# Regulatory Sequence Analysis Tools
# Tutorial
# Command-line utilization of the tools

Jacques van Helden
*jvanheld@scmbb.ulb.ac.be*
*http://www.scmbb.ulb.ac.be/~jvanheld/*
Service de Conformation des Macromolécules Biologiques et de Bioinformatique,
Université Libre de Bruxelles,
Campus Plaine, CP 263, Boulevard du Triomphe, B-1050 Bruxelles, Belgium.
Tel: +32 2 650 2013 - Fax: +32 2 650 5425

October 4, 2006

# Contents

# 1 Warning

This tutorial is under construction. Some sections are still to be written, and are only mentioned as a title without any following text. The tutorial will be progressively completed. We provided it as it is, because it can already provide a good starter for the analysis of regulatory sequences.

# 2 Introduction

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

***RSAT***is 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://www.ucmb.ulb.ac.be/bioinformatics/rsa-tools/*

All programs can also be used directly from the unix shell. The shell access is less intuitive than the web interface, but is very convenient for automatizing repetitive tasks.

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

# 3 Accessing the programs

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

For this tutorial, let us assume that ***RSAT***is installed in the directory `/usr/local/rsa-tools`

1. Open a telnet or ssh session to your account.

2. If your default shell is **tcsh**, type the following commands.

   ```
   set RSAT=/usr/local/rsa-tools
   set path=($path $RSAT/perl-scripts)
   set path=($path $RSAT/perl-scripts/parsers)
   set path=($path $RSAT/bin/)
   rehash
   ```

   If you are using a different shell (e.g. bash), you might need a slightly different command to obtain the same result. See you system manager in case of doubt.

3. The previous step should have included all the ***RSAT***programs in your path. To check if it worked, just type:

   ```
   random-seq -l 350
   ```

   If your configuration is correct, this command should return a random sequence of 350 nucleotides.

You are now able to use any program from the ***RSAT***package, untill you quit your telnet session. It is however not very convenient to set the path manually each time you open a new connection. You can modify your default configuration by including the above commands in the file `.personal-cshrc` in the root of your home directory. If you don't know how to modify this file, see the system administrator.

# 4 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 `-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.

# 5 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 a series of genes, where regulatory elements are generally found, at least in microbial organisms.

### 5.0.1 Retrieving a single upstream sequence

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

```
retrieve-seq -type upstream -org Escherichia_coli_K12 \
    -q metA -from -200 -to -1
```

This command retrieves a 200 bp upstream sequence for the gene metA of *Escherichia coli*. Note the negative coordinates, indicating the upstream side. Also note that all coordinates are calculated starting relative to the start codon (position 0 is the A from the start ATG).

### 5.0.2 Combining upstream and coding sequence

For 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. `retrieve-seq` allows you to extract a sequence that combines an upstream and a coding segment. Try:

```
retrieve-seq -type upstream -org Escherichia_coli_K12 \
    -q metA -from -200 -to 49
```

### 5.0.3 Retrieving a few upstream sequences

The option `-q` can be used iteratively in a command to retrieve sequences for several genes.

```
retrieve-seq -org Escherichia_coli_K12 \
    -from -200 -to 49 -q metA -q metB -q metC
```

### 5.0.4 Retrieving many 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.

To create a test file, you can execute the following steps:

1. to create a new file, call the standard unix command

   ```
   cat > PHO_genes.txt
   ```

2. You can now type a list of gene names, for example:

```
PHO5
PHO8
PHO11
PHO81
PHO84
```

3. Once you have finished typing gene names, press `Ctrl-D`

4. 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 genes.

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

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

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

Check the sequence file:

```
more PHO_up800.fasta
```

#### 5.0.5 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_K12 \
    -from 0 -to 2 \
    -all -format wc -nocomments -label id,name \
    -o Escherichia_coli_K12_start_codons.wc
```

Check the result:

```
more Escherichia_coli_K12_start_codons.wc
```

### 5.0.6 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
hisR
```

The default behaviour will return 200bp for each gene.

```
retrieve-seq -type upstream -org Escherichia_coli_K12 \
    -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_K12 \
    -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).

### 5.0.7 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 returns all the stop codons for *Escherichia coli*.

