Cassis
Detection of genomic rearrangement breakpoints
User manual

Christian Baudet and Claire Lemaitre
Equipe BAMBOO
INRIA Grenoble Rhône-Alpes
Laboratoire de Biométrie et Biologie Évolutive (UMR 5558) CNRS
Université Claude Bernard Lyon 1
F-69100 Villeurbanne, France

version 1.1 (25/02/2013)

Contents

Chapter 1

First steps

1.1 Introduction

The package `Cassis` implements methods for the precise detection of genomic rearrangement breakpoints which were described in Lemaitre *et al.* [1, 2].

This document contains information about how to obtain, install and run the package. It also provides a description of the package organisation.

The code was written in `Perl` and `R` and is distributed under the GNU GPL License. For more information, read the `COPYING` file which is distributed with the package or access `http://www.gnu.org/copyleft/gpl.html`.

1.2 Installation

1.2.1 System Requirements

`Cassis` was implemented in `Perl` and `R` and tested under the Linux and Mac OS environments. It has the following requirements:

- `Linux` or `Mac OS`
- `Perl`
- `R`
- `Ghostscript`[1]
- `LASTZ`

1.2.2 Obtaining the package

The package can be downloaded at the site of the Pôle Bioinformatique Lyonnais `PBIL`:

`http://pbil.univ-lyon1.fr/software/Cassis/`

1.2.3 Installing the package

Before installing the package, the user must check if his system contains the installations of `Perl` and `R`. Nowadays, `Perl` is distributed together with most `Linux` or `Mac` systems. If you need more information, visit `http://www.perl.org/`. On the other hand, `R` is not distributed in the basic installation of most `Linux` or `Mac` systems. If your system does not have `R`, access `http://www.r-project.org/` and follow the download and installation instructions.

To install the package, the user must obtain the file `cassis.zip` (see Section 1.2.2) and decompress it under the desired directory:

```
> cp cassis.zip <directory>
> cd <directory>
> unzip cassis.zip
```

where `<directory>` is the path for the desired directory (Example: `/usr/local/`). After decompressing the file `cassis.zip`, a directory named `cassis` can be found inside of `<directory>` that contains all package files.

To finish the installation, the user must download the source code of the program `LASTZ`. It can be obtained on the *CCGB Miller Lab website*:

`http://www.bx.psu.edu/miller_lab/`

The user must download the latest `<version>` of the source code, decompress and compile it and copy the generated executable file to the directory `<directory>/cassis/lastz`:

```
> tar -xvzf lastz-<version>.tar.gz
> cd lastz-distrib-<version>
> make
> cp src/lastz <directory>/cassis/lastz/
```

By doing these steps, the installation of `Cassis` is concluded and the package is ready to use (See Chapter 2 for instructions on how to use the script `cassis.pl`).

[1]`Ghostscript` is required by the `R` command `bitmap` which is used to create `png` images.

1.3 Ghostscript

To create `png` images (dotplot and segmentation plot of the breakpoints) `Cassis` uses the `R` command `bitmap`. This command requires that the software `gs` (`Ghostscript`) is installed in your computer. Additionaly, `R` should be able to see this command.

Usually, `R` searches for the software `gs` in the locations which are listed in the environment variable `PATH`. Thus, you can add the directory where the software `gs` is located to your `PATH` environment variable.

Alternatively, you can configure `R` by updating its `Renviron` [2] configuration file and add the following line:

```
R_GSCMD=${RGSCMD-'[PATH_GS]/gs'}
```

where `[PATH_GS]` is the directory where the software `gs` is located (`/usr/local/bin`, for example).

To verify if `R` is properly configured, open `R` in the console terminal of you `Linux` or `Mac OS` and type the following commands:

```
y=sample(500)
bitmap("teste.png", type="png16m", height=8, width=7, res=100)
plot(1:500, y, t="p", col="blue", pch=20)
dev.off()
```

If the file `teste.png` is created and contains a plot, `R` is properly configured.

1.4 32 or 64 bits

By default, `Cassis` uses the command `R CMD BATCH`. Thus, the architecture (32 or 64 bits) is associated with the default installation of `R` in your system.

In `Mac OS`, `R` is installed in 32 and 64 bits version. By default, `Cassis` uses the 32 bits version. If you desire to use the 64 bits version, you must edit the following files:

- `core/dotplotBreakpoint.pl`
- `core/identifyBreakpoints.pl`
- `core/segmentation.pl`

and change the line (in the beginning of the scripts):

```
my $R_CMD = "R CMD BATCH --no-restore --no-save";
```

to:

```
my $R_CMD = "R64 CMD BATCH --no-restore --no-save";
```

[2]On `Mac OS`, the `Renviron` file can be found at:
`/Library/Frameworks/R.framework/Versions/Current/Resources/etc/i386/Renviron` (for R 32 bits)
`/Library/Frameworks/R.framework/Versions/Current/Resources/etc/x86_64/Renviron` (for R 64 bits)

Chapter 2

How to use `Cassis`

2.1 The script `cassis.pl`

The package `Cassis` contains one main script called `cassis.pl`. This script coordinates the execution of the whole process of breakpoint identification and refinement. This section describes how this script must be called by the user.

The script `cassis.pl` has the following command line:

```
perl cassis.pl [options] <table> <type> <dirGR> <dirGO> <outputdir>
```

It has five mandatory parameters (< >) and a set of optional parameters.

