can be a reliable basis for subsequent analysis steps like clustering or exon analysis. This means that even genes that contain only one transcribed SNP on different alleles are first treated as different transcripts. The optional last step of the assembly process can be configured as a simple clusterer that can assemble transcripts containing the same exon sequence -- but only differ in SNP positions -- into one consensus sequence. Such SNPs can then be analysed, classified and reliably assigned to their corresponding mRNA transcriptome sequence. However, it is important to note that miraSearchESTSNPs is an assembler and not a full blown clustering tool.

Generally speaking, miraSearchESTSNPs is a three-stage assembly system that was designed to catch SNPs in different strains and reconstruct the mRNA present in those strains. That is, one really should have different strains to analyse (and the information provided to the assembler) to make the most out of miraSearchESTSNPs. Here is a quick overview on what miraSearchESTSNPs does:

1. Step 1: assemble everything together, not caring about strain information. Potential SNPs are not treated as SNPs, but as possible repeat marker bases and are tagged as such (temporarily) to catch each and every possible sequence alignment which might be important later. As a result of this stage, the following information is written out:

   (a) Into `step1_snpsinSTRAIN_<strainname>.caf` all the sequences of a given strain that are in contigs (can be aligned with at least one other sequence) - also, all sequences that are singlets BUT have been tagged previously as containing tagged bases showing that they aligned previously (even to other strains) but were torn apart due to the SNP bases.

   (b) Into `step1_nosnps_remain.caf` all the remaining singlets.

   Obviously, if one did not provide strain information to the assembly of step 1, all the sequences belong to the same strain (named *"default"*). The CAF files generated in this step are the input sequences for the next step.

   Note If you want to apply clippings to your data (poly-A/T or reading clipping information from SSAHA2 or SMALT), then do this only in step 1! Do not try to re-appply them in step 2 or 3 (or only if you think you have very good reasons to do so. Once loaded and/or applied in step 1, the clipping information is carried on by MIRA to steps 2 and 3.

   ---

2. Step 2: Now, miraSearchESTSNPs assembles each strain independently from each other. Again, sequences containing SNPs are torn apart into different contigs (or singlets) to give a clean representation of the "really sequenced" ESTs. In the end, each of the contigs (or singlets) coming out of the assemblies for the strains is a representation of the mRNA that was floating around the given cell/strain/organism. The results of this step are written out into one big file (`step2_reads.caf`) and a new straindata file that goes along with those results (`step2_straindata.txt`).

3. Step 3: miraSearchESTSNPs takes the result of the previous step (which should now be clean transcripts) and assembles them together, *this time* allowing transcripts from different strains with different SNP bases to be assembled together. The result is then written to `step3_out.*` files and directories.

miraSearchESTSNPs can also be used for EST data of a single strain or when no strain information is available. In this case, it will cleanly sort out transcripts of almost identical genes or, when eukaryotic ESTs are assembled, according to their respective allele when these contain mutations.

Like the normal mira, miraSearchESTSNPs keeps track on a lot of things and writes out quite a lot of additional information files after each step. Results and and additional information of step 1 are stored in `step1_*` directories. Results and information of step 2 are in `<strainname>_*` directories. For step 3, it's `step3_*` again.

Each step of miraSearchESTSNPs can be configured exactly like mira via command line parameters.

The pipeline of miraSearchESTSNPs is almost as flexible as mira itself: if the defaults set by the quick switches are not right for your use case, you can change about any parameter you wish via the command line. There are only two things which you need to pay attention to

1. a straindata file must be present for step 1 (`*_straindata_in.txt`), but it can very well be an empty file.

2. the naming of the result files is fixed (for all three steps), you cannot change it.

9.6 Walkthroughs

These walkthroughs use "msd" as project name (acronym for My Simple Dataset), please replace that with your own project name according to the MIRA naming convention.

9.6.1 mira with "--job=est"

9.6.1.1 Example: One strain, Sanger without vectors and no XML

Given is just a FASTA and FASTA quality file, where the Sanger sequencing vector sequences and problematic things (like bad quality) have been either completely removed from the data or were masked with "X". Apart from that, no further processing (poly-A removal etc.) was done. Your directory looks like this:

```
bach@arcadia:$ ls -l
-rwxr--r-- 1 bach bach 15486163 2009-02-22 21:01 msd_in.sanger.fasta
-rwxr--r-- 1 bach bach 38017687 2009-02-22 21:01 msd_in.sanger.fasta.qual
```

Then, use this command:

```
$ mira --project=msd
  --job=denovo,est,accurate,sanger
  SANGER_SETTINGS
  -CL:qc=no
  >& log_assembly.txt
```

We switch off the Sanger quality clips because bad quality is already trimmed away by your pipeline.

9.6.1.2 Example: One strain, 454 with XML ancillary data

Like above, but this time 454 sequencing and the FASTA files contain everything (including remaining adaptors and bad quality), but there's a XML with ancillary data which contains all necessary clips (like generated by, e.g., **sff_extract**):

```
bach@arcadia:$ ls -l
-rwxr--r-- 1 bach bach 15486163 2009-02-22 21:01 msd_in.454.fasta
-rwxr--r-- 1 bach bach 38017687 2009-02-22 21:01 msd_in.454.fasta.qual
-rwxr--r-- 1 bach bach 10433244 2009-02-22 21:01 msd_traceinfo_in.454.xml
```

Then, use this command:

```
bach@arcadia:$  mira --project=msd
  --job=denovo,est,accurate,454
  454_SETTINGS
  -CL:qc=no
  >& log_assembly.txt
```

We just switch off our quality clip for 454 (and load the quality clips from the XML), poly-A removal is performed by MIRA. Loading of TRACEINFO XML data must not be switched on as it's the default for 454 data.

9.6.1.3 Example: One strain, 454 with XML ancillary data, poly-A already removed.

Like above, but this time the data was pre-processed by another program to mask the poly-A stretches with X:

```
bach@arcadia:$ ls -l
-rwxr--r-- 1 bach bach 15486163 2009-02-22 21:01 msd_in.454.fasta
-rwxr--r-- 1 bach bach 38017687 2009-02-22 21:01 msd_in.454.fasta.qual
-rwxr--r-- 1 bach bach 10433244 2009-02-22 21:01 msd_traceinfo_in.454.xml
```

Then, use this command:

```
bach@arcadia:$ mira --project=msd
  --job=denovo,est,accurate,454
  454_SETTINGS
  -CL:qc=no:cpat=no
  >& log_assembly.txt
```

We just switch off our quality clip (and load the quality clips from the XML) and also switch off poly-A clipping. Remember, never perform poly-A/T clipping twice on a data set.

9.6.1.4 Example: Two strains, 454 with XML ancillary data, poly-A already removed.

Like above, but this time we assign reads to different strains. This can happen either by putting the strain information into the XML file (using the `strain` field of the NCBI TRACEINFO format definition) or by using a two column, tab-delimited file which mira loads on request.

As written. when using XML no change to the command line from the last example would be needed. This example uses the extra file with strain information. The file `msd_straindata_in.txt` contains key value pair information on the relationship of reads to strains and looks like this (gnlti* are name of reads):

```
bach@arcadia:$ cat msd_straindata_in.454.txt
gnlti136478626 tom
gnlti136479357 tom
gnlti136479063 tom
gnlti136478624 jerry
gnlti136479522 jerry
gnlti136477918 jerry
```

Then, use this command (note the additional [-LR:lsd] option):

```
bach@arcadia:$ mira --project=msd
  --job=denovo,est,accurate,454
  454_SETTINGS
  -LR:lsd=yes
  -CL:qc=no:cpat=no
  >& log_assembly.txt
```

9.6.2 miraSearchESTSNPs

9.6.2.1 Example: Two strains, Sanger with masked sequences, no XML

Given just a FASTA and FASTA quality file, where the Sanger sequencing vectors and all sequencing related things (like bad quality) have been either completely removed from the data or were masked with "X". Apart from that, no further processing (poly-A removal etc.) was done.

You have *n* strains (in this example *n*=2) called "tom" and "jerry"

Your directory looks like this:

```
bach@arcadia:$ ls -l
-rw-r--r-- 1 bach bach  5276 2009-02-22 21:23 msd_in.sanger.fasta
-rw-r--r-- 1 bach bach 13827 2009-02-22 21:23 msd_in.sanger.fasta.qual
-rw-r--r-- 1 bach bach   120 2009-02-22 21:27 msd_straindata_in.txt
```

The file `msd_straindata_in.txt` contains key value pair information on the relationship of reads to strains and looks like this (gnlti* are name of reads):

```
bach@arcadia:$ cat msd_straindata_in.txt
gnlti136478626 tom
gnlti136479357 tom
gnlti136479063 tom
gnlti136478624 jerry
gnlti136479522 jerry
gnlti136477918 jerry
```

To assemble, use this:

```
bach@arcadia:$ miraSearchESTSNPs
  --project=msd
  --job=denovo,accurate,sanger,esps1
  >&log_assembly_esps1.txt
```

Note that the results of this first step are in sub-directories prefixed with "step1".

When the first step finished, continue with this (note that no "--project" is given here):

```
bach@arcadia:$ miraSearchESTSNPs
  --job=denovo,accurate,esps2
  >&log_assembly_esps2.txt
```

Note that the results of this second step are in sub-directories prefixed with "tom", "jerry" and "remain". You will find in each directory the clean transcripts from every strain/organism.

To see which SNPs exist between both "tom" and "jerry", launch the third step:

```
bach@arcadia:$ miraSearchESTSNPs
  --job=denovo,accurate,esps3
  >&log_assembly_esps3.txt
```

Note that the results of this third step are in sub-directories prefixed with "step3".

In the `step3_d_results` directory for example, you can transform the CAF file into a gap4 database and then look at the SNPs searching for the tags SROr, SIOr and SAOr.

9.7 Solving common problems of EST assemblies

... continue here ...

Megahubs => track down reason (high expr, seqvec or adaptor: see mira_hard) and eliminate it

Chapter 10

Working with the results of MIRA

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'You have to know what you're looking for before you can find it. '

—Solomon Short

MIRA makes results available in quite a number of formats: CAF, ACE, FASTA and a few others. The preferred formats are CAF and MAF, as these format can be translated into any other supported format.

10.1 MIRA output directories and files

For the assembly MIRA creates a directory named `projectname_assembly` in which a number of sub-directories will have appeared.

Note The `projectname` is determined by the mira parameter `--project=...` or, if used, the specific `--proout=...` parameter.

These sub-directories (and files within) contain the results of the assembly itself, general information and statistics on the results and -- if not deleted automatically by MIRA -- a tmp directory with log files and temporary data:

- `projectname_d_results`: this directory contains all the output files of the assembly in different formats.
- `projectname_d_info`: this directory contains information files of the final assembly. They provide statistics as well as, e.g., information (easily parseable by scripts) on which read is found in which contig etc.
- `projectname_d_tmp`: this directory contains log files and temporary assembly files. It can be safely removed after an assembly as there may be easily a few GB of data in there that are not normally not needed anymore.

  The default settings of MIRA are such that really big files are automatically deleted when they not needed anymore during an assembly.
- `projectname_d_chkpt`: this directory contains checkpoint files needed to resume assemblies that crashed or were stopped (not implemented yet, but soon)

10.1.1 The `*_d_results` directory

The following files in `projectname_d_results` contain results of the assembly in different formats. Depending on the output options of MIRA, some files may or may not be there. As long as the CAF or MAF format are present, you can translate your assembly later on to about any supported format with the **convert_project** program supplied with the MIRA distribution:

- `projectname_out.txt`: this file contains in a human readable format the aligned assembly results, where all input sequences are shown in the context of the contig they were assembled into. This file is just meant as a quick way for people to have a look at their assembly without specialised alignment finishing tools.
- `projectname_out.padded.fasta`: this file contains as FASTA sequence the consensus of the contigs that were assembled in the process. Positions in the consensus containing gaps (also called 'pads', denoted by an asterisk) are still present. The computed consensus qualities are in the corresponding `projectname_out.padded.fasta.qual` file.
- `projectname_out.unpadded.fasta`: as above, this file contains as FASTA sequence the consensus of the contigs that were assembled in the process, put positions in the consensus containing gaps were removed. The computed consensus qualities are in the corresponding `projectname_out.unpadded.fasta.qual` file.
- `projectname_out.caf`: this is the result of the assembly in CAF format, which can be further worked on with, e.g., tools from the *caftools* package from the Sanger Centre and later on be imported into, e.g., the Staden gap4 assembly and finishing tool.
- `projectname_out.ace`: this is the result of the assembly in ACE format. This format can be read by viewers like the TIGR clview or by consed from the phred/phrap/consed package.
- `projectname_out.gap4da`: this directory contains the result of the assembly suited for the *direct assembly* import of the Staden gap4 assembly viewer and finishing tool.

10.1.2 The `*_d_info` directory

The following files in `projectname_info` contain statistics and other information files of the assembly:

- `projectname_info_assembly.txt`: This file should be your first stop after an assembly. It will tell you some statistics as well as whether or not problematic areas remain in the result.
- `projectname_info_callparameters.txt`: This file contains the parameters as given on the mira command line when the assembly was started.
- `projectname_info_consensustaglist.txt`: This file contains information about the tags (and their position) that are present in the consensus of a contig.
- `projectname_info_contigreadlist.txt`: This file contains information which reads have been assembled into which contigs (or singlets).
- `projectname_info_contigstats.txt`: This file contains in tabular format statistics about the contigs themselves, their length, average consensus quality, number of reads, maximum and average coverage, average read length, number of A, C, G, T, N, X and gaps in consensus.
- `projectname_info_debrislist.txt`: This file contains the names off all the reads which were not assembled into contigs (or singlets if appropriate MIRA parameters were chosen).
- `projectname_info_readrepeats`: This file helps to find out which parts of which reads are quite repetitive in a project. Please consult the chapter on how to tackle "hard" sequencing projects to learn how this file can help you in spotting sequencing mistakes and / or difficult parts in a genome or EST / RNASeq project.
- `projectname_info_readstooshort`: A list containing the names of those reads that have been sorted out of the assembly only due to the fact that they were too short, before any processing started.
- `projectname_info_readtaglist.txt`: This file contains information about the tags and their position that are present in each read. The read positions are given relative to the forward direction of the sequence (i.e. as it was entered into the the assembly).
- `projectname_error_reads_invalid`: A list of sequences that have been found to be invalid due to various reasons (given in the output of the assembler).

10.2 First look: the assembly info

Once finished, have a look at the file `*_info_assembly.txt` in the info directory. The assembly information given there is split in three major parts:

1. some general assembly information (number of reads assembled etc.). This part is quite short at the moment, will be expanded in future
2. assembly metrics for 'large' contigs.
3. assembly metrics for all contigs.