```
retrieve-seq -type downstream -org Escherichia_coli_K12 \
    -from -2 -to 0 \
    -all -format wc -nocomments -label orf_gene \
    -o Escherichia_coli_K12_stop_codons.wc
```

### 5.0.8 Retrieving random sequences

`retrieve-seq` can also be used to retrieve random sequences, with two flavours :

- retrieving upstream, downstream or ORF sequence for a random selection of ORFs
- retrieving random genomic fragments

In both cases, the number of sequences to retrieve is specified with the option `-random`, followed by a natural number.

The type of sequence is specified with the option `-type`, as usual. Random genomic segments can be obtained by specifying "random" as sequence type.

For example, To retrieve upstream sequences for a random selection of 100 yeast ORFs :

```
retrieve-seq -org Saccharomyces_cerevisiae \
    -randsel 100 -type upstream
```

Another example: to retrieve 100 random genomic segments of size 200 in *Saccharomyces cerevisiae* :

```
retrieve-seq  -org Saccharomyces_cerevisiae \
    -randsel 100 -type random
```

# 6 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. For yeast regulatory elements, string-based approaches give excellent results. The advantages:

- Simple to use
- Deterministic (if you run it repeatedly, you always get the same result)
- Easily parametrizable
- Easy to interpret
- Fast
- Able to return a negative answer: if no motif is significant, the programs return an empty list of motifs. This is particularly important to reduce the rate of false positive.

Matrix-based approach can provide a more refined description of motifs presenting a high degree of degeneracy. The 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. Last problem: the result is not easily interpretable because the programs always return an answer.

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). The ideal would thus be to combine string-based pattern discovery and matrix-based pattern matching.

## 6.1 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:

```
ls -1
```

You should see the following file list:

```
PHO_genes.txt
PHO_up800.fasta
Escherichia_coli_K12_start_codons.wc
Escherichia_coli_K12_stop_codons.wc
```

## 6.2 oligo-analysis

The program `oligo-analysis` is the simplest pattern discovery program. It counts the number of occurrences of all oligonucleotides (word) of a given length (typically 6), and compares, for each word, the observed and expected occurrences, and return words with a significant level of over-representation.

Despite its simplicity, this program already returns good results for many families of co-regulated genes in yeast.

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 5 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.

### 6.2.1 Counting word occurrences and frequencies

Try the following command:

```
oligo-analysis -i Escherichia_coli_K12_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 (verbose) and store the result in a file:

```
oligo-analysis -i Escherichia_coli_K12_start_codons.wc \
    -format wc -l 3 -1str \
    -return occ,freq -v \
    -o Escherichia_coli_K12_start_codon_frequencies
```

Read the result file:

```
more Escherichia_coli_K12_start_codon_frequencies
```

Note the effect of the verbose. You receive information about sequence length, number of possible oligonucleotides, the content of the output columns, ...

**Exercise:** check the frequencies of *E.coli* stop codons.

### 6.2.2 Pattern discovery in yeast upstream regions

Try the following command:

```
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
```

Note that the return fields ("occ", and "proba") are separated by a comma *without* space.

Call the on-line help to understand the meaning of the parameters.

```
oligo-analysis -h
```

Note that we used pre-calibrated tables as estimators of expected word frequencies. These tables have been previously calculated (with oligo-analysis) by counting hexanucleotide frequencies in the whole set of yeast upstream regions. Our experience is that these frequencies are the optimal estimator for discovering regulatory elements in upstream sequences of co-regulated genes.

Look the result file:

```
more PHO_up800_6nt_2str_ncf_sig0
```

A few questions:

1. How many hexanucleotides can be formed with the 4-letter alphabet A,T,G,C ?
2. How many possible oligonucleotides are indicated ? Is it the number you would expect ? Why ?
3. How many patterns have been selected as significant ?
4. Do you see some similarity between some of the selected patterns ?

### 6.2.3 Answers

1. $4^6 = 4,096$
2. $2,080$. This is due to the fact that the analysis was performed on both strands. Each oligonucleotide is thus equivalent to its reverse complement.
3. $9$
4. some pairs of words are mutually overlapping (e.g. `cACGTG` and `ACGTGc`).