2.1.1 Mandatory parameters

`<table>` Path for the file which contains the list of pairs of *one2one* orthologous genes or synteny blocks that can be found in the genomes G_r and G_o. For more information about the file format, see Section 2.2.

`<type>` This parameter must be set with one of these two possible values, according to the type of input table:

- `G` – Table of *one2one* orthologous genes
- `B` – Table of *one2one* orthologous synteny blocks

`<dirGR>` Path for the directory which contains FASTA files that have the complete DNA sequences of the chromosomes of the genome G_r. See Section 2.2 for more information.

`<dirGO>` Path for the directory which contains FASTA files that have the complete DNA sequences of the chromosomes of the genome G_o. See Section 2.2 for more information.

`<outputdir>` Path for the directory where the script will write all results of the process of breakpoint identification and refinement. See Section 2.3 for more information.

2.1.2 Optional parameters

`--lastzlevel N`: This optional parameter offers to the user the possibility of choosing between three different sets of configuration parameters for the alignments of the sequences S_r, S_{oA} and, S_{oB}. There are three possible values for this parameter:

- `N` = 1 – Configuration for alignment of closely related species. It is based on the parameters that were defined for the alignments between *Homo sapiens* and *Macaca mulatta*.
- `N` = 2 – Default configuration. It is based on the parameters that were defined for the alignments between *Homo sapiens* and *Mus musculus*.
- `N` = 3 – Configuration for alignment of distantly related species. It is based on the parameters that were defined for the alignments between *Homo sapiens* and *Gallus gallus*.

`--max_length_sr N`: It defines the maximum length for the sequence S_r. The value of `N` must be an integer bigger than zero. If the length of S_r is bigger than `N`, the breakpoint receives *status* -6 (see Section 2.3.1) and it is not processed by the segmentation step. The default value for this parameter is 1000000000.

`--max_length_sab N`: It defines the maximum length for the sequences S_{oA} and S_{oB}. The value of `N` must be an integer bigger than zero. If S_{oA} (or S_{oB}) has length bigger than N, its length is reduced to `N` bp. The default value for this parameter is 3000000.

`--minimum_length N`: It defines the minimum length for the sequences S_r, S_{oA} and, S_{oB}. The value of `N` must be an integer bigger than zero. If S_r, S_{oA} or, S_{oB} have lengths smaller than N, the breakpoint receives status -2, -3, -4 or, -5 (see Section 2.3.1) and it is not processed by the segmentation step. The default value for this parameter is 1 when the input table contains orthologous genes and 50000 when it contains orthologous synteny blocks.

`--extend_before [T/F]`: It determines if the breakpoint identification method must verify the minimum sequence length before (`F`) or after (`T`) applying the sequence extension (adding genes or fragments to the sequences S_r, S_{oA} and S_{oB}). The default value for this parameter is `T`.

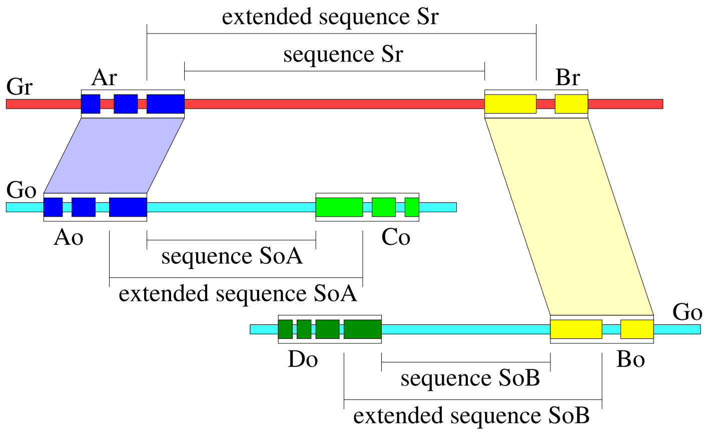

Figure 2.1: Breakpoint schema

`--extend_by_adding_gene [T/F]`: It determines if the breakpoint identification method must (`T`) or must not (`F`) add the first or last genes of the synteny blocks A_r, B_r, A_o and B_o to the sequences S_r, S_{oA} and, S_{oB} (see Figure 2.1). This parameter is valid only when the input table contains orthologous genes and its default value is `T`.

`--extend_ab_by_adding_gene [T/F]`: It determines if the breakpoint identification method must (`T`) or must not (`F`) add the first or last genes of the synteny blocks C_o and D_o

to the sequences S_{oA} and S_{oB} (see Figure 2.1). This parameter is valid only when the input table contains orthologous genes and its default value is `T`.

`--extend_by_adding_fragment N`: It determines the length of the fragments that must be added to the extremities of S_r, S_{oA} and S_{oB} in order to simulate the inclusion of the first last genes of the synteny blocks A_r, B_r, A_o and B_o. The value of `N` must be an integer bigger than or equal to zero. This parameter is valid only when the input table contains orthologous synteny blocks and its default value is 50000.

`--extend_ab_by_adding_fragment N`: It determines the length of the fragments that must be added to the extremities of S_{oA} and S_{oB} in order to simulate the inclusion of the first last genes of the synteny blocks C_o and D_o. The value of `N` must be an integer bigger than or equal to zero. This parameter is valid only when the input table contains orthologous synteny blocks and its default value is 50000.