The first part for large contigs contains several sections. The first of these shows what MIRA counts as large contig for this particular project. As example, this may look like this:

```
Large contigs:
--------------
With    Contig size             >= 500
        AND (Total avg. Cov     >= 19
             OR Cov(san)        >= 0
             OR Cov(454)        >= 8
             OR Cov(pbs)        >= 0
             OR Cov(sxa)        >= 11
             OR Cov(sid)        >= 0
            )
```

The above is for a 454 and Solexa hybrid assembly in which MIRA determined large contigs to be contigs

1. of length of at least 500 bp and
2. having a total average coverage of at least 19x or an average 454 coverage of 8 or an average Solexa coverage of 11

The second section is about length assessment of large contigs:

```
  Length assessment:
  ------------------
  Number of contigs:    44
  Total consensus:      3567224
  Largest contig:       404449
  N50 contig size:      186785
  N90 contig size:      55780
  N95 contig size:      34578
```

In the above example, 44 contigs totalling 3.56 megabases were built, the largest contig being 404 kilobases long and the N50/N90 and N95 numbers give the respective lengths.

The next section shows information about the coverage assessement of large contigs. An example:

```
  Coverage assessment:
  --------------------
  Max coverage (total): 563
  Max coverage
        Sanger: 0
        454:    271
        PacBio: 0
        Solexa: 360
        Solid:  0
  Avg. total coverage (size >= 5000): 57.38
  Avg. coverage (contig size >= 5000)
        Sanger: 0.00
```

```
        454:    25.10
        PacBio: 0.00
        Solexa: 32.88
        Solid:  0.00
```

Maximum coverage attained was 563, maximum for 454 alone 271 and for Solexa alone 360. The average total coverage (computed from contigs with a size $\geq$ 5000 bases is 57.38. The average coverage by sequencing technology (in contigs $\geq$ 5000) is 25.10 for 454 and 32.88 for Solexa reads.

Note The value for "Avg. total coverage (size >= 5000)" is currently always calculated for contig having 5000 or mor consensus bases. While this gives a very effective measure for genome assemblies, EST assemblies will often have totally irrelevant values here as most genes in eukaryotes (and prokaryotes) tend to be smaller than 5000 bases.

The last section contains some numbers useful for quality assessment. It looks like this:

```
Quality assessment:
-------------------
Average consensus quality:                    90
Consensus bases with IUPAC:                   11      (you might want to check these)
Strong unresolved repeat positions (SRMc):    0       (excellent)
Weak unresolved repeat positions (WRMc):      19      (you might want to check these)
Sequencing Type Mismatch Unsolved (STMU):     0       (excellent)
Contigs having only reads wo qual:            0       (excellent)
Contigs with reads wo qual values:            0       (excellent)
```

Beside the average quality of the contigs and whether they contain reads without quality values, MIRA shows the number of different tags in the consensus which might point at problems.

The above mentioned sections (length assessemnt, coverage assessment and quality assessment) for *large* contigs will then be re-iterated for *all* contigs, this time including also contigs which MIRA did not take into account as large contig.

10.3 Converting results

10.3.1 Converting to and from gap4

The **gap4** program from the Staden package is a pretty useful finishing tool and assembly viewer. It has an own database format which MIRA does not read or write, but there are interconversion possibilities using the CAF format and the **caf2gap** and **gap2caf** utilities.

Conversion is pretty straightforward. From MIRA to gap4, it's like this:

```
$ caf2gap -project YOURGAP4PROJECTNAME -ace mira_result.caf >&/dev/null
```

Note Don't be fooled by the `-ace` parameter of **caf2gap**. It needs a CAF file as input, not an ACE file.

From gap4 to CAF, it's like this:

```
$ gap2caf -project YOURGAP4PROJECTNAME >tmp.caf
$ convert_project -f caf -t caf -r c tmp.caf somenewname
```

Note Using **gap2caf**, be careful to use the simple > redirection to file and *not* the `>&` redirection.

Note Using first **gap2caf** and then **convert_project** is needed as gap4 writes an own consensus to the CAF file which is not necessarily the best. Indeed, gap4 does not know about different sequencing technologies like 454 and treats everything as Sanger. Therefore, using **convert_project** with the [-r c] option recalculates a MIRA consensus during the "conversion" from CAF to CAF.

10.3.2 Converting to and from gap5

The **gap5** program is the successor for gap4. It comes with on own import utility (**tg_index**) which can read CAF files, and gap5 itself can export to CAF.

Conversion is pretty straightforward. From MIRA to gap5, it's like this:

```
$ tg_index -C YOURGAP4PROJECTNAME_out.caf
```

This creates a gap5 database named `YOURGAP4PROJECTNAME_out.g5d` which can be directly loaded with gap5 like this:

```
$ gap5 YOURGAP4PROJECTNAME_out.g5d
```

10.3.3 Converting to and from other formats: convert_project

convert_project is tool in the MIRA package which reads and writes a number of formats, ranging from full assembly formats like CAF and MAF to simple output view formats like HTML or plain text.

Please read the chapter on MIRA utilities in this manual to learn more on **convert_project** and also have a look at `convert_project -h` which lists all possible formats and other command line options.

10.4 Filtering results

It is important to remember that some assembly options of mira improve the overall assembly while increasing the number of *contig debris*, i.e. small contigs with low coverage that can probably be discarded. One infamous option is the option to use *uniform read distribution* ([-AS:urd]) which helps to reconstruct identical repeats across multiple locations in the genome but as a side effect, some redundant reads will end up as typical contig debris. You probably do not want to have a look at contig debris when finishing a genome unless you are really, really, really picky.

By default, the result files of MIRA contain everything which might play a role in automatic assembly post-processing pipelines as most sequencing centers have implemented.

Many people prefer to just go on with what would be large contigs. Therefore the **convert_project** program from the MIRA package can selectively filter CAF or MAF files for contigs with a certain size, average coverage or number of reads.

The file `*_info_assembly.txt` in the info directory at the end of an assembly might give you first hints on what could be suitable filter parameters. For example, in assemblies being made with a in a normal (whatever this means) fashion I routinely only consider contigs larger than 500 bases and have at least one third of the average coverage of the N50 contigs.

Here's an example: In the "Large contigs" section, there's a "Coverage assessment" subsection. It looks a bit like this:

```
...
Coverage assessment:
--------------------
Max coverage (total): 43
Max coverage
Sanger: 0
454:    43
Solexa: 0
Solid:  0
Avg. total coverage (size ≥ 5000): 22.30
Avg. coverage (contig size ≥ 5000)
```

```
Sanger: 0.00
454:    22.05
Solexa: 0.00
Solid:  0.00
...
```

This project was obviously a 454 only project, and the average coverage for it is ~22. This number was estimated by MIRA by taking only contigs of at least 5kb into account, which for sure left out everything which could be categorised as debris. It's a pretty solid number.

Now, depending on how much time you want to invest performing some manual polishing, you should extract contigs which have at least the following fraction of the average coverage:

- 2/3 if a quick and "good enough" is what you want and you don't want to do some manual polishing. In this example, that would be around 14 or 15.
- 1/2 if you want to have a "quick look" and eventually perform some contig joins. In this example the number would be 11.
- 1/3 if you want quite accurate and for sure not loose any possible repeat. That would be 7 or 8 in this example.

Example (useful with assemblies of Sanger data): extracting only contigs $\geq$ 1000 bases and with a minimum average coverage of 4 into FASTA format:

```
$ convert_project -f caf -t fasta -x 1000 -y 4 sourcefile.caf targetfile.fasta
```

Example (useful with assemblies of 454 data): extracting only contigs $\geq$ 500 bases into FASTA format:

```
$ convert_project -f caf -t fasta -x 500 sourcefile.caf targetfile.fasta
```

Example (e.g. useful with Sanger/454 hybrid assemblies): extracting only contigs $\geq$ 500 bases and with an average coverage $\geq$ 15 reads into CAF format, then converting the reduced CAF into a Staden GAP4 project:

```
$ convert_project -f caf -t caf -x 500 -y 15 sourcefile.caf tmp.caf
$ caf2gap -project somename -ace tmp.caf
```

Example (e.g. useful with Sanger/454 hybrid assemblies): extracting only contigs $\geq$ 1000 bases and with $\geq$ 10 reads from MAF into CAF format, then converting the reduced CAF into a Staden GAP4 project:

```
$ convert_project -f maf -t caf -x 500 -z 10 sourcefile.maf tmp
$ caf2gap -project somename -ace tmp.caf
```

Start **convert_project** with the *-h* option for help on available options.

10.5 Finishing and data analysis: finding places of importance in the assembly

10.5.1 Tags set by MIRA

MIRA sets a number of different tags in resulting assemblies. They can be set in reads (in which case they mostly end with a *r*) or in the consensus.(then ending with a *c*).

If you use the Staden **gap4** or **consed** assembly editor to tidy up the assembly, you can directly jump to places of interest that MIRA marked for further analysis by using the search functionality of these programs.

You should search for the following "consensus" tags for finding places of importance (in this order).

- IUPc
- UNSc

- SRMc
- WRMc
- STMU (only hybrid assemblies)
- MCVc (only when assembling different strains, i.e., mostly relevant for mapping assemblies)
- SROc (only when assembling different strains, i.e., mostly relevant for mapping assemblies)
- SAOc (only when assembling different strains, i.e., mostly relevant for mapping assemblies)
- SIOc (only when assembling different strains, i.e., mostly relevant for mapping assemblies)
- STMS (only hybrid assemblies)

of lesser importance are the "read" versions of the tags above:

- UNSr
- SRMr
- WRMr
- SROr (only when assembling different strains, i.e., mostly relevant for mapping assemblies)
- SAOr (only when assembling different strains, i.e., mostly relevant for mapping assemblies)
- SIOr (only when assembling different strains, i.e., mostly relevant for mapping assemblies)

In normal assemblies (only one sequencing technology, just one strain), search for the IUPc, UNSc, SRMc and WRMc tags.

In hybrid assemblies, searching for the IUPc, UNSc, SRMc, WRMc, and STMU tags and correcting only those places will allow you to have a qualitatively good assembly in no time at all.

Columns with SRMr tags (SRM in **R**eads) in an assembly without a SRMc tag at the same consensus position show where mira was able to resolve a repeat during the different passes of the assembly ... you don't need to look at these. SRMc and WRMc tags however mean that there may be unresolved trouble ahead, you should take a look at these.

Especially in mapping assemblies, columns with the MCVc, SROx, SIOx and SAOx tags are extremely helpful in finding places of interest. As they are only set if you gave strain information to MIRA, you should always do that.

For more information on tags set/used by MIRA and what they exactly mean, please look up the according section in the reference chapter.

10.5.2 Other places of importance

The read coverage histogram as well as the template display of gap4 will help you to spot other places of potential interest. Please consult the gap4 documentation.

10.5.3 Joining contigs

I recommend to invest a couple of minutes (in the best case) to a few hours in joining contigs, especially if the uniform read distribution option of MIRA was used (but first filter for large contigs). This way, you will reduce the number of "false repeats" in improve the overall quality of your assembly.

10.5.3.1 Joining contigs at true repetitive sites

Joining contigs at repetitive sites of a genome is always a difficult decision. There are, however, two rules which can help:

1. If the sequencing was done without a paired-end library, don't join.
2. If the sequencing was done with a paired-end library, but no pair (or template) span the join site, don't join.

The following screenshot shows a case where one should not join as the finishing program (in this case **gap4**) warns that no template (read-pair) span the join site:

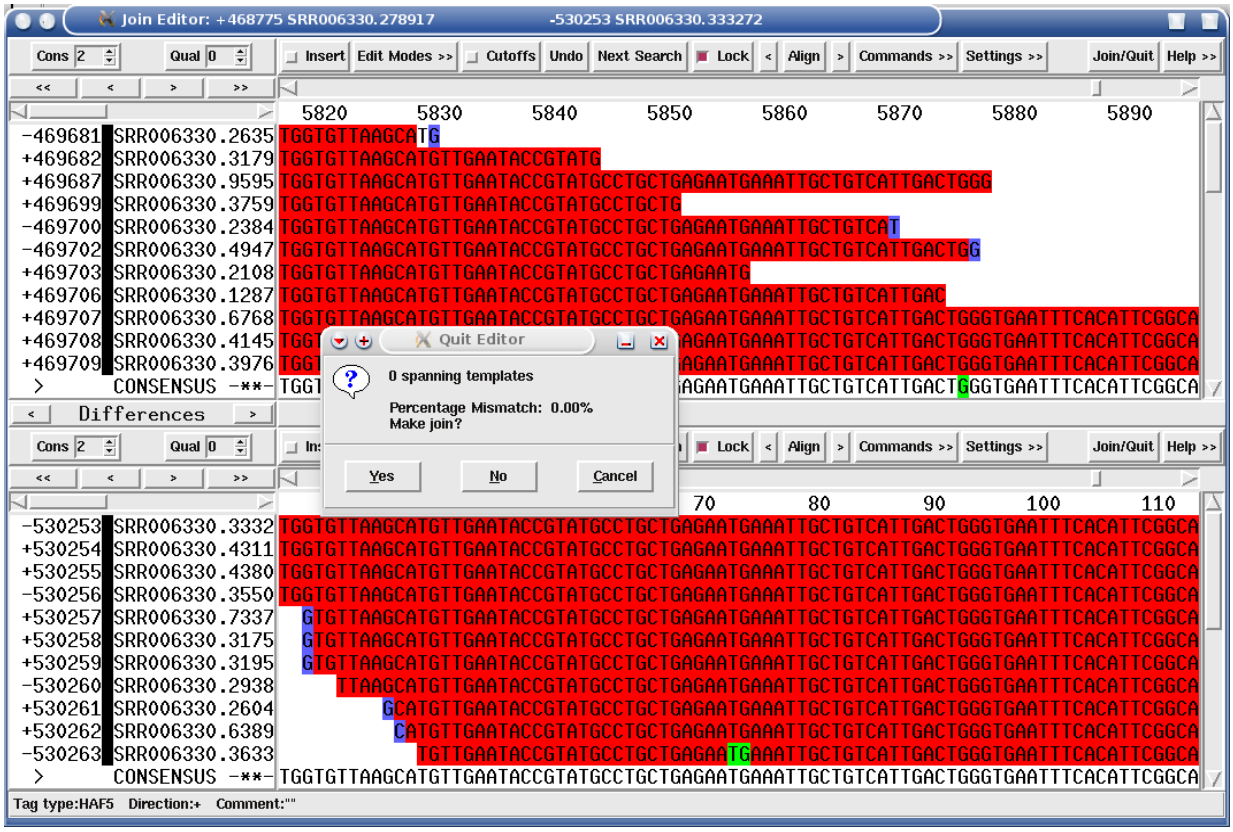

Figure 10.1: Join at a repetitive site which should not be performed due to missing spanning templates.