### 6.2.4 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_6nt_2str_ncf_sig0 \
    -v -sc 7 -subst 1 \
    -2str -o PHO_up800_6nt_2str_ncf_sig0.assemb
```

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

```
pattern-assembly -h
```

Look at the result. There are two alignments (with two contigs), and two isolated patterns. Each alignment is made of strongly overlapping patterns. The first alignment (cgcacgtgcg) corresponds to the high affinity binding site for Pho4p, the protein controlling transcriptional response to Phosphate in yeast. the second alignment (cgcacgttt) corresponds to the medium affinity binding site for Pho4p. Medium affinity binding sites have been shown to participate in the transcriptional response to some PHO genes.

```
more PHO_up800_6nt_2str_ncf_sig0.assemb
```

### 6.2.5 Suboptimal settings

This chapter only aims at emphasizing how crucial is the choice of appropriate statistical parameters. We saw above that the optimal parameters give 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: each word is considered equiprobable. For this, we simply suppress the options `-bg upstream -org yeast` fom the above commands. We also omit to specify the output file, so results will immediately appear on the screen.

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

Note that

- The number of selected motifs is higher (27) than in the previous trial
- The most significant motifs have nothing to do with Pho4p binding sites. All these false positives are A-rich motifs (or T-rich, since we are grouping patterns with their reverse-complement).

- Two patterns (`acgttt` and `acgtgc`) are selected which are related to Pho4p binding site. However, they come at the 12th and 14th positions only.

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

```
oligo-analysis -i PHO_up800.fasta -format fasta -v \
    -l 6 -2str  -return occ,proba -lth occ_sig 0 -sort \
    | pattern-assembly -2str -sc 7 -subst 1 -v
```

On unix systems, this special 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 most significant patterns are associated to the poly-A (aaaaaa) contig. The true positive come isolated. Due to the bad choice of expected frequencies (all hexanucleotides were considered equiprobable here), regulatory sites were lost within a majority of false positive, and their description is much less accurate than with the option `-bg upstream`.

***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`, 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.

```
oligo-analysis -markov 0 \
    -i PHO_up800.fasta -format fasta \
    -l 6 -lth occ_sig 0 -sort \
    -2str -return occ,proba \
    | pattern-assembly -2str -sc 7 -subst 1 -v

oligo-analysis -markov 1 \
    -i PHO_up800.fasta -format fasta \
    -2str -return occ,proba \
    -l 6 -lth occ_sig 0 -sort \
    | pattern-assembly -2str -sc 7 -subst 1 -v

oligo-analysis -markov 2 \
    -i PHO_up800.fasta -format fasta \
    -2str -return occ,proba \
    -l 6 -lth occ_sig 0 -sort \
    | pattern-assembly -2str -sc 7 -subst 1 -v

oligo-analysis -markov 3 \
    -i PHO_up800.fasta -format fasta \
    -2str -return occ,proba \
    -l 6 -lth occ_sig 0 -sort \
```

```
    | pattern-assembly -2str -sc 7 -subst 1 -v

oligo-analysis -markov 4 \
    -i PHO_up800.fasta -format fasta \
    -2str -return occ,proba \
    -l 6 -lth occ_sig 0 -sort \
    | pattern-assembly -2str -sc 7 -subst 1 -v
```

***Remarks***

- Markov 0 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.

## 6.3 dyad-analysis

## 6.4 gibbs motif sampler (program developed by Andrew Neuwald)

## 6.5 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.

### 6.5.1 Getting help

As for RSAT programs, there are two ways to get help from Jerry Hertz' proigrams: 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
```

### 6.5.2 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.fasta -from fasta -to wc \
    -o PHO_up800.wc
```

### 6.5.3 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.wc -A a:t c:g -c2 -N 10
```

The two main options of this command 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.

Notice that several matrices are returned. Each matrix is followed by the alignment of the sites on which it is based. Notice 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.

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.

To save the result in a file, you can use the symbol "greater than" (>) which redirects the output of a program to a file.