2.1.3 Important observations

Some important observations about the input parameters:

1. The list of pairs of *one2one* orthologous genes or *one2one* orthologous synteny blocks cannot have duplicated genes or blocks.
2. When `Cassis` receives a list of pairs of *one2one* orthologous genes, it performs a step of elimination of overlapping genes and a step of identification of synteny blocks before executing the breakpoint identification step.
3. When `Cassis` receives a list of pairs of *one2one* orthologous synteny blocks, it directly executes the breakpoint identification step. Overlapping blocks are handled during the definition of the coordinates of the sequences S_r, S_{oA} and S_{oB}. If one of these sequences has length smaller than an arbitrary limit (50 kbp), the breakpoint candidate is discarded.
4. We strongly recommend the use of chromosome sequences which are previously submitted to a repeat masking process. The time required to perform the `LASTZ` alignments decreases significantly and the segmentation results are more sensitive in these conditions.

2.2 Input file formats

This section describes the format of the files which are received as input by the script `cassis.pl`.

2.2.1 Table of *one2one* orthologous genes

The table of *one2one* orthologous genes is a TSV file (Tab Separated Values file) which has information about the pairs of *one2one* orthologous genes of G_r and G_o. This file must have just rows with values (no header row is allowed) and each row must have the definition of one pair of orthologous genes which is defined by the following columns:

Name	**Definition**
$g1$	Name of the gene in the genome G_r.
$c1$	Chromosome of the genome G_r where $g1$ is located.
$inf1$	Start position of the gene $g1$ in the chromosome $c1$.
$sup1$	End position of the gene $g1$ in the chromosome $c1$.
$strand1$	Strand where the gene $g1$ can be found (1 or -1)
$g2$	Name of the gene in the genome G_o.
$c2$	Chromosome of the genome G_o where $g2$ is located.
$inf2$	Start position of the gene $g2$ in the chromosome $c2$.
$sup2$	End position of the gene $g2$ in the chromosome $c2$.
$strand2$	Strand where the gene $g2$ can be found (1 or -1)

The file must have just *one2one* orthologous genes. In the case of duplicated genes, the execution of the script is aborted. Additionally, every chromosome which appears in the column $c1$ ($c2$) must have a FASTA file in the directory `<dirGR>` (`<dirGO>`).

2.2.2 Table of *one2one* orthologous blocks

The table of *one2one* orthologous synteny blocks is a TSV file (Tab Separated Values file) which has information about the pairs of *one2one* orthologous synteny blocks that were identified in the genomes G_r and G_o. Each line of this file contains the definition of one pair of orthologous synteny blocks. This file must have just rows with values (no header row is allowed) and each row must have the following columns:

Name	**Definition**
id	Name of the pair of orthologous synteny block.
$c1$	Chromosome of the genome G_r where the block is located.
$inf1$	Start position of the block in the chromosome $c1$.
$sup1$	End position of the block in the chromosome $c1$.
$c2$	Chromosome of the genome G_o where the block is located.
$inf2$	Start position of the block in the chromosome $c2$.
$sup2$	End position of the block in the chromosome $c2$.
$strand$	If this column is set with the value 1, the synteny blocks of the genomes G_r and G_o are on the same strand. Otherwise (-1), the synteny blocks are on different strands.

The file must have just *one2one* orthologous synteny blocks. In the case of duplicated genes, the execution of the script is aborted. Additionally, every chromosome which appears in the column $c1$ ($c2$) must have a FASTA file into the directory `<dirGR>` (`<dirGO>`).

2.2.3 FASTA files

Each chromosome of the genomes G_r and G_o must have a FASTA file with just one DNA sequence. This sequence must represent the complete sequence of the chromosome. The first line of the FASTA file must have a header starting with the symbol “>” and the remaining lines must contain the DNA sequence. Example:

```
>17 dna_rm:chromosome chromosome:GRCh37:17:1:81195210:1
AAGCTTCTCACCCTGTTCCTGCATAGATAATTGCATGACAATTGCCTTGTCCCTGCTGAA
TGTGNNNNNNNNNNNNNNNNNNNNNACCCACGACCAACTCCCTGGGCCTGGCACCAGGGA
GCTTAACAAACATCNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNTCTGCCCAGTTCCTCTCCAGAAAGGCTGCATGGTTGACACACA
GTGNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNAAAATTGTGACTTTCATGGCATA
```

The FASTA files of the chromosomes of G_r (G_o) must be placed inside the directory defined by the parameter `<dirGR>` (`<dirGO>`). Additionally, the name of the FASTA file must have the following pattern:

```
[chromosomename].fasta
```

For example, if the chromosome has for name 1, the file must be named `1.fasta`.

2.3 Script output

The script `cassis.pl` writes all output results inside the directory `<outputdir>`. This section describes each one of the items which are output by it.

