Université Libre de Bruxelles
Service de Bioinformatique des Génomes et Reséaux (BiGRe)
Laboratory of Genome and Network Biology
http://www.bigre.ulb.ac.be/

Regulatory Sequence Analysis Tools (*RSAT*)
Tutorial: command-line utilization of the tools

Jacques VAN HELDEN & the *RSAT*team

January 13, 2021

Contents

1 Introduction **9**
- 1.1 Prerequisites . . . 9
- 1.2 Creating a directory for this tutorial . . . 9
- 1.3 Warning . . . 10
- 1.4 Configuring ***RSAT*** . . . 11
  - 1.4.1 Adding ***RSAT*** to your path . . . 11

2 Getting help **14**

3 Retrieving sequences **15**
- 3.1 Retrieving a single upstream sequence . . . 15
- 3.2 Combining upstream and coding sequence . . . 16
- 3.3 Retrieving a few upstream sequences . . . 16
- 3.4 Retrieving a larger list of upstream sequences . . . 16
- 3.5 Preventing the inclusion of upstream ORFs . . . 17
- 3.6 Getting information about genes . . . 18
  - 3.6.1 Getting gene location, names and description . . . 18
  - 3.6.2 Selecting gene by name or description . . . 18
  - 3.6.3 Selecting genes by their description . . . 19
  - 3.6.4 Adding selected fields to a list of gene . . . 19
- 3.7 Retrieving sequences of a random selection of genes . . . 20
- 3.8 Retrieving all upstream sequences . . . 20
- 3.9 Retrieving downstream sequences . . . 21
- 3.10 Inferring operons . . . 21
  - 3.10.1 Inferring operon from a list of query genes . . . 22
  - 3.10.2 Selecting custom return fields . . . 22
  - 3.10.3 Operons with non-CDS genes . . . 23
  - 3.10.4 Inferring all operons for a given organism . . . 23
  - 3.10.5 Retrieving operon leader genes and inferred operon promoters . . . 23
  - 3.10.6 Collecting all upstream regions from the query gene up to the leader gene 23
  - 3.10.7 Automatic inference . . . 24
- 3.11 Purging sequences . . . 24

4 Pattern discovery **25**
- 4.1 Requirements . . . 26

5 String-based pattern discovery **27**
5.1 Requirements . . . 28
5.2 oligo-analysis . . . 28
5.2.1 Counting word occurrences and frequencies . . . 29
5.2.2 Pattern discovery in yeast upstream regions . . . 29
5.2.3 Answers . . . 31
5.2.4 Assembling the patterns . . . 32
5.2.5 Alternative background models . . . 33
5.3 Genome-scale pattern discovery . . . 36
5.3.1 Detection of over-represented words in all the yeast upstream sequences 36
5.3.2 Detection of under-represented words in bacterial genomes . . . 37
5.4 dyad-analysis . . . 37

6 String-based pattern matching **38**
6.1 dna-pattern . . . 38
6.2 Matching a single pattern . . . 38
6.3 Matching on both strands . . . 39
6.4 Allowing substitutions . . . 39
6.5 Extracting flanking sequences . . . 40
6.6 Changing the origin . . . 40
6.7 Matching degenerate patterns . . . 40
6.8 Matching regular expressions . . . 41
6.9 Matching several patterns . . . 42
6.10 Counting pattern matches . . . 42
6.11 Getting a count table . . . 43

7 Drawing graphs **44**
7.1 feature-map . . . 44
7.1.1 Converting *dna-pattern* matches into features . . . 44
7.1.2 Basic feature maps . . . 45
7.1.3 Refining the feature map . . . 45
7.1.4 Map orientation . . . 45
7.1.5 Export formats . . . 46
7.1.6 HTML maps . . . 46
7.1.7 Other options . . . 46
7.1.8 Feature converters . . . 47
7.2 XYgraph . . . 47
7.2.1 Exercise: drawing features from patser . . . 48

8 Markov models **49**
8.0.2 Transition frequency tables . . . 49
8.0.3 Oligonucleotide frequency tables . . . 49
8.0.4 Converting oligonucleotide frequencies into transition frequencies . . . 49
8.0.5 Bernoulli models . . . 50

9 Matrix-based Pattern discovery **51**
9.1 consensus (program developed by Jerry Hertz) . . . 51
9.1.1 Getting help . . . 51
9.1.2 Sequence conversion . . . 51
9.1.3 Running consensus . . . 51
9.2 Random expectation . . . 52

10 Matrix-based pattern matching **53**
10.1 Prerequisite . . . 53
10.2 patser (program developed by by Jerry Hertz) . . . 53
10.2.1 Getting help . . . 53
10.2.2 Extracting the matrix from the ***consensus*** result file . . . 53
10.2.3 Getting information about a matrix . . . 54
10.2.4 Detecting Pho4p sites in the PHO genes . . . 54
10.2.5 Detecting Pho4p sites in all upstream regions . . . 55
10.2.6 Interpretation of the P-value returned by ***patser*** . . . 55
10.2.7 Score distributions in promoter sequences . . . 58
10.3 Scanning sequences with ***matrix-scan*** . . . 58
10.3.1 Bernoulli background models . . . 59
10.3.2 Higher order (Markov) background models . . . 59
10.3.3 P-values . . . 60
10.3.4 Observed distribution of scores and site enrichment . . . 61
10.3.5 Scanning sequences with multiple matrices . . . 62
10.3.6 Detecting Cis-Regulatory element Enriched Regions (CRER) . . . 64
10.4 Computing the theoretical score distribution of a PSSM . . . 64
10.4.1 Estimating the quality of a PSSM . . . 66

11 Evaluating the quality of position-specific scoring matrices **67**
11.1 Prerequisite . . . 67
11.2 Why is important to estimate the quality of a matrix? . . . 67
11.3 How to estimate the theoretical distribution of a matrix? . . . 68
11.4 How to compare the theoretical distribution with the scores of the known binding sites? 70
11.5 Distribution in full collections of promoters . . . 71
11.6 Negative control with random sequences . . . 72
11.7 Negative controls with permuted matrices . . . 73
11.8 ROC curves indicate the trade-off between sensitivity and false positive rate . 73

12 Generating random sequences **75**
12.1 Sequences with identically and independently distributed (IID) nucleotides . . 75
12.2 Sequences with nucleotide-specific frequencies . . . 75
12.3 Markov chain-based random sequences . . . 75

13 Pattern comparisons **77**
13.1 Comparing patterns with patterns . . . 77

13.2 Comparing discovered patterns wirth a library of TF-binding consensus . . . 77

14 Comparing classes, sets and clusters 78

15 Comparative genomics 79
15.1 Genome-wise comparison of protein sequences . . . 79
15.1.1 Applying genome-blast between two genomes . . . 79
15.1.2 Applying genome-blast between a genome and a taxon . . . 81
15.2 Getting putative homologs, orthologs and paralogs . . . 81
15.2.1 Getting genes by similarities . . . 81
15.2.2 Obtaining information on the BLAST hits . . . 82
15.2.3 Selecting bidirectional best hits . . . 82
15.2.4 Selecting hits with more stringent criteria . . . 83
15.2.5 Restricting the number of reference organisms . . . 84
15.3 Retrieving sequences for multiple organisms . . . 84
15.4 Detection of phylogenetic footprints . . . 85
15.5 Phylogenetic profiles . . . 85
15.6 Detecting pairs of genes with similar phylogenetic profiles . . . 86
15.6.1 Comparing binary profiles with ***compare-profiles*** . . . 86
15.6.2 Comparing binary profiles with ***compare-classes*** . . . 87

16 Automated analysis of multiple gene clusters 88
16.1 Input format . . . 89
16.2 Example of utilization . . . 89
16.3 Loading the results in a relational database . . . 91
16.4 Comparing programs . . . 91
16.5 The negative control: analyzing random gene selections . . . 92
16.6 Analyzing a large set of regulons . . . 92

17 Utilities 93
17.1 gene-info . . . 93
17.2 On-the-fly compression/uncompression . . . 93

18 Downloading genomes 95
18.1 Original data sources . . . 95
18.2 Requirement : wget . . . 95
18.3 Importing organisms from the ***RSAT*** main server . . . 96
18.3.1 Obtaining the list of organisms supported on the ***RSAT*** server . . . 96
18.3.2 Importing a single organism . . . 96
18.3.3 Importing a few selected organisms . . . 97
18.3.4 Importing all the organisms from a given taxon . . . 97

19 Installing additional genomes on your machine 98
19.1 Adding support for Ensembl genomes . . . 98
19.1.1 Handling genomes from Ensembl . . . 98

19.2 Installing genomes and variations from ***EnsEMBL*** . . . 100
19.2.1 Installing genomes from Ensembl . . . 100
19.2.2 Installing genomes from EnsemblGenomes . . . 101
19.2.3 Downloading variations . . . 102
19.3 Importing genomes from NCBI BioProject . . . 103
19.4 Importing multi-genome alignment files from UCSC . . . 104
19.4.1 Warning: disk space requirement . . . 104
19.4.2 Checking supported genomes at UCSC . . . 104
19.4.3 Downloading multiz files from UCSC . . . 104
19.5 Installing genomes from NCBI/Genbank files . . . 104
19.5.1 Organization of the genome files . . . 105
19.5.2 Downloading genomes from NCBI/Genbank . . . 107
19.5.3 Parsing a genome from NCBI/Genbank . . . 108
19.5.4 Parsing a genome from the Broad institute (MIT) . . . 108
19.5.5 Updating the configuration file . . . 109
19.5.6 Checking the start and stop codon composition . . . 109
19.5.7 Calibrating oligonucleotide and dyad frequencies with ***install-organisms*** 110
19.5.8 Installing a genome in your own account . . . 110
19.6 Installing genomes from EMBL files . . . 111

20 Regulatory variations (rSNPs and insertion/deletions) 113
20.1 Requirements . . . 113
20.2 Getting variation sequences . . . 113
20.2.1 Checking the installation of variations on your ***RSAT*** server . . . 113
20.3 File formats for variations . . . 114
20.4 Detecting regulatory variations . . . 115
20.4.1 Scanning a selected variation with selected matrices . . . 115
20.4.2 Obtaining a list of disease-associated variation IDs . . . 115
20.4.3 Scanning a list of selected variations with a list of matrices . . . 115
20.4.4 Scanning all variations with a selected matrix . . . 115

21 Exercises 116
21.1 Some hints . . . 116
21.1.1 Sequence retrieval . . . 116
21.1.2 Detection of over-represented motifs . . . 116

22 Using RSAT Web Services 117
22.1 Introduction . . . 117
22.2 Examples of WS clients in Perl with SOAP::WSDL 2.00 (or above) . . . 117
22.2.1 Requirements . . . 117
22.2.2 Retrieving sequences from RSATWS . . . 117
22.3 Examples of WS clients in Perl with SOAP::WSDL 1.27 (or below) . . . 120
22.3.1 Requirements . . . 120
22.3.2 Getting gene-info from RSATWS . . . 120

22.3.3 Documentation . . . 122
22.3.4 Retrieving sequences from RSATWS . . . 122
22.3.5 Work flow using RSATWS . . . 124
22.3.6 Discover patterns with RSATWS . . . 127
22.3.7 Example of clients using property files . . . 129
22.3.8 Other tools in RSATWS . . . 131
22.4 Examples of WS client in java . . . 131
22.4.1 Same workflow as above with RSATWS . . . 131
22.5 Examples of WS client in python . . . 133
22.5.1 Get infos on genes having methionine or purine in their description, as above in perl 133
22.6 Full documentation of the RSATWS interface . . . 134

23 Graph analysis **135**
23.1 Introduction . . . 135
23.1.1 Definition . . . 135
23.1.2 Some types of graphs . . . 135
23.1.3 Graph files formats . . . 136
23.2 RSAT Graph tools . . . 136
23.2.1 ***convert-graph*** . . . 136
23.2.2 ***graph-node-degree*** . . . 137
23.2.3 ***graph-neighbours*** . . . 137
23.2.4 ***compare-graphs*** . . . 137
23.2.5 ***graph-get-clusters*** . . . 137
23.2.6 ***compare-graph-clusters*** . . . 138

24 Pathway extraction tools **139**
24.1 Using pathway extraction tools . . . 139
24.1.1 Listing tools and getting help . . . 139
24.1.2 Abbreviating tool names . . . 139
24.1.3 Increasing JVM memory . . . 139
24.2 Obtaining metabolic networks . . . 139
24.2.1 Downloading MetaCyc and KEGG generic metabolic networks from the NeAT web server 1
24.2.2 Building KEGG generic metabolic networks . . . 140
24.2.3 Building KEGG organism-specific metabolic networks . . . 140
24.2.4 Building metabolic networks from biopax files . . . 141
24.3 Finding k-shortest paths . . . 141
24.4 Linking genes to reactions . . . 143
24.4.1 Prerequisites . . . 143
24.4.2 Linking genes of the isoleucine-valine operon to reactions . . . 143
24.5 Predicting metabolic pathways . . . 143
24.5.1 Predicting a metabolic pathway for the isoleucine-valine operon . . . 144
24.5.2 Mapping reference pathways onto the predicted pathway . . . 144
24.5.3 Annotating the predicted pathway . . . 145
24.5.4 Visualizing the predicted pathway . . . 145

25 References 146

1 Introduction

This tutorial aims at introducing how to use Regulatory Sequence Analysis Tools (***RSAT***) directly from the Unix shell.

RSATis a package combining a series of specialized programs for the detection of regulatory signals in non-coding sequences. A variety of tasks can be performed: retrieval of upstream or downstream sequences, pattern discovery, pattern matching, graphical representation of regulatory regions, sequence conversions, ...

A web interface has been developed for the most common tools, and is freely available for academic users.

http://rsat.ulb.ac.be/rsat/

All the programs in ***RSAT***can also be used directly from the Unix shell. The shell access is less intuitive than the web interface, but it allows to perform more complex analyses, and it is very convenient for automatizing repetitive tasks.

This tutorial was written by Jacques van Helden (*Jacques.van.Helden@ulb.ac.be*). Unless otherwise specified, the programs presented here were written by Jacques van Helden.

1.1 Prerequisites

This program requires a basic knowledge of the Unix environment. Before starting you should be familiar with the concepts of Unix shell, directory, file, path.

1.2 Creating a directory for this tutorial

During this tutorial, we wll frequently save data and result files. I propose to create a dedicated directory for these files. In the following chapters, we will assume that this directory is named *practical_rsat* and is located at the root of your personnal account (everyone is of course allowed to change the name and location of this directory).

To create the directory for the tutorials, you can simply type the following commands.

```
cd $HOME ## Go to your home directory
mkdir -p practical_rsat ## Create the directory for the tutorial
cd  practical_rsat ## Go to this directory
pwd ## Check the path of your directory
```

From now on, we will assume that all the exercises are executed from this directory.

1.3 Warning

This tutorial is under construction. Some sections are still to be written, and only appear as a title without any further text. The tutorial will be progressively completed. We provided it as it is.

1.4 Configuring *RSAT*

In order to use the command-line version of ***RSAT***, you first need an account on a Unix machine where ***RSAT***has been installed, and you should know the directory where the tools have been installed (if you don't know, ask assistance to your system administrator).

In the following instruction, we will assume that ***RSAT***is installed in the directory `/home/fred/rsat`.

Note: in all subsequent instructions, this path has to be replaced by the actual path where ***RSAT***has been installed on your computer.

1.4.1 Adding *RSAT*to your path

Before starting to use the tools, you need to define an environment variable (`RSAT`), and to add some directories to your path.

For the following instructions, we will denote as

RSAT_PARENT_PATH

the path of the parent folder in which the ***RSAT***suite has been downloaded.

For example, if the *rsat* folder has been installed in your home directory (it is thus found at */home/fred/rsat*), you should replace

RSAT_PARENT_PATH

by */home/fred* in all the following instructions.

1. Identify your `SHELL`.

   The way to execute the following instructions depends on your "shell" environment. If you don't know which is your default shell, type

   ```
   echo $SHELL
   ```

   The answer should be something like

   ```
   /sbin/bash
   ```

   or

   ```
   /bin/tcsh
   ```

2. Declare the ***RSAT***environment variables.

   We will define configuration parameters necessary for ***RSAT***. This includes an environment variable named `RSAT`. We will then add the path of the ***RSAT*** perl scripts, python scripts and binaries to your path.

   - *If your shell is bash*, you should copy the following lines in a file named *.bash_profile* at the root of your account (depending on the Unix distribution, the bash custom

parameters may be declered in a file named *.bash_profile* or) *.bashrc*, in case of doubt ask your system administrator).

If you dispose of admin rights, you may decide to store this configuration ina file that will be laoded for all users of the server (e.g. */etc/profile.d/rsat.sh* or */etc/bashrc*, depending on the operating system).

Note: replace [PARENT_PATH] by the full path of the directory in which the rsat folder has been created.

```
#################################################################
## Configuration for Regulatory Sequence Analysis Tools (RSAT)
export RSAT=[PARENT_PATH]/rsat
export PATH=${RSAT}/bin:${PATH}
export PATH=${RSAT}/perl-scripts:${PATH}
export PATH=${RSAT}/python-scripts:${PATH}
if  [ ${CLASSPATH} ]; then
       export CLASSPATH=${CLASSPATH}:${RSAT}/java/lib/NeAT_javatools.jar
else
       export CLASSPATH=.:${RSAT}/java/lib/NeAT_javatools.jar
fi

## Pass through ssh for CVS (required to install Ensembl API)
export CVS_RSH=ssh
```

- *If your default shell is* ***tcsh*** *or* ***csh***, copy the following lines in a file named *.chsrc* that should be found at root of your account.

  Note: replace [PARENT_PATH] by the full path of the directory in which the rsat folder has been created.

```
#################################################################
## Configuration for Regulatory Sequence Analysis Tools (RSAT)
setenv RSAT [PARENT_PATH]/rsat
set path=($RSAT/bin $path)
set path=($RSAT/perl-scripts $path)
set path=($RSAT/python-scripts $path)
if  [ ${CLASSPATH} ]; then
       setenv CLASSPATH ${CLASSPATH}:${RSAT}/java/lib/NeAT_javatools.jar
else
       setenv CLASSPATH .:${RSAT}/java/lib/NeAT_javatools.jar
fi

## Pass through ssh for CVS (required to install Ensembl API)
setenv CVS_RSH=ssh
```

- If you are using a different shell than bash, csh or tcsh, the specification of environment variables might differ from the syntax above. In case of doubt, ask your system administrator how to configure your environment variables and your path.

3. In order for the variables to be taken in consideration, you need to log out and open a new terminal session. To check that the variables are correctly defined, type.

```
echo $RSAT
```

In the example above, this command should return

```
/home/fred/rsat
```

2 Getting help

The first step before using any program is to read the manual. All programs in the ***RSAT*** package come with an on-line help, which is obtained by typing the name of the program followed by the option *-h*. For example, to get a detailed description of the functionality and options for the program `retrieve-seq`, type

```
retrieve-seq -h
```

The detailed help is specially convenient before using the program for the first time. A complementary functionality is offered by the option `-help`, which prints a short list of options. Try:

```
retrieve-seq -help
```

which is convenient to remind the precise formulation of arguments for a given program.

3 Retrieving sequences

The program ***retrieve-seq*** allows you to retrieve sequences from a genome (provided this genome is supported on your machine). In particular (and by default), this program extracts the non-coding sequences located upstream the start codon of the query genes. The reason for selecting upstream sequences (rather than coding) is that regulatory elements are generally found upstream of the coding regions, at least in microbial organisms.

3.1 Retrieving a single upstream sequence

First trial: we will extract the upstream sequence for a single gene. Try:

```
retrieve-seq -type upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -q metA -from -200 -to -1
```

This command retrieves a 200 bp upstream sequence for the gene *metA* of the bacteria *Escherichia coli K12*.

By default, coordinates are calculated from the start codon. Ideally, we would prefer to retrieve sequences upstream of the Transcription Start Site (*TSS*), since this is the place where the RNA polymerase starts to transcribe the gene. Unfortunately, the precise location of the TSS is unknown for most genes, in most sequecned genome. For this reason, the default reference is the start codon rather than the TSS.

Note that for some organisms (e.g. *Homo sapiens*), genome annotations include mRNA boundaries. In this case, the option *-feattype mRNA* allows you to specify that the reference point is the start of the mRNA (thus the TSS) rather than the start codon.

Whichever reference point you decide to use, negative coordinates indicate sequences upstream to this reference point, and positive coordinates downstream sequences.

With the default parameters,

- the reference point is the start codon;
- position -1 corresponds to the first residue upstream of the coding sequence;
- position 0 is the first letter from the start codon (the A from ATG);
- positive coordinates indicate the coding sequence (downstream from the start codon).

To better understand the system of coordinates, try to locate the start codon in the sequence obtained with the following commands.

```
retrieve-seq -type upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -q metA -from -5 -to 6
```

3.2 Combining upstream and coding sequence

For *E.coli* genes, regulatory signals sometimes overlap the 5' side of the coding sequence. By doing so, they exert a repression effect by preventing RNA-polymerase from binding DNA. The command ***retrieve-seq*** allows you to extract a sequence that overlaps the start codon, to combine an upstream and a coding segment.

```
retrieve-seq -type upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -q metA -from -200 -to 49
```

3.3 Retrieving a few upstream sequences

The option *-q* (query gene) can be used iteratively in a command to retrieve sequences for several genes.

```
retrieve-seq -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -from -200 -to 49 -q metA -q metB -q metC
```

3.4 Retrieving a larger list of upstream sequences

If you have to retrieve a large number of sequences, it might become cumbersome to type each gene name on the command-line. A list of gene names can be provided in a text file, each gene name coming as the first word of a new line.

As an example, we willuse the command

`gene-info`

to collect all genes whose matches the prefix `PHO` followed by one or several numbers (this will return a list of genes involved in phosphate metabolism).

```
gene-info -org Saccharomyces_cerevisiae -q 'PHO\d+' -o PHO_genes.txt
```

We can check the content of your file by typing

```
cat PHO_genes.txt
```

This file can now be used as input to indicate the list of query genes for

`retrieve-seq`

, with the option *-i*.

```
retrieve-seq -type upstream -i PHO_genes.txt \
    -org Saccharomyces_cerevisiae \
    -from -800 -to -1
```

The option *-o* allows you to indicate the name of a file where the sequence will be stored.

```
retrieve-seq -type upstream -i PHO_genes.txt \
    -org Saccharomyces_cerevisiae \
    -from -800 -to -1   -label name \
    -o PHO_up800.fasta
```

Check the sequence file:

```
more PHO_up800.fasta
```

3.5 Preventing the inclusion of upstream ORFs

With the command above, we retrieved sequences covering precisely 200 bp upstream the start codon of the selected genes. Intergenic regions are sometimes shorter than this size. In particular, in bacteria, many genes are organized in operons, and the intergenic distance is very short (typically between 0 and 50 bp). If your gene selection contains many intra-operon genes, the sequences will be mainly composed of coding sequences (more precisely ORF, open reading frame), which will bias subsequent analyses.

The option *-noorf* of *retrieve-seq* indicates that, if the upstream gene is closer than the specified limit, the sequence should be clipped in order to return only intergenic regions.

As an example, we will store the list of histidin genes in a file and compare the results obtained with and without the option *-noorf*.

Create a text file named *his_genes.txt* with the following genes.

```
hisL
hisG
hisD
hisC
hisH
hisA
hisF
hisI
hisP
hisM
hisQ
hisJ
hisS
```

The default behaviour will return 200bp for each gene.

```
retrieve-seq -type upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -i his_genes.txt -from -200 -to -1
```

With the option *-noorf*, sequences are clipped depending on the position of the closest upstream neighbour.

```
retrieve-seq -type upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -i his_genes.txt -from -200 -to -1 -noorf \
    -o his_up200_noorf.fasta

more his_up200_noorf.fasta
```

You can measure the length of the resulting sequences with the program ***sequence-lengths***.

```
sequence-lengths -i his_up200_noorf.fasta
```

Notice that some genes have very short upstream sequences (no more than a few bp, or even 0bp). These are the internal genes of the *his* operon.

We will now apply the same option to the list of PHO genes entered above, in order to obtaine the corresponding non-coding upstream sequences, with a size up to 800bp.

```
retrieve-seq -type upstream -i PHO_genes.txt \
    -org Saccharomyces_cerevisiae \
    -from -800 -to -1 -noorf  -label name \
    -o PHO_up800-noorf.fasta
```

Check the sequence file:

```
more PHO_up800-noorf.fasta
```

We can now use the command

`sequence-lengths`

to compare the sequence sizes of the files *PHO_up800.fasta*, and *PHO_up800-noorf.fasta*, respectively.

```
sequence-lengths -i PHO_up800.fasta

sequence-lengths -i PHO_up800-noorf.fasta
```

3.6 Getting information about genes

RSATinclude several utilities to obtain information about a set of genes, we will illustrate some basic features.

3.6.1 Getting gene location, names and description

In the previous section, we created a text file with the names of a set of genes related to phosphate metabolism. The command

`gene-info`

returns the complete information concerning a set of genes. By default, the first word of each row of the input file is considered as a query.

```
gene-info -i PHO_genes.txt -org Saccharomyces_cerevisiae
```

3.6.2 Selecting gene by name or description

Another common need is to search all the names whose name or description matches some string. For example, let us assume that we want to ollect all the genes whose name indicates a role in the methionine metabolism, in the yeast *Saccharomyces cerevisiae*. The program *gene-info* allows us to specify this type of query. according to the naming convention in the yeast community, gene names start with three letters indicating the function (e.g. PHO for

phosphate, MET for methionine), wollowed by a number. We can ask the program to return all the gene names having the string "MET" in their names.

In this example, we will enter the query string with the option *-q* on the command line, rather than in a file.

```
gene-info -q 'MET' -org Saccharomyces_cerevisiae
```

We could also refine the query by taking advantage of our knowledge of the yeast gene nomenclature, and selecting the genes whose name starts with the prefix "MET", followed by one or several numbers.

```
gene-info -q '^MET\d+' -org Saccharomyces_cerevisiae
```

The query is formulatd as a *regular expression*, where \d indicates a number, and the symbol + is a multiplier, so \d+, indicates that we accept a succession of one or more numbers after the string "MET". The character ^ indicates that the string MET should be at the start of the name (thus, there can be no letter before MET).

We can now store this list of genes in a separate file, and retrieve the coresponding upstream sequences.

```
gene-info -q '^MET\d+' -org Saccharomyces_cerevisiae -o MET_genes.txt

retrieve-seq -type upstream -i MET_genes.txt \
    -org Saccharomyces_cerevisiae \
    -from -800 -to -1 -noorf  -label name \
    -o MET_up800-noorf.fasta
```

3.6.3 Selecting genes by their description

By default, the program *gene-info* matches a query string against the list of gene names for the selected organism. The option *-descr* extends the search to the gene descriptions. For instance, we could search all the genes having the word "methionine" in their description.

```
gene-info -descr -q methionine -org Saccharomyces_cerevisiae
```

3.6.4 Adding selected fields to a list of gene

As we saw in the previous section, the program ***gene-info*** takes as input a list of gene names or identifiers, and return the complete description of each gene.

In some cases, one needs only a part of this information (e.g. the common name, or the descripion), in order to to add some columns to a pre-existing tab-delimited file where each row represents one gene. For example, imagine that you have a file containing expression profiles for 6,000 yeast genes, measured by microarray experiments under 200 conditions. The file contains 201 columns: the first column indicates the ID of each gene, and the 200 next column give expression values measured in the 200 microarrays. In such case, you would typically use ***add-gene-info*** to add a few columns after each profile, in order to indicate the common name and the description of each gene.

The program *add-gene-info* allows add columns to an input file, with user-selected fields of information about the genes. For example, the options below will add the gene identifier and the list of synonym to each row of our PHO gene list.

```
add-gene-info -i PHO_genes.txt -org Saccharomyces_cerevisiae \
  -info id,names
```

If the input file contains additional columns (e.g. expression profiles), these will be preserved in the output, and the requested information columns will be added at the end of each row.

You can check the list of fields supported by ***add-gene-info*** by consulting the help message.

```
add-gene-info -help
```

3.7 Retrieving sequences of a random selection of genes

It is also sometimes interesting to select a set of random genes, which canbe used as negative control or some analyses. This is exactly the purporse of the program ***random-genes***. We will perform a random selection of 20 yeast genes, and retrieve their upstream sequences. This selection will also be used in the next chapters.

```
random-genes -org Saccharomyces_cerevisiae -n 20 -o RAND_genes.txt

retrieve-seq -type upstream -i RAND_genes.txt \
    -org Saccharomyces_cerevisiae \
    -from -800 -to -1 -noorf  -label name \
    -o RAND_up800-noorf.fasta
```

3.8 Retrieving all upstream sequences

For genome-scale analyses, it is convenient to retrieve upstream sequences for all the genes of a given genome, without having to specify the complete list of names. For this, simply use the option *-all*.

As an illustration, we will use

```
retrieve-seq
```

to retrieve all the start codons from *Escherichia coli*. As we saw before, negative coordinates specify upstream positions, 0 being the first base of the coding sequence. Thus, by specifying positions 0 to 2, we will extract the three first coding bases, i.e. the start codon.

```
retrieve-seq -type upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -from 0 -to 2 \
    -all -format wc -nocomments -label id,name \
    -o Escherichia_coli_K_12_substr__MG1655_uid57779_start_codons.wc
```

Check the result:

```
more Escherichia_coli_K_12_substr__MG1655_uid57779_start_codons.wc
```

3.9 Retrieving downstream sequences

retrieve-seq can also be used to retrieve downstream sequences. In this case, the origin (position 0) is the third base of the stop codon, positive coordinates indicate downstream (3') location, and negative coordinates locations upstream (5') from the stop codon (i.e. coding sequences).

For example, the following command will retrieve 200pb downstream sequences for a few yeast genes. The first nucleotides of the retrieved sequences are those immediately after the stop codon.

```
retrieve-seq -type downstream -org Saccharomyces_cerevisiae \
    -from 1 -to 200 -label id,name -q PHO5 -q MET4
```

Since with the option *-type downstream*, the coordinates smaller than 1 indicate positions upstream of the stop codon, we can use ***retrieve-seq*** to extract the stop codons for all the genes of *Escherichia coli*.

```
retrieve-seq -type downstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -from -2 -to 0 \
    -all -format wc -nocomments -label id,name \
    -o Escherichia_coli_K_12_substr__MG1655_uid57779_stop_codons.wc
```

3.10 Inferring operons

In Bacteria, genes are organized in operon, which means that several genes are transcribed in a single transcription unit. The transcription of a whole operon is driven by a single promoter, located upstream of the so-called *leader gene*.

Let us assume that we dispose of a set of bacterial genes for which we want to predict cis-acting elements (e.g. co-expressed genes in a microarray experiment). A good fraction of these genes might be located inside operons. For these, the putative regulatory elements should be searched in the promoter of the operon leader gene, rather than in the upstream sequence of the gene itself.