The next screenshot shows a case where one should join as the finishing program (in this case **gap4**) finds templates spanning the join site and all of them are good:

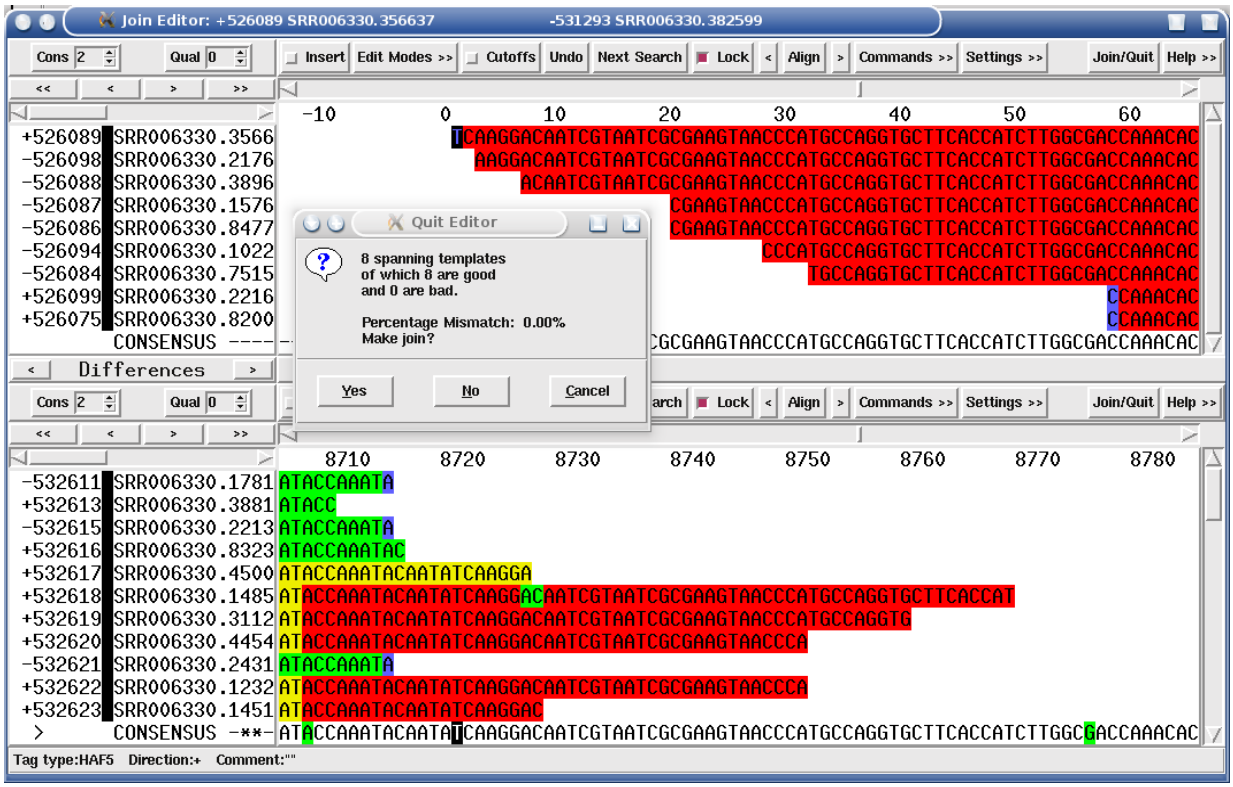

Figure 10.2: Join at a repetitive site which should be performed due to spanning templates being good.

10.5.3.2 Joining contigs at "wrongly discovered" repetitive sites

Remember that MIRA takes a very cautious approach in contig building, and sometimes creates two contigs when it could have created one. Three main reasons can be the cause for this:

1. when using *uniform read distribution*, some non-repetitive areas may have generated so many more reads that they start to look like repeats (so called pseudo-repeats). In this case, reads that are above a given coverage are *shaved off* (see [-AS:urdcm] and kept in reserve to be used for another copy of that repeat ... which in case of a non-repetitive region will of course never arrive. So at the end of an assembly, these shaved-off reads will form short, low coverage contig debris which can more or less be safely ignored and sorted out via the filtering options ([-x -y -z]) of **convert_project**.

   Some 454 library construction protocols -- especially, but not exclusively, for paired-end reads -- create pseudo-repeats quite frequently. In this case, the pseudo-repeats are characterised by several reads starting at exact the same position but which can have different lengths. Should MIRA have separated these reads into different contigs, these can be -- most of the time -- safely joined. The following figure shows such a case:

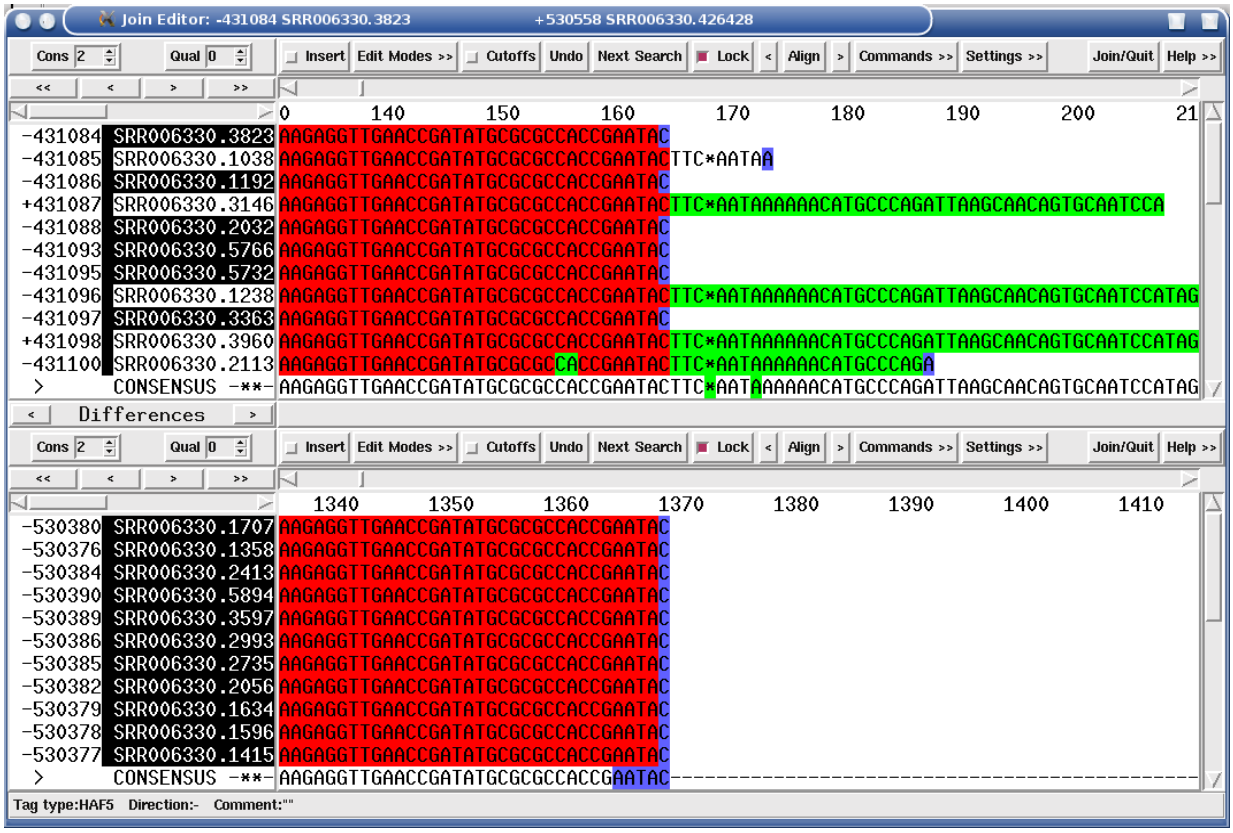

Figure 10.3: Pseudo-repeat in 454 data due to sequencing artifacts

For Solexa data, a non-negligible GC bias has been reported in genome assemblies since late 2009. In genomes with moderate to high GC, this bias actually favours regions with lower GC. Examples were observed where regions with an average GC of 10% less than the rest of the genome had between two and four times more reads than the rest of the genome, leading to false "discovery" of duplicated genome regions.

2. when using unpaired data, the above described possibility of having "too many" reads in a non-repetitive region can also lead to a contig being separated into two contigs in the region of the pseudo-repeat.

3. a number of reads (sometimes even just one) can contain "high quality garbage", that is, nonsense bases which got - for some reason or another - good quality values. This garbage can be distributed on a long stretch in a single read or concern just a single base position across several reads.

   While MIRA has some algorithms to deal with the disrupting effects of reads like, the algorithms are not always 100% effective and some might slip through the filters.

Chapter 11

Utilities in the MIRA package

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'Ninety percent of success is just growing up. '

—Solomon Short

11.1 convert_project

11.1.1 Synopsis

`convert_project` [options] *`input_file output_basename`*

11.1.2 Description

convert_project is a tool to convert, extract and sometimes recalculate all kinds of data related to sequence assembly files.

More specifically, **convert_project** can

1. convert from multiple alignment files (CAF, MAF) to other multiple alignment files (CAF, MAF, ACE, SAM), and -- if wished -- selecting contigs by different criteria like name, length, coverage etc.
2. extract the consenus from multiple alignments in CAF and MAF format, writing it to any supported output format (FASTA, FASTQ, plain text, HTML, etc.) and -- if wished -- recalculating the consensus using the MIRA consensus engine with MIRA parameters
3. extract read sequences (clipped or unclipped) from multiple alignments and save to any supported format
4. Much more, need to document this.

...

11.1.3 Options

...

11.1.3.1 General options

-f ***caf | maf | fasta | fastq | gbf | phd | fofnexp*** 'From-type', the format of the input file. CAF and MAF files can contain full assemblies and/or unassembled (single) sequences while the other formats contain only unassembled sequences.

-t ***ace | asnp | caf | crlist | cstats | exp | fasta | fastq | gbf | hsnp | html | maf | phd | tex*** 'To-type', the format of the output file. Multiple mentions of [-t] are allowed, in which case **convert_project** will convert to multiple types.

-a Append. Results of conversion are appended to existing files instead of overwriting them.

-A ***MIRA-PARAMETERSTRING*** Additional MIRA parameters. Allows to initialise the underlying MIRA routines with specific parameters. A use case can be, e.g., to recalculate a consensus of an assembly in a slightly different way (see also [-r]) than the one which is stored in assembly files. Example: to tell the consensus algorithm to use a minimum number of reads per group, use: "454_SETTINGS -CO:mrpg=4".

Consult the MIRA reference manual for a full list of MIRA parameters.

-C Hard clip reads. When the input is a format which contains clipping points in sequences and the requested output consists of sequences of reads, only the unclipped parts of sequences will be saved as results.

-m Make contigs. Encase single reads as contig singlets into a CAF/MAF file.

-n ***namefile*** Name select. Only contigs or reads are selected for output which name appears in `namefile`. `namefile` is a simple text file having one name entry per line.

-o ***offset*** Offset of quality values in FASTQ files. Only valid if -f is FASTQ.

-R ***namestring*** Rename contigs/singlets/reads with given name string to which a counter is added.

Known bug: will create duplicate names if input (CAF or MAF)contains contigs/singlets as well as free reads, i.e. reads not in contigs nor singlets.

11.1.3.2 Options for input containing contig data

The following switches will work only if the input file contains contigs (i.e., CAF or MAF with contig data). Though infrequent, note that both CAF and MAf can contain single reads only.

-M Do not extract contigs (or their consensus), but the sequence of the reads they are composed of.

-N ***namefile*** Name select, sorted. Only contigs are selected for output which name appears in `namefile`. Regardless of the order of contigs in the input, the output is sorted according to the appearance of names in `namefile`. `namefile` is a simple text file having one name entry per line.

Note that for this function to work, all contigs are loaded into memory which may be straining your RAM for larger projects.

-r ***c | C | q | f*** Recalculate consensus and / or consensus quality values and / or SNP feature tags of an assembly. This feature is useful in case third party programs create own consensus sequences without handling different sequencing technologies (e.g. the combination of **gap4** and **caf2gap**) or when the CAF/MAF files do not contain consensus sequences at all.

- **c** recalculate consensus & consensus qualities using IUPAC where necessary
- **C** recalculate consensus & consensus qualities forcing ACGT calls and without IUPAC codes
- **q** recalculate consensus quality values only
- **f** recalculate SNP features

Note Only the last of cCq is relevant, 'f' works as a switch and can be combined with the others (e.g. '-r Cf').

Note If the CAF/MAF contains reads from multiple strains, recalculation of consensus & consensus qualities is forced, you can just influence whether IUPACs are used or not. This is due to the fact that CAF/MAF do not provide facilities to store consensus sequences from multiple strains.

-s Split. Split outpout into single files, one file per contig. Files are named according to name of contig.

-u fillUp strain genomes. In assemblies made of multiple strains, holes in the consensus of a strain (bases 'N' or '@') can be filled up with the consensus of the other strains. Takes effect only when '-r' is active.

-q ***quality_value*** Defines minimum quality a consensus base of a strain must have, consensus bases below this will be set to 'N'. Only used when -r is active.

-v ***coverage_value*** Defines minimum coverage a consensus base of a strain must have, consensus bases with a coverage below this will be set to 'N'. Only used when -r is active.

-x ***length*** Minimum length a contig (in full assemblies) or read (in single sequence files) must have. All contigs / reads with a length less than this value are discarded. Default: 0 (=switched off).

Note: this is of course not applied to reads in contigs! Contigs passing the [-x] length criterium and stored as complete assembly (CAF, MAF, ACE, etc.) still contain all their reads.

-X ***length*** Similar to [-x], but applies only to clipped reads (input file format must have clipping points set to be effective).

-y ***contig_coverage*** Minimum average contig coverage. Contigs with an average coverage less than this value are discarded.

-z ***min_reads*** Minimum number of reads in contig. Contigs with less reads than this value are discarded.

-l ***line_length*** On output of assemblies as text or HTML: number of bases shown in one alignment line. Default: 60.

-c ***endgap_character*** On output of assemblies as text or HTML: character used to pad endgaps. Default: ' ' (a blank)

11.1.4 Examples

In the following examples, the CAF and MAF files used are expected to contain full assembly data like the files created by MIRA during an assembly or by the gap2caf program. CAF and MAF could be used interchangeably in these examples, depending on which format currently is available. In general though, MAF is faster to process and smaller on disk.

Simple conversion: the consensus of an assembly to FASTA, at the same time coverage data for contigs to WIG and furth

```
convert_project -f caf -t fasta -t wig -t ace source.caf dest
```

```
convert_project -f maf -t caf -x 2000 -y 10 source.caf dest
```

Filtering an assembly for contigs of length ≥2000 and an average coverage ≥ 10, while translating from MAF to CAFFilteri

```
convert_project -f fastq -x 55 -R newname source.fastq dest
```

Filtering and sortig contigs of an assembly according to external contig name list. This example will fetch the contigs named bchoc_c14, ...3, ...5 and ...13 and save the result in exactly that order to a new file:

```
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users  231698898 2007-10-21 15:16 bchoc_out.caf
-rw-r--r-- 1 bach users         38 2007-10-21 15:16 contigs.lst
arcadia:/path/to/myProject$ cat contigs.lst
bchoc_c14
bchoc_c3
bchoc_c5
bchoc_c13
arcadia:/path/to/myProject$ convert_project -f caf -N contigs.lst bchoc_out.caf  ↩
    myfilteredresult
```

```
[...]
arcadia:/path/to/myProject$ ls -l
-rw-r--r-- 1 bach users  231698898 2007-10-21 15:16 bchoc_out.caf
-rw-r--r-- 1 bach users         38 2007-10-21 15:16 contigs.lst
-rw-r--r-- 1 bach users     828726 2007-10-21 15:24 myfilteredresult.caf
```

11.2 mirabait

11.2.1 Synopsis

`mirabait` [options] *`bait_file input_file output_basename`*

While input and output file can have any of the supported formats (see -f and -t options), the bait file needs to be in FASTA format.