2.3.1 File `NonRefinedBreakpoints.txt`

This TSV file contains the results of the breakpoint identification process. Each row contains the information related to a breakpoint which was found in the genome G_r. The following columns can be found in this file:

Name	Definition
id	Breakpoint identifier.
type	Breakpoint type: `inter` or `intra`.
sRgeneA	Name of the gene (block) A_r in the genome G_r.
sRgeneB	Name of the gene (block) B_r in the genome G_r.
sRchr	Chromosome of the genome G_r where the sequence S_r is located.
sRstrandA	Strand of the gene (block) A_r in the genome G_r.
sRstrandB	Strand of the gene (block) B_r in the genome G_r.
sRinf	Start position of S_r in the chromosome *sRchr*.
sRsup	End position of S_r in the chromosome *sRchr*.
sOgeneA	Name of the gene (block) A_o in the genome G_o.
sOgeneB	Name of the gene (block) B_o in the genome G_o.
sOchrA	Chromosome of the genome G_o where the sequence S_{oA} is located.
sOchrB	Chromosome of the genome G_o where the sequence S_{oB} is located.
sOstrandA	Strand of the gene (block) A_o in the genome G_o.
sOstrandA	Strand of the gene (block) B_o in the genome G_o.
sOinfA	Start position of S_{oA} in the chromosome *sOchrA*.
sOAsupA	End position of S_{oA} in the chromosome *sOchrA*.
sOinfB	Start position of S_{oB} in the chromosome *sOchrB*.
sOsupB	End position of S_{oB} in the chromosome *sOchrB*.
bkpBegin	Relative position of the start position of the breakpoint in the sequence S_r.
bkpEnd	Relative position of the end position of the breakpoint in the sequence S_r.
status	Breakpoint status.

Breakpoint type

A breakpoint can be classified either as `inter` or as `intra`. An `inter` breakpoint occurs when the sequences S_{oA} and S_{oB}, which are orthologous to fragments on the sequence S_r, are located in different chromosomes of the genome G_o (*interchromosomic*). On the other hand, an `intra` breakpoint occurs when the sequences S_{oA} and S_{oB} are in the same chromosomes of the genome G_o (*intrachromosomic*).

Breakpoint status

The column *status* of the file `NonRefinedBreakpoints.txt` can be filled with the following values:

Value	Description
1	Valid breakpoint.
−2	The sequence S_r is smaller than the allowed limit.
−3	The sequence S_{oA} is smaller than the allowed limit.
−4	The sequence S_{oB} is smaller than the allowed limit.
−5	The sequences S_{oA} and S_{oB} are smaller than the allowed limit.
−6	The sequence S_r is bigger than the allowed limit.

When the script processes pairs of orthologous synteny blocks, it imposes a minimum length of 50 bkp for the length of the sequences S_r, S_{oA} and, S_{oB}. The script verifies this limit after extending the sequences (it adds 50 bkp on each side of the sequence). This verification is performed to eliminate breakpoints whose flanking synteny blocks overlap too much.

When the script processes pairs of orthologous genes, it only requires that the sequences S_r, S_{oA} and, S_{oB} are non empty.

The default configuration of the script determines a maximum length of 1 Gbp for the sequence S_r. It is a value big enough to avoid discarding any breakpoint. However, if an advanced user wishes filtering big sequences, he can edit the script and change this value to fit his needs.

Relative position in the sequence S_r

While defining S_r for each breakpoint, `Cassis` considers the extended version of this sequence. If we are working with orthologous genes, the extended sequence S_r includes the genes A_r and B_r which are at the boundaries of the breakpoint. In the case of synteny blocks, the extended version of S_r includes fragments with an arbitrary length of 50 kbp on each side of the breakpoint.

The columns *bkpBegin* and *bkpEnd* refer to the start and end positions of the breakpoint inside of the sequence S_r. The values of these columns are defined in relation to the value of the column $sRinf$. For example, if the columns $sRinf$, *bkpBegin* and, *bkpEnd* have, respectively, the values 100000, 50000 and, 120000, it means that the sequence S_r starts at the position 100000 and that the breakpoint starts at the position 150000 and ends at the position 220000 of the chromosome *sRchr*.

2.3.2 Directory `fasta`

The directory `fasta` contains the FASTA files of the sequences S_r, S_{oA} and S_{oB} for each valid breakpoint (*status* = 1 in the file `NonRefinedBreakpoints.txt`). Each file is named with the following name pattern:

`Breakpoint_[ID]_S[X].fasta`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A`, `B` or `R` which are related, respectively, with the sequences S_{oA}, S_{oB} and S_r.

2.3.3 Directory `alignments`

The directory `alignments` contains the results of the alignments S_r against S_{oA} and S_r against S_{oB} for each valid breakpoint (*status* = 1 in the file `NonRefinedBreakpoints.txt`). Each file is named with the following name pattern:

`Breakpoint_[ID]_S[X].lastz`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A` or `B` which are related, respectively, with the alignments S_r against S_{oA} and S_r against S_{oB}.

2.3.4 Directory `dotplot`

After performing the alignments S_r against S_{oA} and S_r against S_{oB}, the script generates, for verification purposes, *dot plot* graphs with the obtained hits. These graphs are written inside of the directory `dotplot` and have the following name pattern:

`Breakpoint_[ID].png`,

where `[ID]` is the number that identifies the breakpoint. Figure 2.2 shows an example of a dot plot graph.

2.3.5 Directory `segmentation`

The directory `segmentation` contains the result of the segmentation process for each valid breakpoint (*status* $= 1$ in the file `NonRefinedBreakpoints.txt`).