The program ***infer-operons*** allows to infer the operons and return the corresponding leader genes for a set of input genes. The approach is inspired by the Salgado-Hagelsieb method, which consists in predicting, for each upstream region, if it is within an operon (WO) or t a transcription unit border (TUB). This prediction is based on two rules:

1. **Orientation rule** If the intergenic region is flanked by two genes located on different strands, it is a TUB.

2. **[Distance rule** If the intergenic region is flanked two *tandem* genes (adjacent genes transcribed in the same direction), it is classified as WO if the intergenic distance is lower than some threshold (by default, 55bp), and as TUB otherwise.

The default distance threshold was chosen to obtain a good balance between *sensitivity* (*Sn*, fraction of annotated WO regions which are correctly predicted) and *positive predictive value* (*PPV*, fraction of predicted WO region which indeed correspond to annotations).

The option *-dist* allows to specify a custom distance threshold. By increasing the threshold, the number of regions predicted as WO increases, at the expense of those predicted as WO. This will thus increase the Sn and decrease PPV.

The *accuracy* measures the balance between Sn and PPV by taking their arithmetic average. With the default value, one can expect 78% of accuracy (Reki's janky and Jacques van Helden, unpublished results).

We will illustrate the use of ***infer-operons*** with a few examples.

3.10.1 Inferring operon from a list of query genes

With the following command, we infer the operon for a set of input genes.

```
infer-operons -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 -q hisD -q mhpR \
  -q mhpA -q mhpD
```

Note that the prediction is incorrect for the gene *hisD*: the program predict *hisG* as operon leader, whereas the well known leader of the *his* operon is *hisL*. This is due to the fact that the intergenic distance between *hisL* and *hisG* is 145bp, which exceeds the default distance threshold (55bp).

One option would be to increase the distance threshold to 150bp.

```
infer-operons -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 -q hisD -q mhpR \
  -q mhpA -q mhpD -dist 150
```

However, we should be very careful with this option, since it has a strong consequence on all the other operon inferenes in the same genome. Since a good fraction of promoters of *Escherichia coli* are shorted than 150bp, by increasing the distance threshold to 150, we will undully consider these promoters as WO.

3.10.2 Selecting custom return fields

The option *-return* allows to specify custom return fields.

```
infer-operons -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 -q hisD -q lacI -
  -return q_info,up_info,leader,trailer,operon
```

Note that the famous *lac* operon contains three genes: *lacZ*, *lacY* and *lacA*, but the inferred operon only returns the two first genes because the distance between *lacY* and *lacA* is 65bp. This can be checked with the return field *down_info*.

```
infer-operons -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 -q lacZ -q lacY \
  -return q_info,up_info,down_info,operon
```

3.10.3 Operons with non-CDS genes

Note that operons can contain non-coding genes. For example, the metT operon contains a series of tRNA genes for methionine, leucine and glutamina, respectively.

```
infer-operons -org Escherichia_coli_K_12_substr__MG1655_uid57779 -q glnV -q metU -q ile
-return q_info,up_info,operon
```

3.10.4 Inferring all operons for a given organism

The option *-all* allows to infer operons for all the genes of an organism.

```
infer-operons -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 -all \
  -return q_info,up_info,leader,operon
```

3.10.5 Retrieving operon leader genes and inferred operon promoters

As explained above, a common usage of operon inference is to predict a list of leader genes from a set of query genes, in order to retrieve the corresponding promoter sequences. For this, we will use the option *-return* to obtain the leader gene in the first column of the result table.

```
infer-operons -org Escherichia_coli_K_12_substr__MG1655_uid57779 -return leader,q_info,
   -q lacI -q lacZ -q lacY -q mhpD -q mhpF
```

The first column now indicates the inferred leader genes rather than tne query genes, and that this column contains some redundancy: the same leader gene appears multiple times. This comes from the fact that several of our query genes were part of the same operon (e.g.:lacZ and lacY).

To avoid including twice their leader, we use the unix command sort -u (unique).

```
infer-operons -org Escherichia_coli_K_12_substr__MG1655_uid57779 -return leader,q_info,
   -q lacI -q lacZ -q lacY -q mhpD -q mhpF \
   | cut -f 1 \
   | sort -u
```

We can now use the resulting non-redundant list of operon leaders as input for retrieve-seq.

```
infer-operons -org Escherichia_coli_K_12_substr__MG1655_uid57779 -return leader,q_info,
   -q lacI -q lacZ -q lacY -q mhpD -q mhpF \
   | cut -f 1 \
   | sort -u \
   | retrieve-seq -org Escherichia_coli_K_12_substr__MG1655_uid57779 -noorf
```

3.10.6 Collecting all upstream regions from the query gene up to the leader gene

TO BE IMPLEMENTED

3.10.7 Automatic inference

TO BE IMPLEMENTED

3.11 Purging sequences

TO BE WRITTEN

4 Pattern discovery

In a pattern discovery problem, you start from a set of functionally related sequences (e.g. upstream sequences for a set of co-regulated genes) and you try to extract motifs (e.g. regulatory elements) that are characteristic of these sequences.

Several approaches exist, either string-based or matrix-based. *String-based pattern discovery* is based on an analysis of the number of occurrences of all possibles words (***oligo-analysis***), or spaced pairs (***dyad-analysis***). The methods for *matrix-based pattern discovery* rely on the utlisation of some machine-learning method (e.g. greedy algorithm, expectation-maxiisation, gibbs sampling, ...) in order to optimise of some scoring function (log-likelihood, information,...) which is likely to return significant motifs.

In this chapter we will mainly focus on string-based approaches, and illustrate some of their advantages. A further chapter will be dedicated to matrix-based pattern discovery.

For microbial cis-acting elements, string-based approaches give excellent results. The main advantages of these methods:

+ Simple to use
+ Deterministic (if you run it repeatedly, you always get the same result), in contrast with stochastic optimization methods.
+ Exhaustive : each word or space pair is tested independently. Consequently, if a set of sequences contains several exceptional motifs, all of them can be detected in a single run.
+ The tests of significances can be performed on both tails of the theoretical distribution, in order to detect either over-represented, or under-represented patterns.
+ Fast.
+ Able to return a negative answer: if no motif is significant, the programs return no motif at all. This is particularly important to reduce the rate of false positive.

An obvious advantage of matrix-based approach is that they provide a more refined description of motifs presenting a high degree of degeneracy. However, a general problem of matrix-based approaches is that it is impossible to analyze all possible position-weight matrices, and thus one has to use heuristics. There is thus a risk to miss the global optimum because the program is attracted to local maxima. Another problem is that there are more parameters to select (typically, matrix width and expected number of occurrences of the motif), and their choice drastically affects the quality of the result.

Basically, I would tend to prefer string-based approaches for any problem of pattern discovery. On the contrary, matrix-based approaches are much more sensitive for pattern matching problems (see below). My preference is thus to combine string-based pattern discovery and matrix-based pattern matching.

But I am obviously biased because I developed string-based approaches. An important factor in the success obtained with a program is to understand precisely its functioning. I thus think that each user should test different programs, compare them and select the one that best suits his/her needs.

4.1 Requirements

This part of the tutorial assumes that you already performed the tutorial about sequence retrieval (above), and that you have the result files in the current directory. Check with the command:

```
cd ${HOME}/practical_rsat
ls -1
```

You should see the following file list:

```
Escherichia_coli_K_12_substr__MG1655_uid57779_start_codons.wc
Escherichia_coli_K_12_substr__MG1655_uid57779_stop_codons.wc
MET_genes.txt
MET_up800-noorf.fasta
PHO_genes.txt
PHO_up800-noorf.fasta
PHO_up800.fasta
RAND_genes.txt
RAND_up800-noorf.fasta
his.genes.txt
his.up200.noorf.fasta
```

5 String-based pattern discovery

In a pattern discovery problem, you start from a set of functionally related sequences (e.g. upstream sequences for a set of co-regulated genes) and you try to extract motifs (e.g. regulatory elements) that are characteristic of these sequences.

Several approaches exist, either string-based or matrix-based. *String-based pattern discovery* is based on an analysis of the number of occurrences of all possibles words (***oligo-analysis***), or spaced pairs (***dyad-analysis***). The methods for *matrix-based pattern discovery* rely on the utlisation of some machine-learning method (e.g. greedy algorithm, expectation-maxiisation, gibbs sampling, ...) in order to optimise of some scoring function (log-likelihood, information,...) which is likely to return significant motifs.

In this chapter we will mainly focus on string-based approaches, and illustrate some of their advantages. A further chapter will be dedicated to matrix-based pattern discovery.

For microbial cis-acting elements, string-based approaches give excellent results. The main advantages of these methods:

+ Simple to use
+ Deterministic (if you run it repeatedly, you always get the same result), in contrast with stochastic optimization methods.
+ Exhaustive : each word or space pair is tested independently. Consequently, if a set of sequences contains several exceptional motifs, all of them can be detected in a single run.
+ The tests of significances can be performed on both tails of the theoretical distribution, in order to detect either over-represented, or under-represented patterns.
+ Fast.
+ Able to return a negative answer: if no motif is significant, the programs return no motif at all. This is particularly important to reduce the rate of false positive.

An obvious advantage of matrix-based approach is that they provide a more refined description of motifs presenting a high degree of degeneracy. However, a general problem of matrix-based approaches is that it is impossible to analyze all possible position-weight matrices, and thus one has to use heuristics. There is thus a risk to miss the global optimum because the program is attracted to local maxima. Another problem is that there are more parameters to select (typically, matrix width and expected number of occurrences of the motif), and their choice drastically affects the quality of the result.

Basically, I would tend to prefer string-based approaches for any problem of pattern discovery. On the contrary, matrix-based approaches are much more sensitive for pattern matching problems (see below). My preference is thus to combine string-based pattern discovery and matrix-based pattern matching.

But I am obviously biased because I developed string-based approaches. An important factor in the success obtained with a program is to understand precisely its functioning. I thus think that each user should test different programs, compare them and select the one that best suits his/her needs.

5.1 Requirements

This part of the tutorial assumes that you already performed the tutorial about sequence retrieval (above), and that you have the result files in the current directory. Check with the command:

```
cd ${HOME}/practical_rsat
ls -1
```

You should see the following file list:

```
Escherichia_coli_K_12_substr__MG1655_uid57779_start_codons.wc
Escherichia_coli_K_12_substr__MG1655_uid57779_stop_codons.wc
MET_genes.txt
MET_up800-noorf.fasta
PHO_genes.txt
PHO_up800-noorf.fasta
PHO_up800.fasta
RAND_genes.txt
RAND_up800-noorf.fasta
his.genes.txt
his.up200.noorf.fasta
```

5.2 oligo-analysis

The program ***oligo-analysis*** is the simplest pattern discovery program. It counts the number of occurrences of all oligonucleotides (words) of a given length (typically 6), and calculates the statistical significance of each word by comparing its observed and expected occurrences. The program returns words with a significant level of over-representation.

Despite its simplicity, this program generally returns good results for groups of co-regulated genes in microbes.

For a first trial, we will simply use the program to count word occurrences. The application will be to check the start and stop codons retrieved above. We will then use ***oligo-analysis*** in a pattern discovery process, to detect over-represented words from the set of upstream sequences retrieved above (the PHO family). In a first time, we will use the appropriate parameters, which have been optimized for pattern discovery in yeast upstream sequences (van Helden et

al., 1998). We will then use the sub-optimal settings to illustrate the fact that the success of word-based pattern-discovery crucially depends on a rigorous statistical approach (choice of the background model and of the scoring function).

5.2.1 Counting word occurrences and frequencies

Try the following command:

```
oligo-analysis -v 1 -i Escherichia_coli_K_12_substr__MG1655_uid57779_start_codons.wc \
    -format wc -l 3 -1str
```

Call the on-line option description to understand the meaning of the options you used:

```
oligo-analysis -help
```

Or, to obtain more details:

```
oligo-analysis -h
```

You can also ask some more information by speifying a verbosity of 1 (option *-v 1*), and store the result in a file:

```
oligo-analysis -v 1 -i Escherichia_coli_K_12_substr__MG1655_uid57779_start_codons.wc \
    -format wc -l 3 -1str -return occ,freq \
    -o Escherichia_coli_K_12_substr__MG1655_uid57779_start_codon_frequencies.tab
```

Read the result file:

```
more Escherichia_coli_K_12_substr__MG1655_uid57779_start_codon_frequencies.tab
```

Note the effect of the verbose option (*-v 1*). You receive information about sequence length, number of possible oligonucleotides, the content of the output columns, ...

Exercise 5.1 *Follow the same procedure as above to check the frequencies of stop codons in the genomes of Escherichia coli K12, and Saccharomyces cerevisia, respectively.*

5.2.2 Pattern discovery in yeast upstream regions

Try the following command:

```
oligo-analysis -i PHO_up800-noorf.fasta -format fasta    \
    -v 1 -l 6 -2str -lth occ_sig 0 -noov \
    -return occ,proba,rank -sort \
    -bg upstream-noorf -org Saccharomyces_cerevisiae \
    -o PHO_up800-noorf_6nt-2str-noov_ncf_sig0
```

Note that the return fields (“occ”, “proba”, and “rank”) are separated by a comma *without* space. Call the on-line help to understand the meaning of the parameters.

```
oligo-analysis -h
```

For this analysis, the expected frequency of each word was estimate on the basis of pre-calibrated frequency tables. These tables have been previously calculated (with oligo-analysis) by counting hexanucleotide frequencies in the whole set of yeast upstream regions (*-bg upstream*). Our experience is that these frequencies are the optimal estimator for discovering regulatory elements in upstream sequences of co-regulated genes.

Analyze the result file:

```
more PHO_up800-noorf_6nt-2str-noov_ncf_sig0
```

```
; Counted on both strands
;  grouped by pairs of reverse complements
; Background model              upstream
; Organism                      Saccharomyces_cerevisiae
; Method                        Frequency file
...
; Nb of sequences               19
; Sum of sequence lengths       11352
; discarded residues            0  (other letters than ACGT)
; discarded occurrences         0  (contain discarded residues)
; nb possible positions         11257
; total oligo occurrences       11257
[...]
; nb possible oligomers         2080
; oligomers tested for significance 2080
[...]
; column headers
; 1 seq             oligomer sequence
; 2 identifier      oligomer identifier
; 3 exp_freq        expected relative frequency
; 4 occ             observed occurrences
; 5 exp_occ         expected occurrences
; 6 occ_P           occurrence probability (binomial)
; 7 occ_E           E-value for occurrences (binomial)
; 8 occ_sig         occurrence significance (binomial)
; 9 rank            rank
; 10 ovl_occ         number of overlapping occurrences (discarded from the count)
; 11 forbocc         forbidden positions (to avoid self-overlap)
; 12 test
;seq identifier exp_freq occ exp_occ occ_P occ_E occ_sig rank ovl_occ forbocc
acgtgc acgtgc|gcacgt 0.0002182431087 16 2.46 8.4e-09 1.7e-05 4.76 1 2 76
cccacg cccacg|cgtggg 0.0001528559297 11 1.72 2e-06 4.2e-03 2.37 2 0 55
acgtgg acgtgg|ccacgt 0.0002257465554 13 2.54 2.8e-06 5.9e-03 2.23 3 1 65
cacgtg cacgtg|cacgtg 0.0001299168211 10 1.46 3.3e-06 6.8e-03 2.17 4 0 100
cgcacg cgcacg|cgtgcg 0.0001322750472 10 1.49 3.8e-06 8.0e-03 2.10 5 0 50
cgtata cgtata|tatacg 0.0005113063008 17 5.76 0.00011 2.2e-01 0.65 6 1 85
agagat agagat|atctct 0.0006913890231 19 7.78 0.00047 9.8e-01 0.01 7 0 95
```

A few questions:

1. How many hexanucleotides can be formed with the 4-letter alphabet A,T,G,C ?

2. How many possible oligonucleotides were analysed here ? Is it the number you would expect ? Why ?

3. How many patterns have been selected as significant ?

4. By simple visual inspection, can you identify some sequence similarities between the selected patterns?

5.2.3 Answers

1. The number of possible hexanucleotides is $4^6 = 4,096$.

2. The result file however reports $2,080$ possible oligonucleotides. This is due to the fact that the analysis was performed on both strands. Each oligonucleotide is thus regrouped with its reverse complement. The number of pairs is hovever larger than 4096/2, because there are $4^3 = 64$ motifs (e.g. CACGTG) which are identical to their reverse complements. The number of motifs distinct from their reverse complement is thus $4,069 - 64 = 4,032$, and they are regrouped into $4,032/2 = 2,016$ pairs. The total number of motifs is thus $T = 64 + 2016 = 2080$.

3. Among the 2080 tested oligonucleotides (+reverse complement), no more than 7 were selected as significantly over-represented.

4. Some pairs of words are mutually overlapping (e.g. `ACGTGc` and `cACGTG`).

We can now interpret these results in terms of statistics.

exp_freq The expected frequency of an oligonucleotide is the probability to find it by chance at any position of the sequences analyzed. The expected frequencies are estimated on the basis of the background model.

The program ***oligo-analysis*** uses the binomial statistics to compare the observed and expected number of occurrences, an to calculate the over-representation statistics.

Pval P-value: probability for a given oligonucleotide to be a false positive, i.e. to be considered as over-represented whereas it is not.

$Eval = T \cdot Pval$ number of false positive patterns expected by chance given the P-value of the considered pattern.

$occ_sig = -log_{10}(Eval)$ significance of the oligonucleotide occurrences. This is a simple minus-log conversion of th E-value.

5.2.4 Assembling the patterns

A separate program, ***pattern-assembly***, allows to assemble a list of patterns, in order to group those that overlap mutually. Try:

```
pattern-assembly -i PHO_up800-noorf_6nt-2str-noov_ncf_sig0 \
    -v  1 -subst 1 -2str -o PHO_up800-noorf_6nt-2str-noov_ncf_sig0.asmb
```

Read the on-line help to have a look at the assembly parameters.

```
pattern-assembly -h
```

Let us have a look at the assembled motifs.

```
more PHO_up800-noorf_6nt-2str-noov_ncf_sig0.asmb
```

Should give something llike this (the precise result might be slightly different depending on the version of the genome).

```
; pattern-assembly  -i PHO_up800-noorf_6nt-2str-noov_ncf_sig0 -v 1 -subst 1 -2str -o PHO_up800-noorf_6nt-2str-noov_ncf_sig0.asmb
; Input file    PHO_up800-noorf_6nt-2str-noov_ncf_sig0
; Output file   PHO_up800-noorf_6nt-2str-noov_ncf_sig0.asmb
; Input score column            8
; Output score column           0
; two strand assembly
; max flanking bases            1
; max substitutions             1
; max assembly size             50
; max number of patterns        100
; number of input patterns      7
;

;assembly # 1   seed: acgtgc    9 words length
;   alignt         rev_cpl      score
cccacg....      ....cgtggg      2.37
cgcacg....      ....cgtgcg      2.10
.gcacgt...      ...acgtgc.      4.76
.ccacgt...      ...acgtgg.      2.23
..cacgtg..      ..cacgtg..      2.17
...acgtgc.      .gcacgt...      4.76
...acgtgg.      .ccacgt...      2.23
....cgtggg      cccacg....      2.37
....cgtgcg      cgcacg....      2.10
cgcacgtgcg      cgcacgtgcg      4.76    best consensus

; Isolated patterns: 2
;alignt rev_cpl score
cgtata  tatacg  0.65    isol
agagat  atctct  0.01    isol
;Job started 26/10/06 09:58:21 CDT
;Job done    26/10/06 09:58:21 CDT
```

The result of the assembly shows us that several of the significant hexanucleotides actually reflect various fragments of a same motif. We also see that, despite the fact that ***oligo-analysis*** only analyzed the 4-letters DNA alphabet, the assembly indicates some degeneracy in the motif, revealed by the presence of alternative letters at the same position. For instance, in the penultimate position of the assembly, we can observe either `C` or `G`. In addition, the scores besides each oligonucleotide indicate us that these alternative letters can be more or less significantly over-represented in our sequence set. In summary, the result of *pattern-assembly* is the real key to the interpretation of *oligo-analysis*: the discovered motifs are not each separate oligo-analysis, but the assemblies that can be formed out of them.

The *best consensus* indicates, for each position of the alignment, the letter corresponding to the oligonucleotide with the highest significance. This consensus should be considered with

caution, because its complete sequence is built from the collection of various oligonucleotides, and might not correspond to any real site in the input sequences. Also, this "best consensus" is generally too stringent to perform pattern matching (see next chapters), and we usually prefer to search all the oligonucleotides separately, and analyze their feature map to identify the putative cis-acting elements.

Exercise 5.2 *Use the same procedure as above to discover over-represented hexanucleotides in the upstream sequences of the MET genes obtained in the chapter on sequence retrieval. Analyze the results of* ***oligo-analysis*** *and* ***pattern-assembly****.*

Exercise 5.3 *Use the same procedure as above to discover over-represented hexanucleotides in the upstream sequences of the RAND genes (randoms election of genes) obtained in the chapter on sequence retrieval. Analyze the results of* ***oligo-analysis*** *and* ***pattern-assembly****.*

5.2.5 Alternative background models

One of the most important parameters for the detectin of significant motifs is the choice of an appropriate background model.

This chapter aims at emphasizing how crucial is the choice of appropriate statistical parameters. We saw above that a background model calibrated on all the yeast upstream sequences gives good results with the PHO family: despite the simplicity of the algorithm (counting non-degenerate hexanucleotide occurrences), we were able to extract a description of the regulatory motif over a larger width than 6 (by pattern assembly), and we got some description of the degeneracy (the high and low affinity sites).

We will now intentionally try other parameter settings and see how they affect the quality of the results.

Equiprobable oligonucleotides

Let us try the simplest approach, where each word is considered equiprobable. For this, we simply suppress the options *-bg upstream -org Saccharomyces_cerevisiae* fom the above commands. We also omit to specify the output file, so results will immediately appear on the screen.

```
oligo-analysis -v 1 -i PHO_up800-noorf.fasta -format fasta \
  -l 6 -2str  -return occ,proba,rank -lth occ_sig 0 -sort -bg equi
```

You can combine ***oligo-analysis*** and ***pattern-assembly*** in a single command, by using the pipe character as below.

```
oligo-analysis -i PHO_up800-noorf.fasta -format fasta -v 1 \
    -l 6 -2str  -return occ,proba -lth occ_sig 0 -sort \
    | pattern-assembly -2str -subst 1 -v 1
```

On unix systems, the "pipe" character is used to concatenate commands, i.e. the output of the first command (in this case ***oligo-analysis***) is not printed to the screen, but is sent as input for the second command (in this case ***pattern-assembly***).

Note that

- The number of selected motifs is higher than in the previous trial. with the 2006 version of the sequences, I obtain 92 patterns, instead of the 7 obtained with the background model calibrated on yeast upstream sequences.
- The most significant motifs are not related to the Pho4p binding sites. All these false positives are AT-rich motifs.
- Two of the selected patterns (`acgttt` and `acgtgc`) are related to Pho4p binding site. However, they come at the 56^{th} and 65^{th} positions only.
- With this background model, we would thus not be able to detect the Pho4p binding sites.

Markov chains

Another possibility is to use Markov chain models to estimate expected word frequencies. Try the following commands and compare the results. None is as good as the option *-bg upstream-noorf*, but in case one would not have the pre-calibrated non-coding frequencies (for instance if the organism has not been completely sequenced), Markov chains can provide an interesting approach.

in a Markov chain model, the probability of each oligonucleotide is estimated on the basis of the probabilities smaller oligonucleotides that enter in its composition.

We will first apply a Markov model of order 1.

```
## Markov chain of order 1
oligo-analysis -v 1 -markov 1 \
    -i PHO_up800-noorf.fasta -format fasta \
    -l 6 -lth occ_sig 0 -sort \
    -2str -return occ,proba,rank \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv1

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv1
```

The number of patterns is strongly reduced, compared to the equiprobable model. A few AT-rich patterns are still present, but the Pho4p motif now appears at the 3rd position. We can assemble these oligos in order to highligh the different motifs.

```
pattern-assembly -i PHO_up800-noorf_6nt-2str-noov_sig0_mkv1 \
    -2str -sc 7 -subst 1 -v 1 \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv1.asmb

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv1.asmb
```

We can now increase the stringency, by using a Markov model of order 2.

```
## Markov chain of order 2
oligo-analysis -v 1 -markov 2 \
    -i PHO_up800-noorf.fasta -format fasta \
    -l 6 -lth occ_sig 0 -sort \
    -2str -return occ,proba,rank \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv2

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv2
```

We now have a very restricted number of patterns, with onnly 2 remaining AT-rich motifs. Besides these, most of the selected oligos can be assembled to form a moti corresponding to the Pho4p binding site.

```
pattern-assembly -i PHO_up800-noorf_6nt-2str-noov_sig0_mkv2 \
    -2str -sc 7 -subst 1 -v 1 \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv2.asmb

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv2.asmb
```

We can still increase the stringency with a Markov model of order 3.

```
## Markov chain of order 3
oligo-analysis -v 1 -markov 3 \
    -i PHO_up800-noorf.fasta -format fasta \
    -l 6 -lth occ_sig 0 -sort \
    -2str -return occ,proba,rank \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv3

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv3
```

If we further increase the order of the Markov chain, there is not a single significant oligonucleotide.

```
## Markov chain of order 4
oligo-analysis -v 1 -markov 4 \
    -i PHO_up800-noorf.fasta -format fasta \
    -l 6 -lth occ_sig 0 -sort \
    -2str -return occ,proba,rank,rank \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv4

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv4
```

Bernoulli model

Note that the Markov order 0 means that there is no dependency between successive residues. The probability of a word is thus simply the product of its residue probabilities. This is a *Bernoulli model*, but, by extension of the concepts of Markov chain, it is accepted to call it markov chain of order 0.

```
## Markov chain of order 0 = Bernoulli model
oligo-analysis -v 1 -markov 0 \
    -i PHO_up800-noorf.fasta -format fasta \
    -l 6 -lth occ_sig 0 -sort \
    -2str -return occ,proba,rank \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv0
pattern-assembly -i PHO_up800-noorf_6nt-2str-noov_sig0_mkv0 \
    -2str -sc 7 -subst 1 -v 1 \
    -o PHO_up800-noorf_6nt-2str-noov_sig0_mkv0.asmb

more PHO_up800-noorf_6nt-2str-noov_sig0_mkv0.asmb
```

Summary about the Markov chain background models

- The Markov model of order 1 returns AT-rich patterns with the highest significance, but the Pho4p high affinity site is described with a good accuracy. The medium affinity site appears as a single word (acgttt) in the isolated patterns.
- Markov order 1 returns less AT-rich motifs. The poly-A (aaaaaa) is however still associated with the highest significance, but comes as isolated pattern.
- The higher the order of the markov chain, the most stringent are the conditions. For small sequence sets, selecting a too high order prevents from selecting any pattern. Most of the patterns are missed with a Markov chain of order 2, and higher orders don't return any single significant word.

5.3 Genome-scale pattern discovery

The detection of exceptional words can also be used to detect signals in large sequence sets, such as the whole set of upstream sequences for a given organism, or even its complete genome. We will illustrate this with two examples.

5.3.1 Detection of over-represented words in all the yeast upstream sequences

```
retrieve-seq -org Saccharomyces_cerevisiae -type upstream -all \
    -from -1 -to -800 -noorf -o Saccharomyces_cerevisiae_allup_800-noorf.fasta.gz
```

Note that we added the extension `.gz` to the name of the output file. This suffix is interpreted by all the ***RSAT*** programs as an indication to compress the result using the command

```
gzip
```

. The result file occupies a much smaller space on your hard drive.

We will now analyze the frequency of all the heptanucleotides, and analyze their level of over- or under-representation (for this, we use the option *-two_tails*). To estimate expected frequencies, we will use a Markov model of order 4 (the other models are left as exercise).

```
oligo-analysis -v 1 -i Saccharomyces_cerevisiae_allup_800-noorf.fasta.gz \
    -l 7 -2str -noov -return occ,freq,proba,zscore,rank -sort -markov 4 \
    -two_tails \
    -o   Saccharomyces_cerevisiae_allup_800-noorf_7nt-2str-noov_mkv4.tab
```

you can now compare the most significant oligonucleotides with the transcription factor binding sites annotated in SCPD, the Sacharomyces cerevisiae Promoter Database (http://rulai.cshl.

5.3.2 Detection of under-represented words in bacterial genomes

Exercise 5.4 *Analyze the frequencies of all the hexanucleotides in Escherichia coli K12. One of them shows a very high degree of under-representation. Try to understand the reason why this hexanucleotide is avoided in this genome.*

Info: *the full genome of Escherichia coli K12 can be found in the* ***RSAT****genome directory.*

```
ls $RSAT/data/genomes/Escherichia_coli_K_12_substr__MG1655_uid57779/genome/contigs.txt
```

This file contains the list of chromosomes of the bacteria (in this cxase there is a single one, but for S.cerevisiae there are 16 nuclear and one mitochondrial chromosomes). It can be idrectly used as input by specifying the format -format filelist.

5.4 dyad-analysis

In the previous chapter, we saw that ***oligo-analysis*** allows to detect over- and under-represented motifs in biological sequences, according to a user-specified background model. Since 1997, this program has been routinely used to predict cis-acting elements from groups of co-expressed genes.

However, some motifs escape to ***oligo-analysis***, because they do not correspond to an oligonucleotide, but to a spaced pair of very short oligonucleotides (*dyads*). To address this problem, we developed another program called ***dyad-analysis***.

TO BE WRITTEN

6 String-based pattern matching

In a pattern matching problem, you start from one or several predefined patterns, and you match this pattern against a sequence, i.e. you locate all occurrences of this pattern in the sequences.

Patterns can be represented as strings (with *dna-pattern*) or position-weight matrices (with *patser*).

6.1 dna-pattern

dna-pattern is a string-based pattern matching program, specialized for searching patterns in DNA sequences.

- This specialization mainly consists in the ability to search on both the direct and reverse complement strands.
- A single run can either search for a single pattern, or for a list of patterns.
- multi-sequence file formats (fasta, filelist, wc, ig) are supported, allowing to match patterns against a list of sequences with a single run of the program.
- String descriptions can be refined by using the 15-letters IUPAC code for uncompletely specified nucleotides, or by using regular expressions.
- The program can either return a list of matching positions (default behaviour), or the count of occurrences of each pattern.
- Imperfect matches can be searched by allowing substitutions. Insertions and deletions are not supported. The reason is that, when a regulatory site presents variations, it is generally in the form of a tolerance for substitution at a specific position, rather than insertions or deletions. It is thus essential to be able distinguishing between these types of imperfect matches.