11.2.2 Description

mirabait selects reads from a read collection which are partly similar or equal to sequences defined as target baits. Similarity is defined by finding a user-adjustable number of common k-mers (sequences of k consecutive bases) which are the same in the bait sequences and the screened sequences to be selected, either in forward or reverse complement direction.

The search performed is exact, that is, sequences selected are guaranteed to have the required number of k-mers equal to the bait sequences while sequences not selected are guaranteed not have these.

11.2.3 Options

`-f` ***`caf | maf | fasta | fastq | gbf | phd`*** 'From-type', the format of the input file. Default: fastq.

`-t` ***`caf | maf | fasta | fastq`*** 'To-type', the format of the output file. Default: format of the input.
Multiple mentions of -t are allowed, in which case the selected sequences are written to all file formats chosen.

`-k` ***`k-mer-length`*** k-mer, length of bait in bases ($\leq$32, default=31)

`-n` ***`minoccurence`*** Minimum number of k-mers needed for a sequence to be selected. Default: 1.

`-i` Inverse hit: selects only sequence that do not meet the -k and -n criteria.

`-r` Does not check for hits in reverse complement direction.

Chapter 12

Assembly of *hard* genome or EST / RNASeq projects

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'If it were easy, it would have been done already. '

—Solomon Short

12.1 Getting 'mean' genomes or EST / RNASeq data sets assembled

For some EST data sets you might want to assemble, MIRA will take too long or the available memory will not be sufficient. For genomes this can be the case for eukaryotes, plants, but also for some bacteria which contain high number of (pro-)phages, plasmids or engineered operons. For EST data sets, this concerns all projects with non-normalised libraries.

This guide is intended to get you through these problematic genomes. It is (cannot be) exhaustive, but it should get you going.

12.1.1 For the impatient

Use [-SK:mnr=yes:nrr=10] and give it a try. If that does not work, decrease [-SK:nrr] to anywhere between 5 and 9. If it worked well enough increase the [-SK:nrr] parameter up to 15 or 20. But please also read on to see how to choose the "nrr" threshold.

12.1.2 Introduction to 'masking'

The SKIM phase (all-against-all comparison) will report almost every potential hit to be checked with Smith-Waterman further downstream in the MIRA assembly process. While this is absolutely no problem for most bacteria, some genomes (eukaryotes, plants, some bacteria) have so many closely related sequences (repeats) that the data structures needed to take up all information might get much larger than your available memory. In those cases, your only chance to still get an assembly is to tell the assembler it should disregard extremely repetitive features of your genome.

There is, in most cases, one problem: one doesn't know beforehand which parts of the genome are extremely repetitive. But MIRA can help you here as it produces most of the needed information during assembly and you just need to choose a threshold from where on MIRA won't care about repetitive matches.

The key to this are the two fail-safe command line parameters which will mask "nasty" repeats from the quick overlap finder (SKIM): [-SK:mnr] and [-SK:nrr=10]. [-SK:bph] also plays a role in this, but I'll come back to this later).

12.1.3 How does "nasty repeat" masking work?

If switched on [-SK:mnr=yes], MIRA will use SKIM3 k-mer statistics to find repetitive stretches. K-mers are nucleotide stretches of length k. In a perfectly sequenced genome without any sequencing error and without sequencing bias, the k-mer frequency

can be used to assess how many times a given nucleotide stretch is present in the genome: if a specific k-mer is present as many times as the average frequency of all k-mers, it is a reasonable assumption to estimate that the specific k-mer is not part of a repeat (at least not in this genome).

Following the same path of thinking, if a specific k-mer frequency is now two times higher than the average of all k-mers, one would assume that this specific k-mer is part of a repeat which occurs exactly two times in the genome. For 3x k-mer frequency, a repeat is present three times. Etc.pp. MIRA will merge information on single k-mers frequency into larger 'repeat' stretches and tag these stretches accordingly.

Of course, low-complexity nucleotide stretches (like poly-A in eukaryotes), sequencing errors in reads and non-uniform distribution of reads in a sequencing project will weaken the initial assumption that a k-mer frequency is representative for repeat status. But even then the k-mer frequency model works quite well and will give a pretty good overall picture: most repeats will be tagged as such.

Note that the parts of reads tagged as "nasty repeat" will not get masked per se, the sequence will still be present. The stretches dubbed repetitive will get the "MNRr" tag. They will still be used in Smith-Waterman overlaps and will generate a correct consensus if included in an alignment, but they will not be used as seed.

Some reads will invariably end up being completely repetitive. These will not be assembled into contigs as MIRA will not see overlaps as they'll be completely masked away. These reads will end up as debris. However, note that MIRA is pretty good at discerning 100% matching repeats from repeats which are not 100% matching: if there's a single base with which repeats can be discerned from each other, MIRA will find this base and use the k-mers covering that base to find overlaps.

12.1.4 Selecting a "nasty repeat ratio"

The ratio from which on the MIRA SKIM algorithm won't report matches is set via [-SK:nrr]. E.g., using [-SK:nrr=10] will hide all k-mers which occur at a frequency 10 times (or more) higher than the median of all k-mers.

The nastiness of a repeat is difficult to judge, but starting with 10 copies in a genome, things can get complicated. At 20 copies, you'll have some troubles for sure.

The standard values of *10* for the [-SK:nrr] parameter is a pretty good 'standard' value which can be tried for an assembly before trying to optimise it via studying the hash statistics calculated by MIRA. For the later, please read the section 'Examples for hash statistics' further down in this guide.

12.2 How MIRA tags different repeat levels

During SKIM phase, MIRA will assign frequency information to each and every k-mer in all reads of a sequencing project, giving them different status. Additionally, tags are set in the reads so that one can assess reads in assembly editors that understand tags (like gap4, gap5, consed etc.). The following tags are used:

HAF2 coverage below average (default: < 0.5 times average)

HAF3 coverage is at average (default: $\geq$ 0.5 times average and $\leq$ 1.5 times average)

HAF4 coverage above average (default: > 1.5 times average and < 2 times average)

HAF5 probably repeat (default: $\geq$ 2 times average and < 5 times average)

HAF6 'crazy' repeat (default: > 5 times average)

MNRr stretches which were masked away by [-SK:*mnr=yes*] being more that [-SK:*nnr=...*] repetitive.

12.3 The readrepeats info file

If [-SK:mnr=yes] is used, MIRA will write an additional file into the info directory: `<projectname>_info_readrepeats.lst`

The "readrepeats" file makes it possible to try and find out what makes sequencing data nasty. It's a key-value-value file with the name of the sequence as "key" and then the type of repeat (HAF2 - HAF7 and MNRr) and the repeat sequence as "values". "Nasty" in this case means *everything which was masked via [-SK:mnr=yes]*.

The file looks like this:

```
read1     HAF5   GCTTCGGCTTCGGCTTCGGCTTCGGCTTCGGCTTCGGCTTCGGCT ...
read2     HAF7   CCGAAGCCGAAGCCGAAGCCGAAGCCGAAGCCGAAGCCGAAGC ...
read2     MNRr   AAAAAAAAAAAAAAAAAAAAAAAAAAAA ...
read3     HAF6   GCTTCGGCTTCGGCTTCGGCTTCGGCTTCGGCTTCGGCTTCGGCT ...
...
etc.
```

That is, each line consists of the read name where a stretch of repetitive sequences was found, then the MIRA repeat categorisation level (HAF2 to HAF7 and MNRr) and then the stretch of bases which is seen to be repetitive.

Note that reads can have several disjunct repeat stetches in a single read, hence they can occur more than one time in the file as shown with *read2* in the example above.

One will need to search some databases with the "nasty" sequences and find vector sequences, adaptor sequences or even human sequences in bacterial or plant genomes ... or vice versa as this type of contamination happens quite easily with data from new sequencing technologies. After a while one gets a feeling what constitutes the largest part of the problem and one can start to think of taking countermeasures like filtering, clipping, masking etc.

12.4 Pipeline to find worst contaminants or repeats in sequencing data

Note
In case you are not familiar with UNIX pipes, now would be a good time to read an introductory text on how this wonderful system works. You might want to start with a short introductory article at Wikipedia: http://en.wikipedia.org/wiki/Pipeline_%28Unix%29
In a nutshell: instead of output to files, a pipe directs the output of one program as input to another program.

There's one very simple trick to find out whether your data contains some kind of sequencing vector or adaptor contamination which I use. it makes use of the read repat file discussed above.

The following example shows this exemplarily on a 454 data where the sequencing provider used some special adaptor in the wet lab but somehow forgot to tell the Roche pre-processing software about it, so that a very large fraction of reads in the SFF file had unclipped adaptor sequence in it (which of course wreaks havoc with assembly programs):

```
arcadia:$ grep MNRr badproject_info_readrepeats.lst | cut -f 3| sort | uniq -c |sort -g -r ↩
    | head -15
    504 ACCACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    501 CAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    489 GGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    483 GCCACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    475 AATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    442 GATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    429 CGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    424 TTGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    393 ACTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    379 CAGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    363 ATTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    343 CATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    334 GTTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    328 AACACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    324 GGTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
```

You probably see a sequence pattern CTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC in the above screenshot. Before going into details of what you are actually seeing, here's the explanation how this pipeline works:

grep MNRr *badproject*_info_readrepeats.lst From the file with the information on repeats, grab all the lines containing repetitive sequence which MIRA categorised as 'nasty' via the 'MNRr' tag. The result looks a bit like this (first 15 lines shown):

```
C6E3C7T12GKN35  MNRr    GCGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12JLIBM  MNRr    TTCACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12HQOM1  MNRr    CAGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12G52II  MNRr    CAGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12JRMPO  MNRr    TCTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12H1A8V  MNRr    GCGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12H34Z7  MNRr    AAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12H4HGC  MNRr    GGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12FNA1N  MNRr    AATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12F074V  MNRr    CTTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12I1GYO  MNRr    CAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12I53C8  MNRr    CACACTCGTATAGTGACACGCAACAGGGG
C6E3C7T12I4V6V  MNRr    ATCACTCGTATAGTGACACGCAACAGGGG
C6E3C7T12H5R00  MNRr    TCTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
C6E3C7T12IBA5E  MNRr    AATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
...
```

cut -f 3 We're just interested in the sequence now, which is in the third column. The above 'cut' command takes care of this. The resulting output may look like this (only first 15 lines shown):

```
GCGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
TTCACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
CAGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
CAGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
TCTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
GCGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
AAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
GGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
AATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
CTTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
CAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
CACACTCGTATAGTGACACGCAACAGGGG
ATCACTCGTATAGTGACACGCAACAGGGG
TCTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
AATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
...
```

sort Simply sort all sequences. The output may look like this now (only first 15 line shown):

```
AAACTCGTATAGTGACACGCA
AAACTCGTATAGTGACACGCAACAGG
AAACTCGTATAGTGACACGCAACAGGG
AAACTCGTATAGTGACACGCAACAGGGG
AAACTCGTATAGTGACACGCAACAGGGG
AAACTCGTATAGTGACACGCAACAGGGG
AAACTCGTATAGTGACACGCAACAGGGG
AAACTCGTATAGTGACACGCAACAGGGG
AAACTCGTATAGTGACACGCAACAGGGGAT
AAACTCGTATAGTGACACGCAACAGGGGATA
AAACTCGTATAGTGACACGCAACAGGGGATA
AAACTCGTATAGTGACACGCAACAGGGGATA
AAACTCGTATAGTGACACGCAACAGGGGATA
AAACTCGTATAGTGACACGCAACAGGGGATA
AAACTCGTATAGTGACACGCAACAGGGGATA
...
```

uniq -c This command counts how often a line repeats itself in a file. As we previously sorted the whole file by sequence, it effectively counts how often a certain sequence has been tagged as MNRr. The output consists of a tab delimited format in two columns: the first column contains the number of times a given line (sequence in our case) was seen, the second column contains the line (sequence) itself. An exemplariy output looks like this (only first 15 lines shown):

```
      1 AAACTCGTATAGTGACACGCA
      1 AAACTCGTATAGTGACACGCAACAGG
      1 AAACTCGTATAGTGACACGCAACAGGG
      5 AAACTCGTATAGTGACACGCAACAGGGG
      1 AAACTCGTATAGTGACACGCAACAGGGGAT
     13 AAACTCGTATAGTGACACGCAACAGGGGATA
      6 AAACTCGTATAGTGACACGCAACAGGGGATAGAC
      4 AAACTCGTATAGTGACACGCAACAGGGGATAGACAA
      9 AAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGC
      3 AAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCA
    257 AAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
      1 AACACTCGTATAGTGACACGCAAC
      2 AACACTCGTATAGTGACACGCAACAGGG
     23 AACACTCGTATAGTGACACGCAACAGGGG
      6 AACACTCGTATAGTGACACGCAACAGGGGATA
...
```

sort -g -r We now sort the output of the previous uniq-counting command by asking 'sort' to perform a numerical sort (via '-g') and additionally sort in reverse order (via '-r') so that we get the sequences encountered most often at the top of the output. And that one looks exactly like shown previously:

```
    504 ACCACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    501 CAACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    489 GGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    483 GCCACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    475 AATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    442 GATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    429 CGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    424 TTGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    393 ACTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    379 CAGACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    363 ATTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    343 CATACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    334 GTTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    328 AACACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
    324 GGTACTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC
...
```

So, what is this ominous CTCGTATAGTGACACGCAACAGGGGATAGACAAGGCAC you are seeing? To make it short: a modified 454 B-adaptor with an additional MID sequence.

Note

These adaptor sequences have absolutely no reason to exist in your data, none! Go back to your sequencing provider and ask them to have a look at their pipeline as they should have had it set up in a way that you do not see these things anymore. Yes, due to sequencing errors, sometimes some adaptor or sequencing vectors remnants will stay in your sequencing data, but that is no problem as MIRA is capable of handling that very well.

But having much more than 0.1% to 0.5% of your sequence containing these is a sure sign that someone goofed somewhere ... and it's very probably not your fault.