For each breakpoint, the segmentation process outputs a TSV file containing information on the refined breakpoint (it has the same format as the file `segmentation.txt` described in Section 2.3.6). If the segmentation could be calculated, the breakpoint also has an image file with the graph representation of the segmentation. These files have the following name pattern:

`Breakpoint_[ID].txt` (TSV file) and `Breakpoint_[ID].png` (image file),

where `[ID]` is the number that identifies the breakpoint. Figure 2.3 shows an example of a segmentation graph.

2.3.6 File `segmentation.txt`

The file `segmentation.txt` contains the final results of the process of breakpoint identification and refinement for all breakpoints. It is a TSV file which has the coordinates of all breakpoints before and after the segmentation. It has the following columns:

Name	**Definition**
id	Breakpoint identifier.
chr	Chromosome where the breakpoint is located on the genome G_r.
oldBegin	Start position of the breakpoint (in the sequence of the chromosome *chr*) before the segmentation process.
oldEnd	Start position of the breakpoint (in the sequence of the chromosome *chr*) before the segmentation process
oldLength	Length of the breakpoint sequence before the segmentation process
newBegin	Start position of the breakpoint (in the sequence of the chromosome *chr*) after the segmentation process.
newEnd	Start position of the breakpoint (in the sequence of the chromosome *chr*) after the segmentation process
newLength	Length of the breakpoint sequence after the segmentation process
status	Breakpoint status.

Refined breakpoint status

The column *status* of the file `segmentation.txt` can be filled with the following values:

Value	**Segmentation**	**Description**
1	YES	Segmentation passed through the statistical test.
0	YES	Segmentation did not pass through the statistical test.
-1	NO	Alignments S_r against S_{oA} and S_r against S_{oB} showed no hits.
-2	NO	Sequence S_r is smaller than the allowed limit.
-3	NO	Sequence S_{oA} is smaller than the allowed limit.
-4	NO	Sequence S_{oB} is smaller than the allowed limit.
-5	NO	Sequences S_{oA} and S_{oB} are smaller than the allowed limit.
-6	NO	Sequence S_r is bigger than the allowed limit.
-7	NO	R aborted (out of memory).