6.2 Matching a single pattern

We will start by searching all positions of a single pattern in a sequence set. The sequence is the set of upstream regions from the PHO genes, that was obtained in the tutorial on sequence retrieval. We will search all occurrences of the most conserved core of the Pho4p medium affinity binding site (`CACGTT`) in this sequence set.

Try the following command:

```
dna-pattern -v 1 -i PHO_up800.fasta -format fasta \
    -1str -p cacgtt -id 'Pho4p_site'
```

You see a list of positions for all the occurrences of CACGTT in the sequence.
Each row represents one match, and the columns provide the following information:

1. pattern identifier
2. strand
3. pattern searched
4. sequence identifier
5. start position of the match
6. end position of the match
7. matched sequence
8. matching score

6.3 Matching on both strands

To perform the search on both strands, type:

```
dna-pattern -v 1 -i PHO_up800.fasta -format fasta \
    -2str -p cacgtt -id 'Pho4p_site'
```

Notice that the strand column now contains two possible values: D for "direct" and R for "reverse complement".

6.4 Allowing substitutions

To allow one substitutions, type:

```
dna-pattern -i PHO_up800.fasta -format fasta \
    -2str -p cacgtt -id 'Pho4p_site' -subst 1
```

Notice that the score column now contains 2 values: 1.00 for perfect matches, 0.83 (=5/6) for single substitutions. This si one possible use of the score column: when substitutions are allowed, the score indicates the percentage of matching nucleotides.

Actually, for regulatory patterns, allowing substitutions usually returns many false positive, and this option is usually avoided. We will not use it further in the tutorial.

6.5 Extracting flanking sequences

The matching positions can be extracted along with their flanking nucleotides. Try:

```
dna-pattern -i PHO_up800.fasta -format fasta \
     -2str -p cacgtt \
     -id  'Pho4p_site' -N 4
```

Notice the change in the matched sequence column: each matched sequence contains the pattern CACGTT in uppercase, and 4 lowercase letters on each side (the flanks).

6.6 Changing the origin

When working with upstream sequences, it is convenient to work with coordinates relative to the start codon (i.e. the right side of the sequence). Sequence matching programs (including dna-pattern) return the positions relative to the beginning (i.e. the left side) of the sequence. The reference (coordinate 0) can however be changed with the option `-origin`. In this case, we retrieved upstream sequences over 800bp. the start codon is thus located at position 801. Try:

```
dna-pattern -i PHO_up800.fasta -format fasta \
    -2str -p cacgtt \
    -id  'Pho4p_site' -N 4 -origin 801
```

Notice the change in coordinates.

In some cases, a sequence file will contain a mixture of sequences of different length (for example if one clipped the sequences to avoid upstream coding sequences). The origin should thus vary from sequence to sequence. A convenient way to circumvent the problem is to use a negative value with the option `origin`. for example, `-origin -100` would take as origin the 100th nucleotide starting from the right of each sequence in the sequence file. But in our case we want to take as origin the position immediately after the last nucleotide. For this, there is a special convention: `-origin -0`.

```
dna-pattern -i PHO_up800.fasta -format fasta \
    -2str -p cacgtt \
    -id  'Pho4p_site' -N 4 -origin -0
```

In the current example, since all sequences have exactly 800bp length, the result is identical to the one obtained with `-origin 801`.

6.7 Matching degenerate patterns

As we said before, there are two forms of Pho4p binding sites: the protein has high affinity for motifs containing the core CACGTG, but can also bind, with a medium affinity, CACGTT

sites. The IUPAC code for partly specified nucleotides allows to represent any combination of nucleotides by a single letter.

A		(Adenine)
C		(Cytosine)
G		(Guanine)
T		(Thymine)
R	= A or G	(puRines)
Y	= C or T	(pYrimidines)
W	= A or T	(Weak hydrogen bonding)
S	= G or C	(Strong hydrogen bonding)
M	= A or C	(aMino group at common position)
K	= G or T	(Keto group at common position)
H	= A, C or T	(not G)
B	= G, C or T	(not A)
V	= G, A, C	(not T)
D	= G, A or T	(not C)
N	= G, A, C or T	(aNy)

Thus, we could use the string CACGT**K** to represent the Pho4p consensus, and search both high and medium affinity sites in a single run of the program.

```
dna-pattern -i PHO_up800.fasta -format fasta \
    -2str -p cacgtk \
    -id  'Pho4p_site' -N 4 -origin -0
```

6.8 Matching regular expressions

Another way to represent partly specified strings is by using regular expressions. This not only allows to represent combinations of letters as we did above, but also spacings of variable width. For example, we could search for tandem repeats of 2 Pho4p binding sites, separated by less than 100bp. This can be represented by the following regular expression:

```
cacgt[gt].{0,100}cacgt[gt]
```

which means

- cacgt
- followed by either g or t [gt]
- followed by 0 to 100 unspecified letters .0,100
- followed by cacgt
- followed by either g or t [gt]

Let us try to use it with dna-pattern

```
dna-pattern -i PHO_up800.fasta -format fasta \
    -2str -id 'Pho4p_pair' \
    -N 4 -origin -0 \
    -p 'cacgt[gt].{0,100}cacgt[gt]'
```

Note that the pattern has to be quoted, to avoid possible conflicts between special characters used in the regular expression and the unix shell.

6.9 Matching several patterns

TO match a series of patterns, you first need to store these patterns in a file. Let create a pattern file:

```
cat > test_patterns.txt
cacgtg high
cacgtt medium
```

(then type Ctrl-d to close)
check the content of your pattern file.

```
more test_patterns.txt
```

There are two lines, each representing a pattern. The first word of each line contains the pattern, the second word the identifier for that pattern. This column can be left blank, in which case the pattern is used as identifier.

We can now use this file to search all matching positions of both patterns in the PHO sequences.

```
dna-pattern -i PHO_up800.fasta -format fasta \
    -2str  -N 4 -origin -0 \
    -pl test_patterns.txt
```

6.10 Counting pattern matches

In the previous examples, we were interested in matching positions. It is sometimes interesting to get a more synthetic information, in the form of a count of matching positions for each sequences. Try:

```
dna-pattern -i PHO_up800.fasta -format fasta \
     -2str  -N 4 -origin -0 -c \
     -pl test_patterns.txt
```

With the option `-c`, the program returns the number of occurrences of each pattern in each sequence. The output format is different: there is one row for each combination pattern-sequence. The columns indicate respectively

1. sequence identifier
2. pattern identifier
3. pattern sequence
4. match count

An even more synthetic result can be obtained with the option `-ct` (count total).

```
dna-pattern -i PHO_up800.fasta -format fasta -2str \
    -pl test_patterns.txt -N 4 -origin -0 -ct
```

This time, only two rows are returned, one per pattern.

6.11 Getting a count table

Another way to display the count information is in the form of a table, where each row represents a gene and each column a pattern.

```
dna-pattern -i PHO_up800.fasta -format fasta -2str \
    -pl test_patterns.txt -N 4 -origin -0 -table
```

This representation is very convenient for applying multivariate statistics on the results (e.g. classifying genes according to the patterns found in their upstream sequences)

Last detail: we can add one column and one row for the totals per gene and per pattern.

```
dna-pattern -i PHO_up800.fasta -format fasta -2str \
    -pl test_patterns.txt -N 4 -origin -0 -table -total
```

7 Drawing graphs

7.1 feature-map

The program ***feature-map*** draws a graphical map of a list of features. A typical usage of feature-map is to draw maps with the positions of regulatory motifs detected by pattern matching programs such ***dna-pattern*** (string-based matching) or ***patser*** (matrix-based matching).

7.1.1 Converting *dna-pattern* matches into features

We will analyze the same PHO family as in the tutorial on pattern discovery. We will use successively ***oligo-analysis***, ***dna-pattern*** and ***convert-features*** to obtain a list of features with the matching locations of the over-represented hexanucleotides.

1. Run ***oligo-analysis*** to detect over-represented hexanucleotides in the upstream sequences of the PHO genes.

```
oligo-analysis -i PHO_up800.fasta -format fasta      \
    -v -l 6 -2str                                    \
    -return occ,proba -lth occ_sig 0 -bg upstream        \
    -org Saccharomyces_cerevisiae -sort              \
    -o PHO_up800_6nt_2str_ncf_sig0
```

2. Run ***dna-pattern*** to locate these patterns in the upstream sequences.

```
dna-pattern -i PHO_up800.fasta -format fasta    \
    -pl PHO_up800_6nt_2str_ncf_sig0 -origin -0  \
    -o  PHO_up800_6nt_2str_ncf_sig0_matches.tab
```

3. Run ***convert-features*** to convert these pattern matches into features.

```
convert-features                                \
    -from dnapat -to ft                         \
    -i PHO_up800_6nt_2str_ncf_sig0_matches.tab \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.ft
```

We will now play with this feature file, in order to obtain different drawings.

7.1.2 Basic feature maps

```
feature-map -format jpg                              \
    -i PHO_up800_6nt_2str_ncf_sig0_matches.ft  \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg
```

You can now open the file *PHO_up800_6nt_2str_ncf_sig0_matches.jpg* with a web browser or a drawing application.

This is a very simple representation: each feature is represented as a box. A specific color is associated to each pattern (feature ID).

7.1.3 Refining the feature map

We will use a few additional options to add information on this feature map.

```
feature-map -format jpg                          \
    -i PHO_up800_6nt_2str_ncf_sig0_matches.ft  \
    -legend -scalebar -scalestep 50              \
    -from -800 -to 0 -scorethick                 \
    -title 'Over-represented 6nt in PHO upstream sequences' \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg
```

This example illustrates some capabilities of ***feature-map***:

- A title has been added to the drawing.
- A specific height is associated to each box, to reflect the score associated to the corresponding feature.
- The scale bar indicates the location, in base pairs.
- A legend indicates the color associated to each pattern, as well as its score.

7.1.4 Map orientation

Feature-maps can be oriented horizontally or vertically. The horizontal orientation is usually the most convenient, but when labels are attached to each feature, the vertical orientation prevents them from expanding over each other.

```
feature-map -format jpg                          \
    -i PHO_up800_6nt_2str_ncf_sig0_matches.ft  \
    -legend -scalebar -scalestep 50              \
    -from -800 -to 0                             \
    -vertical -symbol -label pos                 \
    -title 'Over-represented 6nt in PHO upstream sequences'  \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg
```

In this representation, a *label* is written besides each feature box. In addition, a *symbol* has been attached to each feature ID (pattern). This symbol improves the readability of the map, and is convenient for monochrome printers.

7.1.5 Export formats

Feature-map can be exported in different formats, specified with the option *-format*.

jpg (default) The *jpg* format (also called *jpeg*) is a bitmap format recognized by all the web browsers and most drawing applications. The jpg standard includes a compression protocol, so that the resulting images occupy a reasonable space on the hard disk.

png The *png* format is a bitmap format which gives a better color rendering than jpg. It is not compressed, and requires more space for storage. It is recognized by most browsers.

ps The *postscript* (*ps*) format is a vectorial format, which ensures a high quality result on printing devices. Postscript files can be opened with specific applications, depending on the operating system (ghostview, ghostscript). This format is recommended for drawing graphs to be included in publications.

7.1.6 HTML maps

A HTML map can be created, which allows to display dynamically the feature-map in a web browser. When the users positions the mouse over a feature, the corresponding information is displayed in the status bar.

```
feature-map -format jpg                          \
    -i PHO_up800_6nt_2str_ncf_sig0_matches.ft    \
    -legend -scalebar -scalestep 50              \
    -from -800 -to 0                             \
    -scorethick  -dots                           \
    -title 'Over-represented 6nt in PHO upstream sequences' \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg \
    -htmap > PHO_up800_6nt_2str_ncf_sig0_matches.html
```

Notice that we used the option *-dot* to attach a colored filled circle to each feature box.

Open the file *PHO_up800_6nt_2str_ncf_sig0_matches.html* with a web browser (e.g. Netscape, Mozilla, Safari). Position the mouse cursor over a feature (either the box or the filled circle attached to it), and look the status bar at the bottom of the browser window.

7.1.7 Other options

The program ***feature-map*** includes a few other options.

```
feature-map -help
```

A complete description of their functionality is provided in the help pages.

```
feature-map -h
```

7.1.8 Feature converters

In the previous tutorial, we used the program ***convert-features*** to convert matches from ***dna-pattern*** to features.

RSATincludes a few additional converters (these are older versions, and their functionalities will progressively be incorporated in ***convert-features***).

features-from-dssp extracts features from the output file of ***dssp*** (secondary structures)

features-from-fugue extracts features from the output file of ***fugue***

features-from-gibbs extracts features from the ***gibbs*** motif sampler, developed by Andrew Neuwald.

features-from-matins extracts features from the result of ***matinspector***, developed in Thomas Werner's team.

features-from-msf converts a multiple alignment file from format *msf* for features.

features-from-patser extracts features from the result of the matrix-based pattern matching ***patser***, developed by Jerry Hertz.

features-from-sigscan extracts features from the results of the ***sigscan*** program.

features-from-swissprot extracts features from a ***Swissprot*** file.

If you need to draw features from any other type of program output, it is quite simple to write your own converter. The feature-map input is a tab-delimited text file, with one row per feature, and one column per attribute.

1. map label (eg gene name)
2. feature type
3. feature identifier (ex: GATAbox, Abf1_site)
4. strand (D for Direct, R for Reverse),
5. feature start position
6. feature end position
7. (optional) description
8. (optional) score

7.2 XYgraph

The program ***XYgraph*** is a simple utility which plots graphs from a series of (x,y) coordinates.

7.2.1 Exercise: drawing features from patser

In the section on pattern-matching, we scanned all yeast upstream sequences with the PHO matrix and stored the result in a file (`PHO_matrix_matches_allup.txt`).

With the programs *features-from-patser* and *feature-map*, draw a map of the sites found in this analysis.

8 Markov models

Markov models allow to represent local dependencies between successive residues. A Markov model of order m assumes that the probability to find the residue r at position i of a sequence depends on the m preceding residues.

8.0.2 Transition frequency tables

Markov models are described by transition frequencies $P(R|W_m)$, i.e. the probability to osberve residue R at a certain position, depending on the preceding word W_m of size m.

8.0.3 Oligonucleotide frequency tables

RSAT allows to derive organism-specific Markov models from oligonucleotide frequency tables.

Pre-calibrated oligonucleotide frequency tables are stored in the form of oligonucleotide frequency tables (see chapter on pattern discovery).

The calibration tables for *Escherichia coli K12* can be found in the ***RSAT*** directory *oligo-frequencies*.

```
cd $RSAT/data/genomes/Escherichia\_coli\_K12/oligo-frequencies
ls -ltr
```

For example, the file *4nt_upstream-noorf_Escherichia_coli_K12-1str.freq.gz* indicates the tetranucleotide frequencies for all the upstream sequences of *E.coli*.

```
cd $RSAT/data/genomes/Escherichia_coli_K_12_substr__MG1655_uid57779/oligo-freque

## Have a look at the content of the 4nt frequency file
gunzip -c 4nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-1str.
```

8.0.4 Converting oligonucleotide frequencies into transition frequencies

Transition frequencies are automatically derived from the table of oligonucleotide frequencies, but one should take care of the fact that, in order to estimate the transition frequencies for a Markov model of order m, we need to use the frequency tables for oligonucleotides of size $m+1$.

We can illustrate this by converting the table of dinucleotide frequencies into a transition matrix of first order. For this, we can use the program ***convert-background-model***.

```
convert-background-model \
  -i 2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-1str.freq.
  -from oligo-analysis -to tab
```

The output displays the transition matrix of a Markov model of order 1. Each row of the transition matrix indicates the prefix W_m, and each column the suffix r. For a Markov model of order 1, the prefixes are single residues.

We can now calculate a Markov model of 2nd order, from the table of trinucleotide frequencies.

```
convert-background-model \
  -i 3nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-1str.freq.
  -from oligo-analysis -to tab
```

The transition matrix contains 16 rows (prefixes, corresponding to dinucleotides) and 4 columns (the suffixes, corresponding to nucleotides).

The same operation can be extended to higher order markov models.

8.0.5 Bernoulli models

In contrast with Markov model, Bernoulli models assume that the residue probabilities are independent from the position. By extension of the concept of Markov order, Bernoulli models can be conceived as a Markov model of order 0. We can thus derive a Bernoulli model ($m = 0$) from the nucleotide frequencies ($m + 1 = 1$).

```
convert-background-model \
  -i 1nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-1str.freq.
  -from oligo-analysis -to tab
```

The suffix column is now empty (there is no suffix, since the order is 0), and the matrix simply displays 4 columns with the frequencies of `A`, `C`, `G` and `T`.

9 Matrix-based Pattern discovery

RSAT does not (yet) contain programs for matrix-based pattern discovery. However, several excellent programs exist for matrix-based pattern discovery, and it is often useful to combine various approaches in order to compare the results and select the most consistent ones. We show hereafter some examples of utilization for some of these programs:

- ***consensus***, a greedy approach of pattern discovery, developed by Jerry Hertz.

9.1 consensus (program developed by Jerry Hertz)

An alternative approach for matrix-based pattern discovery is *consensus*, a program written by Jerry hertz, an based on a greedy algorithm. We will see how to extract a profile matrix from upstream regions of the PHO genes.

9.1.1 Getting help

As for RSAT programs, there are two ways to get help from Jerry Hertz' programs: a detailed manual can be obtained with the option *-h*, and a summary of options with *-help*. Try these options and read the manual.

```
consensus -h
consensus -help
```

9.1.2 Sequence conversion

consensus uses a custom sequence format. Fortunately, the RSAT package contains a sequence conversion program (*convert-seq*) which supports Jerry Hertz' format. We will thus start by converting the fasta sequences in this format.

```
convert-seq -i PHO_up800-noorf.fasta -from fasta -to wc -o PHO_up800-noorf.wc
```

9.1.3 Running consensus

Using consensus requires to choose the appropriate value for a series of parameters. We found the following combination of parameters quite efficient for discovering patterns in yeast upstream sequences.

```
consensus -L 10 -f PHO_up800-noorf.wc -A a:t c:g -c2 -N 10
```

The main options used above are

-L 10 we guess that the pattern has a length of about 10 bp;

-N 10 we expect about 10 occurrences in the sequence set. Since there are 5 genes in the family, this means that we expect on average 2 regulatory sites per gene, which is generally a good guess for yeast.

-c2 indicates ***consensus*** that the motif can be searched on both strands.

-A a:t c:g specifies the alphabet. Indeed, ***consensus*** can be used to extract motif from DNA sequences, proteins, or a text based on an arbitrary alphabet. In thus tutorial we are only interested in DNA sequences, we wpecify thus *-A a:t c:g* (the semicolons indicate the complementary residues).

By default, several matrices are returned. Each matrix is followed by the alignment of the sites on which it is based. Note that the 4 matrices are highly similar, basically they are all made of several occurrences of the high afinity site `CACGTG`, and matrices 1 and 3 contain one occurrence of the medium affinity site `CACGTT`. These matrices are thus redundant, and it is generally appropriate to select the first one of the list for further analysis, because it is the most significant matrix found by the program.

Also notice that these matrices are not made of exactly 10 sites each. *consensus* is able to adapt the number of sites in the alignment in order to get the highest information content. The option *-N 10* was an indication rather than a rigid requirement.

We can use the options *-pt 1* and *-pf 1* to restrict the result to a single matrix (the most significant one). To save the result in a file, we can use the symbol "greater than" (>) which redirects the output of a program to a file.

```
consensus -L 10 -f PHO_up800-noorf.wc -A a:t c:g -c2 -N 10 -pf 1 -pt 1 \
    > PHO_consensus_L10_N10_c2.matrix
```

(this may take a few minutes)

Once the task is achieved, check the result.

```
more PHO_consensus_L10_N10_c2.matrix
```

9.2 Random expectation

```
random-seq -format wc -r 10 -l 800 -bg upstream-noorf \
  -org Saccharomyces_cerevisiae -ol 6 -lw 0 -o rand_Sc_ol6_n10_l800.wc

consensus -L 10 -f rand_Sc_ol6_n10_l800.wc -A a:t c:g -c2 -N 10 -pf 1 -pt 1 \
    > rand_Sc_ol6_n10_l800_L10_N10_c2.matrix
```

10 Matrix-based pattern matching

10.1 Prerequisite

This tutorial assumes that you already followed the tutorial on *Matrix-based pattern discovery*.

To check this, list the files contained in directory with the results of your tutorial.

```
cd ${HOME}/practical_rsat
ls -1
```

You should find the following files.

```
PHO_up800-noorf.fasta
PHO_up800-noorf.wc
PHO_consensus_L10_N10_c2.matrix
```

10.2 patser (program developed by by Jerry Hertz)

We will now see how to match a profile matrix against a sequence set. For this, we use *patser*, a program written by Jerry Hertz.

10.2.1 Getting help

help can be obtained with the two usual options.

```
patser -h
patser -help
```

10.2.2 Extracting the matrix from the *consensus* result file

Patser requires two input data:

- a sequence file (option -*f*),
- a position-specific scoring matrix (option -*m*), like the one we obtained in the previouschapter, with *consensus*.

The output from *consensus* can however not be used directly because it contains additional information (the parameters of analysis, the sequences used to build the matrix, ...) besides the matrix itself. One possibility is to cut the matrix of interest and save it in a separate file.

To avoid manual editing, RSAT contains a program *convert-matrix*, which automaticaly extracts a matrix from various file formats, including consensus.

```
convert-matrix -in_format consensus -i PHO_consensus_L10_N10_c2.matrix  \
    -return counts -o PHO_consensus_L10_N10_c2_matrix.tab

more PHO_consensus_L10_N10_c2_matrix.tab
```

10.2.3 Getting information about a matrix

The program *convert-matrix* includes several output options, which allow you to get additional information about your matrix. For example you can obtain the degenerate consensus from a matrix with the following options.

```
convert-matrix -v 1 -pseudo 1 -in_format consensus -i PHO_consensus_L10_N10_c2.matrix \
    -return consensus


convert-matrix -v 1 -pseudo 1 -in_format consensus -i PHO_consensus_L10_N10_c2.matrix \
    -return parameters
```

The program ***convert-matrix*** also allows to derive frequencies, weights or information from the count matrix.

```
convert-matrix -v 1 -pseudo 1 -in_format consensus -i PHO_consensus_L10_N10_c2.matrix \
    -return frequencies,weights,information
```

Additional information can be otbained with the on-line help for *convert-matrix*.

```
convert-matrix -h
```

10.2.4 Detecting Pho4p sites in the PHO genes

After having extracted the matrix, we can match it against the PHO sequences to detect putative regulatory sites.

```
patser -m PHO_consensus_L10_N10_c2_matrix.tab -f PHO_up800-noorf.wc -A a:t c:g -c -ls 9
```

By default, patser uses equiprobable residue frequencies. However, we can impose our own priors in the following way.

```
patser -m PHO_consensus_L10_N10_c2_matrix.tab -f PHO_up800-noorf.wc -A a:t 0.325 c:g 0.
```

We an also adapt our expected frequencies from pre-calibrated genome frequencies, for example, residue frequencies from all the yeast upstream sequences.

```
## Calculate prior frequencies
convert-background-model -from oligo-analysis -to patser \
   -i ${RSAT}/data/genomes/Saccharomyces_cerevisiae/oligo-frequencies/1nt_upstream-noor
   -o 1nt_upstream-noorf_Saccharomyces_cerevisiae-noov-2str_patser.tab

more 1nt_upstream-noorf_Saccharomyces_cerevisiae-noov-2str_patser.tab

patser -m PHO_consensus_L10_N10_c2_matrix.tab -f PHO_up800-noorf.wc -a 1nt_upstream-noo
```

10.2.5 Detecting Pho4p sites in all upstream regions

We will now match the PHO matrix against the whole set of upstream regions from the ≈ 6000 yeast genes. This should allow us to detect new genes potentially regulated by Pho4p.

One possibility would be to use *retrieve-seq* to extract all yeast upstream regions, and save the result in a file, which will then be used as input by *patser*. Alternatively, in order to avoid occupying too much space on the disk, we can combine both tasks in a single command, and immediately redirect the output of *retrieve-seq* as input for *patser*. This can be done with the pipe character || as below.

patser result can be redirected to a file with the unix "greater than" (>) symbol. We will store the result of the genome-scale search in a file *PHO_matrix_matches_allup.txt*.

```
retrieve-seq -type upstream -from -1 -to -800  \
    -org Saccharomyces_cerevisiae \
    -all -format wc -label id,name  \
    | patser -m PHO_consensus_L10_N10_c2_matrix.tab -ls 9 -A a:t c:g \
    > PHO_consensus_L10_N10_c2_matrix.tab_matches_allup.txt

more PHO_consensus_L10_N10_c2_matrix.tab_matches_allup.txt
```

10.2.6 Interpretation of the P-value returned by *patser*

The program ***patser*** returns a column with the P-value of each mach. The P-value indicates the probability of false-positive, i.e. the probability to consider a site as an instance of the motif whereas it is not.

In other terms, the P-value represents the probability to observe a score (X) at least as high as that of the current sequence segment ($x_{i,i+w-1}$)

$$Pval = P(X \geq x_{i,i+w-1}|B)$$

where

X is a random variable representing the matrix score,

$x_{i,i+w-1}$ is the score assigned to the sequence segment of width w starting at position i of the sequence,

B is the background model.

We will evaluate the reliability of this P-value by analyzing the distribution of estimated P-value for all the positions of a random sequence. By default, ***patser*** only calculates the P-value for the weight scores > 0. We will add the option *-M -999* to force patser to calculate P-values for all the score.

The raw results from patser will be processed in the fillowing way:

1. ***features-from-patser*** converts the patser result into a tab-delimited file;
2. ***awk*** is used to cut the 8^{th} column of this file, and convert the P-value into a significance (sig=-log10(Pval))
3. ***classfreq*** calculates the distribution of ln(P-value);
4. ***XYgraph*** is used to draw an XY plot, representinf the theoretical P-value on the X axis, and on the Y axis the frequency observed for this P-value in the random sequence.

```
random-seq -l 100000 -format wc \
  | patser  -A a:t c:g -m PHO_consensus_L10_N10_c2_matrix.tab -b 1 -d1  -p -M -999 \
  | features-from-patser  \
  | XYgraph -xcol 8 -ycol 9 -o PHO_consensus_L10_N10_c2_rand_score_versus_Pval.png

random-seq -l 100000 -format wc \
  | patser  -A a:t c:g -m PHO_consensus_L10_N10_c2_matrix.tab -b 1 -d1  -p -M -999 \
  | features-from-patser  \
  | awk -F '\t' '{print -$9/log(10)}' \
  | classfreq -v -ci 0.01 -o PHO_consensus_L10_N10_c2_rand_sig_distrib.tab

more PHO_consensus_L10_N10_c2_rand_sig_distrib.tab

XYgraph -i PHO_consensus_L10_N10_c2_rand_sig_distrib.tab \
  -title1 'Validation of P-values returned by patser' \
  -title2 'Distribution of these P-values in random sequences' \
  -xcol 1 -ycol 9 -xleg1 'theoretical sig=-log10(P-value)' -ymax 1 \
  -yleg1 'inverse cumulative frequency' -ylog 10 \
  -xsize 800 -format png -lines \
  -o PHO_consensus_L10_N10_c2_rand_sig_distrib.png
```

The image file can be opened with any graphical display application (e.g. ***xv***), or with a web browser (e.g. ***Mozilla***).

The distribution almost perfectly follows a diagonal, indicating that the theoretical P-value calculated by ***patser*** corresponds to the empirical one.

However, we should bear in mind that this P-value is based on the basis of a Bernoulli model, i.e. it assumes that successive residues are independent from each other.

The previous test was based on th simplest possible model for generating the random sequence: equiprobable and independent nucleotides. We can thus wonder if the P-value will still be valid with random sequences generated following a more complex model. We will successively test two models:

- random sequences generated according to a Bernoulli model, with unequal residue frequencies;
- random sequences generated according to a higher-order Markov model.

Bernoulli model with unequal frequencies

```
## Generate a bg model for patser
convert-background-model -from oligo-analysis -to patser \
  -i $RSAT/data/genomes/Saccharomyces_cerevisiae/oligo-frequencies/1nt_upstream-noorf_S
  -o 1nt_upstream-noorf_Saccharomyces_cerevisiae-1str_freq.tab


## Generate a random sequence with a Bernouli model
## and analyze it with patser using the same expected residue frequencies
random-seq -l 100000 -format wc -bg upstream-noorf -ol 1 -org Saccharomyces_cerevisiae
  | patser  -a 1nt_upstream-noorf_Saccharomyces_cerevisiae-1str_freq.tab \
    -m PHO_consensus_L10_N10_c2_matrix.tab -b 1 -d1  -p -M -999 \
  | features-from-patser  \
  | awk -F '\t' '{print -$9/log(10)}' \
  | classfreq -v -ci 0.01 -o PHO_consensus_L10_N10_c2_rand_Mkv0_sig_distrib.tab

XYgraph -i PHO_consensus_L10_N10_c2_rand_Mkv0_sig_distrib.tab \
  -title1 'Validation of P-values returned by patser' \
  -title2 'Distribution of these P-values in random sequences' \
  -xcol 1 -ycol 9 -xleg1 'theoretical sig=-log10(P-value)' -ymax 1 \
  -yleg1 'inverse cumulative frequency' -ylog 10 \
  -xsize 800 -format png -lines \
  -o PHO_consensus_L10_N10_c2_rand_Mkv0_sig_distrib.png
```

Markov model of order 1

```
random-seq -l 100000 -format wc -bg upstream-noorf -ol 2 -org Saccharomyces_cerevisiae
  | patser -a  1nt_upstream-noorf_Saccharomyces_cerevisiae-1str_freq.tab \
    -m PHO_consensus_L10_N10_c2_matrix.tab -b 1 -d1  -p -M -999 \
  | features-from-patser  \
  | awk -F '\t' '{print -$9/log(10)}' \
  | classfreq -v -ci 0.01 -o PHO_consensus_L10_N10_c2_rand_Mkv1_sig_distrib.tab

XYgraph -i PHO_consensus_L10_N10_c2_rand_Mkv1_sig_distrib.tab \
  -title1 'Validation of P-values returned by patser' \
  -title2 'Distribution of these P-values in random sequences' \
```

```
  -xcol 1 -ycol 9 -xleg1 'theoretical sig=-log10(P-value)' -ymax 1 \
  -yleg1 'inverse cumulative frequency' -ylog 10 \
  -xsize 800 -format png -lines \
  -o PHO_consensus_L10_N10_c2_rand_Mkv1_sig_distrib.png
```

Markov model of order 5

```
random-seq -l 100000 -format wc -bg upstream-noorf -ol 6 -org Saccharomyces_cerevisiae
  | patser -a 1nt_upstream-noorf_Saccharomyces_cerevisiae-1str_freq.tab \
    -m PHO_consensus_L10_N10_c2_matrix.tab -b 1 -d1  -p -M -999 \
  | features-from-patser  \
  | awk -F '\t' '{print -$9/log(10)}' \
  | classfreq -v -ci 0.01 -o PHO_consensus_L10_N10_c2_rand_Mkv5_sig_distrib.tab

XYgraph -i PHO_consensus_L10_N10_c2_rand_Mkv5_sig_distrib.tab \
  -title1 'Validation of P-values returned by patser' \
  -title2 'Distribution of these P-values in random sequences' \
  -xcol 1 -ycol 9 -xleg1 'theoretical sig=-log10(P-value)' -ymax 1 \
  -yleg1 'inverse cumulative frequency' -ylog 10 \
  -xsize 800 -format png -lines \
  -o PHO_consensus_L10_N10_c2_rand_Mkv5_sig_distrib.png
```

10.2.7 Score distributions in promoter sequences

```
retrieve-seq -all -noorf -org Saccharomyces_cerevisiae -format wc \
  | patser -a 1nt_upstream-noorf_Saccharomyces_cerevisiae-1str_freq.tab \
    -m PHO_consensus_L10_N10_c2_matrix.tab -b 1 -d1  -p -M -999 \
  | features-from-patser  \
  | awk -F '\t' '{print -$9/log(10)}' \
  | classfreq -v -ci 0.01 -o PHO_consensus_L10_N10_c2_allup_sig_distrib.tab

XYgraph -i PHO_consensus_L10_N10_c2_allup_sig_distrib.tab \
  -title1 'Validation of P-values returned by patser' \
  -title2 'Distribution of these P-values in random sequences' \
  -xcol 1 -ycol 9 -xleg1 'theoretical sig=-log10(P-value)' -ymax 1 \
  -yleg1 'inverse cumulative frequency' -ylog 10 \
  -xsize 800 -format png -lines \
  -o PHO_consensus_L10_N10_c2_allup_sig_distrib.png
```

10.3 Scanning sequences with *matrix-scan*

The program ***matrix-scan*** allows to scan sequences with a position-specific scoring matrix (*PSSM*), in the same way as patser. However, it presents some differences:

1. ***matrix-scan*** is much slower than ***patser***, because it is a perl script (whereas ***patser*** is compiled). However, for most tasks, we can affor dto spend a few minuts per genome rather than a few seconds.