12.5 Examples for hash statistics

Selecting the right ratio so that an assembly fits into your memory is not straight forward. But MIRA can help you a bit: during assembly, some frequency statistics are printed out (they'll probably end up in some info file in later releases). Search for the

term "Hash statistics" in the information printed out by MIRA (this happens quite early in the process)

12.5.1 Caveat: -SK:bph

Some explanation how bph affects the statistics and why it should be chosen >=16 for [-SK:mnr]

12.5.2 Sanger sequencing, a simple bacterium

This example is taken from a pretty standard bacterium where Sanger sequencing was used:

```
Hash statistics:
=============================================================
Measured avg. coverage: 15

Deduced thresholds:
-------------------
Min normal cov: 7
Max normal cov: 23
Repeat cov: 29
Crazy cov: 120
Mask cov: 150

Repeat ratio histogram:
-----------------------
0       475191
1       5832419
2       181994
3       6052
4       4454
5       972
6       4
7       8
14      2
16      10
=============================================================
```

The above can be interpreted like this: the expected coverage of the genome is 15x. Starting with an estimated hash frequency of 29, MIRA will treat a k-mer as 'repetitive'. As shown in the histogram, the overall picture of this project is pretty healthy:

- only a small fraction of k-mers have a repeat level of '0' (these would be k-mers in regions with quite low coverage or k-mers containing sequencing errors)
- the vast majority of k-mers have a repeat level of 1 (so that's non- repetitive coverage)
- there is a small fraction of k-mers with repeat level of 2-10
- there are almost no k-mers with a repeat level >10

12.5.3 454 Sequencing, a somewhat more complex bacterium

Here's in comparison a profile for a more complicated bacterium (454 sequencing):

```
Hash statistics:
=============================================================
Measured avg. coverage: 20

Deduced thresholds:
-------------------
```

```
Min normal cov: 10
Max normal cov: 30
Repeat cov: 38
Crazy cov: 160
Mask cov: 0

Repeat ratio histogram:
-----------------------
0       8292273
1       6178063
2       692642
3       55390
4       10471
5       6326
6       5568
7       3850
8       2472
9       708
10      464
11      270
12      140
13      136
14      116
15      64
16      54
17      54
18      52
19      50
20      58
21      36
22      40
23      26
24      46
25      42
26      44
27      32
28      38
29      44
30      42
31      62
32      116
33      76
34      80
35      82
36      142
37      100
38      120
39      94
40      196
41      172
42      228
43      226
44      214
45      164
46      168
47      122
48      116
49      98
50      38
51      56
52      22
53      14
```

```
54      8
55      2
56      2
57      4
87      2
89      6
90      2
92      2
93      2
1177    2
1181    2
=======================================================
```

The difference to the first bacterium shown is pretty striking:

- first, the k-mers in repeat level 0 (below average) is higher than the k-mers of level 1! This points to a higher number of sequencing errors in the 454 reads than in the Sanger project shown previously. Or at a more uneven distribution of reads (but not in this special case).
- second, the repeat level histogram does not trail of at a repeat frequency of 10 or 15, but it has a long tail up to the fifties, even having a local maximum at 42. This points to a small part of the genome being heavily repetitive ... or to (a) plasmid(s) in high copy numbers.

Should MIRA ever have problems with this genome, switch on the nasty repeat masking and use a level of 15 as cutoff. In this case, 15 is OK to start with as a) it's a bacterium, it can't be that hard and b) the frequencies above level 5 are in the low thousands and not in the tens of thousands.

12.5.4 Solexa sequencing, E.coli MG1655

```
Hash statistics:
=========================================================
Measured avg. coverage: 23

Deduced thresholds:
-------------------
Min normal cov: 11
Max normal cov: 35
Repeat cov: 44
Crazy cov: 184
Mask cov: 0

Repeat ratio histogram:
-----------------------
0       1365693
1       8627974
2       157220
3       11086
4       4990
5       3512
6       3922
7       4904
8       3100
9       1106
10      868
11      788
12      400
13      186
14      28
15      10
```

```
16      12
17      4
18      4
19      2
20      14
21      8
25      2
26      8
27      2
28      4
30      2
31      2
36      4
37      6
39      4
40      2
45      2
46      8
47      14
48      8
49      4
50      2
53      2
56      6
59      4
62      2
63      2
67      2
68      2
70      2
73      4
75      2
77      4
=======================================================
```

This hash statistics shows that MG1655 is pretty boring (from a repetitive point of view). One might expect a few repeats but nothing fancy: The repeats are actually the rRNA and sRNA stretches in the genome plus some intergenic regions.

- the k-mers number in repeat level 0 (below average) is considerably lower than the level 1, so the Solexa sequencing quality is pretty good respectively there shouldn't be too many low coverage areas.
- the histogram tail shows some faint traces of possibly highly repetitive k-mers, but these are false positive matches due to some standard Solexa base-calling weaknesses of earlier pipelines like, e.g., adding poly-A, poly-T or sometimes poly-C and poly-G tails to reads when spots in the images were faint and the base calls of bad quality

12.5.5 (NEED EXAMPLES FOR EUKARYOTES)

12.5.6 (NEED EXAMPLES FOR PATHOLOGICAL CASES)

Vector contamination etc.

Chapter 13

Some advice when going into a sequencing project

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *' Reliable information lets you say 'I don't know' with real confidence. '*
>
> —Solomon Short

13.1 Talk to your sequencing provider(s) before sequencing

Well, duh! But it's interesting what kind of mails I sometimes get. Like in:

> 'We've sequenced a one gigabase, diploid eukaryote with Solexa 36bp paired-end with 200bp insert size at 25x coverage. Could you please tell us how to assemble this data set de-novo to get a finished genome?'

A situation like the above should have never happened. Good sequencing providers are interested in keeping customers long term and will therefore try to find out what exactly your needs are. These folks generally know their stuff (they're making a living out of it) and most of the time propose you a strategy that fulfills your needs for a near minimum amount of money.

Listen to them.

If you think they try to rip you off or are overselling their competencies (which most providers I know won't even think of trying, but there are some), ask a quote from a couple of other providers. You'll see pretty quickly if there are some things not being right.

Note As a matter of fact, a rule which has saved me time and again for finding sequencing providers is not to go for the cheapest provider, especially if their price is far below quotes from other providers. They're cutting corners somewhere others don't cut for a reason.

13.2 A word or two on coverage ...

For de-novo assembly of genomes, the MIRA quick switches (--job=...) are optimised for 'decent' coverages that are commonly seen to get you something useful, i.e., $\geq$ 7x for Sanger, >=18x for 454 FLX, $\geq$ 25x for 454 GS20. Should you venture into lower coverages, you will need to adapt a few parameters (clipping etc.) via extensive switches.

There's one thing to be said about coverage and de-novo assembly: especially for bacteria, getting more than 'decent' coverage with 454 FLX or Titanium is *cheap*. Every assembler I know will be happy to assemble de-novo genomes with coverages of 25x, 30x, 40x ... and the number of contigs will still drop dramatically between a 15x 454 and a 30x 454 project.

With the introduction of the Titanium series, a full 454 plate may seem to be too much: you should get *at least* 200 megabase out of a plate, press releases from 454 seem to suggest 400 to 600 megabases.

In any case, do some calculations: if the coverage you expect to get reaches 50x (e.g. 200MB raw sequence for a 4MB genome), then you (respectively the assembler) can still throw away the worst 20% of the sequence (with lots of sequencing errors) and concentrate on the really, really good parts of the sequences to get you nice contigs.

Including library prep, a full 454 plate will cost you between 8000 to 12000 bucks (or less). The price for a Solexa lane is also low ... 2000 or less. Then you just need to do the math: is it worth to invest 10, 20, 30 or more days of wet lab work, designing primers, doing PCR sequencing etc. and trying to close remaining gaps or hunt down sequencing errors when you went for a 'low' coverage or a non-hybrid sequencing strategy? Or do you invest a few thousand bucks to get some additional coverage and considerably reduce the uncertainties and gaps which remain?

Remember, you probably want to do research on your bug and not research on how to best assemble and close genomes. So even if you put (PhD) students on the job, it's costing you time and money if you wanted to save money earlier in the sequencing. Penny-wise and pound-foolish is almost never a good strategy :-)

I do agree that with eukaryotes, things start to get a bit more interesting from the financial point of view.

Warning
There is, however, a catch-22 situation with coverage: too much coverage isn't good either. Without going into details: sequencing errors sometimes interfere heavily when coverage exceeds ~60x to 80x for 454 & IonTorrent and approximately 150x to 200x for Solexa/Illumina.
In those cases, do yourself a favour: there's more than enough data for your project ... just cut it down to some reasonable amount: 40x to 50x for 454 & IonTorrent, 100x for Solexa/Illumina.

13.3 A word of caution regarding your DNA in hybrid sequencing projects

So, you have decided that sequencing your bug with 454 paired-end with unpaired 454 data (or Sanger and 454, or Sanger and Solexa, or 454 and Solexa or whatever) may be a viable way to get the best bang for your buck. Then please follow this advice: prepare enough DNA *in one go* for the sequencing provider so that they can sequence it with all the technologies you chose without you having to prepare another batch ... or even grow another culture!

The reason for that is that as soon as you do that, the probability that there is a mutation somewhere that your first batch did not have is not negligible. And if there is a mutation, even if it is only one base, there is a >95% chance that MIRA will find it and thinks it is some repetitive sequence (like a duplicated gene with a mutation in it) and splits contigs at those places.

Now, there are times when you cannot completely be sure that different sequencing runs did not use slightly different batches (or even strains).

One example: the SFF files for SRA000156 and SRA001028 from the NCBI short trace archive should both contain E.coli K12 MG-16650 (two unpaired half plates and a paired-end plate). However, they contain DNA from different cultures. Furthermore, the DNA was prepared by different labs. The net effect is that the sequences in the paired-end library contain a few distinct mutations from the sequences in the two unpaired half-plates. Furthermore, the paired-end sequences contain sequences from phages that are not present in the unpaired sequences.

In those cases, provide strain information to the reads so that MIRA can discern possible repeats from possible SNPs.

13.4 For bacteria: beware of (high copy number) plasmids!

This is a source of interesting problems and furthermore gets people wondering why MIRA sometimes creates more contigs than other assemblers when it usually creates less.

Here's the short story: there are data sets which include one ore several high-copy plasmid(s). Here's a particularly ugly example: SRA001028 from the NCBI short read archive which contains a plate of paired-end reads for Ecoli K12 MG1655-G (ftp://ftp.ncbi.nih.gov/pub/TraceDB/ShortRead/SRA001028/).

The genome is sequenced at ~10x coverage, but during the assembly, three intermediate contigs with ~2kb attain a silly maximum coverage of ~1800x each. This means that there were ~540 copies of this plasmid (or these plasmids) in the sequencing.

When using the uniform read distribution algorithm - which is switched on by default when using "--job=" and the quality level of 'accurate' - MIRA will find out about the average coverage of the genome to be at ~10x. Subsequently this leads MIRA to dutifully create ~500 additional contigs (plus a number of contig debris) with various incarnations of that plasmid at an average of ~10x, because it thought that these were repetitive sites within the genome that needed to be disentangled.

Things get even more interesting when some of the plasmid / phage copies are slightly different from each other. These too will be split apart and when looking through the results later on and trying to join the copies back into one contig, one will see that this should not be done because there are real differences.

DON'T PANIC!

The only effect this has on your assembly is that the number of contigs goes up. This in turn leads to a number of questions in my mailbox why MIRA is sometimes producing more contigs than Newbler (or other assemblers), but that is another story (hint: Newbler either collapses repeats or leaves them completely out of the picture by not assembling repetitive reads).

What you can do is the following:

1. either you assemble everything together and the join the plasmid contigs manually after assembly, e.g. in gap4 (drawback: on really high copy numbers, MIRA will work quite a bit longer ... and you will have a lot of fun joining the contigs afterwards)
2. or, after you found out about the plasmid(s) and know the sequence, you filter out reads in the input data which contain this sequence and assemble the remaining reads.

Chapter 14

Frequently asked questions

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'Every question defines its own answer. Except perhaps 'Why a duck?' '*
>
> —Solomon Short

This list is a collection of frequently asked questions and answers regarding different aspects of the MIRA assembler.

Note This document needs to be overhauled.

14.1 Assembly quality

1. *Test question 1*
   Test answer 1
2. *Test question 2*
   Test answer 2

14.1.1 What is the effect of uniform read distribution (-AS:urd)?

```
I have a project which I once started quite normally via
"--job=denovo,genome,accurate,454"
and once with explicitly switching off the uniform read distribution
"--job=denovo,genome,accurate,454 -AS:urd=no"
I get less contigs in the second case and I wonder if that is not better.
Can you please explain?
```

Since 2.9.24x1, MIRA has a feature called "uniform read distribution" which is normally switched on. This feature reduces overcompression of repeats during the contig building phase and makes sure that, e.g., a rRNA stretch which is present 10 times in a bacterium will also be present approximately 10 times in your result files.

It works a bit like this: under the assumption that reads in a project are uniformly distributed across the genome, MIRA will enforce an average coverage and temporarily reject reads from a contig when this average coverage multiplied by a safety factor is reached at a given site.

It's generally a very useful tool disentangle repeats, but has some slight secondary effects: rejection of otherwise perfectly good reads. The assumption of read distribution uniformity is the big problem we have here: of course it's not really valid. You

sometimes have less, and sometimes more than "the average" coverage. Furthermore, the new sequencing technologies - 454 perhaps but especially the microreads from Solexa & probably also SOLiD - show that you also have a skew towards the site of replication origin.

One example: let's assume the average coverage of your project is 8 and by chance at one place you have 17 (non-repetitive) reads, then the following happens:

p= parameter of -AS:urdsip

Pass 1 to $p-1$: MIRA happily assembles everything together and calculates a number of different things, amongst them an average coverage of ~8. At the end of pass '$p-1$', it will announce this average coverage as first estimate to the assembly process.

Pass p: MIRA has still assembled everything together, but at the end of each pass the contig self-checking algorithms now include an "average coverage check". They'll invariably find the 17 reads stacked and decide (looking at the -AS:urdct parameter which I now assume to be 2) that 17 is larger than 2*8 and that this very well may be a repeat. The reads get flagged as possible repeats.

Pass $p+1$ to end: the "possibly repetitive" reads get a much tougher treatment in MIRA. Amongst other things, when building the contig, the contig now looks that "possibly repetitive" reads do not overstack by an average coverage multiplied by a safety value (-AS:urdcm) which I'll assume in this example to be 1.5. So, at a certain point, say when read 14 or 15 of that possible repeat want to be aligned to the contig at this given place, the contig will just flatly refuse and tell the assembler to please find another place for them, be it in this contig that is built or any other that will follow. Of course, if the assembler cannot comply, the reads 14 to 17 will end up as contiglet (contig debris, if you want) or if it was only one read that got rejected like this, it will end up as singlet or in the debris file.

Tough luck. I do have ideas on how to reintegrate those reads at the and of an assembly, but I had deferred doing this as in every case I had looked up, adding those reads to the contigs wouldn't have changed anything ... there's already enough coverage. What I do in those cases is simply filter away the contiglets (defined as being of small size and having an average coverage below the average coverage of the project / 3 (or 2.5)) from a project.

14.1.2 There are too many contig debris when using uniform read distribution, how do I filter for "good" contigs?

```
When using uniform read distribution there are too many contig with low
coverage which I don't want to integrate by hand in the finishing process. How
do I filter for "good" contigs?
```

OK, let's get rid of the cruft. It's easy, really: you just need to look up one number, take two decisions and then launch a command.

The first decision you need to take is on the minimum average coverage the contigs you want to keep should have. Have a look at the file `*_info_assembly.txt` which is in the info directory after assembly. In the "Large contigs" section, there's a "Coverage assessment" subsection. It looks a bit like this:

```
...
Coverage assessment:
--------------------
Max coverage (total): 43
Max coverage
Sanger: 0
454:    43
Solexa: 0
Solid:  0
Avg. total coverage (size ≥ 5000): 22.30
Avg. coverage (contig size ≥ 5000)
Sanger: 0.00
454:    22.05
Solexa: 0.00
Solid:  0.00
...
```

This project was obviously a 454 only project, and the average coverage for it is ~22. This number was estimated by MIRA by taking only contigs of at least 5Kb into account, which for sure left out everything which could be categorised as debris. It's a pretty solid number.