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

(this may take a few minutes)

Once the task is achieved, check the result.

```
more PHO_consensus
```

# 7 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*).

## 7.1 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.

### 7.1.1 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

### 7.1.2 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".

### 7.1.3 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.

### 7.1.4 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).

### 7.1.5 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`.

### 7.1.6 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
```

### 7.1.7 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.

### 7.1.8 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
```

### 7.1.9 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.

#### 7.1.10 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
```

### 7.2 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.

#### 7.2.1 Getting help

help can be obtained with the two usual options.

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

#### 7.2.2 Matrix conversion

Patser expects as input a matrix like the 4 matrices we obtained above with *consensus*. The output from *consensus* can however not be used directly because it contains several matrices, and a lot of additional information. One possibility is to copy-paste the matrix of interest to 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 -format consensus -i PHO_consensus \
    -return counts -o PHO_matrix

more PHO_matrix
```

Note that the program *convert-matrix* includes many other output options, fo example you can obtain the degenerate consensus from a matrix with the following options.

```
convert-matrix -format consensus -i PHO_consensus \
    -return consensus
```

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

```
convert-matrix -h
```

#### 7.2.3 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_matrix -f PHO_up800.wc -A a:t c:g -c -ls 9
```

#### 7.2.4 Detecting Pho4p sites in all upstream regions

We will now match our PHO matrix against the whole set of upstream regions from the 6200 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*. To avoid occupying too much space on the disk, we could 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 gene  \
    | patser -m PHO_matrix -ls 9 -A a:t c:g \
    > PHO_matrix_matches_allup.txt

more PHO_matrix_matches_allup.txt
```

# 8 Drawing graphs

## 8.1 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).

### 8.1.1 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.

### 8.1.2 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).

### 8.1.3 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.

### 8.1.4 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.

### 8.1.5 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.

### 8.1.6 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.

### 8.1.7 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
```

### 8.1.8 Feature converters

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

***RSAT***includes 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

## 8.2 XYgraph

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

### 8.2.1 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.

# 9 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.

## 9.1 Sequences with identically and independently distributed (IID) nucleotides

```
random-seq -l 200 -r 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
```

## 9.2 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 -r 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
```

## 9.3 Introduction to Markov order

Markov chain models involve local dependencies between successive residues.

A Markov model of order $m$ means that the probability of the residue at each position $i$ of the sequence depends on the $m$ preceding residues.

***RSAT***includes organism-specific Markov models, stored in the form of oligonucleotide frequency tables (see chapter on pattern discovery). for example, the calibration tables for Escherichia coli K12 can be found in the following directory.

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

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_K12/oligo-frequencies/

## Have a look at the content of the 4nt frequency file
zless zless 4nt_upstream-noorf_Escherichia_coli_K12-1str.freq.gz
```

However, Markov models are described by transition frequencies $P(r|W_m)$, i.e. the probability to osberve a given residue $r$ given the preceding word $W_m$ of size $m$.

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

We can illustrate this convertion by converting the table of dinucleotide frequencies into a transition matrix of first order. each row of the matrix indicates the prefix $W_m$, and each column the following residue $r$. For a Markov model of order 1, the prefixes are single residues.

```
convert-background-model \
  -i 2nt_upstream-noorf_Escherichia_coli_K12-1str.freq.gz  \
  -from oligo-analysis -to tab
```

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

```
convert-background-model \
  -i 3nt_upstream-noorf_Escherichia_coli_K12-1str.freq.gz  \
  -from oligo-analysis -to tab
```

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

## 9.4 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 -r 20 \
  -expfreq $RSAT/data/genomes/Escherichia_coli_K12/oligo-frequencies/3nt_upstream-noorf_E
  -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 -r 20 \
  -org Escherichia_coli_K12 -bg upstream-noorf -ol 3 \
  -o rand_L200_N20_mkv2.fasta
```

