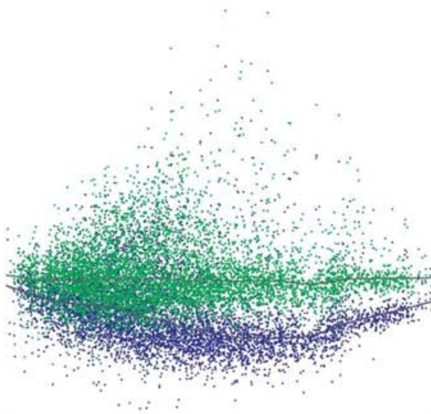
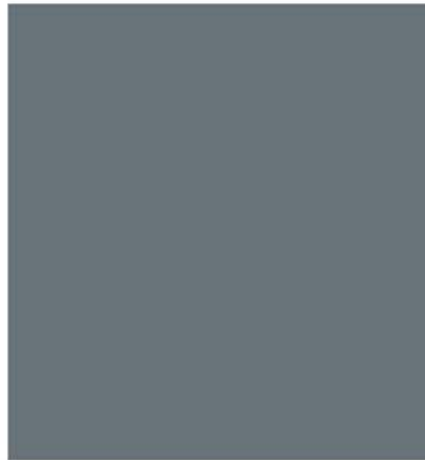
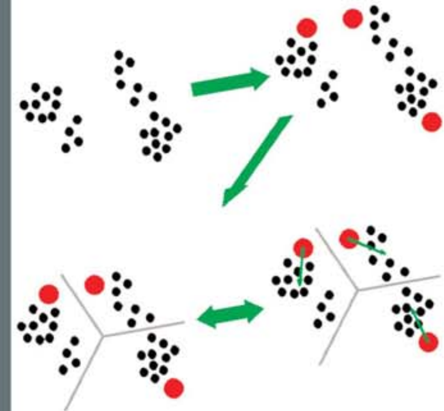

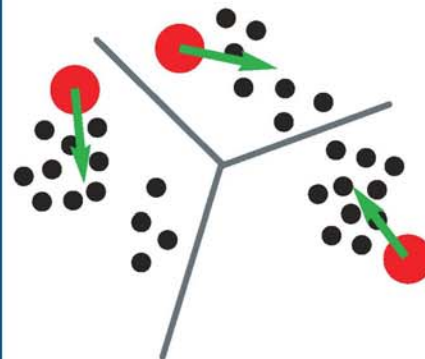
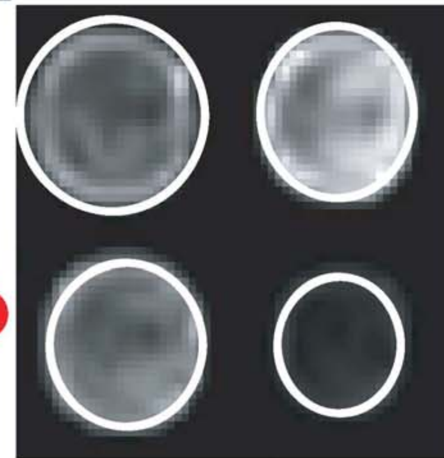

DNA Microarray Data Analysis

IIRIS HOVATTA, KATJA KIMPPA, ANTTI LEHMUSSOLA, TOMI PASANEN, JANNA SAARELA, ILANA SAARIKKO, JUHA SAHARINEN, PEKKA TIIKKAINEN TEEMU TOIVANEN, MARTTI TOLVANEN, MAUNO VIHINEN AND GARRY WONG

EDITORS JARNO TUIMALA AND M. MINNA LAINE

CSC

DNA Microarray Data Analysis

Editors
Jarno Tuimala
M. Minna Laine

CSC, the Finnish IT center for Science

CSC – Scientific Computing Ltd. is a non-profit organization for high-performance computing and networking in Finland. CSC is owned by the Ministry of Education. CSC runs a national large-scale facility for computational science and engineering and supports the university and research community. CSC is also responsible for the operations of the Finnish University and Research Network (Funet).

All rights reserved. The PDF version of this book or parts of it can be used in Finnish universities as course material, provided that this copyright notice is included. However, this publication may not be sold or included as part of other publications without permission of the publisher.