Now, depending on how much time you want to invest performing some manual polishing, you should extract contigs which have at least the following fraction of the average coverage:

- 2/3 if a quick and "good enough" is what you want and you don't want to do some manual polishing. In this example, that would be around 14 or 15.
- 1/2 if you want to have a "quick look" and eventually perform some contig joins. In this example the number would be 11.
- 1/3 if you want quite accurate and for sure not loose any possible repeat. That would be 7 or 8 in this example.

The second decision you need to take is on the minimum length your contigs should have. This decision is a bit dependent on the sequencing technology you used (the read length). The following are some rules of thumb:

- Sanger: 1000 to 2000
- 454 GS20: 500
- 454 FLX: 1000
- 454 Titanium: 1500

Let's assume we decide for an average coverage of 11 and a minimum length of 1000 bases. Now you can filter your project with convert_project

```
convert_project -f caf -t caf -x 1000 -y 14 sourcefile.caf filtered.caf
```

14.1.3 When finishing, which places should I have a look at?

```
I would like to find those places where MIRA wasn't sure and give it a quick
shot. Where do I need to search?
```

Search for the following tags in gap4 or any other finishing program for finding places of importance (in this order).

- IUPc
- UNSc
- SRMc
- WRMc
- STMU (only hybrid assemblies)
- STMS (only hybrid assemblies)

14.2 454 data

1. *What are little boys made of?*
   Snips and snails and puppy dog tails.
2. *What are little girls made of?*
   Sugar and spice and everything nice.

14.2.1 What do I need SFFs for?

```
I need the .sff files for MIRA to load ...
```

Nope, you don't, but it's a common misconception. MIRA does not load SFF files, it loads FASTA, FASTA qualities, FASTQ, XML, CAF, EXP and PHD. The reason why one should start from the SFF is: those files can be used to create a XML file in TRACEINFO format. This XML contains the absolutely vital information regarding clipping information of the 454 adaptors (the sequencing vector of 454, if you want).

For 454 projects, MIRA will then load the FASTA, FASTA quality and the corresponding XML. Or from CAF, if you have your data in CAF format.

14.2.2 What's sff_extract and where do I get it?

```
How do I extract the sequence, quality and other values from SFFs?
```

Use the **sff_extract** script from Jose Blanca at the University of Valencia to extract everything you need from the SFF files (sequence, qualities and ancillary information). The home of sff_extract is: http://bioinf.comav.upv.es/sff_extract/index.html but I am thankful to Jose for giving permission to distribute the script in the MIRA 3rd party package (separate download).

14.2.3 Do I need the sfftools from the Roche software package?

No, not anymore. Use the **sff_extract** script to extract your reads. Though the Roche sfftools package containes a few additional utilities which could be useful.

14.2.4 Combining SFFs

```
I am trying to use MIRA to assemble reads obtained with the 454 technology
but I can't combine my sff files since I have two files obtained with GS20
system and 2 others obtained with the GS-FLX system. Since they use
different cycles (42 and 100) I can't use the sfffile to combine both.
```

You do not need to combine SFFs before translating them into something MIRA (or other software tools) understands. Use **sff_extract** which extracts data from the SFF files and combines this into input files.

14.2.5 Adaptors and paired-end linker sequences

```
I have no idea about the adaptor and the linker sequences, could you send me
the sequences please?
```

Here are the sequences as filed by 454 in their patent application:

```
>AdaptorA
CTGAGACAGGGAGGGAACAGATGGGACACGCAGGGATGAGATGG
>AdaptorB
CTGAGACACGCAACAGGGGATAGGCAAGGCACACAGGGGATAGG
```

However, looking through some earlier project data I had, I also retrieved the following (by simply making a consensus of sequences that did not match the target genome anymore):

```
>5prime454adaptor???
GCCTCCCTCGCGCCATCAGATCGTAGGCACCTGAAA
>3prime454adaptor???
GCCTTGCCAGCCCGCTCAGATTGATGGTGCCTACAG
```

Go figure, I have absolutely no idea where these come from as they also do not comply to the "tcag" ending the adaptors should have.

I currently know one linker sequence (454/Roche also calls it *spacer* for GS20 and FLX paired-end sequencing:

```
>flxlinker
GTTGGAACCGAAAGGGTTTGAATTCAAACCCTTTCGGTTCCAAC
```

For Titanium data using standard Roche protocol, you need to screen for two linker sequences:

```
>titlinker1
TCGTATAACTTCGTATAATGTATGCTATACGAAGTTATTACG
>titlinker2
CGTAATAACTTCGTATAGCATACATTATACGAAGTTATACGA
```

Warning Some sequencing labs modify the adaptor sequences for tagging and similar things. Ask your sequencing provider for the exact adaptor and/or linker sequences.

14.2.6 What do I get in paired-end sequencing?

```
Another question I have is does the read pair sequences have further
adaptors/vectors in the forward and reverse strands?
```

Like for normal 454 reads - the normal A and B adaptors can be present in paired-end reads. That theory this could could look like this:

A-Adaptor - DNA1 - Linker - DNA2 - B-Adaptor.

It's possible that one of the two DNA fragments is *very* short or is missing completely, then one has something like this:

A-Adaptor - DNA1 - Linker - B-Adaptor

or

A-Adaptor - Linker - DNA2 - B-Adaptor

And then there are all intermediate possibilities with the read not having one of the two adaptors (or both). Though it appears that the majority of reads will contain the following:

DNA1 - Linker - DNA2

There is one caveat: according to current paired-end protocols, the sequences will **NOT** have the direction

```
---> Linker <---
```

as one might expect when being used to Sanger Sequencing, but rather in this direction

```
<--- Linker --->
```

14.2.7 Sequencing protocol

```
Is there a way I can find out which protocol was used?
```

Yes. The best thing to do is obviously to ask your sequencing provider.

If this is - for whatever reason - not possible, this list might help.

Are the sequences ~100-110 bases long? It's GS20.

Are the sequences ~220-250 bases long? It's FLX.

Are the sequences ~350-450 bases long? It's Titanium.

Do the sequences contain a linker (GTTGGAACCGAAAGGGTTTGAATTCAAACCCTTTCGGTTCCAAC)? It's a paired end protocol.

If the sequences left and right of the linker are ~29bp, it's the old short paired end (SPET, also it's most probably from a GS20). If longer, it's long paired-end (LPET, from a FLX).

14.2.8 Filtering sequences by length and re-assembly

```
I have two datasets of ~500K sequences each and the sequencing company
already did an assembly (using MIRA) on the basecalled and fully processed
reads (using of course the accompanying *qual file). Do you suggest that I
should redo the assembly after filtering out sequences being shorter than a
certain length (eg those that are <200bp)? In other words, am I taking into
account low quality sequences if I do the assembly the way the sequencing
company did it (fully processed reads + quality files)?
```

I don't think that filtering out "shorter" reads will bring much positive improvement. If the sequencing company used the standard Roche/454 pipeline, the cut-offs for quality are already quite good, remaining sequences should be, even when being < 200bp, not of bad quality, simply a bit shorter.

Worse, you might even introduce a bias when filtering out short sequences: chemistry and library construction being what they are (rather imprecise and sometimes problematic), some parts of DNA/RNA yield smaller sequences per se ... and filtering those out might not be the best move.

You might consider doing an assembly if the company used a rather old version of MIRA (<3.0.0 for sure, perhaps also <3.0.5).

14.3 Solexa / Illumina data

14.3.1 Can I see deletions?

```
Suppose you ran the genome of a strain that had one or more large
deletions. Would it be clear from the data that a deletion had occurred?
```

In the question above, I assume you'd compare your strain *X* to a strain *Ref* and that *X* had deletions compared to *Ref*. Furthermore, I base my answer on data sets I have seen, which presently were 36 and 76 mers, paired and unpaired.

Yes, this would be clear. And it's a piece of cake with MIRA.

Short deletions (1 to 10 bases): they'll be tagged SROc or WRMc. General rule: deletions of up to 10 to 12% of the length of your read should be found and tagged without problem by MIRA, above that it may or may not, depending a bit on coverage, indel distribution and luck.

Long deletions (longer than read length): they'll be tagged with MCVc tag by MIRA ins the consensus. Additionally, when looking at the FASTA files when running the CAF result through convert_project: long stretches of sequences without coverage (the @ sign in the FASTAs) of *X* show missing genomic DNA.

14.3.2 Can I see insertions?

```
Suppose you ran the genome of a strain X that had a plasmid missing from the
reference sequence. Alternatively, suppose you ran a strain that had picked
up a prophage or mobile element lacking in the reference. Would that
situation be clear from the data?
```

Short insertions (1 to 10 bases): they'll be tagged SROc or WRMc. General rule: deletions of up to 10 to 12% of the length of your read should be found and tagged without problem by MIRA, above that it may or may not, depending a bit on coverage, indel distribution and luck.

Long insertions: it's a bit more work than for deletions. But if you ran a de-novo assembly on all reads not mapped against your reference sequence, chances are good you'd get good chunks of the additional DNA put together

Once the Solexa paired-end protocol is completely rolled out and used on a regular base, you would even be able to place the additional element into the genome (approximately).

14.3.3 De-novo assembly with Solexa data

```
Any chance you could assemble de-novo the sequence of a from just the Solexa
data?
```

Warning Highly opinionated answer ahead, your mileage may vary.

Allow me to make a clear statement on this: maybe.

But the result would probably be nothing I would call a good assembly. If you used anything below 76mers, I'm highly sceptical towards the idea of de-novo assembly with Solexa (or ABI SOLiD) reads that are in the 30 to 50bp range. They're really too short for that, even paired end won't help you much (especially if you have library sizes of just 200 or 500bp). Yes, there are papers describing different draft assemblers (SHARCGS, EDENA, Velvet, Euler and others), but at the moment the results are less than thrilling to me.

If a sequencing provider came to me with N50 numbers for an *assembled genome* in the 5-8 Kb range, I'd laugh him in the face. Or weep. I wouldn't dare to call this even 'draft'. I'd just call it junk.

On the other hand, this could be enough for some purposes like, e.g., getting a quick overview on the genetic baggage of a bug. Just don't expect a finished genome.

14.4 Hybrid assemblies

14.4.1 What are hybrid assemblies?

Hybrid assemblies are assemblies where one used more than one sequencing technology. E.g.: Sanger and 454, or 454 and Solexa, or Sanger and Solexa etc.pp

14.4.2 What differences are there in hybrid assembly strategies?

Basically, one can choose two routes: multi-step or all-in-one-go.

Multi-steps means: to assemble reads from one sequencing technology (ideally the one from the shorter tech like, e.g., Solexa), fragment the resulting contigs into pseudo-reads of the longer tech and assemble these with the real reads from the longer tech (like, e.g., 454). The advantage of this approach is that it will be probably quite faster than the all-in-one-go approach. The disadvantage is that you loose a lot of information when using only consensus sequence of the shorter read technology for the final assembly.

All-in-one-go means: use all reads in one single assembly. The advantage of this is that the resulting alignment will be made of true reads with a maximum of information contained to allow a really good finishing. The disadvantage is that the assembly will take longer and will need more RAM.

14.5 Masking

14.5.1 Should I mask?

```
In EST projects, do you think that the highly repetitive option will get rid
of the repetitive sequences without going to the step of repeat masking?
```

For eukaryotes, yes. Please also consult the -SK:mnr option.

Remember: you still **MUST** have sequencing vectors and adaptors clipped! In EST sequences the poly-A tails should be also clipped (or let mira do it.

For prokaryotes, I´m a big fan of having a first look at unmasked data. Just try to start MIRA without masking the data. After something like 30 minutes, the all-vs-all comparison algorithm should be through with a first comparison round. grep the log for the term "megahub" ... if it doesn't appear, you probably don't need to mask repeats

14.5.2 How can I apply custom masking?

```
I want to mask away some sequences in my input. How do I do that?
```

First, if you want to have Sanger sequencing vectors (or 454 adaptor sequences) "masked", please note that you should rather use ancillary data files (CAF, XML or EXP) and use the sequencing or quality clip options there.

Second, please make sure you have read and understood the documentation for all -CL parameters in the main manual, but especially -CL:mbc:mbcgs:mbcmfg:mbcmeg as you might want to switch it on or off or set different values depending on your pipeline and on your sequencing technology.

You can without problem mix your normal repeat masking pipeline with the FASTA or EXP input for MIRA, as long as you **mask** and not **clip** the sequence.

An example:

```
>E09238ARF0
tcag GTGTCAGTGTTGACTGTAAAAAAAAAGTACGTATGGACTGCATGTGCATGTCATGGTACGTGTCA
GTCAGTACAAAAAAAAAAAAAAAAAAAAAGTACGT tgctgacgcacatgatcgtagc
```

(spaces inserted just as visual helper in the example sequence, they would not occur in the real stuff)

The XML will contain the following clippings: left clip = 4 (clipping away the "tcag" which are the last four bases of the adaptor used by Roche) right clip= ~90 (clipping away the "tgctgac..." lower case sequence on the right side of the sequence above.

Now, on the FASTA file that was generated with reads_sff.py or with the Roche sff* tools, you can let run, e.g., a repeat masker. The result could look like this:

```
>E09238ARF0
tcag XXXXXXXXX TTGACTGTAAAAAAAAAGTACGTATGGACTGCATGTGCATGTCATGGTACGTGTCA
GTCAGTACAAAAAAAAAAAAAAAAAAAAAGTACGT tgctgacgcacatgatcgtagc
```

The part with the Xs was masked away by your repeat masker. Now, when MIRA loads the FASTA, it will first apply the clippings from the XML file (they're still the same). Then, if the option to clip away masked areas of a read (-CL:mbc, which is normally on for EST projects), it will search for the stretches of X and internally also put clips to the sequence. In the example above, only the following sequence would remain as "working sequence" (the clipped parts would still be present, but not used for any computation.

```
>E09238ARF0
..............TTGACTGTAAAAAAAAAGTACGTATGGACTGCATGTGCATGTCATGGTACGTGTCA
GTCAGTACAAAAAAAAAAAAAAAAAAAAAGTACGT......................
```

Here you can also see the reason why your filters should **mask** and not clip the sequence. If you change the length of the sequence, the clips in the XML would not be correct anymore, wrong clippings would be made, wrong sequence reconstructed, chaos ensues and the world would ultimately end. Or something.

IMPORTANT! It might be that you do not want MIRA to merge the masked part of your sequence with a left or right clip, but that you want to keep it something like DNA - masked part - DNA. In this case, consult the manual for the -CL:mbc switch, either switch it off or set adequate options for the boundaries and gap sizes.

Now, if you look at the sequence above, you will see two possible poly-A tails ... at least the real poly-A tail should be masked else you will get megahubs with all the other reads having the poly-A tail.