2. ***matrix-scan*** supports higher-order Markov chain models, whereas ***paters*** only supports Bernoulli models. The markov models can be defined from different sequence sets: external sequences, input sequences, or even locally (*adaptive background models*).

3. ***matrix-scan*** calculates the P-value associated to each match for Bernouilli models as well as higher-order Markov chain models.

10.3.1 Bernoulli background models

In ***matrix-scan***, the background model can be calculated from the sequences to be scanned. We use the option -bginput in association with -markov 0 to calculate a Bernoulli model from the input sequences. The option -return bg_model displays in the output details on the calculated background model.

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc -bginput -markov 0 \
   -lth score 0 -return sites,limits,bg_model \
   -origin -0 \
   -o PHO_consensus_L10_N10_c2_matches_mkv0.tab

feature-map -i PHO_consensus_L10_N10_c2_matches_mkv0.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o PHO_consensus_L10_N10_c2_matches_mkv0.png
```

10.3.2 Higher order (Markov) background models

Global background models

To use pre-calibrated background model, we use -bgfile option. Such models are available from within RSAT (refer to Chapter 8 - Markov models for more details). As input for ***matrix-scan***, we use the models trained with ***oligo-analysis*** with the options "ovlp" and "1str".

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc \
   -bgfile ${RSAT}/data/genomes/Saccharomyces_cerevisiae/oligo-frequencies/2nt_upstream
   -lth score 0 -return sites,limits,normw\
   -origin -0 \
   -o PHO_consensus_L10_N10_c2_matches_mkv1.tab

 feature-map -i PHO_consensus_L10_N10_c2_matches_mkv1.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o PHO_consensus_L10_N10_c2_matches_mkv1.png
```

In this command, we have used Markov model of order 1, and in addition to the weight, the output displays the normalised weight.

Adaptive Markov models

Adaptative background models are calculated in sliding windows centered on the scored segment. We use option -window to define the size of the window in combination with -markov for the Markov order. The return field bg_residues returns the frequencies of the residues in each background model and can be used to estimate the GC content in the surroundings of the scored segment.

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc -window 200 -markov 2 \
   -lth score 0 -return sites,limits,bg_residues\
   -origin -0 \
   -o PHO_consensus_L10_N10_c2_matches_mkv2.tab

 feature-map -i PHO_consensus_L10_N10_c2_matches_mkv2.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o PHO_consensus_L10_N10_c2_matches_mkv2.png
```

10.3.3 P-values

One of the ***matrix-scan*** innovative features is the estimation of P-values for each match, including for higher-order Markov chain background models. (see below "Computing the theoretical score distribution of a PSSM" for more details on the calculation). For use with adaptative Markov models, it is necessary to provide a threshold on the score to limit computing time. With the rank return field, the matches are sorted by decreasing significativity, and we select only the 3 top scoring matches for each sequences.

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc -window 200 -markov 1 \
   -lth score 0 -return sites,limits,pval,rank -uth rank 3\
   -origin -0 \
   -o PHO_consensus_L10_N10_c2_matches_mkv1_pval.tab

 feature-map -i PHO_consensus_L10_N10_c2_matches_mkv1_pval.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o PHO_consensus_L10_N10_c2_matches_mkv1_pval.png
```

With non-adpatative background models, it is possible to select a threshold on the P-value.

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc -bginput -markov 0 \
   -uth pval 0.0001 -return sites,limits,pval \
   -origin -0 \
```

```
   -o PHO_consensus_L10_N10_c2_matches_mkv0_pval.tab

feature-map -i PHO_consensus_L10_N10_c2_matches_mkv0.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o PHO_consensus_L10_N10_c2_matches_mkv0_pval.png
```

10.3.4 Observed distribution of scores and site enrichment

Distribution of scores

matrix-scan can return the observed distribution of scores instead of each individual matches.

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc -bginput -markov 0 \
 -return distrib \
   -o PHO_consensus_L10_N10_c2_distrib_mkv0.tab
```

We can now draw an XY plot of this distribution.

```
## Draw the theoretical distribution
XYgraph -i PHO_consensus_L10_N10_c2_distrib_mkv0.tab \
  -xcol 2 -ycol 3 \
  -title1 'PHO matrix' \
  -title2 'Observed distribution of weight scores (Bernoulli model)' \
  -ymin 0 -yleg1 'Probability' \
  -xsize 800 -xleg1 'Weight score' \
  -format png  -lines -legend \
  -o PHO_consensus_L10_N10_c2_distrib_mkv0.png
```

Enrichment in sites

A typical use of the distribution of scores is to compare the number of occurences of a given match in the input sequence to the expected number of occurences in the background model. A Binomial test is run for each possible weight and a P-value is returned. This P-value represents the probability to observe at least the observed number of matches with a given weight by chance in a sequence of the same length as the input sequence. If the difference between the observed and expected occurences is significant, the matches with the given weight are considered as true positives. This approach estimates the over-representation of matches in the input sequences and can be used to retrieve significant matches based on the over-representation of these matches in the input sequence. In the following command, results are sorted by decreasing significativity on the overrepresentation of the given scores.

```
matrix-scan -m PHO_consensus_L10_N10_c2_matrix.tab \
   -i PHO_up800-noorf.wc -seq_format wc -bginput -markov 0 \
 -return occ_proba -lth occ_sig 0 -sort_distrib\
```

```
   -o PHO_consensus_L10_N10_c2_occ_proba_mkv0.tab

XYgraph -i PHO_consensus_L10_N10_c2_occ_proba_mkv0.tab \
  -xcol 2 -ycol 11 \
  -title1 'PHO matrix' \
  -title2 'Site enrichment (Bernoulli model)' \
  -ymin 0 -yleg1 'Over-representation significativity' \
  -xsize 800 -xleg1 'Weight score' \
  -format png  -lines -legend \
  -o PHO_consensus_L10_N10_c2_occ_proba_mkv0.png
```

10.3.5 Scanning sequences with multiple matrices

matrix-scan can scan sequences with multiple motifs at a time. There are 3 ways to provide several matrices : (i) by calling repeatedly -m option, (ii) by providing a file containing multiple sequences, (iii) bu using -mlist option to provide a list of matrices filenames.

We will now work with the motifs describing the binding sites of Met31p and Met4p transcription factors that are involved in the regulation of methionine metabolism in the yeast *Saccharomyces cerevisiae* (Gonze et al, 2005).

First, we will retrieve the promoter sequences of the methionine-responding genes of the following list with ***retrieve-seq*** (refer to the Chapter Retrieve sequences if necessary).

```
MET8
MET32
MET18
MET30
MET28
MET6
MET10
MET13
MET3
ECM17
MET14
MET1
MET17
VPS33
MET2
ZWF1
MET4
MET22
MET7
MET31
MET12
MET16
```

The sequences should be in a file named *MET_up800-noorf.fasta*.

Copy the following matrices describing the MET motifs in a file named *MET_matrices.tab*.

```
; MET4 matrix, from Gonze et al. (2005). Bioinformatics 21, 3490-500.
A |   7   9   0   0  16   0   1   0   0  11   6   9   6   1   8
C |   5   1   4  16   0  15   0   0   0   3   5   5   0   2   0
G |   4   4   1   0   0   0  15   0  16   0   3   0   0   2   0
T |   0   2  11   0   0   1   0  16   0   2   2   2  10  11   8
//
; MET31 matrix, from Gonze et al. (2005). Bioinformatics 21, 3490-500.
A |   3   6   9   6  14  18  16  18   2   0   0   0   1   3   8
C |   8   3   3   2   3   0   1   0  13   2   0   1   0   3   6
G |   4   3   4   8   0   0   1   0   2   0  17   1  17  11   1
T |   3   6   2   2   1   0   0   0   1  16   1  16   0   1   3
```

Individual matches

We can search for individual matches with the 2 matrices, with a threshold on the P-value. This threshold is particularly important when dealing with multiple matrices. Indeed, matrices may be very different in terms of size or information content, leading to very different score ranges. Putting a threshold on the score may thus return many false positive predictions for one of the matrices. By putting a threshold on the P-value, the threshold is coherent for all matrices and results are not biased by the differences in weight ranges. Here we only report the 3 top scoring sites for each matrix in each sequences with the option -rank_pm.

```
matrix-scan -m MET_matrices.tab  -consensus_name \
   -i MET_up800-noorf.fasta  -bginput -markov 0\
   -return sites,pval,rank,limits -uth pval 1e-04 -uth rank_pm 3 \
   -origin -0 \
  -o MET_3topsites_matches_mkv0.tab

 feature-map -i MET_3topsites_matches_mkv0.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o MET_3topsites_matches_mkv0.png
```

Sites enrichment

It is also possible to detect the most significant matches, as regards to their enrichment in the input sequence, compared to the background. For each matrix, the 2 most significant scores are returned by using the threshold -uth occ_sig_rank 2.

```
matrix-scan -m MET_matrices.tab  -consensus_name \
   -i MET_up800-noorf.fasta  -bginput -markov 0\
 -return occ_proba -uth occ_sig_rank 2 -sort_distrib\
   -o MET_2topscores_occ_mkv0.tab
```

10.3.6 Detecting Cis-Regulatory element Enriched Regions (CRER)

An extension of the concept of enrichement of sites in the input sequence is the detecttion of CRER, which are local over-representation of matches. The enrichment is calculated in windows of variable sizes, which may be overlapping. This concept is to be related to the search of homo- and hetero-typic modules, also known as CRM (Cis-Regulatory Modules). The rationale is that matches that are located in a region containing multiple predictions are more likely to be binding sites.

Two options are required for CRER search : a threshold on P-value and a maximum size for the CRER (typically between 150 and 300 bp).

```
matrix-scan -m MET_matrices.tab  -consensus_name \
   -i MET_up800-noorf.fasta \
   -bgfile ${RSAT}/data/genomes/Saccharomyces_cerevisiae/oligo-frequencies/2nt_upstream
   noorf_Saccharomyces_cerevisiae-ovlp-1str.freq.gz \
  -uth pval 0.0001 -origin 0 -decimals 1 \
    -return crer,normw,rank  \
      -uth crer_size 200 \
    -o MET_crer_mkv1.tab

  feature-map -i MET_crer_mkv1.tab \
  -format png -legend -scalebar -scalestep 50 -scorethick \
  -o MET_crer_mkv1.png
```

To view individual site matches over CRERs, we use -return sites,crer. The result file is only intended for display with ***feature-map*** since the columns for sites and crer return types are differents.

```
matrix-scan -m MET_matrices.tab  -consensus_name \
   -i MET_up800-noorf.fasta \
   -bgfile ${RSAT}/data/genomes/Saccharomyces_cerevisiae/oligo-frequencies/2nt_upstream
  -uth pval 0.0001 -origin 0 -decimals 1 \
    -return crer,sites,limits  \
      -uth crer_size 200 \
    -o MET_crer_sites_mkv1.tab

  feature-map -i MET_crer_sites_mkv1.tab \
  -format png -legend -scalebar -scorethick -symbol \
  -o MET_crer_sites_mkv1.png
```

10.4 Computing the theoretical score distribution of a PSSM

The program ***matrix-distrib*** returns the probability to observe a given score, on the basis of the theoretical model proposed by Staden (1989). For Bernoulli (Markov order 0) background

models, the distribution of scores is computed with the algorithm described by Bailey (Bioinformatics, 1999). For Markov background models with higher orders, we have extended this algorithm to take into account the dependencies between residues.

```
## Calculat the theoretical distribution of a PSSM
matrix-distrib  -v 1 -matrix_format consensus \
 -m PHO_consensus_L10_N10_c2.matrix  \
  -decimals 2 \
  -bgfile ${RSAT}/data/genomes/Saccharomyces_cerevisiae/oligo-frequencies/2nt_upstream-
  -o PHO_consensus_L10_N10_c2_distrib_theor.tab
```

Note that we restricted here the precision to 2 decimals. Indeed, for computational reasons, the computing time increases exponentially with the number of decimals. You can experiment this by changing the number of decimals, and you will see that the computation time increases drastically above 3 decimals.

In any case, for most practical applications, 2 decimals are more than enough for the detection of matches with matrices (the first decimal would even be sufficient).

We can now draw an XY plot of this distribution.

```
## Draw the theoretical distribution
XYgraph -i PHO_consensus_L10_N10_c2_distrib_theor.tab \
  -xcol 1 -ycol 2 \
  -title1 'PHO matrix' \
  -title2 'Theoretical distribution of weight scores' \
  -ymin 0 -yleg1 'Probability' \
  -xsize 800 -xleg1 'Weight score' \
  -format png  -lines -legend \
  -o PHO_consensus_L10_N10_c2_theor_distrib.png
```

The raw distribution is not very informative. A more interpretable information will be provided by the inverse cumulative distribution, which indicates, for each score, the probability to observe by chance a site with at least that score. This distribution can be considered as an estimation of the P-value, i.e. the risk of error if we consider as significant a site with a given score.

```
## Draw the theoretical distribution
XYgraph -i PHO_consensus_L10_N10_c2_distrib_theor.tab \
  -xcol 1 -ycol 2,4 \
  -title1 'PHO matrix' \
  -title2 'Theoretical distribution of weight scores' \
  -ymin 0 -ymax 1 -yleg1 'Probability' \
  -xsize 800 -xleg1 'Weight score' \
  -format png  -lines -legend \
  -o PHO_consensus_L10_N10_c2_Pval_distrib.png
```

As expected, the distribution of P-value rapidly decreases with increasing values of scores. for the purpose of deteecting binding sites, the most intersting part of this distribution is the

right tail, corresponding to high values of weight scores. We would like to display this tail with a higher detail, in order to distinguish the low P-values. A convenient way to do this is to use a logarithmic scale for the Y axis.

```
## Draw the theoretical distribution
XYgraph -i PHO_consensus_L10_N10_c2_distrib_theor.tab \
  -xcol 1 -ycol 2,4 \
  -title1 'PHO matrix' \
  -title2 'Theoretical distribution of weight scores' \
  -ymin 0 -ymax 1 -ylog -yleg1 'Probability' \
  -xsize 800 -xleg1 'Weight score' \
  -format png  -lines -legend \
  -o PHO_consensus_L10_N10_c2_Pval_distrib_Ylog.png
```

10.4.1 Estimating the quality of a PSSM

The program ***matrix-quality*** can be used to estimate the quality of a position-specific scoring matrix, by comparing the distribution of scores observed in a positive set (typically, the known binding sites for a transcriptiojn factor), and a negative set (for example, a set of randomly selected promoter sequences).

11 Evaluating the quality of position-specific scoring matrices

11.1 Prerequisite

This tutorial assumes that you alredy followed the tutorial on *Matrix based pattern matching*.

11.2 Why is important to estimate the quality of a matrix?

Position-specific scoring matrices are frequently used to predict transcription factor binding sites in genome sequences. At this point, following the tutorial, you have been able to built a matrix from a set of known binding sites for a transcription factor, and use it to detect new putative binding sites on different promoters, so the result is already there. But! What if there was a problem with the original set of biding sites? Where did they came from? Is the original experiment 100% reliable?

Matrices are generally built from a collection of experimentally characterized binding sites, databases as RegulonDB or TRANSFAC gather all the information reported in the literature about the interaction between Transcription Factors and their respective binding sites, on those databases you can get the sequences to built a matrix or download one or several available matrices for your favourite TF.

However, even if you built your own matrix or if you got it from a database, their reliability to predict novel binding sites is highly variable, due to the small number of training sites or the inappropriate choice of parameters when building the matrix.

There are some classical theoretical measures to describe some properties of matrices, but this measures may fail to predict the behaviour in real situations, cause they don't tell if the new detected putative sites might have a biological relevance.

So at the end in order to know if we can trust the sites we detected with pattern matching methodologies we need to:

- Know the composition of the matrix.
- Analyse the sites used to build the matrix.
- Analyze the behaviour of the matrix in a real situation.
- Analyze a negative control of the matrix and it's behavior in a real situation.

All this procedure can be done with the program *matrix-quality* and a correct tune of it's parameters. This is done combining theoretical and empirical score distributions to assess the predictive capability of matrices.

As a example we are going to use the matrix for the *E. coli* K12 transcriptional factor LexA, which is available at RegulonDB.

```
AC  ECK12_ECK120012770_LexA.20.cons
XX
ID  ECK12_ECK120012770_LexA.20.cons
XX
P0       A      T      C      G
1       12      3      3      5
2        0      1     22      0
3        0     23      0      0
4        0      0      0     23
5        1     14      2      6
6       12      5      3      3
7        1     15      5      2
8       12      5      2      4
9        6     15      2      0
10      10      6      5      2
11       7     11      5      0
12      13      5      2      3
13       4     12      4      3
14      12      2      7      2
15       0      0     23      0
16      23      0      0      0
17       0      0      0     23
18       1     13      8      1
19      12      6      2      3
20       6     13      2      2
21      11      8      3      1
XX
//
```

Please copy this matrix and paste it on a file. For the propose of the chapter the file will be named **LexA_matrix.transfac**.

11.3 How to estimate the theoretical distribution of a matrix?

As has been explained in the previous chapter *matrix-scan* gives a Weight Score (WS) to each site, and we usually take this weigth or statistics based on it to decide if the site is good or if it's not.

However, this WS can be misleading, because its range depends on the matrix width and information content. For example: The relevance of a site with a WS of 15 detected with a matrix having a WS range of -5 to 40 is not the same as if the range was -5 to 16.

So depending on the WS range you can decide whether a WS for a given site is relevant. One way to calculate all the possible Weight Scores that a matrix can give, is to generate an endless random sequence , and search for sites with *matrix-scan* but without any threshold, so it will return ALL the evaluated sites, which means a lot of sites with negative WS and few ones with positives WS. This way you'll see not only the highestt and lowest WS, but also you'll be able to see the frequency of each score.

As a little test we generate a long random sequence based on *E. coli* K12 genome composition.

```
random-seq -l 1000 -bg upstream -org Escherichia_coli_K_12_substr__MG1655_uid57779 -ol
 -o random_seq_E.coliK12.fas
```

And now we run search sites with our matrix using *matrix-scan* without any thresholds.

```
matrix-scan -m LexA_matrix.transfac -i random_seq_E.coliK12.fas \
 -bgfile 2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-1str.freq.gz
 -matrix_format transfac \
 -o LexA_bs_search_random_seq.tab
```

So now we can count how many times does a WS appers in a random enviroment just by chance, remeber the count will change a bit for each generated random sequence and the variation in the count will decreas as we increase sequence length.

But instead of doing this manually trying to simulate an infinite randome sequence and scan it, which will take a lot of time, we will use *matrix-distrib* and this program will calculate the number of times a score should appear in an endless random sequence, and oviusly this result contains as well the range for possible Weight Scores (WS).

First of all we will need to convert the matrix in to tab format.

```
convert-matrix  -i LexA_matrix.transfac \
  -from transfac -to tab \
  -return counts,parameters,consensus
  -o LexA_matrix.tab


matrix-distrib -m LexA_matrix.tab \
  -bgfile  2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-noov-2str.f
  -o LexA_matrix_distrib.tab
```

So this simulates a search for sites in an endless random sequence based on the genome of *E. coli* K12.

In this file you can see the frequency (probability) of finding each value of WSs, or in other words we have the *probaility distribution of weight scores* .

```
XYgraph -i LexA_matrix_distrib.tab -format png \
 -xcol 1 -ycol 2 \
 -o LexA_matrix_probability_distrib.png
```

Now we know the range of WS goes from -40 to 17.7, and in the graph showing the probability distribution of scores you can se the probability of having a positive score is low, and since the range goes up to 17 a WS of 15 for a site in the genome, seams to be a good score, at least in theory.

But this graph is only for one matrix, and is a matrix for one of the transcriptional factors with the most conserved binding sites, other matrices based in fewer and/or less conserved sites will have a different shape, e.g. a widder distribution.

In the output file from *matrix-distrib* we also have the inverse cumulative distribution of WS at column num. 4 so we can know how frequent (probable) is to find a WS of a given X value or higher, which is the definition of the **P-value**.

```
XYgraph -i LexA_matrix_distrib.tab -format png \
 -xcol 1 -ycol 2,4 \
 -o LexA_matrix_probability_distrib_invcum.png
```

But we want to be able to se the probabilities for the higher WSs, for this we will apply log to the y-axis.

```
XYgraph -i LexA_matrix_distrib.tab -format png \
 -xcol 1 -ycol 2,4 -ylog \
 -o LexA_matrix_probability_distrib_invcum_ylog.png
```

e.g. As you see in the graph to find a WS of 10 or higher than 10 has a P-value of aprox. $1e^5$, which seems excellent at first sight. However, with this cutoff, we would still expect about 42 false positives if we scan the whole genome of E. coli (4.2Mb) on both strands.

Remember each matrix has a specific theoretical distribution, depending on the particular frequency of each residue in each column.

11.4 How to compare the theoretical distribution with the scores of the known binding sites?

In order to estimate the capability of a matrix to distinguish bona fide binding sites from genome background, *matrix-quality* implements a method that relies on the combined analysis of theoretical and empirical score distributions in positive and negative control sets.

The sensitivity of a matrix is the fraction of correct sites detected above a score threshold. Sensitivity is defined as

$$Sn = TP/(TP+FN) \tag{11.1}$$

where TP is the number true positives (i.e. annotated sites with WS above a threshold), and FN is the number of false negatives (i.e. annotated sites scoring below that threshold).

The logic positive control should be the set of sequences that have been used to build the matri, if we scan this set with the matrix using *matrix-scan* and calculate the invers cumulative frequency of scores they should show a high scores distribution.

matrix-quality calculates the theoretical score distribution and also the distribution of scores on diferent set of sequences.

From RegulonDB we download the set of sites used in the aligment form which the matrix was generated.

```
matrix-quality -v 1 -m LexA_matrix.transfac \
 -seq matrix_sites LexA.fna  \
 -bgfile 2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-ovlp-2str.fre
 -o matrix-quality_tutorial \
 -matrix_format transfac
```

matrix-quality generates various files, we are going to describe them step by step in order to show how they should be interpretate.

Take a view on the graph file matrix-quality_tutorial**_score_distrib_compa.png**. The blue line in the graphs is the same theoretical distribution we saw in the previous chapter, now we can look the distribution of scores for the set of known binding sites, and we can see this distribution has an important number of positive scores.

However, this matrix is probably over-fitted to these particular sites, since each of them is in the alignment from which the matrix is derived. For an unbiased estimate of sensitivity, we would ideally need two separate collections of sites: one for build-ing the PSSM, another for testing it. Unfortunately, for most tran-scription factors, very few binding sites are known. In order to ensure an independent assessment whilst minimizing the loss of information, the program matrix-quality performs a Leave-One-Out (LOO) validation, iteratively discarding one annotated site, re-building the matrix, and scoring the left-out site with the new matrix. The program also discards multiple copies of identical sites, which would otherwise induce the same kind of bias.

The LOO curve (green) provides an unbiased estimate of the sensitivity of a matrix, and the difference with the matrix sites curve indicates the level of over-fitting to the training sites.

11.5 Distribution in full collections of promoters

Matrices are frequently used to predict transcription factor binding sites in genome sequences, for this what we want to know is the behaviour of the matrix in a real situation.

As a example we will take the complete set of upstream regions of the *E.coli* K12 genome.

```
retrieve-seq -org Escherichia_coli_K_12_substr__MG1655_uid57779 -tipe upstream \
 -all -feattype CDS -noorf  \
 -o Escherichia_coli_K_12_substr__MG1655_uid57779_upstream-noorf.fas
```

With *matrix-quality* we can have the distriution of WSs of the matrix in a given sequence set, and we will give thsi set to the program with the same command we used for the **matrix_sites** set.

```
matrix-quality -v 1 -m LexA_matrix.transfac \
 -matrix_format transfac \
 -seq matrix_sites LexA.fna  \
 -bgfile 2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-ovlp-2str.fre
 -o matrix-quality_tutorial \
 -seq allup Escherichia_coli_K_12_substr__MG1655_uid57779_upstream-noorf.fas
```

From the previous section we know now the range of WS we should expect from real sites we know the expected scores are the ones with less frequency, since this might be difficult to analyze on a normal scale the program gives the same graph with a y-log axis matrix-quality_tutorial_**score_distrib_compa_logy.png**

In this graph we can see the light blue line corresponding to the inverse distribution of scores from the *matrix-scan* search over the complete set of upstream regions from *E. coli* K12 genome. At higher weights the curves separate, revealing a small number of sites with a much higher score than expected by chance (WS>= 9), supposedly corresponding to *bona fide* binding sites (see previous section). The abrupt separation between the two curves results in a plateau-like shape in the high score range, suggesting that the matrix efficiently distinguishes binding sites from the background. Now we need a negative control to probe our statment.

11.6 Negative control with random sequences

An ideal negative control would be a set of sequences where the TF of interest does not bind. Unfortunately, experimental results of this type are generally not available. An alternative is to select a random set of promoters, but this could accidentally include some real binding sites. Another possibility is to generate random sequences using some background model (e.g. Markov chain).

For this we are going to simulate a set of *E. coli* K12 upstream regions using 3000 random sequences of length 2000.

```
random-seq -l 200 -n 3000 -bg upstream -org Escherichia_coli_K_12_substr__MG1655_uid577
 -o random_seq_upstream_E.coliK12.fas
```

and we will add this new set to the *matrix-quality* command.

```
matrix-quality -v 1 -m LexA_matrix.transfac \
 -matrix_format transfac \
 -seq matrix_sites LexA.fna  \
 -bgfile 2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-ovlp-2str.fre
 -o matrix-quality_tutorial \
 -seq allup Escherichia_coli_K_12_substr__MG1655_uid57779_upstream-noorf.fas \
 -seq random random_seq_upstream_E.coliK12.fas
```

However, nothing guarantees that Markov chains provide realistic models of biological sequences.

11.7 Negative controls with permuted matrices

To circumvent the common problems to obtain a negative control, *matrix-quality* supports an original type of negative controls by scanning input sequences with randomized matrices, obtained by permuting the columns of the original matrix. This presents the advantage of preserving important characteristics of the PSSM such as residue composition (sum of each row), number of sites (sum of any column), total information content, and even the complete theoretical score distribution (for Bernoulli models).

Now we are going to add a permutation instruction for each of our sequence sets, we will make 3 permutations of the matrix and scan with this 3 matrices the matrix_sites set, and we will make 5 permutatios for the other two sets.

```
matrix-quality -v 1 -m LexA_matrix.transfac \
 -matrix_format transfac \
 -seq matrix_sites LexA.fna  \
 -bgfile 2nt_upstream-noorf_Escherichia_coli_K_12_substr__MG1655_uid57779-ovlp-2str.fre
 -o matrix-quality_tutorial \
 -seq allup Escherichia_coli_K_12_substr__MG1655_uid57779_upstream-noorf.fas \
 -seq random random_seq_upstream_E.coliK12.fas \
 -perm 3 matrix_sites
 -perm 5 allup
 -perm 5 random
```

We scanned all the promoters of *E. coli* using 5 randomized versions of the matrix (in total, 5Mb of sequences were scanned on both strands). The cyan curve closely follows the blue curve for low scores (weight <= 7), without showing any separation at high scores. This confirms that the plateau observed for the original non-permuted matrix corresponds to sites specifically found in the genome by this matrix.

The column-permuted distribution can be considered an empirical estimate of the FPR. This distribution is however estimated from scanning a few Mb of sequences, and its precision is thereby limited. To combine the advantages of theoretical and empirical FPR curves, we propose the following strategy: (1) scan a representative set of biological sequences with column-permuted matrices; (2) if the results fit the theoretical distribution, use the latter to estimate the P-value of predicted sites.

11.8 ROC curves indicate the trade-off between sensitivity and false positive rate

We still have tow output figures we have not described yet.

The Receiver Operating Characteristic (ROC) curve is a standard representation of the trade-off between False Positive Rate (FPR) and sensitivity. You can see the ROC curve displayed for each distribution of scores in the figure matrix-quality_tutorial**_score_distrib_compa_roc.png**

However, the risk of false positive applies to every position of the scanned sequences. Even with an apparently low FPR, the actual number of FP can be very high when scanning a genome. For example, the E. coli promoters scanned on both strands represent more than

1 million scored positions, so that an FPR of 0.001 would return 1,159 FP on all E. coli promoters. Consequently, regular ROC curves are of no use for estimating the discriminatory power of a matrix. For the same reason, the Area Under the Curve (AUC), classically used to assess the quality of ROC curves, is ineffective. Indeed, this area is obtained by integrating the sensitiv-ity over the full range of FPR from 0 to 1, yet genome-wide predictions performed with an FPR of 90%, 50%, 10% or even 1% are not useful.

