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 1, 2004

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, gaphical 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/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 defaul 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 adiministrator.

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 progam.

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 star codon (position 0 is the A from the start ATG).

5

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 enames 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
```

6

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 orf_gene \
   -o Escherichia_coli_K12_start_codons.wc
```

Check the result:

```
more Escherichia_coli_K12_start_codons.wc
```

5.0.6 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 th 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
```

5.0.7 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 omtifs 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 on 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 disovery 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 may families of co-regulated genes in yeast.

In a first time, 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
```

Reaad 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 -thosig 0 -bg upstream \
    -org Saccharomyces_cerevisiae -sort               \
    -o PHO_up800_6nt_2str_ncf_sig0
```

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 may 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. there are strong mutual overlap between some words (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 ablt to extract a description of the regulatory motif over a larger width than 6 (by pattern assembly), and we got some decription of the degeneracy (the high and low affinity stes).

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 apper on the screen.

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

Note that

- The number of selected motifs is higher (27) than in the previous trial

- The most significant motifs have nothing to with Pho4p binding sites. All these false positive 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 -thosig 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

12

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 -thosig 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 -thosig 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 -thosig 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 -thosig 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 -thosig 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. te 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. Markov order 2 mises most of the patterns, and higher orders don't return any single significant word.

13

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 ths 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 chose 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 mtrices 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 "¿" 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 minuts)

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 -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 identiifier

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 -i PHO_up800.fasta -format fasta \
    -2str -p cacgtt -id 'Pho4p_site'
```

Notice that the strand colmn 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 iwith 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 egative value with the option origin. for example, -origin -100 would take as origin the 100th neucleotide 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 alos bind, with a medium affinity, CACGTT sites. The IUPAC code for partly specified nucleotides allows to represent any combination of nucleotids by a single letter.

17

A		(Adenine)
C		(Cytosine)
G		(Guanine)
T		(Trymine)
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 CACGTK 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]'
```

18

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 tpye 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 lank, in which case the pattern is used as identifier.

We can now use this file to search all matching psitions 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. Te 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).

19

```
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

Anothe rway 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 aplying multivariate statistics on the results (e.g. classificatying 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 extacts a matrix from various file formats, including consensus.

```
convert-matrix -format consensus -i PHO_consensus -o PHO_matrix

more PHO_matrix
```

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.

```
retrieve-seq -type upstream -from -1 -to -800  \
    -org Saccharomyces_cerevisiae \
    -all -format wc -label gene  \
    | patser -m PHO_matrix -l 9 -A a:t c:g
```

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 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 ***features-from-dnapat*** 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 -thosig 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             \
    -o  PHO_up800_6nt_2str_ncf_sig0_matches.tab
```

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

```
features-from-dnapat                            \
    -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  \
   -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg \
   -legend -scalebar -scalestep 50                  \
   -from 0 -to 800 -origin 801                      \
   -scorethick  \
   -title 'Over-represented 6nt in PHO upstream sequences'
```

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.

22

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  \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg \
    -legend -scalebar -scalestep 50               \
    -from 0 -to 800 -origin 801                   \
    -vertical -symbol -label pos                  \
    -title 'Over-represented 6nt in PHO upstream sequences'
```

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  \
    -o PHO_up800_6nt_2str_ncf_sig0_matches.jpg \
    -legend -scalebar -scalestep 50               \
    -from 0 -to 800 -origin 801                   \
    -scorethick  -dots                            \
```

23

```
-title 'Over-represented 6nt in PHO upstream sequences' \
-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 **features-from-dnapat** to convert matches from **dna-pattern** to features.

RSAT includes a few additional converters.

features-from-dnapat **dna-pattern** is the string-based matching program included in the *RSAT* distribution.

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

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

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

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

24

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.

9   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 anayzed, 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

oligonucleotide analysis *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 *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.

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.

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.

9.1   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 fmily 5 genes expressed under phosphate stress, and the MET family 11 genes expressed when methionine is absent fom the culture medium.

9.2   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,oligos,maps,synthesis,sql,clean,all \
    -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
YGR121C	NIT
YJR152W	NIT
YKR034W	NIT
YKR039W	NIT
YNL142W	NIT
YOR348C	NIT
YPR138C	NIT
YAR071W	PHO
YBR093C	PHO
YDR481C	PHO
YGR233C	PHO
YML123C	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 cluster file.

9.3 Loading the results in arelational 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 crating 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.

10 Utilities

10.1 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:

28

```
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.*'
```

10.2 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.fta.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.fta.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.fta
```

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.fta.gz -p GATAAG -c -th 3
```

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

11 Installing additional organisms

11.1 Parsing a genome from 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***.

---

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

11.2 Manual installation of a genome

If you are not lucky, the genome you want to work with is not available in Genbank format. If this is the case, you cn still try a manual installation. This is a bit tricky, but basically you will need two informations :

1. the complete genome sequence in raw format

2. a table with gene description (position, nme, ...)

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:

```
~myaccount/rsat-add/data/Mygenus_myspecies/
```

You need two informations to start installing a new genome:

- The genome in fasta format. If the genome contains multiple chromosomes, they should all be included in a common multi-sequence fasta file.

- 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.

- 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.

11.3 Installing the genome locally

One you have this information, start the program

```
install-organism
```

You will be asked to enter the information needed for genome installation.

11.4 Updating your local configuration

- Modify the local config file

- You need to define an environment variable called RSA_LOCAL_CONFIG, and which indicates the loca config file.

11.5 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.

```
retrieve-seq -help
```

Will give you a list of installed organisms.

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).

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

12 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.

12.1 Some hints

12.1.1 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.

12.1.2 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.

13 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. (2002). Prediction of transcriptional regulation by analysis of the non-coding genome. Current Genomics *in press*.