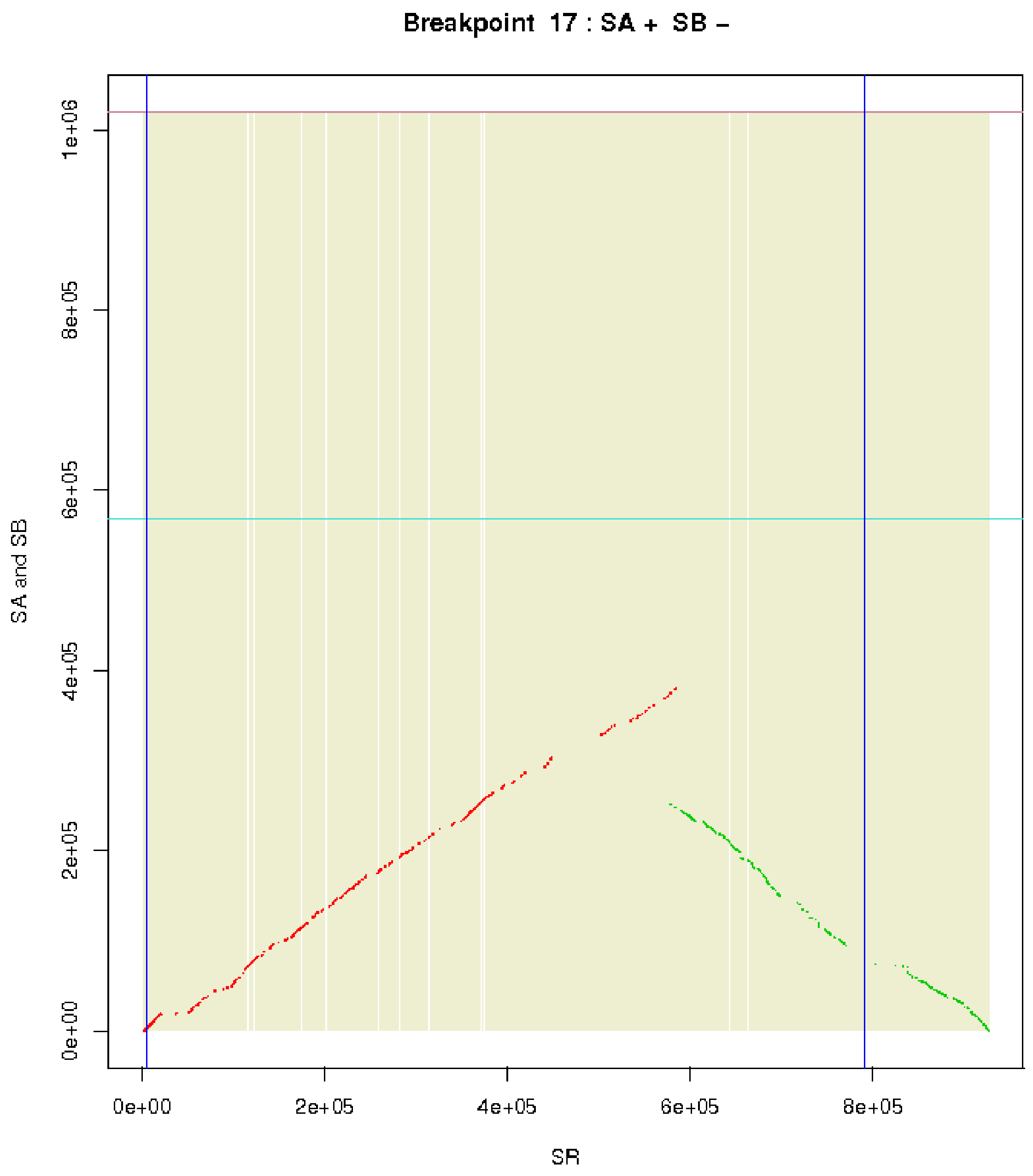

Figure 2.2: Example of a dot plot graph. The red (green) marks identify the hits of the alignment S_r against S_{oA} (S_{oB}). The vertical blue lines indicate the location of the start and end positions of the breakpoint region. The light yellow regions represent the masked regions of the sequence S_r and the horizontal pink (turquoise) line indicates the end of the sequence S_{oA} (S_{oB}).

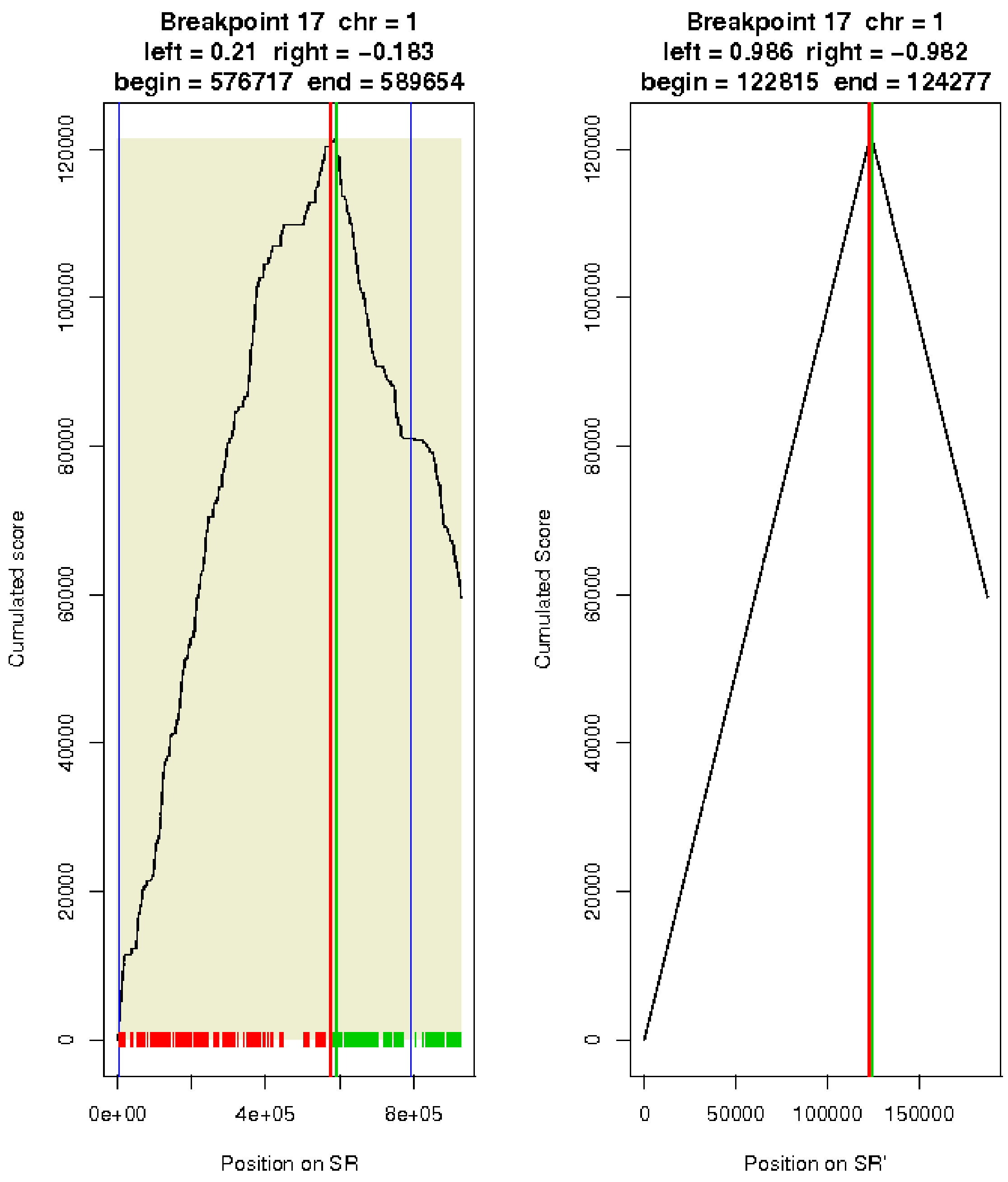

Figure 2.3: Representation of the segmentation of a breakpoint. The score of a position on sequence S_r is equal to (i) 1 if the position is covered by a hit from S_r against S_{oA} alignment and no hit from S_r against S_{oB} alignment, (ii) -1 in the reverse situation and (iii) 0 if it is covered by no hit or by hits from both alignments. The black curve represents the cumulated sum of the scores of all positions along the sequence S_r. The red (green) bars on the bottom of the graph identify the hits of the alignment S_r against S_{oA} (S_{oB}). The vertical blue lines indicate the location of the start and end positions of the non refined breakpoint. The vertical red (green) line represents the start (end) position of the refined breakpoint. The graph which appears on the left side shows the segmentation curve over the sequence S_r. The other graph exhibits the segmentation curve over the sequence S'_r which is built by removing all positions on the sequence S_r that are covered by no hit.