© The authors and
CSC – Scientific Computing Ltd.
2005

Second edition

ISBN 952-5520-11-0 (print)

ISBN 952-5520-12-9 (PDF)

http://www.csc.fi/oppaat/siru/

http://www.csc.fi/molbio/arraybook/

Printed at
Picaset Oy
Helsinki 2005

Preface

This is the second, revised and slightly expanded edition of the DNA microarray data analysis guidebook. As a change to the previous edition, some relatively quickly changing material such as software tutorials have been exclusively published on the book's web site. Please see `http://www.csc.fi/molbio/arraybook/` for more information and to access the extra material.

DNA microarrays generate large amounts of numerical data, which should be analyzed effectively. In this book, we hope to offer a broad view of basic theory and techniques behind the DNA microarray data analysis. Our aim was not to be comprehensive, but rather to cover the basics, which are unlikely to change much over years. Especially, we hope that researchers starting their data analysis can benefit from the book.

The text emphasizes gene expression analysis. Topics, such as genotyping, are discussed shortly. This book does not cover the wet-lab practises, such as sample preparation or hybridization. Rather, we start when the microarrays have been scanned, and the resulting images are being analyzed. Also, we take the files with signal intensities, which usually generate questions such as: "How is the data normalized?" or "How do I identify the genes which are upregulated?", and provide some simple solutions to these specific questions and many others.

Each chapter has a section on suggested reading, which introduces some of the relevant literature. Some chapters have additional information available on the web. In the first edition the software examples were included in the book, but we have now moved them into Internet. This allows us to better keep the material up to date.

Juha Haataja and Leena Jukka are warmly acknowledged for their support during the production of this book.

We are very interested in receiving feedback about this publication. Especially, if you feel that some essential technique has been missed, let us know. Please send your comments to the e-mail address Jarno.Tuimala@csc.fi.

Espoo, 23rd December 2005

The authors

List of Contributors

Iiris Hovatta
National Public Health Institute
Haartmaninkatu 8
FI-00290 Helsinki
Finland

Katja Kimppa
PerkinElmer Life Sciences and Analytical Sciences - Wallac Oy
P.O.Box 10
FI-20101 Turku
Finland

M. Minna Laine
CSC, the Finnish IT center for science
Keilaranta 14
FI-02101 Espoo
Finland

Antti Lehmussola
Tampere University of Technology
P.O.Box 553
FI-33101 Tampere
Finland

Tomi Pasanen
University of Helsinki
P.O.Box 68
FI-00014 University of Helsinki
Finland

Janna Saarela
Biomedicum Biochip Center
Haartmaninkatu 8
FI-00290 Helsinki
Finland

Ilana Saarikko
University of Helsinki
P.O.Box 68
FI-00014 University of Helsinki
Finland

Juha Saharinen
National Public Health Institute
Haartmaninkatu 8
FI-00290 Helsinki
Finland

Pekka Tiikkainen
VTT
P.O.Box 106
FI-20521 Turku
Finland

Teemu Toivanen
Centre for Biotechnology
Tykistökatu 6
FI-20521 Turku
Finland

Martti Tolvanen
Institute of Medical Technology
Biokatu 8
FI-33520 Tampere
Finland

Jarno Tuimala
CSC, the Finnish IT center for science
Keilaranta 14
FI-02101 Espoo
Finland

Mauno Vihinen
Institute of Medical Technology
Biokatu 8
FI-33520 Tampere
Finland

Garry Wong
A. I. Virtanen -institute
University of Kuopio
FI-70211 Kuopio
Finland

Contents

Preface 4

List of Contributors 5

I WEB extra: Introduction 7

II WEB extra: Analysis 8

III WEB extra: Data mining 9

1 Web extra: Data mining for promoter sequences 10

- 1.1 Using BioMart to retrieve promoter regions . . . 10
  - 1.1.1 Final warnings regarding upstream data . . . 13
- 1.2 GeneSpring and promoter analysis . . . 15

2 Web extra: Biological sequence annotations 16

- 2.1 Using Ensembl server to batch process of annotations . . . 16
  - 2.1.1 Example 1 . . . 16
  - 2.1.2 Example 2 . . . 17

3 Web extra: Software issues 18

- 3.1 Programming . . . 18
  - 3.1.1 Perl . . . 18
  - 3.1.2 Awk . . . 19
  - 3.1.3 R . . . 19
- 3.2 Common code examples . . . 20
  - 3.2.1 Read tabular data . . . 20
  - 3.2.2 Filtering/Rows . . . 23
  - Filtering/Columns . . . 24
  - 3.2.3 File convertions/print/write . . . 24