To emphasize the lower, more relevant, range of FPR, we draw ROC curves with a logarithmic abscissa (matrix-quality_tutorial_**score_distrib_compa_roc_xylog.png**), emphasizing the smaller FPR values. For example, for TrpR, we estimate that 70% sensitivity can be reached at a cost of 1 FP per Mb. Note that given the LOO procedure, our estimate of sensitivity is unbiased, but it is based only on five non-redundant sites, thus being of questionable robustness (it could change if new binding sites become available).

For the LexA matrix, built from 23 binding sites, the ROC curves shows a gradual increase: for a sensitivity of 50%, the expected FPR remains reasonably low ($FPR_{0.5} = 1.3x10^{-}5$), whereas collect-ing 90% of the sites would include almost 1FP per 100bp ($FPR_{0.9} = 8.3x10^{-}3$).

12 Generating random sequences

Th program ***random-seq*** allows to generate random sequences wih different random models.

It supports Bernoulli models (independence between successive residues) and Markov models of any order. Markov models are generally more suitable to represent biological sequences.

We will briely illustrate different ways to use this program.

12.1 Sequences with identically and independently distributed (IID) nucleotides

```
random-seq -l 200 -n 20 -o rand_L200_N20.fasta
```

We can now check th residue cmposition of this random sequence.

```
oligo-analysis -v 1 \
  -i rand_L200_N20.fasta \
  -l 1 -1str -return occ,freq \
  -o rand_L200_N20_1nt-1str.tab
```

12.2 Sequences with nucleotide-specific frequencies

In general, the residue composition of biological sequences is biased. We can impose residue-specific probabilities for the random sequence generation.

```
random-seq -l 200 -n 20 -a a:t 0.3 c:g 0.2 \
  -o rand_L200_N20_at30.fasta

oligo-analysis -v 1 \
  -i rand_L200_N20_at30.fasta \
  -l 1 -1str -return occ,freq \
  -o rand_L200_N20_at30_1nt-1str.tab
```

12.3 Markov chain-based random sequences

The random generator ***random-seq*** supports Markov chains of any order (as far as the corresponding ferquency table has previously been calculated). The Markov model is specified by indicating an oligonucleotide frequency table. The table of oligonucleotides of length k is automatically converted in a transition table of order $m = k - 1$ duing the execution of ***random-seq***.

```
random-seq -l 200 -n 20 \
  -expfreq $RSAT/data/genomes/Escherichia_coli_K_12_substr__MG1655_uid57779/oligo
  -o rand_L200_N20_mkv2.fasta
```

A simpler way to obtain organism-specific Markov models is to use the options *-bg* and *-org* of ***random-seq***.

```
## This command generates random sequences with a Markov model of order 2,
## calibrated on all the non-coding upstream sequences of E.coli.
random-seq -l 200 -n 20 \
  -org Escherichia_coli_K_12_substr__MG1655_uid57779 -bg upstream-noorf -ol 3 \
  -o rand_L200_N20_mkv2.fasta
```

13 Pattern comparisons

TO BE WRITTEN

13.1 Comparing patterns with patterns

`compare-patterns`

13.2 Comparing discovered patterns wirth a library of TF-binding consensus

Let us suppose that we dispose of a collection of experimentally characterized binding consensus for the organism of interest, in a file called *known_consensus.pat*.

```
compare-patterns -v 1 \
  -file1 dyads.tab \
  -file2 RegulonDB_sites.tab \
  -return weight,offset,strand,length,Pval,Eval_p,sig_p,Eval_f,sig_f,id,seq \
  -2str -1th weight 6 \
  -o dyads_vs_RegulonDB.tab
```

14 Comparing classes, sets and clusters

TO BE WRITTEN

15 Comparative genomics

15.1 Genome-wise comparison of protein sequences

In this section, we explain how to use the program ***genome-blast***, which runs the sequence similarity search program ***BLAST*** to detect significant similarities between all the proteins of a set of genomes.

This operation can take time, and the result tables occupy a considerable amount of space on the hard disk. For this reason, the ***RSAT***distribution does thus not include the complete comparison of all genomes against all other ones, but is restricted to some model genomes (*Escherchia coli K12* versus all bacteria, *Saccharomyces cerevisiae* against all Fungi, ...).

Depending on your organism of interest, you might wish to perform additional comparisons for your own purpose. In this section, we explain how to compute the similiraty tables between a query organism (e.g. *Mycoplasma pneumoniae*) and a reference taxon (e.g. all Bacteria).

In order to install the tables of similarities between gene products in ***RSAT***, you need writing permissions in the directory $RSAT/data$. If this is not the case, ask your system administrator to do it for you.

15.1.1 Applying genome-blast between two genomes

As a first test, we will use ***genome-blast*** to compare all the gene products (proteins) of a query organism (e.g. *Mycoplasma pneumoniae*) against all the gene products of a reference organism (e.g. *Bacillus subtilis*).

This protocol assumes that the two organisms are already installed on your ***RSAT***site, as explained in the installation guide.

We will perform in two steps:

1. Use the program ***formatdb*** (which is part of the ***BLAST*** distribution) to create a BLAST-formatted structure (the "database") with all proteins of the reference organism (*Bacillus subtilis*).

2. Use the program ***blastall*** (part of the ***BLAST*** distribution) to detect similarities between each protein of the query organism (*Mycoplasma pneumoniae*) and the reference organism.

Formatting the BLAST DB

This DB formatting step is very efficient, it should be completed in a few seconds.

```
genome-blast -v 1 -task formatdb \
   -q Mycoplasma_pneumoniae \
   -db Bacillus_subtilis
```

The result is found in the data directory containing *Bacillus subtilis*. A new directory *blastdb* has been created, which contains the BLAST-formatted database with all the proteins of the reference organism.

```
ls -ltr $RSAT/data/genomes/Bacillus_subtilis/blastdb
```

These are binary files, that you should in principle not open as such.

Searching similarities

The program ***blastall*** compares all the sequences of an input set against all the sequences of a database (the one we just created above). The program ***genome-blast*** generates the appropriate ***blastall*** command to find the BLAST database directory, and query it with the proteins of the query organism.

```
genome-blast -v 1 -task blastall  \
   -q Mycoplasma_pneumoniae \
    -db Bacillus_subtilis
```

This task takes a bit less that one minute for *Mycoplasma pneumoniae* (because we chose a very small genomes), and can take around 10 minutes for medium-sized bacterial genomes (4,000 genes).

Note that the ***blastall*** command is written in the verbosity message. If you have specific reasons to customize this command, you can adapt it to apply different parameters.

Searching reciprocal similarities

One classical orthology criterion (which is not perfect but has practical advantages) is to select the bidirectional best hits as candidate orthologs.

For this, we need to run the reciprocal blast, i.e. using *Bacillus subtilis* as query organism, and *Mycoplasma genitalium* as reference organism.

Note that you can run the two BLAST commands (***formatdb*** and ***blastall***) in a single shot, by specifying multiple tasks for ***genome-blast***.

```
genome-blast -v 1 -task formatdb,blastall  \
    -q Bacillus_subtilis \
    -db Mycoplasma_pneumoniae
```

We can now perform a quick test: select the bidirectional best hit (*-rank 1*) for the gene *NP_109706.1*.

```
get-orthologs -q NP_109706.1 -uth rank 1 -return all \
    -org Mycoplasma_pneumoniae -taxon Bacillus_subtilis
```

15.1.2 Applying genome-blast between a genome and a taxon

Generally, we want to compare a query organism to all the organisms of a given taxon (the *reference taxon*). This can be done with the option *-dbtaxon*.

As an example, we will BLAST all the proteins of *Mycoplasma pneumoniae* against all the proteins of each species of *Mollicutes*.

We can also use the option *-reciprocal* to activate the reciprocal search: BLAST all gene products of each bacteria of the reference taxon *Mollicutes* against those of the query organism *Mycoplasma pneumoniae*.

```
genome-blast -v 1 -task formatdb,blastall -reciprocal \
    -q Mycoplasma_pneumoniae \
    -dbtaxon Mollicutes
```

We can now retrieve the orthologs of a *Mycoplasma pneumoniae* gene (e.g. *NP_109706.1*) in all Mollicutes.

```
get-orthologs -q NP_109706.1 -uth rank 1 -return all \
    -org Mycoplasma_pneumoniae -taxon Mollicutes
```

15.2 Getting putative homologs, orthologs and paralogs

In this section, I will explain how to use the program ***get-orthologs***. This program takes as input one or several query genes belonging to a given organism (the *reference organism*), and return the genes whose product (peptidic sequence) show significant similarities with the products of the query genes. The primary usage of ***get-orthologs*** is thus to return lists of similar genes, not specialy orthologs. Additional criteria can be imposed to infer orthology. In particular, one of the most common criterion is to select *bidirectional best hits (BBH)*. This can be achieved by imposing the rank 1 with the option *-uth rank 1*.

We will illustrate the concept by retrieving the genes whose product is similar to the protein LexA of *Escherichia coli K12*, in all the Enterobacteriales. We will then refine the query to extract putative orthologs.

15.2.1 Getting genes by similarities

```
get-orthologs -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
  -taxon Enterobacteriales \
  -q lexA -o lexA_all_hits_Enterobacteriales.tab
```

The result file is a list of all the Enterobacteriales genes whose product shows some similarity with the LexA protein from E.coli K12.

```
...
#ref_id ref_org query
Sde_1787        Saccharophagus_degradans_2-40   b4043
CPS_0237        Colwellia_psychrerythraea_34H   b4043
```

```
CPS_2683        Colwellia_psychrerythraea_34H   b4043
CPS_1635        Colwellia_psychrerythraea_34H   b4043
IL0262  Idiomarina_loihiensis_L2TR      b4043
...
c5014   Escherichia_coli_CFT073 b4043
c3190   Escherichia_coli_CFT073 b4043
b4043   Escherichia_coli_K_12_substr__MG1655_uid57779    b4043
...
```

Each similarity is reported by the ID of the gene, the organism to which is belong, and the ID of the query gene. In this case, the third column contains the same ID on all lines: b4043, which is the ID of the gene lexA in *Escherichia coli K12*. It seems thus poorly informative, but this column becomes useful when several queries are submitted simultaneously.

15.2.2 Obtaining information on the BLAST hits

The program ***get-orthologs*** allows to return additional information on the hits. The list of supported return fields is obtained by calling the command with the option *-help*. For example, we can ask to return the percentage of identity, the alignment length, the E-value and the rank of each hit.

```
get-orthologs -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
  -taxon Enterobacteriales \
  -q lexA -return ident,ali_len,e_value,rank \
  -o lexA_all_hits_Enterobacteriales.tab \
```

Which gives the following result:

```
...
#ref_id ref_org query   ident   ali_len e_value rank
Sde_1787        Saccharophagus_degradans_2-40   b4043   65.33   199     1e-68   1
CPS_0237        Colwellia_psychrerythraea_34H   b4043   65.69   204     6e-75   1
CPS_2683        Colwellia_psychrerythraea_34H   b4043   33.94   109     1e-10   2
CPS_1635        Colwellia_psychrerythraea_34H   b4043   34.12   85      1e-06   3
IL0262  Idiomarina_loihiensis_L2TR      b4043   66.83   202     1e-75   1
...
c5014   Escherichia_coli_CFT073 b4043   100.00  202     2e-111  1
c3190   Escherichia_coli_CFT073 b4043   43.33   90      2e-14   2
b4043   Escherichia_coli_K_12_substr__MG1655_uid57779    b4043   100.00  202     2e-111  1
...
```

Not surprisingly, the answer includes the self-match of lexA (ID b4043) in *Escherichia coli K12*, with 100% of identify.

15.2.3 Selecting bidirectional best hits

We can see that the output contains several matches per genome. For instance, there are 3 matches in *Colwellia psychrerythraea 34H*. If we assume that these similarities reflect homologies, the result contains thus a combination of paralogs and orthologs.

The simplest criterion to select ortholog is that of *bidirectional best hit (BBH)*. We can select BBH by imposing an upper threshold on the rank, with the option *-uth*.

```
get-orthologs -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
  -taxon Enterobacteriales \
  -q lexA -return ident,ali_len,e_value,rank \
  -uth rank 1 \
  -o lexA_best_hits_Enterobacteriales_bbh.tab
```

The result has now been reduced to admit at most one hit per genome.

```
...
#ref_id ref_org query   ident   ali_len e_value rank
Sde_1787        Saccharophagus_degradans_2-40   b4043   65.33   199     1e-68   1
CPS_0237        Colwellia_psychrerythraea_34H   b4043   65.69   204     6e-75   1
IL0262  Idiomarina_loihiensis_L2TR      b4043   66.83   202     1e-75   1
...
c5014   Escherichia_coli_CFT073 b4043   100.00  202     2e-111  1
b4043   Escherichia_coli_K_12_substr__MG1655_uid57779    b4043   100.00  202     2e-111  1
...
```

15.2.4 Selecting hits with more stringent criteria

It is well known that the sole criterion of BBH is not sufficient to infer orthology between two genes. In particular, there is a risk to obtain irrelevant matches, due to partial matches between a protein and some spurious domains. To avoid this, we can add a constraint on the percentage of identity (min 30%), and on the alignment length (min 50 aa). These limits are somewhat arbitrary, we use them to illustrate the principe, and leave to each user the responsibility to choose the criteria that she/he considers as relevant. Finally, we will use a more stringent threshold on E-value than the default one, by imposing an upper threshold of 1e-10.

```
## Note that or this test we suppress the BBH constraint (-uth rank 1)
get-orthologs -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
  -taxon Enterobacteriales \
  -q lexA \
  -return ident,ali_len,e_value,rank \
  -lth ident 30 -lth ali_len 50 -uth e_value 1e-10 \
  -o lexA_filtered_hits_Enterobacteriales_id30_len50_eval-10.tab
```

We can now combine the constrains above with the criterion of BBH.

```
## Note that or this test we include the BBH constraint (-uth rank 1)
get-orthologs -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
  -taxon Enterobacteriales \
  -q lexA \
  -return ident,ali_len,e_value,rank \
  -lth ident 30 -lth ali_len 50  -uth e_value 1e-10 \
  -uth rank 1 \
  -o lexA_BBH_Enterobacteriales_bbh_id30_len50_eval-10.tab
```

As expected, the number of selected hits is reduced by adding these constraints. In Sept 2006, we obtained the following number of hits for lexA in Enterobacteriales.

- 122 hits without any constraint;

- 107 hits with contrains on ident,ali_len and e_value;
- 69 hits with the constraint of BBH;
- 69 hits with the combined constraint of BBH, at least 30% identity and an alignment over more than 50 aminoacids, and an E-value <= 1.e-10.

Actually, in the particular case of *lexA*, the BBH constraint already filtered out the spurious matches, but inother cases they can be useful.

15.2.5 Restricting the number of reference organisms

The decrease of sequencing cost impulsed the multiplication of genome sequencing projects. In 2015, some bacterial species are represented by several hundreds of strains in the EnsemblGenomes database (e.g. *Escherichia coli*). A conseuence is that the comparative genomics analyses can become quite heavy. A more importance drawback is that some taxonomic branches are over-represented relative to other ones.

To reduce this problem, we added an option *-depth* to the taxonomic tools.

```
## Note that or this test we include the BBH constraint (-uth rank 1)
get-orthologs -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
  -taxon Enterobacteriales \
  -q lexA -o lexA_orthologs_Enterobacteriales_bbh_id30_len50_eval-10.tab \
  -return ident,ali_len,e_value,rank \
  -lth ident 30 -lth ali_len 50  -uth e_value 1e-10 \
  -uth rank 1
```

15.3 Retrieving sequences for multiple organisms

The program ***retrieve-seq-multigenome*** can be used to retrieve sequences for a group of genes belonging to different organisms.This program takes as input a file with two columns. Each row of this file specifies one query gene.

1. The first column contains the name or identifier of the gene (exactly as for the single-genome program ***retrieve-seq***).
2. The second column indicates the organism to which the gne belongs.

The output of ***get-orthologs*** can thus directly be used as input for ***retrieve-seq-multigenome***.

```
retrieve-seq-multigenome -noorf \
  -i lexA_orthologs_Enterobacteriales_bbh_id30_len50_eval-10.tab \
  -o lexA_orthologs_Enterobacteriales_up-noorf.fasta
\end{footnotesize}
```

15.4 Detection of phylogenetic footprints

TO BE WRITTEN

```
dyad-analysis  -v 1 \
  -i lexA_orthologs_Enterobacteriales_up-noorf.fasta \
  -sort -2str -noov -lth occ 1 -lth occ_sig 0 \
  -return occ,freq,proba,rank \
  -l 3 -spacing 0-20 -bg monads \
  -o lexA_orthologs_Enterobacteriales_up-noorf_dyads-2str-noov.tab
```

15.5 Phylogenetic profiles

The notion of *phylogenetic profile* was introduced by Pellegrini et al. (1999). They identified putative orthologs for all the genes of *Escherichia coli K12* in all the complete genomes available at that time, and built a table with one row per gene, one column per genome. Each cell of this table indicates if an ortholog of the considered gene (row) has been identified in the considered genome (column). Pellegrini et al. (1999) showed that genes having similar phylogenetic profiles are generally involved in common biological processes. The analysis of phylogenetic profiles is thus a powerful way to identify functional grouping in completely sequenced genomes.

The program ***get-orthologs*** can be used to obtain the phylogenetic profiles. The principle is to submit the complete list of protein-coding genes of the query organism. We process in two steps :

1. With ***get-orthologs***, we can identify the putative ortholgos for all the genes of the query organism, using the criterion of *bidirectional best hit (BBH)*. This generate a large table with one row per pair of putative orthologs.

2. We then use ***convert-classes*** to convert the ortholog table into profiles (one row per gene, one column per genome).

We will illustrate this by calculating the phylogenetic profiles of all the genes from *Saccharomyces cerevisiae* across all the Fungi. We use a level of verbosity of 2, in order to get information about the progress of the calculations.

```
## Identify all the putative orthologs (BBH)
get-orthologs -v 2 \
  -i $RSAT/data/genomes/Saccharomyces_cerevisiae/genome/cds.tab  \
  -org Saccharomyces_cerevisiae \
  -taxon Fungi \
  -uth rank 1 -lth ali_len 50 -lth ident 30 -uth e_value 1e-10 \
  -return e_value,bit_sc,ident,ali_len \
  -o Saccharomyces_cerevisiae_vs_Fungi_bbh.tab

## Convert ortholog table into a profile table
```

```
## with the IDs of the putative orthologs
convert-classes -v 2 \
  -i Saccharomyces_cerevisiae_vs_Fungi_bbh.tab  \
  -from tab -to profiles \
  -ccol 2 -mcol 3 -scol 1 -null "<NA>" \
  -o Saccharomyces_cerevisiae_vs_Fungi_phyloprofiles_ids.tab
```

The resulting table indicates the identifier of the ortholog genes. The option *-null* was used to specify that the string <NA> should be used to indicate the absence of putative orhtolog.

Another option would be to obtain a "quantitative" profile, where each cell indicates the E-value of the match between the two orthologs. This can be done by specifying a different score column with the option *-scol* of ***convert-classes***.

```
## Convert ortholog table into a profile table
## with the E-value of the putative orthologs
convert-classes -v 2 \
  -i Saccharomyces_cerevisiae_vs_Fungi_bbh.tab  \
  -from tab -to profiles \
  -ccol 2 -mcol 3 -scol 4 -null "<NA>" \
  -o Saccharomyces_cerevisiae_vs_Fungi_phyloprofiles_evalue.tab
```

15.6 Detecting pairs of genes with similar phylogenetic profiles

In the previous section, we generated tables indicating the phylogenetic profiles of each gene from *Saccharomyces cerevisiae*. This table contains one row per gene, and one column per fungal genome.

We will now use the program ***compare-profiles*** to compare each gene profile to each other, to select the pairs of genes with significantly similar profiles. The problem is of course to choose our criterion of similarity between two gene profiles.

15.6.1 Comparing binary profiles with *compare-profiles*

For the binary profiles, the most relevant statistics is the *hypergeometric significance*.

```
## Compare the binary phylogenetic profiles
## using the hypergeometric significance
compare-profiles -v 2 \
  -i Saccharomyces_cerevisiae_vs_Fungi_phyloprofiles_evalue.tab \
  -lth AB 1 -lth sig 0 \
  -return counts,jaccard,hyper,entropy \
  -o Saccharomyces_cerevisiae_vs_Fungi_phyloprof_gene_pairs.tab
```

In the previous commands, we set the verbosity to 2, in order to keep track the progress of the task. Actually, the processing can take a few minuts, it is probably the good moment for a coffee break.

15.6.2 Comparing binary profiles with *compare-classes*

Another way to compare the phylogenetic profiles is to directly analyze with ***compare-classes*** the table of orthology (previously obtained from ***get-orthologs***).

This is just another way of considering the same problem: in order to compare genes *A* and *B*, we will consider as a first class (*Q*) the set of genomes in which gene *A* is present, and as a second class (*R*) the set of genomes in which gene *B* is present. We will then calculate the intersection between these two classes, and assess the significance of this intersection, given the total number of genomes.

Thus, ***compare-classes*** will calculate the hypergeometric statistics, exactly in the same way as ***compare-profiles***.

```
## Convert the orthology into "classes", where each class (second column)
## corresponds to a gene from Saccharomyces cerevisiae, and indicates
## the set of genomes (first column) in which this gene is present.
convert-classes -from tab -to tab -mcol 2 -ccol 3 -scol 5 \
  -i Saccharomyces_cerevisiae_vs_Fungi_bbh.tab \
  -o Saccharomyces_cerevisiae_vs_Fungi_bbh_classes.tab

## Compare the classes to detect significant overlaps
compare-classes -v 3 \
  -i Saccharomyces_cerevisiae_vs_Fungi_bbh_classes.tab \
  -lth QR 1 -lth sig 0 -sort sig -sc 3 \
  -return occ,proba,dotprod,jac_sim,rank \
  -o phyloprof_gene_pairs.tab
```

16 Automated analysis of multiple gene clusters

The main interest of using ***RSAT*** from the shell is that it allows to automatize the analysis of multiple data sets. The different programs of the package can be combined in different ways to apply an extensive analysis of your data. A typical example is the analysis of clusters obtained from gene expression data.

When a few tens or hundreds of gene clusters have to be analyzed, it becomes impossible to manage it manually. ***RSAT*** includes a program, ***multiple-family-analysis***, which takes as input a file with the composition of gene clusters (the *cluster file*), and automatically performs the following analyses on each cluster :

directory management: the results are stored in a separate directory for each cluster. Directories are automatically created during the execution, and bear the name of the cluster.

sequence retrieval: upstream sequences are retrieved and stored in fasta format

sequence purging: upstream sequences are purged (with the program ***purge-sequences*** to remove redundant fragments. Purged sequences are then used for pattern discovery, and non-purged sequences for pattern matching.

oligonucleotide analysis: the program ***oligo-analysis*** is used to detect over-represented oligonucleotides. ***dna-pattern*** and ***feature-map*** are used to draw a feature map of the significant patterns.

dyad analysis: the program ***dyad-analysis*** is used to detect over-represented oligonucleotides. ***dna-pattern*** and ***feature-map*** are used to draw a feature map of the significant patterns.

other pattern discovery programs: several matrix-based pattern discovery programs developed by other teams can be managed by ***multiple-family-analysis***. These programs have to be installed separately they are not part of the ***RSAT*** distribution).

feature map drawing: The patterns discovered by the different programs are matched against the upsteram sequences, and the result is displayed as a feature map.

synthesis of the results: A synthetic table is generated (in HTML format) to facilitate the analysis of the results, and the navigation between result files.

result export: The results can be exported to tab-delimited files, which can then automatically be loaded in a relational database (mySQL, PostgreSQL or Oracle).

In addition to this cluster-per-cluster analysis, results are summarized in two format.

synthetic table A HTML table is generated with one row per cluster, and a summary of the results (gene composition, significant oligonucleotides, significant dyads). This table contains links to the feature maps, making it easy to browse the results.

sql table The list of significant patterns detected in all the cluster are compiled in a single result table (a tab-delimited text file), with one row per pattern and cluster, and one column per criterion (pattern type, occurrences, significance, ...).

The program also automatically exports SQL scripts which allow to create the appropriate table in a relational database management system (RDBMS) and load the data.

16.1 Input format

The input format is a tab-delimited text file with two columns, providing respectively :

1. gene identifier or name
2. cluster name

An example of cluster file is displayed in Table 16.1. This file describes 3 yeast regulons, each responding to some specific environmental condition: the NIT family contains 7 genes expressed under nitrogen depletion, the PHO family 5 genes expressed under phosphate stress, and the MET family 11 genes expressed when methionine is absent fom the culture medium.

Beware: the columns must be separate by tabulations, spaces are not valid separators.

Note that genes can be specified either by their name (as for the NIT and PHO families in Table 16.1), or by their systematic identifier (MET family in Table 16.1).

16.2 Example of utilization

Let us assume that the file displayed in Table 16.1 has been saved under the name *test.fam*. The following command will automatically perform all the analyses.

```
multiple-family-analysis  -i test.fam -v 1 \
    -org Saccharomyces_cerevisiae \
    -2str -noorf -noov \
    -task upstream,purge,oligos,oligo_maps,synthesis,sql,clean \
    -outdir test_fam_results
```

Once the analysis is finished, you can open the folder *synthetic_tables* with aweb browser and follow the links.

; gene	cluster
DAL5	NIT
GAP1	NIT
MEP1	NIT
MEP2	NIT
MEP3	NIT
PUT4	NIT
DAL80	NIT
PHO5	PHO
PHO11	PHO
PHO8	PHO
PHO84	PHO
PHO81	PHO
YDR502C	MET
YER091C	MET
YHL036W	MET
YIL046W	MET
YJR010W	MET
YKL001C	MET
YKR069W	MET
YLR180W	MET
YLR303W	MET
YNL241C	MET
YNL277W	MET

Table 16.1: Example of family file.

16.3 Loading the results in a relational database

The results were exported in tab-delimited text files in the directory *test_fam_results/sql_export/*. This directory contains 3 files and one subdirectory :

```
Family_genes.tab
Family.tab
Pattern.tab
sql_scripts/
```

The subdirectory *sql_scripts* contains several SQL scripts for creating tables in a relational database management system (*RDBMS*), loading data into these tables, and dropping these tables when you don't need them anymore.

```
family_genes_table_load.ctl
family.mk
family_table_create.sql
family_table_drop.sql
family_table_load.ctl
makefile
pattern.mk
pattern_table_create.sql
pattern_table_drop.sql
pattern_table_load.ctl
```

The file *makefile* allows you to automatically create the tables and load the data in two operations.

```
make create MYSQL='mysql -u [your login] -D multifam'
make load MYSQL='mysql -u [your login] -D multifam'
```

This requires the existence of a database space 'multifam' in your *RDBMS*. If you are not familar with relational databases, you probably need to contact your system administrator to create this space for you.

16.4 Comparing programs

The program ***multiple-family-analysis*** allows you to compare the results obtained by different pattern discovery programs. Two of these programs are part of the ***RSAT*** distribution : ***oligo-analysis*** and ***dyad-analysis***. The other programs have been developed by other teams, and can be downloaded from their original site. The command below assumes that these programs were installed and included in your path.

```
multiple-family-analysis  -i test.fam -v 1 \
    -org Saccharomyces_cerevisiae \
    -2str -noorf -noov \
    -task upstream,purge,oligos,oligo_maps \
    -task dyads,dyad_maps,consensus,gibbs \
    -task meme,synthesis,sql,clean \
    -outdir test_fam_results
```

Note that you can define multipe tasks either with a single call to the option *-task*, or by insering iteratively the option in the command line.

16.5 The negative control: analyzing random gene selections

An essential quality of pattern discovery programs is their ability to return a negative answer when there are no specific patterns in a sequence set.

The program ***random-genes*** allows to select random sets of genes, which can then be used by ***multiple-family-analysis*** to check the rate of false positive of pattern discovery programs.

The simplest way to use random-gene is to ask a set of *n* genes:

```
random-genes -org Saccharomyces_cerevisiae -n 10
```

You can also use the option *-r* to select *r* distinct sets of *n* genes.

```
random-genes -org Saccharomyces_cerevisiae -n 10 -r 5
```

Another possibility is to specify a template family file with the option *-fam*.

```
random-genes -org Saccharomyces_cerevisiae -fam test.fam
```

This will return a family file with the same number of gene family as in the input file (*test.fam*). Each output family will contain the same number of gene as the corresponding input family. This option provides thus a very convenient way to generate a negative control of exactly the same size as the real family file.

16.6 Analyzing a large set of regulons

To get a better feeling about the potentialities of the different pattern discovery programs, you can analyze the collection of regulons collected by Nicolas Simonis (2004), which is available at:

http://rsat.ulb.ac.be/rsat/data/published_data/Simonis_Bioinformatics_2004/

17 Utilities

17.1 gene-info

gene-info allows you to get information on one or several genes, given a series of query words. Queries are matched against gene identifiers and gene names. Imperfect matches can be specified by using regular expressions.

For example, to get all info about the yeast gene GAT1:

```
gene-info -org Saccharomyces_cerevisiae -q GAT1
```

And to get all the purine genes from *Escherichia coli*, type:

```
gene-info -org Escherichia_coli_K_12_substr__MG1655_uid57779 -q 'pur.*'
```

Note the use of quotes, which is necessary whenever the query contains a *.

You can also combine several queries on the same command line, by using reiteratively the -q option:

```
gene-info -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
    -q 'met.*' -q 'thr.*' -q 'lys.*'
```

17.2 On-the-fly compression/uncompression

All programs from ***RSAT*** support automatic compression and uncompression of gzip files. This can be very convenient when dealing with big sequence files.

To compress the result of a query, simply add the extension `.gz` to the output file name.

```
retrieve-seq -all -org Saccharomyces_cerevisiae \
        -from -1 -to -200 -noorf -format fasta \
        -o all_up200.fa.gz
```

The result file is a compressed archive. Check its size with the command

```
ls -l
```

Uncompress the file with the command

```
gunzip all_up200.fa.gz
```

The file has now lost the `.gz` extension. Check the size of the uncompressed file.

Recompress the file with the command

```
gzip all_up200.fa
```

Similarly, you can directly use a compressed archive as input for ***RSAT***, it will be uncompressed on the fly, without occupying space on the hard drive. For example :

```
dna-pattern -i all_up200.fa.gz -p GATAAG -c -th 3
```

will return all the genes having at least three occurrences of the motif `GATAAG` in their 200 bp upstream region.

18 Downloading genomes

RSAT includes a series of tools to install and maintain the latest version of genomes.

The most convenient way to add support for one or several organisms on your machine is to use the programs ***supported-organisms*** and ***download-organism***.