Chapter 3

Package organisation

3.1 How is Cassis organised?

Cassis contains one main script called cassis.pl and a set of auxiliary Perl and R scripts which are divided in two groups: core and pipeline scripts.

The objective of this organisation is to offer a code modularity to advanced users who want to extend the package or adopt parameters which are different from the ones that are offered by default in Cassis.

Every Cassis script, when called without parameters, prints a help message which lists and explains the list of parameters that it must receive for performing its work.

3.2 Core scripts

The core set is composed of scripts which were designed to perform atomic tasks: process a list of genes, align a pair of sequences, execute the segmentation of a breakpoint, etc. The following sections give a brief description on each core scripts.

3.2.1 core/alignSequences.pl

This script performs a LASTZ alignment between two DNA sequences. It receives two FASTA files and a list of LASTZ parameters, runs LASTZ and saves the result in the file specified by the parameter <outputfile>.

3.2.2 core/dotplotBreakpoint.pl

This script generates a dot plot graph for a given breakpoint. It receives the table of non refined breakpoints (see Section 2.3.1), the identifier of the desired breakpoint, the results of the alignments S_r against S_{oA} and S_r against S_{oB}, the FASTA file of S_r and outputs an image file containing the dot plot of the specified breakpoint.

3.2.3 core/extractBreakpointRegions.pl

This script generates the FASTA files for the sequences S_r, S_{oA} and S_{oB}. It receives the table of non refined breakpoints (see Section 2.3.1), the name of the chromosome that will be processed, the FASTA file that has the chromosome sequence, a flag that indicates the genome that owns the specified chromosome (1 for G_r and 2 for G_o) and the path for the directory that will receive the new FASTA files.

When the genome flag is set with the value 1 (2), the chromosome and the related FASTA file must belong to G_r (G_o) and the script will generate FASTA files of the sequences S_r (S_{oA} and S_{oB}) of the breakpoints which can be found in the specified chromosome.

3.2.4 `core/getMaskedIntervals.pl`

This script reads a FASTA file and outputs a file that has the list of masked intervals of the DNA sequence (intervals which contain only N).

3.2.5 `core/identifyBreakpoints.pl`

This script performs the task of breakpoint detection by configuring an `R` script and calling `R` to execute it. This script has many optional parameters and three mandatory parameters.

The script must receive a TSV file containing a list of pairs of orthologous genes or orthologous synteny blocks, a flag that identifies the type of the input file (G for genes and B for blocks) and the name of the output file. Note that the input TSV file must have the format described in Section 2.2.1, independently of the type of data (genes or blocks).

By setting the optional parameters, the user can configure the process of breakpoint identification with values different from the default ones. For example, the user can decide if the sequences S_r, S_{oA} and S_{oB} must be extended or not, choose the maximum length of the sequences, etc.

3.2.6 `core/segmentation.pl`

This script executes the segmentation of a single breakpoint by configuring an `R` script and calling `R` to execute it.

The script receives the table of non refined breakpoints (see Section 2.3.1), the identifier of the desired breakpoint, the results of the alignments S_r against S_{oA} and S_r against S_{oB}, the FASTA file of the sequence S_r and the names of two output files (one for the table of results and another for the image file).

3.2.7 `core/verifyFASTAdirectory.pl`

This script reads a given directory to identify all FASTA files and verifies if they are in the correct format (just one sequence per file). After reading the directory, the script will produce a TSV file that maps, for each file, the length of the sequence which it contains.

3.2.8 `core/R/breakpoints.R`

This `R` file contains functions which are developed to execute the step of breakpoint identification. It is used by the `Perl` script `core/identifyBreakpoints.pl` (Section 3.2.5).

3.2.9 `core/R/dotplot.R`

This `R` file contains functions which build a dot plot for a given breakpoint. It is used by the `Perl` script `core/dotplotBreakpoint.pl` (Section 3.2.2).

3.2.10 `core/R/overlappingGenes.R`

The method of synteny blocks definition requires a list of non overlapping pairs of *one2one* orthologous genes. This `R` file has functions that process the original table and output a new table that has no occurrences of overlapping gene.

The method looks for the order and direction of the overlapping genes in each genome. If the overlapping genes are in the same order and have the same direction in both genomes (G_r and G_o), they can be merged to represent a single gene. Otherwise, they are discarded.

This script is used by the `Perl` script `core/identifyBreakpoints.pl` (Section 3.2.5).

3.2.11 `core/R/segmentation.R`

This `R` file contains functions that perform the segmentation of a breakpoint. The functions calculate the new coordinates of the breakpoint, do a statistical test to validate the results and plot a graphical representation of the segmentation curve.

This script is used by the `Perl` script `core/segmentation.pl` (Section 3.2.6).

3.2.12 `core/R/synteny-k2-one2one.R`

Before identifying the breakpoints, `Cassis` processes the list of pairs of *one2one* orthologous genes to identify synteny blocks. This `R` file contains functions which perform this task.