Part III

WEB extra: Data mining

1 Web extra: Data mining for promoter sequences

Martti Tolvanen, Jarno Tuimala and Mauno Vihinen

1.1 Using BioMart to retrieve promoter regions

The preferred option for retrieving upstream sequences is found at the BioMart service (`http://www.ensembl.org/Multi/martview`) of the Ensembl project. This is limited only by the number of Ensembl genes that are annotated and by the accuracy of the annotation. Many genes still have many multiple entries in Ensembl, and some entries are for pseudogenes. However, the service is very easy to use. Note that you will want to use primarily Ensembl data, not Vega (which is manually annotated but still very incomplete data).

For some microarrays (especially Affymetrix chips), BioMart provides direct mappings. Such mappings with microarray contents have become more common in the genome sites, both at Ensembl and at UCSC, and the annotations provided by the manufacturers have improved, too. Therefore, retrieving the upstream sequences is less of a technical problem now, but the problem of correct transcription start sites remains a serious one, especially in the case of alternative promoters..

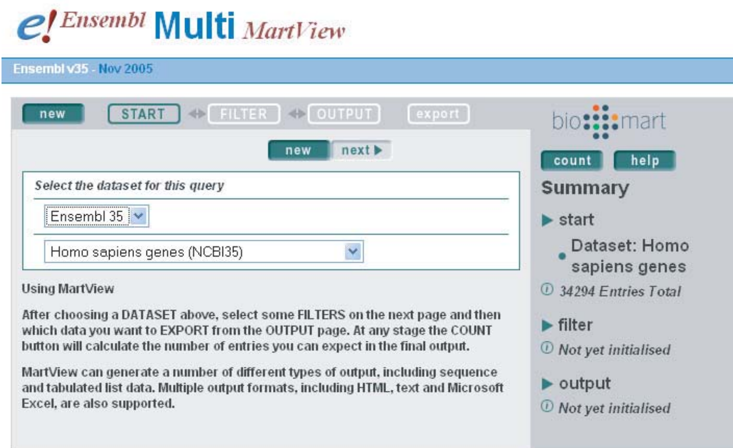

Figure 1.1: *BioMart start screen.*

Next, enter your list of gene identifiers in the Filter step:

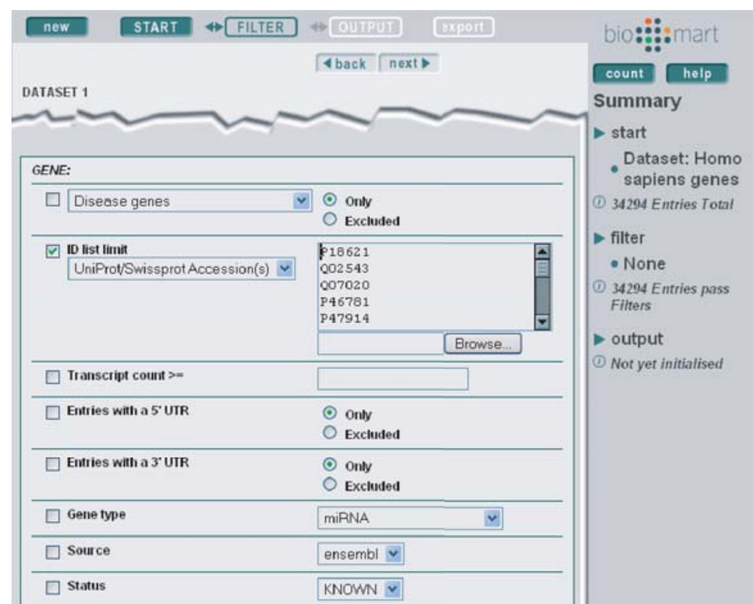

Figure 1.2: *BioMart filtering. There are many options, and you should be aware that some mappings are more complete than others. Next to internal Ensembl references, RefSeq is your best choice. This example comes from data in which reliable SwissProt codes were provided. A long list of other filtering options is omitted from the figure.*

Then, in the following output phase, you can optionally first choose your output as Features, to check the completeness and consistency of your results, and finally as Sequences to retrieve the data. The Features export gives a tabular out-

put which can be imported easily to other programs, e.g. by a direct copy/paste to MS-Excel..