Beware, the complete data required for a single genome may occupy several hundreds of Mb, because ***RSAT*** not only stores the genome sequence, but also the oligonucleotide frequency tables used to estimate background models, and the tables of BLAST hits used to get orthologs for comparative genomics. If you want to install many genomes on your computer, you should thus reserve a sufficient amount of space.

18.1 Original data sources

Genomes supported on ***RSAT*** were obtained from various sources.

Genomes can be installed either from the ***RSAT*** web site, or from their original sources.

- NCBI/Genbank (`ftp://ftp.ncbi.nih.gov/genomes/`) was the primary source for installing genomes on ***RSAT***. Genomes are downloaded from the ftp site and installed locally on the ***RSAT*** server by parsing the .gbk files.
- The EBI genome directory (`ftp://ftp.ebi.ac.uk/pub/databases/genomes/Eukaryota/`) contains supplementary genomes, which can be downloaded and installed on the ***RSAT*** server by parsing files in embl format.
- UCSC (`http://genome.ucsc.edu/`) for the multi-genome alingment files (multiz) used by ***peak-footprints***.
- Since 2008, ENSEMBL (`http://www.ensembl.org/`) genomes are supported by special tools (***retrieve-ensembl-seq***, ***supported-organisms-ensembl***), that remotely address queries to the Ensembl database.
- Since 2013, genomes can be downloaded and installed on ***RSAT*** servers, using the tool ***install-ensembl-genome***. Once installed, ensembl genomes can be queried with the same tools as the other genomes installed on ***RSAT*** servers (***retrieve-seq***, ***gene-info***, ...).

Other genomes can also be found on the web site of a diversity of genome-sequencing centers.

18.2 Requirement : wget

The download of genomes relies on the application ***wget***, which is part of linux distribution[1].

[1] For Linux: `http://www.gnu.org/software/wget/`; for Mac OSX `http://download.cnet.com/Wget/3000-18506_4-128268.html`

wget is a "web aspirator", which allows to download whole directories from ftp and http sites. You can check if the program is installed on your machine.

```
wget --help
```

This command should return the help pages for ***wget***. If you obtain an error message ("command not found"), you need to ask your system administrator to install it.

18.3 Importing organisms from the *RSAT*main server

The simplest way to install organisms on our ***RSAT***site is to download the RSAT-formatted files from the web server. For this, you can use a web aspirator (for example the program ***wget***).

Beware, the full installation (including Mammals) requires a large disk space (several dozens of Gb). You should better start installting a small genome and test it before processing to the full installation. We illustrate the approach with the genome of our preferred model organism: the yeast *Saccharomyces cerevisiae*.

18.3.1 Obtaining the list of organisms supported on the *RSAT*server

By default, the program ***supported-organisms*** returns the list of organisms supported on your local ***RSAT***installation. You can however use the option *-server* to obtain the list of organisms supported on a remote server.

```
supported-organisms-server
```

The command can be refined by restricting the list to a given taxon of interest.

```
supported-organisms-server -taxon Fungi
```

You can also ask additional information, for example the date of the last update and the source of each genome.

```
supported-organisms-server -taxon Fungi -return last_update,source,ID
```

18.3.2 Importing a single organism

The command

```
download-organism
```

allows you to download one or several organisms.

Beware, the complete data for a single genome may occupy several tens of Megabytes (Bacterial genomes) or a few Gigabases (Mammalian). Downloading tenomes thus requires a fast Internet connection, and may take time. If possible, please download genomes during the night (European time).

As a first step, we recommend to download the genome of the yeast *Saccharomyces cerevisiae*, since this is the model organism used in our tutorials.

```
download-organism -v 1 -org Saccharomyces_cerevisiae
```

In principle, the download should start immediately. *Beware*, the data volume to be downloaded is important, because the genome comes together with extra files (blast hits with other genoems, oligonucleotide and dyad frequencies). Depending on the network bandwidth, the download of a genome may take several minutes or tens of minutes. RSAT separated the genomes in different servers depending on the taxon, the option -server can be used to specify which server to use.

After the task is completed, you can check if the configuration file has been correctly updated by typing the command.

```
supported-organisms
```

In principle, the following information should be displayed on your terminal.

```
Saccharomyces_cerevisiae
```

You can also add parameters to get specific information on the supported organisms.

```
supported-organisms -return ID,last_update
```

18.3.3 Importing a few selected organisms

The program ***download-organism*** can be launched with a list of organisms by using iteratively the option *-org*.

```
download-organism -v 1 -org Escherichia_coli_K_12_substr__MG1655_uid57779 \
   -org Mycoplasma_genitalium_G37_uid57707
```

Note: genome names may change with time, since genome centers are occasionally adding new suffixes for genomes. The organism names indicated after the option *-org* should belong to the list of supported organisms collected with the command ***supported-organisms -server***.

18.3.4 Importing all the organisms from a given taxon

For comparative genomics, it is convenient to install on your server all the organisms of a given taxon. For this, you can simply use the option *-taxon* of ***download-organism***.

Before doing this, it is wise to check the number of genomes that belong to this taxon on the server.

```
## Get the list of organisms belonging to the order "Enterobacteriales" on the server
supported-organisms -taxon Enterobacteriales -server

## Count the number of organisms
supported-organisms -taxon Enterobacteriales -server | wc -l
```

In Dec 2013, there are 203 Enterobacteriales supported on the ***RSAT***server. Before starting the download, you should check two things:

1. Has your network a sufficient bandwidth to ensure the transfer in a reasonable time ?
2. Do you have enough free space on your hard drive to store all those genomes ?

If the answer to both questions is “yes”, you can start the download.

```
download-organism -v 1 -taxon Enterobacteriales
```

19 Installing additional genomes on your machine

The easiest way to install genomes on your machine is to download them from the main ***RSAT*** server, as indicated in the Chapter "Downloading genomes" (Chap. 18 of the installation guide).

In some cases, you may however wish to install a genome by yourself, because this genome is not supported on the main ***RSAT*** server. For this, you can use the programs that we use to install new genomes on the main ***RSAT*** server.

19.1 Adding support for Ensembl genomes

In addition to the genomes imported and maintained on your local ***RSAT*** server, the program ***retrieve-ensembl-seq*** allows you to retrieve sequences for any organism supported in the Ensembl database (http://ensembl.org).

For this, you first need to install the Bioperl and Ensembl Perl libraries (see section **??**).

19.1.1 Handling genomes from Ensembl

The first step to work with Ensembl genomes is to check the list of organisms currently supported on their Web server.

```
supported-organisms-ensembl
```

You can then get more precise information about a given organism (build, chromosomes) with the command ***ensembl-org-info***.

```
ensembl-org-info -org Drosophila_melanogaster
```

Sequences can be retrieved from Ensembl with the command ***retrieve-ensembl-seq***.

You can for example retrieve the 2kb sequence upstream of the transcription start site of the gene *PAX6* of the mouse.

```
retrieve-ensembl-seq.pl -org Mus_musculus -q PAX6 \
  -type upstream -feattype mrna -from -2000 -to -1 -nogene -rm \
  -alltranscripts -uniqseqs
```

Options

- *-type upstream* specifies that we want to collect the sequences located upstream of the gene (more procisely, upstream of the mRNA).
- *-feattype mrna* indicates that the reference for computing coordinates is the mRNA. Since we collect upstream sequences, the 5'most position of the mRNA has coordinate 0, and upstream

sequences have negative coordinates. Note that many genes are annotated with multiple RNAs for different reasons (alternative splicing, alternative transcription start sites). By default, the program will return the sequences upstream of each mRNA annotated for the query gene.

- *-nogene* clip the sequences to avoid overlapping the next upstream gene.
- *-rm* repeat masking (important for pattern discovery). Repetitive sequences are replaced by `N` characters.

19.2 Installing genomes and variations from *EnsEMBL*

??

RSAT includes a series of programs to download and install genomes from Ensembl.

1. ***install-ensembl-genome*** is a wrapper enabling to autmoatize the download (genome sequences, features, variations) and configuration tasks.

2. ***download-ensembl-genome*** downloads the genomics sequences and converts them in the raw format required for ***RSAT***.

3. ***download-ensembl-features*** downloads tab-delimited text files describing genomic features (transcripts, CDS, genes, . . .).

4. ***download-ensembl-variations*** downloads tab-delimited text files describing genomic variations (polymorphism).

19.2.1 Installing genomes from Ensembl

The program ***install-ensembl-genome*** manages all the required steps to download and install a genome (sequence, features, and optionally variations) from Ensembl to ***RSAT***.

It performs the following tasks:

1. The option *-task genome* runs the program ***download-ensembl-genome*** to download the complete genomic sequence of a given organism from the ***EnsEMBL*** Web site, and formats it according to ***RSAT*** requirements (conversion from the original fasta sequence file to one file per chromosome, in raw format).

2. The option *-task features* runs ***download-ensembl-features*** to download the positions and descriptions of genomic features (genes, CDS, mRNAs, ...).

3. Optionally, when the option *-task variations* is activated, run ***download-ensembl-variations*** to download the description of genomic variations (polymorphism). Note that variations are supported only for a subset of genomes.

4. Update ***RSAT*** configuration (*-task config*) to make the genome available to other programs in the current ***RSAT*** site.

5. Run the additional tasks (*-task install*) required to have a fully functional genome on the local ***RSAT*** site: compute genomic statisics (intergenic sizes, . . .) and background models (oligonucleotide and dyad frequencies).

6. With the option *-available_species*, the program returns the list species available on the Ensembl server, together with their status of availability for the 3 data types (genome sequence, features, variations). When this option is called, the program does not install any genome.

The detailed description of the program and the list of options can be obtained with the option *-help*.

```
## Get the description of the program + all options
install-ensembl-genome -help
```

Getting the list of available genomes

Before installing a genome, it is generally a good idea to know which genomes are available. For this, use the option *-available_species*.

```
## Retrieve the list of supported species on EnsEMBL
install-ensembl-genome -v 1  -available_species \
  -o available_species_ensembl.tab

## Read the result file
more available_species_ensembl.tab
```

Note: inter-individual variations are available for a subset only of the genomes available in ***EnsEMBL***. The option *-available_species* indicates, for each species, the availability (genome, features, variations). Obviously, the programs to analyse regulatory variations (***convert-variations***, ***retrieve-variation-seq***, ***variation-scan***) are working only for the genomes documented with variations.

Installing a genome from Ensembl

We can now download and install the complete genomic sequence for the species of our choice. For the sake of space and time economy, we will use a small genome for this manual: the budding yeast *Saccharomyces cerevisiae*.

Beware: some installation steps take a lot of time. For large genomes (e.g. Vertebrate organisms), the full installation can thus take several hours. This should in principle not be a big issue, since installing a genome is not a daily task, but it is worth knowing that the whole process requires a continuous connection during several hours.

```
## Install the genome sequences for a selected organism
install-ensembl-genome -v 2 -species Saccharomyces_cerevisiae
```

This command will automatically run all the installation tasks described above, except the installation of variations (see Section 19.2.3).

19.2.2 Installing genomes from EnsemblGenomes

The historical ***EnsEMBL*** project [1] was focused on vertebrate genomes + a few model organisms (*Saccharomyces cerevisiae*, *Drosophila melanogaster*, ...).

A more recent project called ***EnsemblGenomes***[2] extends the ***EnsEMBL*** project to a wider taxonomic range (in Oct 2014, there are >15,000 genomes available at EnsemblGenomes, where as Ensembl only provides 69 genomes).

The program ***install-ensembl-genome*** supports the installation of genomes from ***EnsEMBL*** as well as ***EnsemblGenomes***. By default, it opens a connection to the historical ***EnsEMBL*** database, but the option *-db ensemblgenomes* enables to install genomes from the new project ***EnsemblGenomes***.

```
## Get the list of available species from the extended project
## EnsemblGenomes
install-ensembl-genome -v 2 -available_species -db ensemblgenomes \
   -o available_species_at_EnsemblGenome.txt
```

[1] http://www.ensembl.org/
[2] http://ensemblgenomes.org/

You can then identify your genome of interest in the file *available_species_at_EnsemblGenome.txt*, and start the installation (don't forget the option *-db ensemblgenomes*.

```
## Install Escherichia coli (strain K12 MG1665) from EnsemblGenomes
install-ensembl-genome -v 2 -db ensemblgenomes \
   -species Escherichia_coli_str_k_12_substr_mg1655
```

19.2.3 Downloading variations

The program ***download-ensembl-variations*** downloads variations from the ***EnsEMBL*** Web site, and installs it on the local ***RSAT*** site.

This program relies on ***wget***, which must be installed beforehand on your computer.

```
## Retrieve the list of supported species in the EnsEMBL variation database
download-ensembl-variations -v 1  -available_species
```

We can now download all the variations available for the yeast.

```
## Download all variations for a selected organism on your server
download-ensembl-variations -v 1 -species Saccharomyces_cerevisiae
```

Variation files are stored in a specific subfolder for the specified organism.

```
## Check the content of the variation directory for the yeast
make -f makefiles/variation-scan_demo.mk \
   SPECIES=Saccharomyces_cerevisiae ASSEMBLY=R64-1-1 \
   variation_stats
```

This command will indicate the location of the variation directory on your ***RSAT*** server, and count the number of lines for each variation file (there is one separate file per chromosome or contig).

19.3 Importing genomes from NCBI BioProject

Tne BioProject database hosts the results of genome sequencing and transcriptome projects.

1. Open a connection to the Bioproject Web site
   http://www.ncbi.nlm.nih.gov/bioproject

2. Enter a query to select the organism of interest. E.g. `ostreococcus+tauri[orgn]`

3. If the organism genome has been sequenced, you should see a title "Genome Sequencing Projects" in the record. Find the relevant project and open the link.

   For example, for *Ostreococcus tauri*, the most relevant project is PRJNA51609
   http://www.ncbi.nlm.nih.gov/bioproject/51609

4. Take note of the *Accesssion* of this genome project: since a same organism might have been sequenced several times, it will be useful to include this Accession in the suffix of the name of the file fo be downloaded.

5. On the left side of the page, under *Related information*, click the link "*Nucleotide genomic data*". This will display a list of Genbank entries (one per contig).

6. **Important:** we recommend to create one separate directory per organism, and to name this directory according to the organism name followed by the genome project Accession number. For example, for *Ostreococcus tauri*, the folder name would be *Ostreococcus_tauri_PRJNA51609*.

   This convention will facilitate the further steps of installation, in particular the parsing of genbank-formatted files with the program ***parse-genbank.pl***.

7. In the top corner of the page, click on the *Send to* link and activate the following options.

   Send to > File > Genbank full > Create file

   Save the file in the organism-specific directory described in the previous step.

8. You can now parse the genome with the program ***parse-genbank.pl***. Note that ***parse-genbank.pl*** expects files with extension .gbk or .gbk.gz (as in the NCBI genome repository), whereas the BioProject genome appends the extension *.gb*. You should thus use the option *-ext gb*.

```
parse-genbank.pl -v 2 -i Ostreococcus_tauri_PRJNA51609 -ext gb
```

   After parsing, run the program ***install-organism*** with the following parameters (adapt organism name).

```
install-organism  -v 2 -org Ostreococcus_tauri_PRJNA51609 \
  -task config,phylogeny,start_stop,allup,seq_len_distrib \
  -task genome_segments,upstream_freq,oligos,dyads,protein_freq
```

19.4 Importing multi-genome alignment files from UCSC

19.4.1 Warning: disk space requirement

The UCSC multi-genome alignment files occupy a huge disk space. The alignments of 30 vertebrates onto the mouse genome (mm9 multiz30) requires 70Gb. If you intend to offer support for multi-genome alignments, it might be safe to acquire a separate hard drive for this data.

The complete data set available at UCSC in April 2012 occupies 1Tb in compressed form, and probably 7 times more once uncompressed. For efficiency reasons, it is necessary to uncompress these files for using them with the indexing system of ***peak-footprints***.

19.4.2 Checking supported genomes at UCSC

As a first step, we will check the list of supported genomes at the UCSC Genome Browser.

```
supported-organisms-ucsc
```

Each genome is assocaited with a short identifier, followed by a description. For example, several versions of the mouse genome are currently available.

```
mm10 Mouse Dec. 2011 (GRCm38/mm10) Genome at UCSC
mm9 Mouse July 2007 (NCBI37/mm9) Genome at UCSC
mm8 Mouse Feb. 2006 (NCBI36/mm8) Genome at UCSC
mm7 Mouse Aug. 2005 (NCBI35/mm7) Genome at UCSC
```

19.4.3 Downloading multiz files from UCSC

Multi-genome alignments at UCSC are generated with the program ***multiz***, which produces files in a custom text format called *maf* for Multi-Alignment file.

We show hereafter the command to download the mm9 version of the mouse genome, and install it in the proper directory for ***peak-footprints*** (*$RSAT/data/UCSC_multiz*).

```
download-ucsc-multiz -v 1 -org mm9
```

Beware: the download of all the multi-species alignments can take several hours for one genome.

The program will create the sub-directory for the mm9 genome, download the coresponding compressed multiz files (files with extension *.maf.gz*), uncompress them, and call ***peak-footprint*** with specific options in order to create a position index, which will be further used for fast retrieval of the conserved regions under peaks.

19.5 Installing genomes from NCBI/Genbank files

In the section 18, we saw that the genomes installed on the main ***RSAT*** server can easily be installed on your local site. In some cases, you would like to install additional genomes, which are not published yet, or which are not supported on the main ***RSAT*** server.

If your genomes are available in Genbank (files .gbk) or EMBL (files .embl) format, this can be done without too much effort, using the installation tools of ***RSAT***.

The parsing of genomes from their original data sources is however more tricky than the synchronization from the ***RSAT*** server, so this procedure should be used only if you need to install a genome that is not yet supported.

If this is not your case, you can skip the rest of this section.

19.5.1 Organization of the genome files

In order for a genome to be supported, ***RSAT*** needs to find at least the following files.

1. organism description
2. genome sequences
3. feature tables (CDS, mRNA, ...)
4. lists of names/synonyms

From these files, a set of additional installation steps will be done by ***RSAT*** programs in order to compute the frequencies of oligonucleotides and dyads in upstream sequences.

If you installed ***RSAT*** as specified above, you can have a look at the organization of a supported genome, for example the yeast *Saccharomyces cerevisiae*.

```
cd ${RSAT}/public_html/data/genomes/Saccharomyces_cerevisiae/genome
ls -l
```

As you see, the folder *genome* contains the sequence files and the tables describing the organism and its features (CDSs, mRNAs, ...). The ***RSAT*** parser exports tables for all the feature types found in the original genbank file. There are thus a lot of distinct files, but you should not worry too much, for the two following reasons:

1. ***RSAT*** only requires a subset of these files (basically, those describing organisms, CDSs, mRNAs, rRNAs and tRNAs).
2. All these files can be generated automatically by ***RSAT*** parsers.

Organism description

The description of the organism is given in two separate files.

```
cd ${RSAT}/public_html/data/genomes/Saccharomyces_cerevisiae/genome
ls -l organism*.tab

more organism.tab

more organism_names.tab
```

1. *organism.tab* specifies the ID of the organism and its taxonomy. The ID of an organism is the TAXID defined by the NCBI taxonomical database, and its taxonomy is usually parsed from the .gbk files (but yo may need to specify it yourself in case it would be missing in your own data files).
2. *organism_name.tab* indicates the name of the organism.

Genome sequence

A genome sequence is composed of one or more contigs. A contig is a contigous sequence, resulting from the assembly of short sequence fragments obtained during the sequencing. When a genome is completely sequenced and assembled, each chomosome comes as a single contig.

In ***RSAT***, the genome sequence is specified as one separate file per contig (chromosome) sequence. Each sequence file must be in raw format (i.e. a text file containing the sequence without any space or carriage return).

In addition, the genome directory contains one file indicating the list of the contig (chromosome) files.

```
cd $RSAT/data/genomes/Saccharomyces_cerevisiae/genome/

## The list of sequence files
cat contigs.txt

## The sequence files
ls -l *.raw
```

Feature table

The *genome* directory also contains a set of feature tables giving the basic information about gene locations. Several feature types (CDS, mRNA, tRNA, rRNA) can be specified in separate files (*cds.tab*, *mrna.tab*, *trna.tab*, *rrna.tab*).

Each feature table is a tab-delimited text file, with one row per feature (cds, mrna, ...) and one column per parameter. The following information is expected to be found.

1. Identifier
2. Feature type (e.g. ORF, tRNA, ...)
3. Name
4. Chromosome. This must correspond to one of the sequence identifiers from the fasta file.
5. Left limit
6. Right limit
7. Strand (D for direct, R for reverse complemet)
8. Description. A one-sentence description of the gene function.

```
## The feature table
head -30 cds.tab
```

Feature names/synonyms

Some genes can have several names (synonyms), which are specified in separate tables.

1. ID. This must be one identifier found in the feature table
2. Synonym
3. Name priority (primary or alternate)

```
## View the first row of the file specifying gene names/synonyms
head -30 cds_names.tab
```

Multiple synonyms can be given for a gene, by adding several lines with the same ID in the first column.

```
## An example of yeast genes with multiple names
grep YFL021W cds_names.tab
```

19.5.2 Downloading genomes from NCBI/Genbank

The normal way to install an organism in ***RSAT***is to download the complete genome files from the NCBI [3], and to parse it with the program ***parse-genbank.pl***.

However, rather than downloading genomes directly from the NCBI site, we will obtain them from a mirror [4] which presents two advantages?

- Genome files are compressed (gzipped), which strongly reduces the transfer and storage volume.
- This mirror can be queried by ***rsync***, which facilitates the updates (with the appropriate options, ***rsync*** will only download the files which are newer on the server than on your computer).

RSATincludes a makefile to download genomes from different sources. We provide hereafter a protocol to create a download directory in your account, and download genomes in this directory. Beware, genomes require a lot of disk space, especially for those of higher organisms. To avoid filling up your hard drive, we illustrate the protocol with the smallest procaryote genome to date: *Mycoplasma genitamlium*.

```
## Creating a directory for downloading genomes in your home account
cd $RSAT
mkdir -p downloads
cd downloads

## Creating a link to the makefile which allows you to dowload genomes
ln -s $RSAT/makefiles/downloads.mk ./makefile
```

We will now download a small genome from NCBI/Genbank.

[3] ftp://ftp.ncbi.nih.gov/genomes/
[4] bio-mirror.net/biomirror/ncbigenomes/

```
## Downloading one directory from NCBI Genbank
cd $RSAT/downloads/
make one_genbank_dir NCBI_DIR=Bacteria/Mycoplasma_genitalium
```

We can now check the list of files that have been downloaded.

```
## Downloading one directory from NCBI Genbank
cd $RSAT/downloads/
ls -l ftp.ncbi.nih.gov/genomes/Bacteria/Mycoplasma_genitalium/
```

RSAT parsers only use the files with extension *.gbk.gz*.

You can also adapt the commande to download (for example) all the Fungal genomes in a single run.

```
## Downloading one directory from NCBI Genbank
cd $RSAT/downloads/
make one_ncbi_dir NCBI_DIR=Fungi
```

You can do the same for Bacteria, of for the whole NCBI genome repository, but this requires sveral Gb of free disck space.

19.5.3 Parsing a genome from NCBI/Genbank

The program ***parse-genbank.pl*** extract genome information (sequence, gene location, ...) from Genbank flat files, and exports the result in a set of tab-delimited files.

```
parse-genbank.pl -v 1 \
    -i $RSAT/downloads/ftp.ncbi.nih.gov/genomes/Bacteria/Mycoplasma_genitalium
```

19.5.4 Parsing a genome from the Broad institute (MIT)

The website http://www.broad.mit.edu/ offers a large collection of genomes that are not available on the NCBI website. We wrote a specific parser for the Broad files.

To this, download the following files for the organism of interest : the supercontig file, the protein sequences and the annotation file in the GTF format.

These files contain sometimes too much information that shoud be removed.

This is an example of the beginning of the fasta file containing the protein traduction. In this file, we should remove everything that follows the protein name.

```
>LELG_00001 | Lodderomyces elongisporus hypothetical protein (translation) (1085 aa)
MKYDTAAQLSLINPQTLKGLPIKPFPLSQPVFVQGVNNDTKAITQGVFLDVTVHFISLPA
ILYLHEQIPVGQVLLGLPFQDAHKLSIGFTDDGDKRELRFRANGNIHKFPIRYDGDSNYH
IDSFPTVQVSQTVVIPPLSEMLRPAFTGSRASEDDIRYFVDQCAEVSDVFYIKGGDPGRL
```

This is an example of the beginning of the fasta file containing the contigs. In this file, we should remove everything that follows the name of the contig.

```
>supercontig_1.1 of Lodderomyces elongisporus
AAGAGCATCGGGCAAATGATGTTTTTCAGTCCATCAATGTGATGGATCTGATAGTTGAAG
GTCCTGATGAAGTTCAACCATTTGTAAACCCGATTTACAAAGTGTGAATTATCGAGTGGT
TTATTCATCACAAGGACAAGAGCTTTGTTGGTTGACAGAGATGTTTTGCAGAAAGCCCTT
AAGGATGGTATTGCCTTGTTCAAGAAGAAACCAGTTGTTACTGAAGTAAATCTGACGACC
```

This is an example of the beginning of the GTF file containing the contigs annotation. We should rename the contig name so that it corresponds to the fasta file of contig. To this, we will remove the text in the name of the contig (only keep the supercontig number) and add a prefix.

```
supercont1.1%20of%20Lodderomyces%20elongisporus LE1_FINAL_GENECALL start_codon
322 324 . + 0 gene_id "LELG_00001"; transcript_id "LELT_00001";
supercont1.1%20of%20Lodderomyces%20elongisporus LE1_FINAL_GENECALL stop_codon
3574 3576 . + 0 gene_id "LELG_00001"; transcript_id "LELT_00001";
supercont1.1%20of%20Lodderomyces%20elongisporus LE1_FINAL_GENECALL exon 322
3576 . + . gene_id "LELG_00001"; transcript_id "LELT_00001";
supercont1.1%20of%20Lodderomyces%20elongisporus LE1_FINAL_GENECALL CDS 322 3573
. + 0 gene_id "LELG_00001"; transcript_id "LELT_00001";
```

We use the parse ***parse-broad-mit***.

```
parse-broad-mit.pl -taxid 36914 -org Lodderomyces_elongisporus \
   -nuc_seq lodderomyces_elongisporus_1_supercontigs.fasta \
   -gtf lodderomyces_elongisporus_1_transcripts.gtf \
   -gtf_remove 'supercont' \
   -gtf_remove '%20of%20Lodderomyces%20elongisporus' \
   -contig_prefix LELG_ -nuc_remove supercontig_ \
   -nuc_remove ' of Lodderomyces elongisporus' \
   -aa lodderomyces_elongisporus_1_proteins.fasta -aa_remove ' .*'
```

This will create the raw files, the feature files and the protein sequence file.

19.5.5 Updating the configuration file

After having parsed the genome, you need to perform one additional operation in order for ***RSAT*** to be aware of the new organism: update the configuration file.

```
install-organism -v 1 -org Mycoplasma_genitalium -task config

## Check the last lines of the configuration file
tail -15 $RSAT/data/supported_organisms.pl
```

From now on, the genome is considered as supported on your local ***RSAT*** site. You can check this with the command ***supported-organisms***.

19.5.6 Checking the start and stop codon composition

Once the organism is found in your configuration, you need to check whether sequences are retrieved properly. A good test for this is to retrieve all the start codons, and check whether they are made of the expected codons (mainly ATG, plus some alternative start codons like GTG or TTG for bacteria).

The script ***install-organism*** allows you to perform some additional steps for checking the conformity of the newly installed genome. For example, we will compute the frequencies of all the start and stop codons, i order to check that gene locations were corectly parsed.

```
install-organism -v 1 -org Mycoplasma_genitalium -task start_stop

ls -l $RSAT/data/genomes/Mycoplasma_genitalium/genome/*start*

ls -l $RSAT/data/genomes/Mycoplasma_genitalium/genome/*stop*
```

The stop codons should be TAA, TAG or TGA, for any organism. For eucaryotes, all start codons should be ATG. For some procaryotes, alternative start codons (GTG, TGG) are found with some genome-specific frequency.

```
cd $RSAT/data/genomes/Mycoplasma_genitalium/genome/

## A file containing all the start codons
more Mycoplasma_genitalium_start_codons.wc

## A file with trinucleotide frequencies in all start codons
more Mycoplasma_genitalium_start_codon_frequencies

## A file containing all the stop codons
more Mycoplasma_genitalium_stop_codons.wc

## A file with trinucleotide frequencies in all stop codons
more Mycoplasma_genitalium_stop_codon_frequencies
```

19.5.7 Calibrating oligonucleotide and dyad frequencies with *install-organisms*

The programs ***oligo-analysis*** and ***dyad-analysis*** require calibrated frequencies for the background models. These frequencies are calculated automatically with ***install-organism***.

```
install-organism -v 1 -org Debaryomyces_hansenii \
    -task allup,oligos,dyads,upstream_freq,protein_freq
```

Warning: this task may require several hours of computation, depending on the genome size. For the ***RSAT*** server, we use a PC cluster to regularly install and update genomes. As the task *allup*, is a prerequisite for the computation of all oligonucleotide and dyad frequencies, it should be run directly on the main server before computing oligo and dyad frequencies on the nodes of the cluster. We will thus proceed in two steps. Note that this requires a PC cluster and a proper configuration of the batch management program.

```
## Retrieve all upstream sequences
## Executed directly on the server
install-organism -v 1 -org Debaryomyces_hansenii \
    -task allup

## Launch a batch queue for computing all oligo and dyad frequencies
## Executed on the nodes of a cluster
install-organism -v 1 -org Debaryomyces_hansenii \
    -task oligos,dyads,upstream_freq,protein_freq -batch
```

19.5.8 Installing a genome in your own account

In some cases, you might want to install a genome in your own account rather than in the ***RSAT*** folder, in order to be able to analyze this genome before putting it in public access.

In this chapter, we explain how to add support for an organism on your local configuration of ***RSAT***. This assumes that you have the complete sequence of a genome, and a table describing the predicted location of genes.

First, prepare a directory where you will store the data for your organism. For example:

```
mkdir -p $HOME/rsat-add/data/Mygenus_myspecies/
```

One you have this information, start the program ***install-organism***. You will be asked to enter the information needed for genome installation.

Updating your local configuration

- Modify the local config file
- You need to define an environment variable called RSA_LOCAL_CONFIG, containing the full path of the local config file.

19.6 Installing genomes from EMBL files

RSAT also includes a script ***parse-embl.pl*** to parse genomes from EMBL files. However, for practicaly reasons we generally parse genomes from the NCBI genome repository. Thus, unless you have a specific reason to parse EMBL files, you can skip this section.

The program ***parse-embl.pl*** reads flat files in EMBL format, and exports genome sequences and features (CDS, tRNA, ...) in different files.