# 10 Pattern comparisons

**TO BE WRITTEN**

## 10.1 Comparing patterns with patterns

`compare-patterns`

## 10.2 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 -lth weight 6 \
  -o dyads_vs_RegulonDB.tab
```

# 11 Comparing classes, sets and clusters

**TO BE WRITTEN**

# 12 Comparative genomics

## 12.1 Genome-wise comparison of protein sequences

In this section, I 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. The ***RSAT***distribution does thus not include the complete comparison of all genomes against all other ones. The current ***RSAT***distribution includes the complete comparisons for some model genomes (*Escherchia coli K12* versus all bacteria, *Saccharomyces cerevisiae* against all Fungi, ...), but you might need to perform additional comparisons for your own purpose.

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

**TO BE WRITTEN**

## 12.2 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 Gammaproteobacteria. We will then refine the query to extract putative orthologs.

### 12.2.1 Getting genes by similarities

```
get-orthologs -v 1 -org Escherichia_coli_K12 \
  -taxon Gammaproteobacteria \
  -q lexA -o lexA_orthologs_Gammaproteobacteria.tab
```

The result file is a list of all the Gammaproteobacterial 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_K12    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.

### 12.2.2 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_K12 \
  -taxon Gammaproteobacteria \
  -q lexA -o lexA_orthologs_Gammaproteobacteria.tab \
  -return ident,ali_len,e_value,rank
```

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_K12    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.

### 12.2.3 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_K12 \
  -taxon Gammaproteobacteria \
  -q lexA -o lexA_orthologs_Gammaproteobacteria_bbh.tab \
  -return ident,ali_len,e_value,rank \
  -uth rank 1
```

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_K12    b4043   100.00  202     2e-111  1
...
```

### 12.2.4 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_K12 \
  -taxon Gammaproteobacteria \
  -q lexA -o lexA_orthologs_Gammaproteobacteria_id30_len50_eval-10.tab \
  -return ident,ali_len,e_value,rank \
  -lth ident 30 -lth ali_len 50 -uth e_value 1e-10
```

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_K12 \
  -taxon Gammaproteobacteria \
  -q lexA -o lexA_orthologs_Gammaproteobacteria_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
```

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 Gammaproteobacteria.

- 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.

## 12.3 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_Gammaproteobacteria_bbh_id30_len50_eval-10.tab \
  -o lexA_orthologs_Gammaproteobacteria_up-noorf.fasta
\end{small}
```

## 12.4 Detection of phylogenetic footprints

**TO BE WRITTEN**

```
dyad-analysis  -v 1 \
  -i lexA_orthologs_Gammaproteobacteria_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_Gammaproteobacteria_up-noorf_dyads-2str-noov.tab
```

## 12.5 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_Fungi_bbh.tab

## Convert ortholog table into a profile table
## with the IDs of the putative orthologs
convert-classes -v 2 \
  -i Saccharomyces_cerevisiae_Fungi_bbh.tab  \
  -from tab -to profiles \
  -ccol 2 -mcol 3 -scol 1 -null "<NA>" \
  -o Saccharomyces_cerevisiae_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_Fungi_bbh.tab  \
  -from tab -to profiles \
  -ccol 2 -mcol 3 -scol 4 -null "<NA>" \
  -o Saccharomyces_cerevisiae_Fungi_phyloprofiles_evalue.tab
```

## 12.6 Detecting pairs of genes with similar phylogenetic profiles

**TO BE WRITTEN**

```
compare-profiles -v 1 \
  -i profiles.tab \
  -return counts,jaccard,hyper,entropy \
  -o phyloprof_gene_pairs.tab
```

# 13 Complete 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.

## 13.1 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 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 1), or by their systematic identifier (MET family in Table 1).

## 13.2 Example of utilization

Let us assume that the file displayed in Table 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.

## 13.3 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/
```

| ; 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 1: Example of family file.

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.

## 13.4 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.

## 13.5 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.

## 13.6 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.scmbb.ulb.ac.be/rsat/data/published_data/Simonis_Bioinformatics_2004/*.

# 14 Utilities

## 14.1 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_K12 -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_K12 \
    -q 'met.*' -q 'thr.*' -q 'lys.*'
```

## 14.2 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.

# 15 Installing organisms

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

## 15.1 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/*)
- ENSEMBL (*http://www.ensembl.org/*)
- The EBI genome directory (*ftp://ftp.ebi.ac.uk/pub/databases/genomes/Eukaryota/*)

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

## 15.2 Requirement : wget

The download of genomes relies on the application ***wget***, which is part of linux distribution. ***wget*** is a "web aspirator", which allows to downlaod 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.