You have two possibilities: you mask yourself with an own program or you let MIRA do the job (-CL:cpat, which should normally be on for EST projects but I forgot to set the correct switch in the versions prior to 2.9.26x3, so you need to set it manually for 454 EST projects there).

IMPORTANT! Never ever at all use two poly-A tail masker (an own and the one from MIRA): you would risk to mask too much. Example: assume the above read you masked with a poly-A masker. The result would very probably look like this:

```
>E09238ARF0
tcag XXXXXXXXX TTGACTGTAAAAAAAAAGTACGTATGGACTGCATGTGCATGTCATGGTACGTGTCA
GTCAGTAC XXXXXXXXXXXXXXXXXXX GTACGT tgctgacgcacatgatcgtagc
```

And MIRA would internally make the following out of it after loading:

```
>E09238ARF0
..............TTGACTGTAAAAAAAAAGTACGTATGGACTGCATGTGCATGTCATGGTACGTGTCA
GTCAGTAC................................................
```

and then apply the internal poly-A tail masker:

```
>E09238ARF0
..............TTGACTGT................................................
........................................................
```

You'd be left with ... well, a fragment of your sequence.

14.6 Miscellaneous

14.6.1 What are megahubs?

```
I looked in the log file and that term "megahub" you told me about appears
pretty much everywhere. First of all, what does it mean?
```

Megahub is the internal term for MIRA that the read is massively repetitive with respect to the other reads of the projects, i.e., a read that is a megahub connects to an insane number of other reads.

This is a clear sign that something is wrong. Or that you have a quite repetitive eukaryote. But most of the time it's sequencing vectors (Sanger), A and B adaptors or paired-end linkers (454), unmasked poly-A signals (EST) or non-normalised EST libraries which contain high amounts of housekeeping genes (always the same or nearly the same).

Countermeasures to take are:

- set clips for the sequencing vectors (Sanger) or Adaptors (454) either in the XML or EXP files
- for ESTs, mask poly-A in your input data (or let MIRA do it with the -CL:cpat parameter)
- only after the above steps have been made, use the -SK:mnr switch to let mira automatically mask nasty repeats, adjust the threshold with -SK:rt
- if everything else fails, filter out or mask sequences yourself in the input data that come from housekeeping genes or nasty repeats.

14.6.2 Passes and loops

```
While processing some contigs with repeats i get
"Accepting probably misassembled contig because of too many iterations."
What is this?
```

That's quite normal in the first few passes of an assembly. During each pass (-AS:nop), contigs get built one by one. After a contig has been finished, it checks itself whether it can find misassemblies due to repeats (and marks these internally). If no misassembly, perfect, build next contig. But if yes, the contig requests immediate re-assembly of itself.

But this can happen only a limited number of times (governed by -AS:rbl). If there are still misassemblies, the contig is stored away anyway ... chances are good that in the next full pass of the assembler, enough knowledge has been gained top correctly place the reads.

So, you need to worry only if these messages still appear during the last pass. The positions that cause this are marked with "SRMc" tags in the assemblies (CAF, ACE in the result dir; and some files in the info dir).

14.6.3 Debris

```
What are the debris composed of?
```

- sequences too short (after trimming)
- megahubs
- sequences almost completely masked by the nasty repeat masker ([-SK:mnr])
- singlets, i.e., reads that after an assembly pass did not align into any contig (or where rejected from every contig).
- sequences that form a contig with less reads than defined by [-AS:mrpc]

14.6.4 Log and temporary files: more info on what happened during the assembly

```
I do not understand why ... happened. Is there a way to find out?
```

Yes. The tmp directory contains, beside temporary data, a number of log files with more or less readable information. While development versions of MIRA keep this directory after finishing, production versions normally delete this directory after an assembly. To keep the logs and temporary file also in production versions, use "-OUT:rtd=no".

As MIRA also tries to save as much disk space as possible, some logs and temporary files are rotated (which means that old logs and tmps get deleted). To switch off this behaviour, use "-OUT:rrot=no". Beware, the size of the tmp directory will increase, sometimes dramatically so.

14.6.4.1 Sequence clipping after load

How MIRA clipped the reads after loading them can be found in the file `mira_int_clippings.0.txt`. The entries look like this:

```
   load:  minleft. U13a01d05.t1    Left: 11         -> 30
```

Interpret this as: after loading, the read "U13a01d05.t1" had a left clipping of eleven. The "minleft" clipping option of MIRA did not like it and set it to 30.

```
   load:  bad seq. gnl|ti|1133527649       Shortened by 89 New right: 484
```

Interpret this as: after loading, the read "gnl|ti|1133527649" was checked with the "bad sequence search" clipping algorithm which determined that there apparently is something dubious, so it shortened the read by 89 bases, setting the new right clip to position 484.

14.7 Platforms and Compiling

14.7.1 Windows

```
Also, is MIRA be available on a windows platform?
```

As a matter of fact: it was and may be again. While I haven't done it myself, according to reports I got compiling MIRA 2.9.3* in a Cygwin environment was actually painless. But since then BOOST and multi-threading has been included and I am not sure whether it is still as easy.

I'd be thankful for reports :-)

Chapter 15

The MAF format

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'Design flaws travel in herds. '*
>
> —Solomon Short

This documents describes purpose and format of the MAF format, version 1.

15.1 Introduction: why an own assembly format?

I had been on the hunt for some time for a file format that allow MIRA to quickly save and load reads and full assemblies. There are currently a number of alignment format files on the market and MIRA can read and/or write most of them. Why not take one of these? It turned out that all (well, the ones I know: ACE, BAF, CAF, CALF, EXP, FRG) have some kind of no-go 'feature' (or problem or bug) that makes one life pretty difficult if one wants to write or parse that given file format.

What I needed for MIRA was a format that:

1. is easy to parse
2. is quick to parse
3. contains all needed information of an assembly that MIRA and many finishing programs use: reads (with sequence and qualities) and contigs, tags etc.pp

MAF is not a format with the smallest possible footprint though it fares quite well in comparison to ACE, CAF and EXP), but as it's meant as interchange format, it'll do. It can be easily indexed and does not need string lookups during parsing.

I took the liberty to combine many good ideas from EXP, BAF, CAF and FASTQ while defining the format and if anything is badly designed, it's all my fault.

15.2 The MAF format

This describes version 1 of the MAF format. If the need arises, enhancements like metadata about total number of contigs and reads will be implemented in the next version.

15.2.1 Basics

MAF ...

1. ... has for each record a keyword at the beginning of the line, followed by exactly one blank (a space or a tab), then followed by the values for this record. At the moment keywords are two character keywords, but keywords with other lengths might appear in the future
2. ... is strictly line oriented. Each record is terminated by a newline, no record spans across lines.

All coordinates start at 1, i.e., there is no 0 value for coordinates.

15.2.2 Reads

15.2.2.1 Simple example

Here's an example for a simple read, just the read name and the sequence:

```
RD      U13a05e07.t1
RS      CTTGCATGCCTGCAGGTCGACTCTAGAAGGACCCCGATCA
ER
```

Reads start with RD and end with ER, the RD keyword is always followed by the name of the read, ER stands on its own. Reads also should contain a sequence (RS). Everything else is optional. In the following example, the read has additional quality values (RQ), template definitions (name in TN, minimum and maximum insert size in TF and TT), a pointer to the file with the raw data (SF), a left clip which covers sequencing vector or adaptor sequence (SL), a left clip covering low quality (QL), a right clip covering low quality (QR), a right clip covering sequencing vector or adaptor sequence (SR), alignment to original sequence (AO), a tag (RT) and the sequencing technology it was generated with (ST).

```
RD      U13a05e07.t1
RS      CTTGCATGCCTGCAGGTCGACTCTAGAAGGACCCCGATCA
RQ      ,-+*,1-+/,36;:6≤3327<7A1/,,).('..7=@E8:
TN      U13a05e07
DI      F
TF      1200
TT      1800
SF      U13a05e07.t1.scf
SL      4
QL      7
QR      30
SR      32
AO      1 40 1 40
RT      ALUS 10 15 Some comment to this read tag.
ST      Sanger
ER
```

15.2.2.2 List of records for reads

- RD *string: readname*

  RD followed by the read name starts a read.

- LR *integer: read length*

  The length of the read can be given optionally in LR. This is meant to help the parser perform sanity checks and eventually pre-allocate memory for sequence and quality.

  MIRA at the moment only writes LR lines for reads with more than 2000 bases.

- RS *string: DNA sequence*

  Sequence of a read is stored in RS.

- RQ *string: qualities*

  Qualities are stored in FASTQ format, i.e., each quality value + 33 is written as single as ASCII character.

- SV *string: sequencing vector*

  Name of the sequencing vector or adaptor used in this read.

- TN *string: template name*

  Template name. This defines the DNA template a sequence comes from. In it's simplest form, a DNA template is sequenced only once. In paired-end sequencing, a DNA template is sequenced once in forward and once in reverse direction (Sanger, 454, Solexa). In Sanger sequencing, several forward and/or reverse reads can be sequenced from a DNA template. In PacBio sequencing, a DNA template can be sequenced in several "strobes", leading to multiple reads on a DNA template.

- DI *character: F or R*

  Direction of the read with respect to the template. F for forward, R for reverse.

- TF *integer: template size from*

  Minimum estimated size of a sequencing template. In paired-end sequencing, this is the minimum distance of the read pair.

- TT *integer: template size to*

  Maximum estimated size of a sequencing template. In paired-end sequencing, this is the maximum distance of the read pair.

- SF *string: sequencing file*

  Name of the sequencing file which contains raw data for this read.

- SL *integer: seqvec left*

  Clip left due to sequencing vector. Assumed to be 1 if not present. Note that left clip values are excluding, e.g.: a value of '7' clips off the left 6 bases.

- QL *integer: qual left*

  Clip left due to low quality. Assumed to be 1 if not present. Note that left clip values are excluding, e.g.: a value off '7' clips of the left 6 bases.

- CL *integer: clip left*

  Clip left (any reason). Assumed to be 1 if not present. Note that left clip values are excluding, e.g.: a value of '7' clips off the left 6 bases.

- SR *integer: seqvec right*

  Clip right due to sequencing vector. Assumed to be the length of the sequence if not present. Note that right clip values are including, e.g., a value of '10' leaves the bases 1 to 9 and clips at and including base 10 and higher.

- QR *integer: qual right*

  Clip right due to low quality. Assumed to be the length of the sequence if not present. Note that right clip values are including, e.g., a value of '10' leaves the bases 1 to 9 and clips at and including base 10 and higher.

- CR *integer: clip right*

  Clip right (any reason). Assumed to be the length of the sequence if not present. Note that right clip values are including, e.g., a value of '10' leaves the bases 1 to 9 and clips at and including base 10 and higher.

- AO *four integers: x1 y1 x2 y2*

  AO stands for "Align to Original". The interval [x1 y1] in the read as stored in the MAF file aligns with [x2 y2] in the original, unedited read sequence. This allows to model insertions and deletions in the read and still be able to find the correct position in the original, base-called sequence data.

  A read can have several AO lines which together define all the edits performed to this read.

  Assumed to be "1 x 1 x" if not present, where 'x' is the length of the unclipped sequence.

- RT *string + 2 integers + optional string: type x1 y1 comment*

  Read tags are given by naming the tag type, which positions in the read the tag spans in the interval [x1 y1] and afterwards optionally a comment. As MAF is strictly line oriented, newline characters in the comment are encoded as `\n`.

  If x1 > y1, the tag is in reverse direction.

  The tag type can be a free form string, though MIRA will recognise and work with tag types used by the Staden gap4 package (and of course the MIRA tags as described in the main documentation of MIRA).

- ST *string: sequencing technology*

  The current technologies can be defined: Sanger, 454, Solexa, SOLiD.

- SN *string: strain name*

  Strain name of the sample that was sequenced, this is a free form string.

- MT *string: machine type*

  Machine type which generated the data, this is a free form string.

- BC *string: base caller*

  Base calling program used to call bases

- IB *boolean (0 or 1): is backbone*

  Whether the read is a backbone. Reads used as reference (backbones) in mapping assemblies get this attribute.

- IC *boolean (0 or 1)*

  Whether the read is a coverage equivalent read (e.g. from mapping Solexa). This is internal to MIRA.

- IR *boolean (0 or 1)*

  Whether the read is a rail. This also is internal to MIRA.

- ER

  This ends a read and is mandatory.

15.2.2.3 Interpreting clipping values

Every left and right clipping pair (SL & SR, QL & QR, CL & CR) forms a clear range in the interval [left right[in the sequence of a read. E.g. a read with SL=4 and SR=10 has the bases 1,2,3 clipped away on the left side, the bases 4,5,6,7,8,9 as clear range and the bases 10 and following clipped away on the right side.

The left clip of a read is determined as max(SL,QL,CL) (the rightmost left clip) whereas the right clip is min(SR,QR,CR).

15.2.3 Contigs

Contigs are not much more than containers containing reads with some additional information. Contrary to CAF or ACE, MAF does not first store all reads in single containers and then define the contigs. In MAF, contigs are defined as outer container and within those, the reads are stored like normal reads.

15.2.3.1 Simple example 2

The above example for a read can be encased in a contig like this (with two consensus tags gratuitously added in):

```
CO      contigname_s1
NR      1
LC      24
CS      TGCCTGCAGGTCGACTCTAGAAGG
CQ      -+/,36;:6≤3327<7A1/,,).
CT      COMM 5 8 Some comment to this consensus tag.
```

```
CT      COMM 7 12 Another comment to this consensus tag.
\\
RD      U13a05e07.t1
RS      CTTGCATGCCTGCAGGTCGACTCTAGAAGGACCCCGATCA
RQ      ,-+*,1-+/,36;:6≤3327<7A1/,,).('..7=@E8:
TN      U13a05e07
TF      1200
TT      1800
SF      U13a05e07.t1.scf
SL      4
SR      32
QL      7
QR      30
AO      1 40 1 40
RT      ALUS 10 15 Some comment to this read tag.
ST      Sanger
ER
AT      1 24 7 30
//
EC
```

Note that the read shown previously (and now encased in a contig) is absolutely unchanged. It has just been complemented with a bit of data which describes the contig as well as with a one liner which places the read into the contig.

15.2.3.2 List of records for contigs

- CO *string: contig name*

  CO starts a contig, the contig name behind is mandatory but can be any string, including numbers.

- NR *integer: num reads in contig*

  This is optional but highly recommended.

- LC *integer: contig length*

  Note that this length defines the length of the 'clear range' of the consensus. It is 100% equal to the length of the CS (sequence) and CQ (quality) strings below.

- CT *string + 2 integers + optional string: identifier x1 y1 comment*

  Consensus tags are defined like read tags but apply to the consensus. Here too, the interval [x1 y1] is including and if x1 > y1, the tag is in reverse direction.

- CS *string: consensus sequence*

  Sequence of a consensus is stored in RS.

- CQ *string: qualities*

  Consensus Qualities are stored in FASTQ format, i.e., each quality value + 33 is written as single as ASCII character.

- \\

  This marks the start of read data of this contig. After this, all reads are stored one after the other, just separated by an "AT" line (see below).