As an example, we can parse a yeast genome sequenced by the "Genolevures" project [5].

Let us assume that you want to parse the genome of the species *Debaryomyces hansenii*.

Before parsing, you need to download the files in your account,

- Create a directory for storing the EMBL files. The last level of the directory should be the name of the organism, where spaces are replaced by underscores. Let us assume that you store them in the directory *$RSAT/downloads/Debaryomyces_hansenii*.
- Download all the EMBL file for the selected organism. Save each name under its original name (the contig ID), followed by the extension `.embl`)

We will check the content of this directory.

```
ls -1 $RSAT/downloads/Debaryomyces_hansenii
```

On my computer, it gives the following result

```
CR382133.embl
CR382134.embl
CR382135.embl
CR382136.embl
CR382137.embl
CR382138.embl
CR382139.embl
```

[5] http://natchaug.labri.u-bordeaux.fr/Genolevures/download.php

The following instruction will parse this genome.

```
parse-embl.pl -v 1 -i  $RSAT/downloads/Debaryomyces_hansenii
```

If you do not specify the output directory, a directory is automatically created by combining the current date and the organism name. The verbose messages will indicate you the path of this directory, something like *$HOME/parsed_data/embl/20050309/Debaryomyces_hanseni*.

You can now perform all the steps above (updating the config file, installing oligo- and dyad frequencies, ...) as for genomes parsed from NCBI.

Installing a genome in the main *RSAT* directory

Once the genome has been parsed, the simplest way to make it available for all the users is to install it in the ***RSAT*** genome directory. You can already check the genomes installed in this directory.

```
ls -1 $RSAT/data/genomes/
```

There is one subdirectory per organism. For example, the yeast data is in *$RSAT/data/genomes/Saccharomyces_cerevisiae/*. This directory is further subdivided in folders: *genome* and *oligo-frequencies*.

We will now create a directory to store data about Debaryomyces_hansenii, and transfer the newly parsed genome in this directory.

```
## Create the directory
mkdir -p $RSAT/data/genomes/Debaryomyces_hansenii/genome

## Transfer the data in this directory
mv $HOME/parsed_data/embl/20050309/Debaryomyces_hanseni/* \
  $RSAT/data/genomes/Debaryomyces_hansenii/genome

## Check the transfer
ls -ltr $RSAT/data/genomes/Debaryomyces_hansenii/genome
```

20 Regulatory variations (rSNPs and insertion/deletions)

Jérémy Delerce and Jacques van Helden

This chapter presents a series of tools to analyze the impact of genetic polymorphism (inter-individual variations) on the binding of transcription factors.

It combines a series of tools to

1. download genomes, features, and variations from the ***EnsEMBL*** database and install them on the local ***RSAT*** server;
2. obtain sequences of variants, exended on both sides;
3. scan these variant sequences with position-specific scoring matrices (PSSM) in order to detect variations that may affect transcription factor binding.

20.1 Requirements

The functionalities described below require a connection to the ***EnsEMBL*** database, via their Perl API[1].

In principle, the Ensembl Perl modules should have been installed together with ***RSAT*** (see ***RSAT*** installation guide for details).

20.2 Getting variation sequences

Before detecting regulatory variations, the first step is to obtain the genomic sequences of these variations. For this, variations must have been installed on your server by the ***RSAT*** administrator (the installation protocol is explained in Section 19.2.3).

To check if variations have been installed for your genome of interest, you can proceed in the following way.

20.2.1 Checking the installation of variations on your *RSAT* server

Since the databases about polymorphic variations are growing exponentially, the default genome installation does not include variations. The program ***install-ensembl-genome*** includes the option *-task variations*, which downloads all variations from the Ensembl database, and exports them in one file per chromosome or contig.

[1] API: application programmatic interface

1. Check which versions and assemblies of your organism of interest have been installed on ***RSAT***.
2. Check if variations have been installed for these versions/assemblies.
3. Count the number of variations per chromosome.

Let us assume that we are interested by the Human genome. We want to see if our ***RSAT*** admin installed it, and, if so, to check which versions or assemblies are available locally.

```
supported-organisms -taxon Homo
```

In our case, we get the following answer:

```
Homo_sapiens_GRCh37
Homo_sapiens_GRCh38
```

Variations are installed in the organism-specific folder of the genome directory.

```
ls ${RSAT}/public_html/data/genomes/Homo_sapiens_GRCh38/variations
```

Variations are stored in tab-delimited text files, with one separate file per chromosome. The demo makefile includes a target to check the size of variation files (count the number of lines per file).

```
## Check the content of the variation directory for the yeast
make -f ${RSAT}/makefiles/variation-scan\_demo.mk \
   SPECIES=Homo_sapiens ASSEMBLY=GRCh38 \
   variation_stats
```

This command will indicate the location of the variation directory on your ***RSAT*** server, and count the number of lines for each variation file (there is one separate file per chromosome or contig).

```
Statistics about variations
        SPECIES         Homo_sapiens
        ASSEMBLY        GRCh38
        ORG             Homo_sapiens_GRCh38
        VARIATION_DIR   /data/rsat/public_html/data/genomes/Homo_sapiens_GRCh38/variati
Number of lines per variation file
... (we skip some lines)
   4027724 4.tab
   4134079 3.tab
   4666045 1.tab
   4984715 2.tab
  60448850 total
```

20.3 File formats for variations

Several formats have been defined to store inforamtion about genetic variations.

To ensure compatibility between the different sources of information and ***RSAT*** requirements, the program ***convert-variation*** ensures inter-conversions betwneen different formats. The program includes an on-line help which provides more information about the different formats.

```
convert-variations -h
```

We will treat an example of conversion from the GVF to varbed format. We prepared some demo cases in the file *makefiles/variation-scan_demo.mk*.

```
make -f makefiles/variation-scan_demo.mk
```

TO BE WRITTEN

20.4 Detecting regulatory variations

20.4.1 Scanning a selected variation with selected matrices

TO BE WRITTEN

20.4.2 Obtaining a list of disease-associated variation IDs

In order to test the retrieval of multiple variations, we will first select a realistic case, by getting a list of Human variations associated to some disease (for instance diabetes).

Open a connection to the dbSNP database [2], and paste the following test in the query box:

```
("Homo sapiens"[ORG]) AND (diabetes[Text Word]) AND ("false"[Not Reference Assembly])
```

TO BE WRITTEN

20.4.3 Scanning a list of selected variations with a list of matrices

TO BE WRITTEN

20.4.4 Scanning all variations with a selected matrix

TO BE WRITTEN

[2] http://www.ncbi.nlm.nih.gov/SNP/

21 Exercises

As an exercise, we will now combine the different tools described above to analyse the full set of promoters from *Arabidopsis thaliana*. We define ourselves the following goals :

1. Discover motifs which are over-represented in the complete set of upstream sequences for the selected organism.
2. Try different parameters for this pattern discovery, and compare the results.
3. Use these over-represented patterns to scan full chromosomes with a sliding window, in order to evaluate if we can predict promoter locations on the sole basis of pattern occurrences. Find optimal parameters for the prediction of promoter locations.

21.1 Some hints

21.1.1 Sequence retrieval

The first step will be to retrieve the full complement of upstream sequences. Since we have no precise idea about the best sequence size, we will try several reasonable ranges, each roughly corresponding to a given functionality.

from -1 to -200 this regions is likey to contain mostly 5'UTR.

from -1 to -400 this region is likely to contain the 5' UTR and the proximal promoter.

from -1 to -1000 this region is likely to include the 5'UTR, as well as the proxima and distal promoters.

from -1 to -2000 an even larger range, which probably contains most of the upstream cis-acting elements in *A. thaliana*.

In all cases, we will clip upstream ORFs, because they would bias the oligonucleotide composition.

Write the commands which will retrieve all upstream sequences over the specified range. Beware, the sequence files may occupy a large space on the disk, it is probably wise to directly compress them by adding the extension `.gz` to the output file.

21.1.2 Detection of over-represented motifs

In a first step, we will restrict our analysis to hexanucleotides. Once all the subsequent steps (full chromosome scanning) will be accomplished, we will redo the complete analysis with different oligonucleotide lengths, and compare the efficiency of promoter prediction.

Detect over-represented oligo-nucleotides with different estimators of expected frequencies: Markov chains of different orders, non-coding frequencies.

Do not forget to prevent counting self-overlapping matches.

22 Using RSAT Web Services

Note: in complement of the following instructions, we recommend to run the protocol for using ***RSAT*** Web services [**?**].

22.1 Introduction

RSAT facilities can be used as Web Services (*WS*), i.e. external developers (you) can integrate ***RSAT*** methods in their own code. An important advantage of Web Services is that they are using a standard communication interface between client and server (e.g. WSDL/SOAP), for which libraries exist in various languages (Perl, Python, java).

We explain below how to implement WS clients in Perl, Java and Python for ***RSAT*** programs.

22.2 Examples of WS clients in Perl with SOAP::WSDL 2.00 (or above)

22.2.1 Requirements

Before using such WS clients, You need to install the ***Module::Build::Compat*** and the ***SOAP::WSDL*** Perl modules. These Perl modules can be installed with the program ***cpan***. When required, you will be prompted to install dependency modules for ***SOAP::WSDL***. For all this you need root privileges. If this is not your case, please ask your system administrator to install them for you.

The other thing you need is the RSATWS library that you can download from the following website:
http://rsat.ulb.ac.be/rsat/web_services/RSATWS.tar.gz

Place it in the same directory as your clients, then uncompress if with the following command.

```
tar -xpzf RSATWS.tar.gz
```

22.2.2 Retrieving sequences from RSATWS

The following example is a script to retrieve the start codons of three Escherichia coli genes. It uses ***retrieve-seq*** to do so. The various parameters are passed as a hash table to the method. If there is an error, it will be displayed, otherwise the result is displayed, toghether with the full command generated on the server and the name of the temporary file created on the server to hold the result localy. This file is useful when one wants to feed another program with that output, whithout paying the cost of a useless data transport back and forth between the server and the client.

```
#!/usr/bin/perl -w
# retrieve-seq_client_soap-wsdl.pl - Client retrieve-seq using the SOAP::WSDL
#module
```

```
################################################################
##
## This script runs a simple demo of the web service inerface to the
## RSAT tool retrieve-seq. It sends a request to the server for
## obtaining the start codons of 3 E.coli genes.
##
################################################################

use strict;
use SOAP::WSDL;
use lib 'RSATWS';
use MyInterfaces::RSATWebServices::RSATWSPortType;

warn ``\nThis demo script retrieves the start codons for a set of query
genes\n\n'';

## WSDL location
my $server = 'http://rsat.ulb.ac.be/rsat/web_services';

## Service call
my $soap=MyInterfaces::RSATWebServices::RSATWSPortType->new();

## Output option
my $output_choice = 'both';  ## Accepted values: 'server', 'client', 'both'

## Retrieve-seq parameters
my $organism = 'Escherichia_coli_K_12_substr__MG1655_uid57779';  ## Name of the query o
my @gene = (``metA'', ``metB'', ``metC'');  ## List of query genes
my $all = 0;  ## the -all option (other accepted value = 1). This option is
incompatible with the query list @gene (above)
my $noorf = 1;  ## Clip sequences to avoid upstream ORFs
my $from = 0;  ## Start position of the sequence
my $to = 2;  ## End position of the sequence
my $feattype = '';  ## The -feattype option value is  not specified, the
default is used
my $type = '';  ## The -type option value; other example:'-type downstream'
my $format = '';  ## The -format option value. We use the default (fasta), but
other formats could be specified, for example 'multi'
my $lw = 0;  ## Line width. 0 means all on one line
my $label = 'id,name';  ## Choice of label for the retrieved sequence(s)
my $label_sep = '';  ## Choice of separator for the label(s) of the retrieved
sequence(s)
my $nocom = 0;  ## Other possible value = 1, to get sequence(s) whithout
comments
my $repeat =  0;  ## Other possible value = 1, to have annotated repeat
regions masked
my $imp_pos = 0;  ## Admit imprecise position (value = 1 to do so)

my %args = (
            'output' => $output_choice,
            'organism' => $organism,
```

```
            'query' => \@gene,  ## An array in a hash has to be referenced
            (always?)
            'noorf' => $noorf,
            'from' => $from,
            'to' => $to,
            'feattype' => $feattype,
            'type' => $type,
            'format' => $format,
            'lw' => $lw,
            'label' => $label,
            'label_sep' => $label_sep,
            'nocom' => $nocom,
            'repeat' => $repeat,
            'imp_pos' => $imp_pos
            );

## Send the request to the server
print ``Sending request to the server $server\n'';
my $som = $soap->retrieve_seq({'request' => \%args});

## Get the result
unless ($som) {
        printf ``A fault (%s) occured: %s\n'', $som->get_faultcode(),
  %$som->get_faultstring();
} else {
        my $results = $som->get_response();

    ## Report the remote command
    my $command = $results -> get_command();
    print ``Command used on the server: ``.$command, ``\n'';

    ## Report the result
    if ($output_choice eq 'server') {
                my $server_file = $results -> get_server();
                print ``Result file on the server: ``.$server_file;
   } elsif ($output_choice eq 'client') {
                my $result = $results -> get_client();
                print ``Retrieved sequence(s): \n''.$result;
   } elsif ($output_choice eq 'both') {
                my $server_file = $results -> get_server();
                my $result = $results -> get_client();
                print ``Result file on the server: ``.$server_file.''\n'';
                print ``Retrieved sequence(s): \n''.$result;
    }
}
```

22.3 Examples of WS clients in Perl with SOAP::WSDL 1.27 (or below)

Some of you are maybe already using perl WS clients with an older version of ***SOAP::WSDL*** and would like to stick to it. We show hereafter some simple examples of clients written in perl and using such version of the module. The presented code as well as other can be downloaded from
http://rsat.ulb.ac.be/rsat/web_services.html

22.3.1 Requirements

- ***SOAP::Lite***
- ***SOAP::WSDL***, version 1.27 or below.

These Perl modules can be installed with the program ***cpan***, but for this you need root privileges. If this is not your case, please ask your system administrator to install them for you.

22.3.2 Getting gene-info from RSATWS

The following script allows to get information about three *Escherichia coli* genes from ***RSAT***. The client script passes through the web service to run the ***gene-info*** on the server. A list of genes is provided to the server, which returns the information about those genes.

```
#!/usr/bin/perl -w
# gene-info_client_minimal_soap-wsdl.pl - Client gene-info using the SOAP::WSDL module.

################################################################
##
## This script runs a simple demo of the web service interface to the
## RSAT tool gene-info. It sends a list of 3 gene names to the server,
## in order to obtain the information about these genes.
##
################################################################

use strict;
use SOAP::WSDL;

## Service location
my $server = 'http://rsat.ulb.ac.be/rsat/web_services';
my $WSDL = $server.'/RSATWS.wsdl';
my $proxy = $server.'/RSATWS.cgi';

## Call the service
my $soap=SOAP::WSDL->new(wsdl => $WSDL)->proxy($proxy);
$soap->wsdlinit;

## Gene-info parameters
my $organism = 'Escherichia_coli_K_12_substr__MG1655_uid57779';  ## Name of the query o
my @gene = ("metA", "metB", "metC");  ## List of query genes
my $full = 1;  ## Looking for full match, not substring match.
```

```
my %args = ('organism' => $organism,
    'query' => \@gene,
    'full' => $full);

## Send the request to the server
warn "Sending request to the server $server\n";
my $call = $soap->call('gene_info' => 'request' => \%args);

## Get the result
if ($call->fault){ ## Report error if any
    printf "A fault (%s) occured: %s\n", $call->faultcode, $call->faultstring;
} else {
    my $results_ref = $call->result;  ## A reference to the result hash table
    my %results = %$results_ref;  ## Dereference the result hash table

    ## Report the remote command
    my $command = $results{'command'};
    print "Command used on the server: ".$command, "\n";

    ## Report the result
    my $result = $results{'client'};
    print "Gene(s) info(s): \n".$result;
}
```

We can now use additional parameters of the ***gene-info*** program. For example, we could use regular expressions to ask the server for all the yeast genes whose name starts with 'MET', followed by one or several numbers.

```
... (same as above)

## Gene-info parameters
my $organism = 'Saccharomyces_cerevisiae';  ## Name of the query organism
my @queries = ('MET\d+');  ## This query is a regular expression
my $full = 1;  ## Looking for full match, not substring match.

my %args = ('organism' => $organism,
    'query' => \@queries,
    'full' => $full);

... (same as above)
```

We can also extend the search to match the query strings against gene descriptions (by default, they are only matched against gene names).

```
... (same as above)

## Gene-info parameters
my $organism = 'Escherichia_coli_K_12_substr__MG1655_uid57779';  ## Name of the query o
my @queries = ("methionine", "purine");  ## List of queries
my $full = 0;
```

```
my $descr = 1;  ## Search also in description field of genes

my %args = ('organism' => $organism,
    'query' => \@queries,
    'full' => $full,
    'descr' => $descr);

... (same as above)
```

22.3.3 Documentation

We saw above that the command

```
gene-info
```

can be called with various options. The description of the available options can be found in the documentation of the RSATWS web services at the following URL.

http://rsat.ulb.ac.be/rsat/web_services/RSATWS_documentation.xml

22.3.4 Retrieving sequences from RSATWS

The following example is a script to retrieve the start codons of three Escherichia coli genes. It uses ***retrieve-seq*** to do so. The various parameters are passed as a hash table to the method. If there is an error, it will be displayed, otherwise the result is displayed, toghether with the full command generated on the server and the name of the temporary file created on the server to hold the result localy. This file is useful when one wants to feed another program with that output, whithout paying the cost of a useless data transport back and forth between the server and the client.

```
#!/usr/bin/perl -w
# retrieve-seq_client_soap-wsdl.pl - Client retrieve-seq using the SOAP::WSDL module

################################################################
##
## This script runs a simple demo of the web service interface to the
## RSAT tool retrieve-seq. It sends a request to the server for
## obtaining the start codons of 3 E.coli genes.
##
################################################################

use strict;
use SOAP::WSDL;

warn "\nThis demo script retrieves the start codons for a set of query genes\n\n";

## WSDL location
my $server = 'http://rsat.ulb.ac.be/rsat/web_services';
my $WSDL = $server.'/RSATWS.wsdl';
my $proxy = $server.'/RSATWS.cgi';

## Service call
my $soap=SOAP::WSDL->new(wsdl => $WSDL)->proxy($proxy);
$soap->wsdlinit;
```

```
## Output option
my $output_choice = 'both';  ## Accepted values: 'server', 'client', 'both'

## Retrieve-seq parameters
my $organism = 'Escherichia_coli_K_12_substr__MG1655_uid57779';  ## Name of the query o
my @gene = ("metA", "metB", "metC");  ## List of query genes
my $noorf = 1;  ## Clip sequences to avoid upstream ORFs
my $from = 0;  ## Start position of the sequence
my $to = 2;  ## End position of the sequence
my $lw = 0;  ## Line width. 0 means all the sequence on one line
my $label = 'id,name';  ## Choice of label for the retrieved sequence(s)

my %args = (
    'output' => $output_choice,
    'organism' => $organism,
    'query' => \@gene,
    'noorf' => $noorf,
    'from' => $from,
    'to' => $to,
    'lw' => $lw,
    'label' => $label,
    );

## Send the request to the server
print "Sending request to the server $server\n";
my $call = $soap->call('retrieve_seq' => 'request' => \%args);

## Get the result
if ($call->fault){ ## Report error if any
    printf "A fault (%s) occured: %s\n", $call->faultcode, $call->faultstring;
} else {
    my $results_ref = $call->result;  ## A reference to the result hash table
    my %results = %$results_ref;  ## Dereference the result hash table

    ## Report the remote command
    my $command = $results{'command'};
    print "Command used on the server: ".$command, "\n";

    ## Report the result
    if ($output_choice eq 'server') {
my $server_file = $results{'server'};
print "Result file on the server: ".$server_file;
    } elsif ($output_choice eq 'client') {
my $result = $results{'client'};
print "Retrieved sequence(s): \n".$result;
    } elsif ($output_choice eq 'both') {
my $server_file = $results{'server'};
my $result = $results{'client'};
print "Result file on the server: ".$server_file;
print "Retrieved sequence(s): \n".$result;
    }
```

```
}
```

22.3.5 Work flow using RSATWS

The following example is the script of a typical workflow of RSA Tools programs. First, the upstream sequences of five Saccharomyces cerevisiae genes are retrieved with ***retrieve-seq***. Then, ***purge-sequence*** is applyed to remove any redundancy in the set of sequences. Finally, ***oligo-analysis*** is applied to discover over-represented six letters words. The result of step 1 and 2 are stored on the server, so that the file name can be sent to the following step as input and only the final result needs to be transported from the server to the client.

```
#!/usr/bin/perl -w
# retrieve_purge_oligos_client_soap-wsdl.pl - Client retrieve-seq + oligo-analysis

################################################################
##
## This script runs a simple demo of the web service interface to the
## RSAT tools retrieve-seq, purge-sequence and oligo-analysis linked in a workflow.
##  It sends a request to the server for discovering 6 letter words
## in upstream sequences of 5 yeast genes. The sequences are first
## retrieved and purged for repeated segments
##
################################################################

use strict;
use SOAP::WSDL;

warn "\nThis demo script illustrates a work flow combining three requests to the RSAT w


## Service location
my $server = 'http://rsat.ulb.ac.be/rsat/web_services';
my $WSDL = $server.'/RSATWS.wsdl';
my $proxy = $server.'/RSATWS.cgi';

## Service call
my $soap=SOAP::WSDL->new(wsdl => $WSDL)->proxy($proxy);
$soap->wsdlinit;

################################################
## Retrieve-seq part

## Output option
my $output_choice = 'server'; ## The result will stay in a file on the server

## Parameters
my $organism = 'Saccharomyces_cerevisiae';  ## Name of the query organism
my @gene = ("PHO5", "PHO8", "PHO11", "PHO81", "PHO84");  ## List of query genes
my $noorf = 1;  ## Clip sequences to avoid upstream ORFs
my $from;  ## Start position of the sequence. Default is used (-800).
my $to;  ## End position of te sequence. Default is used (-1).
```

```
my $feattype;  ## -feattype option value is not defined, default is used (CDS).
my $type;  ## -type option value; other example:'-type downstream'
my $format = 'fasta';  ## the format of the retrieved sequence(s)
my $label;  ## Choice of label for the retrieved sequence(s). Default is used.
my $label_sep;  ## Choice of separator for the label(s) of the retrieved sequence(s). D

my %args = ('output' => $output_choice,
    'organism' => $organism,
    'query' => \@gene,  ## An array in a hash has to be referenced
    'noorf' => $noorf,
    'from' => $from,
    'to' => $to,
    'feattype' => $feattype,
    'type' => $type,
    'format' => $format,
    'label' => $label,
    'label_sep' => $label_sep
    );

## Send request to the server
print "\nRetrieve-seq: sending request to the server\t", $server, "\n";
my $call = $soap->call('retrieve_seq' => 'request' => \%args);

## Get the result
my $server_file;  ## That variable needs to be declared outside the if..else block to b
if ($call->fault){  ## Report error if any
  printf "A fault (%s) occured: %s\n", $call->faultcode, $call->faultstring;
} else {
  my $results_ref = $call->result;  ## A reference to the result hash table
  my %results = %$results_ref;  ## Dereference the result hash table

  ## Report the remote command
  my $command = $results{'command'};
  print "Command used on the server:\n\t".$command, "\n";

  ## Report the result file name on the server
  $server_file = $results{'server'};
  print "Result file on the server:\n\t".$server_file;
}

################################################
## Purge-sequence part

## Define hash of parameters
%args = ('output' => $output_choice,  ## Same 'server' output option
 'tmp_infile' => $server_file);  ## Output from retrieve-seq part is used as input here

## Send the request to the server
print "\nPurge-sequence: sending request to the server\t", $server, "\n";
$call = $soap -> call('purge_seq' => 'request' => \%args);

## Get the result
```

```
if ($call->fault){  ## Report error if any
  printf "A fault (%s) occured: %s\n", $call->faultcode, $call->faultstring;
} else {
  my $results_ref = $call->result;  ## A reference to the result hash table
  my %results = %$results_ref;  ## Dereference the result hash table

  ## Report the remote command
  my $command = $results{'command'};
  print "Command used on the server: \n\t".$command, "\n";

  ## Report the result file name on the server
  $server_file = $results{'server'};
  print "Result file on the server: \n\t".$server_file;
}
################################################
## Oligo-analysis part

## Output option
$output_choice = 'both'; ## We want to get the result on the client side, as well as th

## Parameters
my $format = 'fasta';  ## The format of input sequences
my $length = 6;  ## Length of patterns to be discovered
my $background = 'upstream-noorf';  ## Type of background used
my $stats = 'occ,proba,rank';  ## Returned statistics
my $noov = 1;  ## Do not allow overlapping patterns
my $str = 2;  ## Search on both strands
my $sort = 1;  ## Sort the result according to score
my $lth = 'occ_sig 0';  ## Lower limit to score is 0, less significant patterns are not

%args = ('output' => $output_choice,
 'tmp_infile' => $server_file,
 'format' => $format,
 'length' => $length,
 'organism' => $organism,
 'background' => $background,
 'stats' => $stats,
 'noov' => $noov,
 'str' => $str,
 'sort' => $sort,
 'lth' => $lth);

## Send request to the server
print "\nOligo-analysis: sending request to the server\t", $server, "\n";
$call = $soap->call('oligo_analysis' => 'request' => \%args);

## Get the result
if ($call->fault){  ## Report error if any
    printf "A fault (%s) occured: %s\n", $call->faultcode, $call->faultstring;
} else {
    my $results_ref = $call->result;
    my %results = %$results_ref;
```

```
    ## Report remote commande
    my $command = $results{'command'};
    print "Command used on the server: ".$command, "\n";

    ## Report the result
    if ($output_choice eq 'server') {
$server_file = $results{'server'};
print "Result file on the server: \n\t".$server_file;
    } elsif ($output_choice eq 'client') {
my $result = $results{'client'};
print "Discovered oligo(s): \n".$result;
    } elsif ($output_choice eq 'both') {
$server_file = $results{'server'};
my $result = $results{'client'};
print "Result file on the server: \n\t".$server_file;
print "Discovered oligo(s): \n".$result;
    }
}
```

22.3.6 Discover patterns with RSATWS

You can, of course, use directly the program ***oligo-analysis***, providing your own sequences. In the following script, the upstream sequences of five yeast genes are sent as input to oligo-analysis. Over-represented hexanucleotides are returned.

```
#!/usr/bin/perl -w
# oligos_client_soap-wsdl.pl - Client oligo-analysis using the SOAP::WSDL module

################################################################
##
## This script runs a simple demo of the web service interface to the
## RSAT tool oligo-analysis. It sends a request to the server for
## discovering 6 letter words in the upstream sequences of 5 yeast genes.
##
################################################################

use strict;
use SOAP::WSDL;

warn "\nINFO: This demo script sends a set of sequences to the RSAT web service, and ru

## WSDL location
my $server = 'http://rsat.ulb.ac.be/rsat/web_services';
my $WSDL = $server.'/RSATWS.wsdl';
my $proxy = $server.'/RSATWS.cgi';

my $soap=SOAP::WSDL->new(wsdl => $WSDL)->proxy($proxy);

$soap->wsdlinit;

## Output option
```

```
my $output_choice = 'both';  ## Accepted values: 'server', 'client', 'both'

## Oligo-analysis parameters
my $sequence = '>NP_009651.1    PHO5; upstream from -800 to -1; size: 800; location: NC
TTTTACACATCGGACTGATAAGTTACTACTGCACATTGGCATTAGCTAGGAGGGCATCCAAGTAATAATTGCGAGAAACGTGACCCA
>NP_010769.1    PHO8; upstream from -180 to -1; size: 180; location: NC_001136.8 142024
CAGCATTGACGATAGCGATAAGCTTCGCGCGTAGAGGAAAAGTAAAGGGATTTTAGTATATAAAGAAAGAAGTGTATCTAAACGTTT
>NP_009434.1    PHO11; upstream from -800 to -1; size: 800; location: NC_001133.6 22465
GCAGCCTCTACCATGTTGCAAGTGCGAACCATACTGTGGCCACATAGATTACAAAAAAAGTCCAGGATATCTTGCAAACCTAGCTTG
>NP_011749.1    PHO81; upstream from -800 to -1; size: 800; location: NC_001139.7 95821
AAACGAGCATGAGGGTTACAAAGAACTTCCGTTTCAAAAATGAATATAATCGTACGTTTACCTTGTGGCAGCACTAGCTAACGCTAC
>NP_013583.1    PHO84; upstream from -800 to -1; size: 800; location: NC_001145.2 25802
AAAAAAAAAGATTCAATAAAAAAAGAAATGAGATCAAAAAAAAAAAAAATTAAAAAAAAAAAGAAACTAATTTATCAGCCGCTCGTT

my $format = 'fasta';  ## The format of input sequences
my $length = 6;  ## Length of patterns to be discovered
my $organism = 'Saccharomyces_cerevisiae';  ## Name of the query organism
my $background = 'upstream-noorf';  ## Type of background used
my $stats = 'occ,proba,rank';  ## Returned statistics
my $noov = 1;  ## Do not allow overlapping patterns
my $str = 2;  ## Search on both strands
my $sort = 1;  ## Sort the result according to score
my $lth = 'occ_sig 0';  ## Lower limit to score is 0, less significant patterns are not

my %args = ('output' => $output_choice,
    'sequence' => $sequence,
    'format' => $format,
    'length' => $length,
    'organism' => $organism,
    'background' => $background,
    'stats' => $stats,
    'noov' => $noov,
    'str' => $str,
    'sort' => $sort,
    'lth' => $lth);

## Send request to the server
print "Sending request to the server $server\n";
my $call = $soap->call('oligo_analysis' => 'request' => \%args);

## Get the result
if ($call->fault){  ## Report error if any
    printf "A fault (%s) occured: %s\n", $call->faultcode, $call->faultstring;
} else {
    my $results_ref = $call->result;  ## A reference to the result hash table
    my %results = %$results_ref;  ## Dereference the result hash table

    ##Report the remote command
    my $command = $results{'command'};
    print "Command used on the server: ".$command, "\n";

    ## Report the result
```

```
    if ($output_choice eq 'server') {
my $server_file = $results{'server'};
print "Result file on the server: ".$server_file;
    } elsif ($output_choice eq 'client') {
my $result = $results{'client'};
print "Discovered oligo(s): \n".$result;
    } elsif ($output_choice eq 'both') {
my $server_file = $results{'server'};
my $result = $results{'client'};
print "Result file on the server: ".$server_file;
print "Discovered oligo(s): \n".$result;
    }
}
```

22.3.7 Example of clients using property files

We have also made clients using an alternative approach. Instead of writing the parameters values in the client code itself, these are read from a property file. Here is the client for retrieve-seq:

```
#!/usr/bin/perl -w
# retrieve-seq_client.pl - Client retrieve-seq using the SOAP::WSDL module
# and a property file

#################################################################
##
## This script runs a simple demo of the web service inerface to the
## RSAT tool retrieve-seq. It sends a request to the server for
## obtaining the start codons of 3 E.coli genes.
##
#################################################################

use strict;
use SOAP::WSDL;
use Util::Properties;

## WSDL location
my $server = 'http://rsat.ulb.ac.be/rsat/web_services';
my $WSDL = $server.'/RSATWS.wsdl';
my $proxy = $server.'/RSATWS.cgi';
my $property_file = shift @ARGV;
die "\tYou must specify the property file as first argument\n"
  unless $property_file;

## Service call
my $soap=SOAP::WSDL->new(wsdl => $WSDL)->proxy($proxy);
$soap->wsdlinit;

my $prop =  Util::Properties->new();
$prop->file_name($property_file);
$prop->load();
my %args = $prop->prop_list();
```

```
## Convert the query string into a list
my @queries = split(",", $args{query});
$args{query} = \@queries;

my $output_choice = $args{output_choice} || 'both';

warn "\nThis demo script retrieves upstream sequences for a set of query genes\n\n";

## Send the request to the server
print "Sending request to the server $server\n";
my $som = $soap->call('retrieve_seq' => 'request' => \%args);

## Get the result
if ($som->fault){ ## Report error if any
    printf "A fault (%s) occured: %s\n", $som->faultcode, $som->faultstring;
} else {
    my $results_ref = $som->result;  ## A reference to the result hash table
    my %results = %$results_ref;  ## Dereference the result hash table

    ## Report the remote command
    my $command = $results{'command'};
    print "Command used on the server: ".$command, "\n";

    ## Report the result
    if ($output_choice eq 'server') {
my $server_file = $results{'server'};
print "Result file on the server: ".$server_file;
    } elsif ($output_choice eq 'client') {
my $result = $results{'client'};
print "Retrieved sequence(s): \n".$result;
    } elsif ($output_choice eq 'both') {
my $server_file = $results{'server'};
my $result = $results{'client'};
print "Result file on the server: ".$server_file;
print "Retrieved sequence(s): \n".$result;
    }
}
```

The property file looks like this:

```
output=both
organism=Escherichia_coli_K_12_substr__MG1655_uid57779
query=metA,metB
all=0
noorf=1
from=0
to=2
feattype=CDS
type=upstream
format=fasta
lw=0
label=id,name
label_sep=
```

```
nocom=0
repeat= 0
imp_pos=0
```

To run the client, give the path of the property file as argument.