## 15.3 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 one of the smallest sequenced genome: *Mycoplasma genitalium*.

To download the genome in your ***RSAT***folder, tye the following command.

```
cd $RSAT
wget -rNL http://rsat.scmbb.ulb.ac.be/rsat/data/genomes/Mycoplasma_genitalium/
```

This will create a local mirror of the ***RSAT***data repository. You can check the result by typing.

```
ls -l $RSAT/rsat.scmbb.ulb.ac.be/rsat/data/genomes/Mycoplasma_genitalium/
```

When the download is complete, move the newly transferred genome to the data directory of your ***RSAT***installation.

```
mv $RSAT/rsat.scmbb.ulb.ac.be/rsat/data/genomes/Mycoplasma_genitalium \
    $RSAT/data/genomes/
```

You need now to declare the newly installed organism.

```
install-organism -v 1 -task config \
    -org Mycoplasma_genitalium -up_from -400 -up_to -1
```

You can now check the configuration file.

```
tail -20 $RSAT/data/supported_organisms.pl
```

If the installation was successfull, you should see something like this :

```
#### Mycoplasma_genitalium      Mycoplasma genitalium   2006/01/04 22:08:42
$supported_organism{'Mycoplasma_genitalium'}->{'name'} = "Mycoplasma genitalium";
$supported_organism{'Mycoplasma_genitalium'}->{'data'} = "$RSA/data/genomes/Mycoplasma_ge
$supported_organism{'Mycoplasma_genitalium'}->{'last_update'} = "2006/01/04 22:08:42";
$supported_organism{'Mycoplasma_genitalium'}->{'features'} = "$RSA/data/genomes/Mycoplasm
$supported_organism{'Mycoplasma_genitalium'}->{'genome'} = "$RSA/data/genomes/Mycoplasma_
$supported_organism{'Mycoplasma_genitalium'}->{'seq_format'} = "filelist";
$supported_organism{'Mycoplasma_genitalium'}->{'taxonomy'} = "Bacteria; Firmicutes; Molli
$supported_organism{'Mycoplasma_genitalium'}->{'synonyms'} = "$RSA/data/genomes/Mycoplasm
$supported_organism{'Mycoplasma_genitalium'}->{'up_from'} = -400;
$supported_organism{'Mycoplasma_genitalium'}->{'up_to'} = -1;

return 1;
```

# 16 Installing genomes from their original source

The parsing of genomes from their original data sources is more tricky than the synchronization from the ***RSAT***server.

In principle, if you succeeded, with the protocol above, to obtain the genomes from the ***RSAT***server, you don't need to proceed to new installations yourself and you can skip the rest of this chapter.

## 16.1 The *RSAT*genome files

1. genome sequence
2. feature table
3. list of names/synonyms

### 16.1.1 Genome sequence

The genome must be in raw format (text files containing the sequence without any space or carriage return). If the organism contains several chromosomes, there should be one separate file per contig (chromosome).

In addition, the genome directory must contain one file listing the contig (chromosome) files. You can find an example in the directory *$RSAT/data/genomes/Saccharomyces_cerevisiae/genome/*.

### 16.1.2 Feature table

A feature-table giving the basic information about genes. This is a tab-delimited text file. Each row contains information about one gene. The columns contain the following information:

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.

### 16.1.3 Gene names (synonyms)

Optionally, you can provide a synonym file, which contains two columns:

1. ID. This must be one identifier found in the feature table
2. Synonym

Multiple synonyms can be given for a gene, by adding several lines with the same ID in the first column.

### 16.1.4 Example

```
cd $RSAT/data/genomes/Saccharomyces_cerevisiae/genome/

## The list of sequence files
cat contigs.txt

## The sequence files
```

```
ls -l *.raw

## The feature table
head -30 feature.tab

## The gene names/synonyms
head -30 feature_names.tab
```

## 16.2 Parsing genomes from NCBI/Genbank

The easiest way to install an organism in ***RSAT***is to download the complete genome files from the NCBI [1], and to parse it with the program ***parse-genbank.pl***.