- AT *Four integers: x1 y1 x2 y2*

  The AT (Assemble_To) line defines the placement of the read in the contig and follows immediately the closing "ER" of a read so that parsers do not need to perform time consuming string lookups. Every read in a contig has exactly one AT line.

  The interval [x2 y2] of the read (i.e., the unclipped data, also called the 'clear range') aligns with the interval [x1 y1] of the contig. If x1 > y1 (the contig positions), then the reverse complement of the read is aligned to the contig. For the read positions, x2 is always < y2.

- //

  This marks the end of read data

- EC

  This ends a contig and is mandatory

Chapter 16

Log and temporary files used by MIRA 3

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

'The amount of entropy in the universe is constant - except when it increases. '

—Solomon Short

16.1 Introduction

The tmp directory used by mira (usually `<projectname>_d_tmp`) may contain a number of files with information which could be interesting for other uses than the pure assembly. This guide gives a short overview.

Note This guide is probably the least complete and most out-of-date as it is updated only very infrequently. If in doubt, ask on the MIRA talk mailing list.

Warning Please note that the format of these files may change over time, although I try very hard to keep changes reduced to a minimum.

Note Remember that mira has two options that control whether log and temporary files get deleted: while [-OUT:rtd] removes the complete tmp directory after an assembly, [-OUT:rrot] removes only those log and temporary files which are not needed anymore for the continuation of the assembly. Setting both options to no will keep all log and temporary files.

16.2 The files

16.2.1 mira_error_reads_invalid

A simple list of those reads that were invalid (no sequence or similar problems).

16.2.2 mira_info_reads_tooshort

A simple list of those reads that were sorted out because the unclipped sequence was too short as defined by [-AS:mrl].

16.2.3 mira_int_alignextends_preassembly1.0.txt

If read extension is used ([-DP:ure]), this file contains the read name and the number of bases by which the right clipping was extended.

16.2.4 mira_int_clippings.0.txt

If any of the [-CL:] options leads to the clipping of a read, this file will tell when, which clipping, which read and by how much (or to where) the clippings were set.

16.2.5 mira_int_posmatch_megahubs_pass.X.lst

Note: replace the *X* by the pass of mira. Should any read be categorised as megahub during the all-against-all search (SKIM3), this file will tell you which.

16.2.6 mira_int_posmatch_multicopystat_preassembly.0.txt

After the initial all-against-all search (SKIM3), this file tells you to how many other reads each read has overlaps. Furthermore, reads that have more overlaps than expected are tagged with ``mc'' (multycopy).

16.2.7 mira_int_posmatch_rawhashhits_pass.X.lst

Note: replace the *X* by the pass of mira. Similar to `mira_int_posmatch_multicopystat_preassembly.0.txt`, this counts the hash hits of each read to other reads. This time however per pass.

16.2.8 mira_int_skimmarknastyrepeats_hist_pass.X.lst

Note: replace the *X* by the pass of mira. Only written if [-SK:mnr] is set to yes. This file contains a histogram of hash occurrences encountered by SKIM3.

16.2.9 mira_int_skimmarknastyrepeats_nastyseq_pass.X.lst

Note: replace the *X* by the pass of mira. Only written if [-SK:mnr] is set to yes. One of the more interesting files if you want to know the repetitive sequences cause the assembly to be really difficult: for each masked part of a read, the masked sequences is shown here.

E.g.

```
U13a04h11.t1    TATATATATATATATATATATATA
U13a05b01.t1    TATATATATATATATATATATATA
U13a05c07.t1    AAAAAAAAAAAAAAA
U13a05e12.t1    CTCTCTCTCTCTCTCTCTCTCTCTCTC
```

Simple repeats like the ones shown above will certainly pop-up there, but a few other sequences (like e.g. SINEs, LINEs in eukaryotes) will also appear.

Nifty thing to try out if you want to have a more compressed overview: sort and unify by the second column.

```
sort -k 2 -u mira_int_skimmarknastyrepeats_nastyseq_pass.X.lst
```

16.2.10 mira_int_vectorclip_pass.X.txt

Note: replace the *X* by the pass of mira. Only written if [-CL:pvlc] is set to yes. Tells you where possible sequencing vector (or adaptor) leftovers were found and clipped (or not clipped).

16.2.11 miratmp.ads_pass.X.forward and miratmp.ads_pass.X.complement

Note: replace the *X* by the pass of mira. Which read aligns with Smith-Waterman against which other read, 'forward-forward' and 'forward-complement'.

16.2.12 miratmp.ads_pass.X.reject

Note: replace the *X* by the pass of mira. Which possible read overlaps failed the Smith-Waterman alignment check.

16.2.13 miratmp.noqualities

Which reads went completely without qualities into the assembly.

16.2.14 miratmp.usedids

Which reads effectively went into the assembly (after clipping etc.).

16.2.15 mira_readpoolinfo.lst

Chapter 17

Bonus material

MIRA Version 3.4.1.1 *Document revision Id* Bastien Chevreux 2011Bastien Chevreux

> *'I'm all in favour of keeping dangerous weapons out of the hand of fools. Let's start with typewriters.'*
>
> —Solomon Short

17.1 Death of the ATINSEQ "bug"

My warmest thanks to Bob Bruccoleri, Lionel Guy, Arun Rawat, Nestor Zaburannyi and numerous other people - who either declined being thanked publicly or could not respond before I wrote this (you still can, just drop me a mail) - who all donated time and computing power to hunt down a "bug" (see http://www.freelists.org/post/mira_talk/Call-for-help-bughunting) which, in the end, turned out to be a RAM defect on my development machine.

Because I haven't much apart some words to thank them, and because I felt like it ... this is the story on how the problem got nailed. It involves lots of hot electrons, a lot less electrons without spin which keel over, the end of a hunt for invisibugs of the imaginary sort, 454, mutants (but no zombies), Illumina, some spider monkeys, PacBio, a chat with Sherlock and, of course, an anthropomorphed star.

In short: don't read if you've got more interesting things to do on a Friday morning or afternoon.

...

Life's a rollercoaster and there are days - or weeks - where moral is on a pretty hefty ride: ups and downs in fast succession ... and the occasional looping here and there.

Today was a day where I had - the first time ever - ups and downs occuring absolutely simultaneously. Something which is physically impossible, I know, but don't tell any physicist or astronomist about that or else they'll embark you on a lengthy discussion on how isochronicity is a myth by telling you stories on lightning, thunder and two poor sobs at the ends of a 300,000 km long train. But I digress ...

So, my lowest low and highest high today were at 09:17 this morning when I prepared leaving for work (hey, it's vacation time, almost everyone else is out and I can go a bit later than usual, right?). A few minutes earlier I had just told MIRA to run on the very same PacBio test set she had successfully worked on the night before to see how stable assemblies with this kind of data are (quite well so far, thank you for asking).

Reaching out to switch off the monitor and leave, MIRA suddenly came back with a warm and cosy little error message which she's taken the habit lately to have a mischievous pleasure to present. This time, she claimed there had been an illegal base in the FASTQ file.

'Hey, MIRA, wait a minute!' I thought. *'Yesterday and tonight you ran on the very same data file with the very same parameters for two times three hours and even gave me back some nice assembly results. And now you claim that the INPUT data has errors?! Come on, you're not serious, are you?'*

As a side note: she then just gave me back *that look*, you know, the one with those big open eyes almost hidden behind by long, dark lashes ... and slightly flushed cheeks accompanied by pointed lips. As if she wanted to say *'I *am* innocent and *I* did no nothing wrong, you disbeliever!'* (http://24.media.tumblr.com/tumblr_lj3efmmDL01qasfhmo1_400.png). This usually announces a major pouting round of hers, something which I'm not looking forward to, I can tell you.

Two restarts later with the same negative result (MIRA can be quite stubborn at times) I had to give in and decided to sit down again and investigate the problem.

'So ... read number 317301 at base position 246, eh? Let's have a look."'

clickedyclick

'Read 317299, 317300 ... 317301 ... there we are.'

hackedyhack

'Base position 239, 240 ... now: C G G G T C F A A... wait! What? 'F' ... 'F'*?!? That's not even an IUPAC code. What's a frakking 'F' doing in the FASTQ input file?!'* (see http://www.youtube.com/watch?v=r7KcpgQKo2I, conditionally safe for work).

Indeed, 'F' is not IUPAC. Even more mysterious to me was the fact that just the night before it apparently had not been there. Or had it? I now was pretty unsure where this path would lead me, as if I had unlocked a door with the key of imagination. Beyond it: another dimension - a dimension of sound, a dimension of sight, a dimension of mind. I was moving into a land of both shadow and substance, of things and ideas. I just crossed over into ... the Twilight Zone ("G#-A-G#-E-G#-A-G#-E" at 128 bpm, for more info see http://www.youtube.com/watch?v=zi6wNGwd84g).

Where was I? Ah, yes, the 'F'.

So, how did that 'F' appear in the FASTQ, and where had it been the night before? Out to town, ashamed of not being a nucleotide and getting a hangover without telling anyone up-front? Or did it subreptitiously sneak in from the outside, murdering an innnocent base and taking its place in hope no one would note? I didn't have the slightest clue, but I was determined to find that out.

First thing to check: the log files of the successful runs the previous night. MIRA's very chatty at times and tidying up after her has always been a chore (there, I'm feeling *that look* again in my back), but now was one of those occasions where not gagging her paid out as poking around the files she left behind proved to be interesting. Read 317301 showed the following at the position in doubt: "C G G G T C ___G___ A A" Without question: a 'G', and no 'F' in sight!

So MIRA had been right and the 'G' in the sequence of the file mysteriously mutated into an 'F' overnight. I must admit that I had grown suspicious of her in the past few weeks as she had seemed to become uncooperative at times. In particular she had been screaming at me a couple of times during rehearsal of combined 454 and Illumina assemblies for the premiere of her new 3.4.0 show. She claimed that some uninvited spider monkeys (see http://dict.leo.org/ende?search=Klammeraffe) had frightened her so much she refused to continue to work and simply scribbled the '@' sign all over her error messages. I had not been able to find out how those critters entered MIRA's data and had even enrolled a few volunteers to rehearse different assemblies with MIRA ... to no avail as she'd performed without flaws there.

While reconsidering all these things, something suddenly made *click*.

The character 'G' has the hexadecimal ASCII table code 0x47 (or in 8-bit binary: 01000111). 'F', as preceding character of 'G' and the table having some logic behind it, has the hex code 0x46, which is 01000110 in 8-bit binary.

The ATINSEQ-bug (@-in-seq) I had been desperately hunting in the past few weeks (and which had held up the release of MIRA 3.4.0) was due to the "@" character sometimes mysteriously appearing in sequences during the assembly of MIRA. The '@' sign in the ASCII table has the hex code 0x40 (binary: 01000000). In the ASCII table, there is one important character for DNA assembly which is very near to the '@' character ..., so near that it is the successor of it: the 'A' character. Hexadecimal 0x41, binary 01000001.

I had always thought that a bug in MIRA somehow corrupted the sequence, but what if ... what if MIRA was actually really innocent?! I had never taken this possibility into account as any other explanation attempt would have seemed too far stretched.

But now I had a similar effect *outside* of MIRA, in the Linux filesystem!

The difference between the characters is in both cases exactly 1 bit which changes, and it's even at the same position (last one in a byte) and changing into the same direction (from '1' to '0').

I was now sure I was on to something: bit decay (see http://en.wikipedia.org/wiki/Bit_rot)

	Filesystem		MIRA	
before	G	01000111	A	01000001
after	F	01000110	@	01000000

But how could I prove it? Well, elementary my dear Watson: When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

Suspects:

- the problem is caused either by MIRA or one of the components of the comeputer: CPU, disk, disk/dma controller, RAM.

Facts:

- an artefact was very sporadically observed during MIRA runs where sequences (containing lot's of 'A') suddenly contained at least one '@'. This occured after several passes, i.e., not on loading.
- an artefact was observed in the Linux filesystem where a 'G' mutated suddenly and overnight to a 'F'.
- both artefacts are based on one bit flipping, perhaps even to the same direction all the time.
- when loading data, MIRA does not use mmap() to mirror data from disk, but physically creates a copy of that data.
- MIRA loaded the data twice flawlessly before the artefact in the filesystem occured.

Deduction 1:

- MIRA is innocent. The artefact in the filesystem happened outside of the address space of MIRA and therefore outside her control. MIRA cannot be responsible as the Linux kernel would have prevented her from writing to some memory she was not allowed to.

Further facts:

- the system MIRA ran on had 24 GiB RAM
- even with a KDE desktop, KMail, Firefox, Emacs and a bunch of terminals open, there is still a lot of free RAM (some 22 to 23 GiB).
- Linux uses free RAM to cache files

Deduction 2:

- when loading the small FASTQ input file in the morning, Linux put it into the file cache in RAM. As MIRA almost immediately stopped without taking much memory, the file stayed in cache.

Further facts:

- the drive with the FASTQ file is run in udma6 mode. That is, when loading data the controller moves the data directly from disk to RAM without going via the processor
- subsequent "loading" of the same FASTQ into MIRA or text viewer like 'less' showed the 'F' character always appearing at the same place.

Deduction 3:

- the CPU is innocent! It did not touch the data while it was transferred from disk to RAM and it afterwards shows always the same data.

- the disk controllers and UDMA controller are innocent! Some of the glitches observed in previous weeks occured during runs of MIRA, inside the MIRA address space, long after initial loading, when UDMA had already finished their job.

From deductions 1, 2 & 3 follows:

- it's not MIRA, not the CPU, nor the disk & UDMA controller

Suspects left:

- RAM
- Disk

Well, that can be easily tested: shut down the computer, restart it and subsequently look at the file again. No file cache in RAM can survive that procedure. Yes, I know, there are some magic incantations one can chant to force Linux to flush all buffers and clear all caches, but in that situation I was somehow feeling conservative.

Lo and behold, after the above procedure the FASTQ file showed an all regular, good old nucleic acid 'G' in the file again. No 'F' to be seen anywhere.

Deduction 4:

- the disk is innocent.

Deduction 5:

- as all other components have been ruled out, the RAM is faulty.

As I wrote: life's a rollercoaster.

Up: MIRA is innocent! There, she's giving me "that look" again and one would have to be blind to oversee the "told you so" she's sending over with it.

Down: My RAM's broken and I need to replace it. Bought it only last March, should still be under guarantee, but still ... time and effort.

Up: I did not sell my old RAMs, so I can continue to work

Down: 12 GiB feels soooooo tight after having had 24.

Up: I can wrap up 3.4.0 end of this week with good conscience!

Down: How the hell am I gonna tie all loose bits and pieces in the documentation in the next 24 to 48 hours?

Looping: later that morning MIRA again helped me at work to locate in a couple of minutes a mutation in an eukaryotic strain important for one of our Biotech groups. Oh boy, do I love sequencing and MIRA.

Have a nice Friday and a good week-end,

Bastien

PS: while celebrating with MIRA tonight, I expressed my fear that some people might find it strange that I anthropomorphise her. They could think I went totally nuts or that I needed an extended vacation (which I do btw). She assured me that no one would dare thinking I were insane ... and if so, she would come over to their place and give them *that look*.

How utterly reassuring.