In the downloadable clients, the ones with a name like *_client.pl use a property file. Examples of property files are in the sub-directory 'property_files'. When the property file contains the path to a file, make sure you edit it according to your system.

22.3.8 Other tools in RSATWS

Following the examples above or using the code that is available for download[1], you can easily access the other RSA Tools for which Web Services have been implemented. You will find all you need to know about the tools (parameters, etc.) in the documentation[2].

22.4 Examples of WS client in java

First, you need to generate the libraries. There are tools, like Axis, which do it from the WSDL document. These usually take the URL of that document as one of their parameters. In our case, it is there:

http://rsat.ulb.ac.be/rsat/web_services/RSATWS.wsdl

Then you write a simple client like the one in the following example.

22.4.1 Same workflow as above with RSATWS

```
import RSATWS.OligoAnalysisRequest;
import RSATWS.OligoAnalysisResponse;
import RSATWS.PurgeSequenceRequest;
import RSATWS.PurgeSequenceResponse;
import RSATWS.RSATWSPortType;
import RSATWS.RSATWebServicesLocator;
import RSATWS.RetrieveSequenceRequest;
import RSATWS.RetrieveSequenceResponse;


public class RSATRetrievePurgeOligoClient {

/**
 * This script runs a simple demo of the web service interface to the
 * RSAT tools retrieve-seq, purge-sequence and oligo-analysis linked in a workflow.
 * It sends a request to the server for discovering 6 letter words
 * in upstream sequences of 5 yeast genes. The sequences are first
 * retrieved and purged for repeated segments
 */
public static void main(String[] args) {
 try
```

[1] http://rsat.ulb.ac.be/rsat/web_services.html

[2] http://rsat.ulb.ac.be/rsat/web_services/RSATWS_documentation.xml

```
        {

System.out.println("This demo script illustrates a work flow combining three requests t

String organism = "Saccharomyces_cerevisiae";

/* Get the location of the service */
RSATWebServicesLocator service = new RSATWebServicesLocator();
RSATWSPortType proxy = service.getRSATWSPortType();

/** Retrieve-seq part **/

            /* prepare the parameters */
            RetrieveSequenceRequest retrieveSeqParams = new RetrieveSequenceRequest();

            //Name of the query organism
            retrieveSeqParams.setOrganism(organism);
            //List of query genes
            String[] q=  { "PHO5", "PHO8", "PHO11", "PHO81", "PHO84" };
            retrieveSeqParams.setQuery(q);
            // Clip sequences to avoid upstream ORFs
            retrieveSeqParams.setNoorf(1);
            retrieveSeqParams.setNocom(0);
            // The result will stay in a file on the server
            retrieveSeqParams.setOutput("server");

          /* Call the service */
            System.out.println("Retrieve-seq: sending request to the server...");
            RetrieveSequenceResponse res = proxy.retrieve_seq(retrieveSeqParams);

            /* Process results  */
            //Report the remote command
            System.out.println("Command used on the server:"+ res.getCommand());
            //Report the server file location
            String retrieveSeqFileServer = res.getServer();
            System.out.println("Result file on the server::\n"+ res.getServer());

            /** Purge-sequence part **/

            /* prepare the parameters */
            PurgeSequenceRequest purgeSeqParams = new PurgeSequenceRequest();
            // The result will stay in a file on the server
            purgeSeqParams.setOutput("server");
            // Output from retrieve-seq part is used as input here
            purgeSeqParams.setTmp_infile(retrieveSeqFileServer);

            /* Call the service */
            System.out.println("Purge-sequence: sending request to the server...");
            PurgeSequenceResponse res2 = proxy.purge_seq(purgeSeqParams);

            /* Process results  */
            //Report the remote command
```

```
            System.out.println("Command used on the server:"+ res2.getCommand());
            //Report the server file location
            String purgeSeqFileServer = res2.getServer();
            System.out.println("Result file on the server::\n"+ res2.getServer());

            /** Oligo-analysis part **/

            /* prepare the parameters */
            OligoAnalysisRequest oligoParams = new OligoAnalysisRequest();
            // Output from purge-seq part is used as input here
            oligoParams.setTmp_infile(purgeSeqFileServer);
            oligoParams.setOrganism(organism);
            // Length of patterns to be discovered
            oligoParams.setLength(6);
            // Type of background used
            oligoParams.setBackground("upstream-noorf");
            // Returned statistics
            oligoParams.setStats("occ,proba,rank");
            // Do not allow overlapping patterns
            oligoParams.setNoov(1);
            // Search on both strands
            oligoParams.setStr(2);
            // Sort the result according to score
            oligoParams.setSort(1);
            // Lower limit to score is 0, less significant patterns are not displayed
            oligoParams.setLth("occ_sig 0");

            /* Call the service */
            System.out.println("Oligo-analysis: sending request to the server...");
            OligoAnalysisResponse res3 = proxy.oligo_analysis(oligoParams);

            /* Process results  */
            //Report the remote command
            System.out.println("Command used on the server:"+ res3.getCommand());
            //Report the result
            System.out.println("Discovered oligo(s):\n"+ res3.getClient());
            //Report the server file location
            System.out.println("Result file on the server::\n"+ res3.getServer());

        }
        catch(Exception e) { System.out.println(e.toString());
        }
}
}
```

22.5 Examples of WS client in python

22.5.1 Get infos on genes having methionine or purine in their description, as above in perl

```
#! /usr/bin/python
```

```
class GeneInfoRequest:

    def __init__(self):

        self.organism = None
        self.query = None
        self.noquery = None
        self.desrc = None
        self.full = None
        self.feattype = None


if __name__ == '__main__':

    import os, sys, SOAPpy

    if os.environ.has_key("http_proxy"):
        my_http_proxy=os.environ["http_proxy"].replace("http://","")
    else:
        my_http_proxy=None

    organism = "Escherichia_coli_K_12_substr__MG1655_uid57779"
    query = ["methionine", "purine"]
    full = 0
    noquery = 0
    descr = 0
    feattype = "CDS"

    url = "http://rsat.ulb.ac.be/rsat/web_services/RSATWS.wsdl"
    server = SOAPpy.WSDL.Proxy(url, http_proxy = my_http_proxy)
    server.soapproxy.config.dumpSoapOutput = 1
    server.soapproxy.config.dumpSoapInput = 1
    server.soapproxy.config.debug = 0

    req = GeneInfoRequest()
    req.organism = organism
    req.query = query
    req.full = 0
    req.descr = 1

    res = server.gene_info(req)

    print res.command
    print res.client
```

22.6 Full documentation of the RSATWS interface

The full documentation can be found there:
http://rsat.ulb.ac.be/rsat/web_services/RSATWS_documentation.pdf
Please refer to the documentation of each RSAT application for further detail on each program.

23 Graph analysis

23.1 Introduction

23.1.1 Definition

Informally speaking, a *graph* is a set of objects called points, nodes, or vertices connected by links called lines or edges.

More formally, a graph or undirected graph G is an ordered pair $G = (V, E)$ that is subject to the following conditions :

- V is a set, whose elements are called vertices or nodes
- E is a set of pairs (unordered) of distinct vertices, called edges or lines.

The vertices belonging to an edge are called the ends, endpoints, or end *vertices* of the *edge*. V (and hence E) are taken to be finite sets.

The *degree* of a vertex is the number of other vertices it is connected to by edges. As graphs are used to model all kinds of problems and situation (networks, maps, pathways, ...), nodes and vertex may present attributes (color, weight, label, ...).

23.1.2 Some types of graphs

Undirected graph

An edge between vertex A and vertex B corresponds to an edge between B and A.

Directed graph (digraph)

An edge between vertex A and vertex B does not correspond to a vertex between B and A. In that case, edges are said to be arcs.

Weighted graph

A weight can be placed either on the nodes or on the edges of the graph. A weight on the edge may for example represent a distance between two nodes or the strength an interaction.

Bipartite graphs

A bipartite graph is a special graph where there are two types of nodes : A and B and where each node of type A is only connected to nodes of type B and vice-versa.

23.1.3 Graph files formats

List of edges

This format is the more intuitive way to encode a graph. It consists in a list of edges between the nodes. The names of the nodes are separated using some field separator, in RSAT, a tabulation. Some attributes of the edges can be placed in the following columns (weight, label, color).

```
n1    n2    3.2
n1    n2    1.4
n2    n3     4
n3    n4     6
```

GML format

Among other, GML format allows to specify the location, the color, the label and the width of the nodes and of the edges. A GML file is made up of pairs of a key and a value. Example for keys are graphs, node and edges. You can then add any specific information for each key. GML format can be used by most graph editors (like cytoscape and yEd).

For more information on the GML format, see http://www.infosun.fim.uni-passau.de/Graphlet/G

DOT format

DOT is a plain text graph description language. The DOT files are generally used by the programs composing the GraphViz suite (dot, neato, dotty, ...). It is a simple way of describing graphs that both humans and computer programs can use. DOT graphs are typically files that end with the *.dot* extension. Like GML, with DOT you can specify a lot of feature for the nodes (color, width, label).

23.2 RSAT Graph tools

23.2.1 *convert-graph*

This program converts a graph encoded in some format (gml, tab) to some other (gml, tab, dot). The source node are in the first column of this file, target nodes in the second column and the edge weights are in the third one. By default, column 1 contains the source node, column 2 the target nodes and there is no weight.

```
convert-graph -i demo_graph.tab -o demo_graph.gml -from tab -to gml -scol 1 -tcol 2 -
```

convert-graph also allows to randomize a graph using *-random* option, each node keeping the same number of neighbours (degree). You can specify the number of required random graphs.

```
convert-graph -i demo_graph.tab -o random_graph -random 100 -from tab -to tab
```

This command will create 100 different random graph from the file demo_graph.tab.

23.2.2 *graph-node-degree*

Calculate the node degree of each node (or of a selection of nodes) and specifies if this node is a seed or a target node.

```
graph-node-degree -all -i demo_graph.tab
```

23.2.3 *graph-neighbours*

Extracts the neighbourhood from a graph (the number of steps may be specified) of all or of a set of seed nodes.

```
graph-neighbours -i demo_graph.tab -steps 1 -seed n2 -self
```

With this command, ***graph-neighbours*** will retrieve all the first neighbours of node *n*2 , *n*2 being included. To also get the neighbours of the neighbours of *n*2, we should use the option *-steps 2*. The output file may then be used with ***compare-classes*** program to compare groups of neighbours to annotated groups of nodes. A file containing a list of seed nodes can be given to ***graph-neighbours*** using *-seedf* option.

Using the -stats option with a weighted graph will return one line for each seed node (-steps must then be equal to 1).

23.2.4 *compare-graphs*

Computes the intersection, union or difference of two graphs (a reference graph and a query graph). The format of each input graph may be specified so that you can compare a gml encoded graph to a edge-list format graph.

```
compare-graphs -Q query_graph.tab -R reference_graph2.gml \
               -return union -out_format tab -outweight Q::R \
               -in_format_R gml -wcol_Q 3
```

With this command, you will compare query_graph.tab and reference_graph2.gml. The output will be an edge list format file. For each edge, it will specify if the edge belongs to the reference graph, to the query graph or to both of them and colour the edges accordingly.

23.2.5 *graph-get-clusters*

Extract from a graph a subgraph specified by a set of *clusters* of nodes. It returns the nodes belonging to the clusters and the intra-cluster arcs, and ignore the inter-cluster arcs.

```
graph-get-clusters -i demo_graph_cl.tab -clusters demo_graph_clusters.tab \
                   -out_format gml -o demo_graph_clusters_ex.gml
```

Using the *-distinct* option, nodes belonging to more than one cluster are duplicated. This option should be used for visualisation purpose only.

Using the *-inducted* option, you can extract a subgraph containing all the nodes specified in the cluster file. In that case, you don't specially need a two-column file.

23.2.6 *compare-graph-clusters*

With the *-return table* option, this program counts the number (or the sum of the weights) of intra cluster (or class) edges in a graph according to some clustering (classification) file and the number of edges in each cluster.

```
compare-graph-clusters -i demo_graph_cl.tab \
                       -clusters demo_graph_clusters.tab -v 1 -return table
```

With the *-return graph* option, this program returns some cluster characteristics for each edge, i.e., the number of time the source node and the target node were found within the same cluster, the number of time the source node was found without the target node, ...

24 Pathway extraction tools

24.1 Using pathway extraction tools

24.1.1 Listing tools and getting help

You can list available tools by typing:

```
java graphtools.util.ListTools
```

All tools provide a *-h* option to display help.

24.1.2 Abbreviating tool names

The command line tool names may be simplified by setting aliases. For example, in the bash shell:

```
alias Pathfinder="java graphtools.algorithms.Pathfinder"
```

allows to type:

```
Pathfinder -h
```

instead of:

```
java graphtools.algorithms.Pathfinder -h
```

24.1.3 Increasing JVM memory

For large graphs, you may need to increase the memory allocated to the java virtual machine. You can do so by specifying the *-Xmx* option.

Example:

```
java -Xmx800m graphtools.algorithms.Pathfinder -h
```

24.2 Obtaining metabolic networks

24.2.1 Downloading MetaCyc and KEGG generic metabolic networks from the NeAT web server

Metabolic networks can be downloaded from the NeAT web server. Go to the menu entry “Path finding and pathway extraction”, open the “Pathway extraction” page and click on “More networks can be downloaded here.” This will open a table with tab-delimited generic MetaCyc and KEGG networks.

24.2.2 Building KEGG generic metabolic networks

Reaction network

To build the directed reaction network, type:

```
java -Xmx800m graphtools.parser.KeggLigandDataManager -m
```

The network is stored in the current directory.

The execution of this command takes quite long, because it fetches the reaction and compound files from KEGG's ftp repository at ftp.genome.jp. To get these files, the ***KeggLigandDataManager*** requires ***wget*** to be installed and in your path. ***wget*** is freely available from http://www.gnu.org/software/wg

Alternatively, you may first download the reaction and compound files yourself from the KEGG ftp server. Type in your browser (or in your favourite ftp client):
ftp://anonymous@ftp.genome.jp/pub/kegg/ligand/compound/compound
and save the compound file into *$RSAT/data/KEGG/KEGG_LIGAND*. Do the same for the reaction file at
ftp://anonymous@ftp.genome.jp/pub/kegg/ligand/reaction/reaction.
Then you can run the command above to generate the reaction network.

RPAIR network

To construct the undirected RPAIR network, type:

```
java -Xmx800m graphtools.parser.KeggLigandDataManager -s -u
```

Creating the RPAIR network will also create the *rpairs.tab* file, which can be placed in the KEGG directory for later use by typing:

```
cp $RSAT/data/KEGG/KEGG_LIGAND/rpairs.tab $RSAT/data/KEGG/rpairs.tab
```

An older version of this file is also available from the ***NeAT*** web server in the data/KEGG directory.

Reaction-specific RPAIR network

For the reaction-specific undirected RPAIR network, type:

```
java -Xmx800m graphtools.parser.KeggLigandDataManager -t -u
```

24.2.3 Building KEGG organism-specific metabolic networks

The MetabolicGraphProvider tool allows you to merge KEGG KGML files into a metabolic network specific to a set of organisms.

Prerequisites

You may first create the list of available KEGG organisms:

```
java -Xmx800m graphtools.parser.MetabolicGraphProvider -O
```

This command will create the file *Kegg_organisms_list.txt* in the current directory. Since this file is needed by the ***MetabolicGraphProvider***, you may copy it to its default location:

```
cp Kegg_organisms_list.txt \$RSAT/data/KEGG/Kegg_organisms_list.txt
```

Alternatively, you may obtain an older version of this file from the ***NeAT*** web server in the data/KEGG directory.

Creating an organism-specific reaction network for E. coli

The command below builds the *E. coli*-specific metabolic reaction network from its KGML files:

```
java -Xmx800m graphtools.util.MetabolicGraphProvider -i eco -o ecoNetwork.tab
```

The KGML files are automatically obtained from the current KEGG database (which may take very long). Alternatively, they can be downloaded manually from http://www.genome.jp/kegg/xml/. If downloaded manually, all organism-specific KGML files have to be placed in a folder named with the organism's KEGG abbreviation (e.g.*eco* for *E. coli*). The folder should be located in the *$RSAT/-data/KEGG* directory.

We can also merge the KGML files of several organisms into one network and apply some filtering as follows (in one line):

```
java -Xmx800m graphtools.util.MetabolicGraphProvider -i ecv/eco -o \
eco_ecv_Network.tab -c C00001/C000002/C00003/C00004/C00005/C00006/C00007/C00008
```

This command will construct a merged metabolic network from two *E. coli* strains (*Escherichia coli K-12 MG1655* and *Escherichia coli O1 (APEC)*) and in addition filter out some highly connected compounds (water, ATP, NAD+, NADH, NADPH, NADP+, oxygen and ADP).

24.2.4 Building metabolic networks from biopax files

Several metabolic databases store their data in biopax format (http://www.biopax.org/), e.g. BioCyc and Reactome. You can create a metabolic network from a biopax file using the ***GDLConverter***.

For instance, you may download the lysine biosynthesis I pathway from http://metacyc.org/ in biopax format and save it into a file named *lysine_pwy1.xml*. You can then obtain a tab-delimited metabolic network from this file using the command below (in one line). Note that the metabolic network preserves the reaction directions indicated in the biopax file, that is irreversible reactions are kept.

```
java graphtools.util.GDLConverter -i lysine_pwy1.xml
-o lysine.txt -O tab -I biopax -b -d
```

Option *-O* indicates the output format (tab-delimited), *-I* specifies the input format (biopax in this case), *-b* flags that attributes required for the metabolic format should be set and *-d* tells the program to construct a directed network.

The ***GDLConverter*** may be applied in general to interconvert networks in different formats.

24.3 Finding k-shortest paths

Pathways may be extracted from metabolic networks by enumerating the *k*-shortest paths between a set of source compounds/reactions and a set of target compounds/reactions.

In metabolic networks, some compounds such as ATP or NADPH are involved in a large number of reactions, thus acting as shortcuts for the path finding algorithm. However, paths crossing these highly connected compounds are not biochemical relevant. In order to prevent the path finding algorithm to traverse these compounds, the metabolic network should be weighted.

For example, assume you have generated (24.2.2) or downloaded (24.2.1) a KEGG RPAIR network stored in the file *KEGG_RPAIR_undirected.txt*. Given this network, we can list the three highest-ranked lightest paths between aspartate (KEGG identifier: C00049) and lysine (KEGG identifier: C00047) with the command below (in one line):

```
java -Xmx800m graphtools.algorithms.Pathfinder -g KEGG_RPAIR_undirected.txt
\
-s C00049 -t C00047 -y con -b -r 3 -f tab
```

where option *-s* specifies the source node (more than one can be given), *-t* the target node (as for the source, more than one target can be specified),*-f* indicates the format of the input network (tab-delimited), *-r* indicates the rank, option-y gives the weight policy to be applied (con sets the weight of compounds to their degree and the weight of reactions to one) and *-b* flags that the input network is metabolic.

This command will yield the following output (with KEGG RPAIR version 49.0):

```
INFO: Pathfinder took 5014 ms to perform its task.
; Experiment exp_0
; Pathfinding results
; Date=Fri Apr 30 16:34:27 CEST 2010
; ==============================
; INPUT
; Source=[C00049]
; Target=[C00047]
; Graph=KEGG_RPAIR_undirected.txt
; Directed=false
; Metabolic=true
; RPAIR graph=true
; CONFIGURATION
; Algorithm=rea
; Weight Policy=con
; Maximal weight=2147483647
; Exclusion attribute=ExclusionAttribute
; Rank=3
; REA timeout in minutes=5
; EXPLANATION OF COLUMNS
; Start node=given start node identifier
; End node=given end node identifier
; Path=path index
; Rank=rank of path (paths having same weight have
the same rank, though their step number might differ)
; Weight=weight of path (sum of edge weights)
; Steps=number of nodes in path
; Path=sequence of nodes from start to end node that forms the path
; ==============================
#start  end     path    rank    weight  steps   path
C00049  C00047  1       1       122.0   15      C00049->RP00932->C03082
->RP02107->C00441->RP02109->C03340->RP00740->C03972->RP03970->C03871
->RP02474->C00680->RP00907->C00047
C00049  C00047  2       2       126.0   15      C00049->RP00932->C03082
->RP02107->C00441->RP02109->C03340->RP00740->C03972->RP11205->C00666
->RP02449->C00680->RP00907->C00047
C00049  C00047  3       3       134.0   11      C00049->RP00116->C00152
```

```
->RP06538->C00151->RP01393->C00405->RP07206->C00739->RP00911->C00047
C00049  C00047  4       4       143.0   13      C00049->RP03035->C04540
->RP01395->C00152->RP06538->C00151->RP01393->C00405->RP07206->C00739
->RP00911->C00047
```

The format of the output can be changed to output the path list as a network. This network can then be visualized using the ***PathwayDisplayer*** as explained in section 24.5.4.

To output the path list as a network in gml format, run the following command (in one line):

```
java -Xmx800m graphtools.algorithms.Pathfinder -g KEGG_RPAIR_undirected.txt
-s C00049 -t C00047 -y con -b -r 3 -f tab -T pathsUnion -O gml
-o asp_lys_paths.gml
```

The file *asp_lys_paths.gml* created in the current directory contains the network in gml format.

24.4 Linking genes to reactions

The main application of pathway extraction is to interpret a set of associated enzyme-coding genes. An association can for example be co-expression in a microarray, co-regulation in an operon or regulon or co-occurrence in a phylogenetic profile.

In this section, we will see how to link enzyme-coding genes to their reactions. This is not a straightforward task, as an N:N relationship exists between genes, EC numbers, reactions and reactant pairs.

24.4.1 Prerequisites

In order to link genes to reactions, the metabolic database needs to be installed. The installation of this database is described in chapter “Metabolic Pathfinder and Pathway extraction" in the ***NeAT***web server install guide, which is available from the ***NeAT***web server download section.

24.4.2 Linking genes of the isoleucine-valine operon to reactions

The isoleucine-valine operon (RegulonDB identifier: ilvLG_1G_2MEDA) in *Escherichia coli* is known to contain enzymes of the isoleucine and valine biosynthesis pathway.

It consists of the following genes:

```
ilvL ilvG_1 ilvG_2 ilvM ilvE ilvD ilvA
```

These genes can be linked to KEGG reactant pairs using the command below (in one line):

```
java graphtools.util.SeedConverter -i ilvL/ilvG_1/ilvG_2/ilvM/ilvE/ilvD/ilvA
-I string -O eco -o ilv_operon_seeds.txt -r
```

Option *-r* flags that genes should be mapped to (main) reactant pairs, *-O* specifies the source organism of the genes, *-i* lists the genes and *-I* specifies the input format.

24.5 Predicting metabolic pathways

Given a set of seeds (compounds or reactions/reactant pairs) and a metabolic network, the task of the pathway extraction tool is to extract a metabolic pathway that connects these seeds in the metabolic

network. The tool is quite generic and can be applied to any network and seed node set. However, it has been tailored to metabolic pathway prediction.

24.5.1 Predicting a metabolic pathway for the isoleucine-valine operon

Assume you have generated the seed input file from section 24.4.2 and the KEGG RPAIR graph as described in section 24.2.2. The KEGG RPAIR graph is assumed to be stored in a tab-delimited file named *KEGG_RPAIR_undirected.txt*. Then we can predict the pathway for the genes in the isoleucine-valine operon with the following command (in one line):

```
java -Xmx800m graphtools.algorithms.Pathwayinference -g
KEGG_RPAIR_undirected.txt -i ilv_operon_seeds.txt -b -f tab
-y con -E Result -a takahashihybrid -U -o ilv_predicted_pathway.tab
```

where option *-b* specifies that the network is a metabolic network, *-f* indicates the input network format (tab-delimited), *-a* specifies the algorithm to be used and *-y* indicates the weight policy to be applied (con stands for connectivity, which means that compound nodes receive a weight corresponding to their degree). Option *-E* is used to indicate the name of the folder where results are stored. This is especially useful when several predictions are carried out in a row, because the output file in this case reports the merged pathway. In the example above, the result folder serves to store the properties of the predicted pathway (obtained with option *-U*).

A variant of the pathway extraction exploits the fact that we work with the KEGG RPAIR graph, which allows us to link adjacent main reactant pairs (i.e. reactant pairs sharing a compound). This is done in a preprocessing step (option *-P*):

```
java -Xmx800m graphtools.algorithms.Pathwayinference -g
KEGG_RPAIR_undirected.txt -i ilv_operon_seeds.txt -b -f tab -y con -P
-a takahashihybrid -o ilv_predicted_pathway_preprocessed.tab
```

24.5.2 Mapping reference pathways onto the predicted pathway

The predicted metabolic pathway can be mapped to reference pathways stored in the metabolic database. This can be done as follows:

```
java graphtools.util.MetabolicPathwayProvider -i ilv_predicted_pathway.tab
-I tab -D KEGG -o ilv_predicted_pathway_mapped.tab
```

where option *-D* indicates that reference pathways should be taken from KEGG and *-I* indicates the input format of the pathway. In the output pathway, nodes mapping to reference pathways are annotated with a color and the name of the corresponding reference pathway. The program also outputs the color-code of mapping reference pathways:

```
INFO: Legend
BurlyWood: Valine,_leucine_and_isoleucine_biosynthesis
orange: no match to any reference pathway
```

24.5.3 Annotating the predicted pathway

The nodes of a predicted metabolic pathway can be labeled with names (compounds), EC numbers (reactions) and genes (reactions). The requires the metabolic database to be installed (see 24.4.1).

The command below annotates the metabolic pathway named *ilv_predicted_pathway.tab* and colors its seed nodes (stored in the seed node file *ilv_operon_seeds.txt*) in blue:

```
java graphtools.util.GraphAnnotator -i ilv_predicted_pathway.tab -I tab
-o ilv_predicted_pathway_annotated.tab -O tab -k -b
-F ilv_operon_seeds.txt
```

Option *-k* tells ***GraphAnnotator*** to associate EC numbers to KEGG genes using the current KEGG database, -b indicates that the pathway is a metabolic pathway, -I specifies the input format of the pathway to be annotated (tab-delimited) and -F indicates the location of the seed node file.

24.5.4 Visualizing the predicted pathway

The visualization of a pathway requires ***graphviz*** to be installed, which is available here http://www.graphviz.or

With ***graphviz*** installed, the pathway can be visualized as follows:

```
java graphtools.util.PathwayDisplayer -i ilv_predicted_pathway_annotated.tab
-I tab -p
```

Option *-p* tells ***PathwayDisplayer*** to generate the image with ***graphviz***, *-I* indicates the input format of the pathway to be displayed (tab-delimited).

25 References

1. van Helden, J., Andre, B. & Collado-Vides, J. (1998). Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. J Mol Biol 281(5), 827-42.

2. van Helden, J., Andre, B. & Collado-Vides, J. (2000). A web site for the computational analysis of yeast regulatory sequences. Yeast 16(2), 177-187.

3. van Helden, J., Olmo, M. & Perez-Ortin, J. E. (2000). Statistical analysis of yeast genomic downstream sequences reveals putative polyadenylation signals. Nucleic Acids Res 28(4), 1000-1010.

4. van Helden, J., Rios, A. F. & Collado-Vides, J. (2000). Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. Nucleic Acids Res. 28(8):1808-18.

5. van Helden, J., Gilbert, D., Wernisch, L., Schroeder, M. & Wodak, S. (2001). Applications of regulatory sequence analysis and metabolic network analysis to the interpretation of gene expression data. Lecture Notes in Computer Sciences 2066: 155-172.

6. van Helden, J. 2003. Prediction of transcriptional regulation by analysis of the non-coding genome. Current Genomics 4: 217-224.

7. van Helden, J. 2003. Regulatory sequence analysis tools. Nucleic Acids Res 31: 3593-3596.

8. van Helden, J. 2004. Metrics for comparing regulatory sequences on the basis of pattern counts. Bioinformatics 20: 399-406.

9. Simonis, N., J. van Helden, G.N. Cohen, and S.J. Wodak. 2004. Transcriptional regulation of protein complexes in yeast. Genome Biol 5: R33.

10. Simonis, N., S.J. Wodak, G.N. Cohen, and J. van Helden. 2004. Combining pattern discovery and discriminant analysis to predict gene co-regulation. Bioinformatics.