This script is used by the `Perl` script `core/identifyBreakpoints.pl` (Section 3.2.5).

3.3 Pipeline scripts

The `pipeline` scripts were developed to control the processing of various sets of information: alignment of the sequences of all breakpoints, segmentation of all breakpoints, etc. They make use of the `core` scripts to execute their tasks.

3.3.1 `pipeline/alignBreakpointRegions.pl`

This script executes the alignments S_r against S_{oA} and S_r against S_{oB} for a set of breakpoints. It receives a directory where it can find the FASTA files of the breakpoint sequences, a directory where it will write the alignment results and the `LASTZ` parameters to perform the alignment.

The script expects that the FASTA files have names which respect the following name pattern:

`Breakpoint_[ID]_S[X].fasta`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A`, `B` or `R` which are related, respectively, with the sequences S_{oA}, S_{oB} and S_r. If one of the sequences S_r, S_{oA} or S_{oB} has a missing FASTA file, the script ignores the corresponding breakpoint.

The alignment results are written inside of the specified directory and have the following name pattern:

`Breakpoint_[ID]_S[X].lastz`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A` or `B` which are related, respectively, with the alignments S_r against S_{oA} and S_r against S_{oB}.

3.3.2 `pipeline/breakpointSegmentation.pl`

This script executes the segmentation of a set of breakpoints. It receives a table of non refined breakpoints (see Section 2.3.1), a directory where it can access the FASTA files of the sequences S_r, a directory where it can find the results of the alignments S_r against S_{oA} and S_r against S_{oB} and a directory to write the output results.

The script expects that the FASTA files of the sequences S_r have names which respect the following name pattern:

`Breakpoint_[ID]_SR.fasta`,

and that the `LASTZ` alignment files have the following name pattern:

`Breakpoint_[ID]_S[X].lastz`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A` or `B` which are related, respectively, with the alignments S_r against S_{oA} and S_r against S_{oB}.

The script generates, for each breakpoint, a TSV file with the segmentation results (see Section 2.3.6) and an image file which contains the graphical representation of the segmentation. The files are written on the specified output directory and have the following pattern:

`Breakpoint_[ID].txt` (TSV file) and `Breakpoint_[ID].png` (image file),

where `[ID]` is the number that identifies the breakpoint.

3.3.3 `pipeline/createBreakpointFastas.pl`

This script creates FASTA files for a set of breakpoints. It receives a table of non refined breakpoints (see Section 2.3.1), two TSV files which were produced by the script `core/verifyFASTAdirectory.pl` (see Section 3.2.7) containing information about the chromosomes of the genomes G_r and G_o and the path for an output directory.

The script writes inside the output directory the FASTA files of the sequences S_r, S_{oA} and, S_{oB} for all valid breakpoints. These files have the following name pattern:

`Breakpoint_[ID]_SR.fasta`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A`, `B` or `R` which are related, respectively, with the sequences S_{oA}, S_{oB} and S_r.

3.3.4 `pipeline/dotplotBreakpointRegions.pl`

This script produces dot plot graphs for a set of breakpoints. It receives a table of non refined breakpoints (see Section 2.3.1), a directory where it can access the FASTA files of the sequences S_r, a directory where it can find the results of the alignments S_r against S_{oA} and S_r against S_{oB} and a directory to write the output results.

The script expects that the FASTA files of the sequences S_r have names which respect the following name pattern:

`Breakpoint_[ID]_SR.fasta`,

and that the `LASTZ` alignment files have the following name pattern:

`Breakpoint_[ID]_S[X].lastz`,

where `[ID]` is the number that identifies the breakpoint and `[X]` is one of the letters `A` or `B` which are related, respectively, with the alignments S_r against S_{oA} and S_r against S_{oB}.

The script generates, for each breakpoint, an image file which contains the dot plot graph. The files are written on the specified output directory and have the following pattern:

`Breakpoint_[ID].png`,

where `[ID]` is the number that identifies the breakpoint.

3.4 Other stuffs

The package `Cassis` also contains some auxiliary or utility scripts.

3.4.1 `core/util.pm`

This is a `Perl` module which contains a set of auxiliary functions that is used by other `Perl` scripts.

3.4.2 `core/R/intervals.R`

This is a `R` script which contains a set of functions that perform operations with intervals and is used by other `R` scripts.

3.4.3 `core/R/util.R`

This is a `R` script which contains a set of auxiliary functions that is used by other `R` scripts.

3.4.4 `util/createOne2OneTable.pl`

This script receives data obtained from the Biomart (Ensembl) and creates a table of *one2one* orthologous genes.

3.4.5 `util/maskSequence.pl`

This script performs the repeat masking of a FASTA file by calling `RepeatMasker` with the appropriated parameters [3].

Bibliography

[1] C. Lemaitre. *Réarrangements chromosomiques dans les génomes de mammifères : caractérisation des points de cassure.* PhD thesis, Université Claude Bernard – Lyon 1, Novembre 2008. In French.

[2] C. Lemaitre, E. Tannier, C. Gautier, and M.-F. Sagot. Precise detection of rearrangement breakpoints in mammalian chromosomes. *BMC Bioinformatics*, 9(286), June 2008. 15 pages.

[3] A. F. A. Smit, R. Hubley, and P. Green. Repeatmasker, 2009. http://www.repeatmasker.org.