### 16.2.1 Downloading genomes from NCBI/Genbank

***RSAT***includes 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 $HOME
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.

```
## Downloading one directory from NCBI Genbank
cd $HOME/downloads/
make one_genbank_dir GB_DIR=genomes/Bacteria/Mycoplasma_genitalium
```

### 16.2.2 Parsing genomes 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.

[1] *ftp://ftp.ncbi.nih.gov/genomes/*

## 16.3 Parsing genomes from EMBL files

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 [2].

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 *$HOME/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 $HOME/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
```

The following instruction will parse this genome.

```
parse-embl.pl -v 1 -i  $HOME/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*.

### 16.3.1 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/
```

[2] *http://natchaug.labri.u-bordeaux.fr/Genolevures/download.php*

There is one subdirectory per organism. For example, the yeast data is in *$RSAT/data/genomes/Saccharomyces_cerevis*
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
```

### 16.3.2 Updating the configuration file

The fact to add a directory is not sufficient for ***RSAT***to be aware of the new organism. For this, we must update the configuration file. We will also specify the default upstream sequence length. For a yeast (*Debaryomyces hansenii*), a good guess is 800bp (this is at least the value I chose for *Saccharomyces cerevisiae*).

```
install-organism -v 1 -org Debaryomyces_hansenii -task config -up_from -800

## Check the last lines of the configuration file
tail -15 $RSAT/data/supported_organisms.pl
```

### 16.3.3 Checking that the organism is installed properly

To check the installation, start by checking whether your newly installed now appears in the list of supported organisms.

```
supported-organisms
```

Will give you a list of installed organisms.

### 16.3.4 Retrieving some sequences

As soon as the configuration file has been updated, we should be in state to retrieve sequences for the newly installed genome. We will check this by retrieving a the start codons.

```
retrieve-seq -org Debaryomyces_hansenii -all -from 0 -to 2
```

### 16.3.5 Checking the start codons

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).

We will now analyze the trinucleotide composition of the start codons. In principle, al of them should be ATG for an eucaryote organism.

```
retrieve-seq -org Debaryomyces_hansenii -all -from 0 -to 2 \
    | oligo-analysis -l 3 -1str -v 1 -return occ,freq -sort
```

### 16.3.6 Checking the start and stop codons with *install-organisms*

The program ***install-organisms*** includes an option to automatically check all the start and stop codons from a parsed organism.

```
install-organism -v 1 -org Debaryomyces_hansenii -task start_stop
```

You can then check the composition of the start and stop codons.

```
cd $RSAT/data/genomes/Debaryomyces_hansenii/genome/
more Debaryomyces_hansenii_start_codon_frequencies
more Debaryomyces_hansenii_stop_codon_frequencies
```

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 frequent.

### 16.3.7 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***.

**Warning:** this task requires several hours of computation (a few hours for small bacterial genomes, and several days for the human genome).

```
install-organism -v 1 -org Debaryomyces_hansenii \
    -task allup,oligos,dyads,upstream_freq,clean
```

### 16.3.8 Installing a genome in your own account

We describe below how this information should be formatted to be used in rsa-tools.

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.

## 16.4 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.

# 17 Using RSAT web services

## 17.1 Introduction

***RSAT***facilities can be used as web services (*WS*), i.e. external developers (you) can use ***RSAT***facilities in their own code. One important advantage of web services is that they are using a standard communication interface between client and server, and the libraries exist in various languages (Perl, Python, java).

We explain below how to implement a WS client in Perl.

## 17.2 Requirements

Before using a WS client, You need to install the Perl module ***SOAP::Lite***. 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 administrato to install them for you.

## 17.3 Examples of WS client in Perl

We show hereafter some simple examples of

### 17.3.1 Retrieving sequences from RSATWS

### 17.3.2 Work flow using RSATWS

## 17.4 Examples of WS client in java

## 17.5 Full documentation of the RSATWS interface

# 18 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.

## 18.1 Some hints

### 18.1.1 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.

### 18.1.2 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.

# 19 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., Andr, 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.