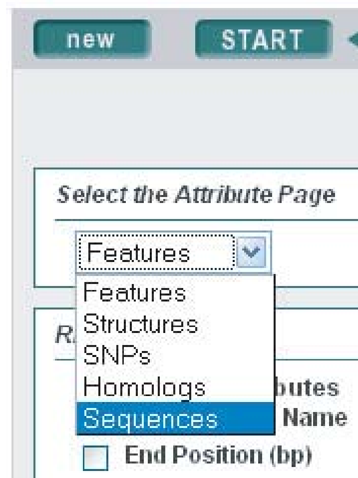

Figure 1.3: *Changing between Features and Sequences.*

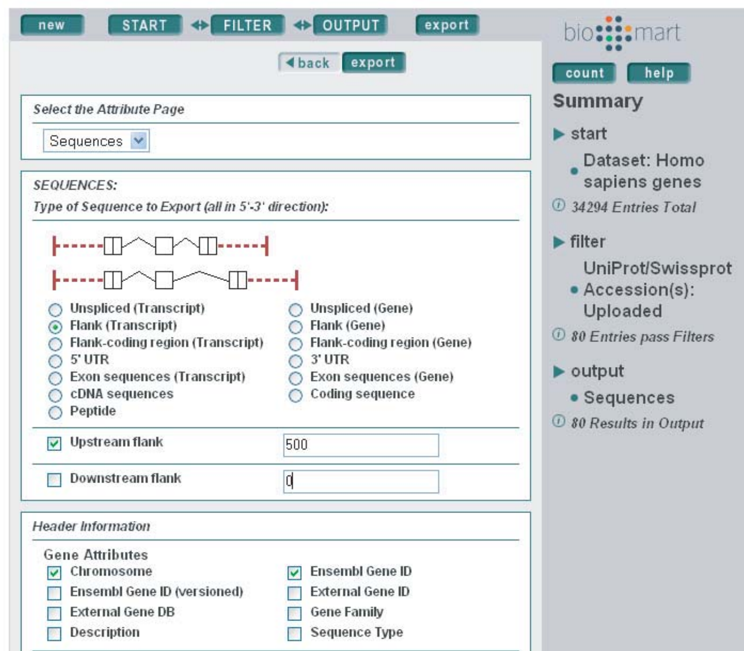

Figure 1.4: *Sequence output options for obtaining only the 5'-upstream flank of your found genes in BioMart. You may want to add more options in the Header information and/or select a longer sequence region. Additional options for data compression, saving locally etc. are not shown.*

```
>7|ENSG00000075624
ACTGCCTGGCCACTCCATGCCCTCCAAGAGCTCCTTCTGCAGGAGCGTACAGAACCCAGGGCCCTGGCACCCGTGCAGAC
CCTGGCCCACCCCACCTGGGCGCTCAGTGCCCAAGAGATGTCCACACCTAGGATGTCCCGCGGTGGGTGGGGGGCCCGAG
AGACGGGCAGGCCGGGGGCAGGCCTGGCCATGCGGGGCCGAACCGGGCACTGCCCAGCGTGGGGCGCGGGGGCCACGGCG
CGCGCCCCCAGCCCCCGGGCCCAGCACCCCAAGGCGGCCAACGCCAAAACTCTCCCTCCTCCTCTTCCTCAATCTCGCTC
TCGCTCTTTTTTTTTTTCGCAAAAGGAGGGGAGAGGGGGTAAAAAAATGCTGCACTGTGCGGCGAAGCCGGTGAGTGAGC
GGCGCGGGGCCAATCAGCGTGCGCCGTTCCGAAAGTTGCCTTTTATGGCTCGAGCGGCCGCGGCGGCGCCCTATAAAACC
CAGCGGCGCGACGCGCCACC

>7|ENSG00000075624
CTTTGGGCGCTAACTGCGTGCGCGCTGGGAATTGGCGCTAATTGCGCGTGCGCGCTGGGACTCAAGGCGCTAACTGCGCG
TGCGTTCTGGGGCCCGGGGTGCCGCGGCCTGGGCTGGGGCGAAGGCGGGCTCGGCCGGAAGGGGTGGGGTCGCCGCGGCT
CCCGGGCGCTTGCGCGCACTTCCTGCCCGAGCCGCTGGCCGCCCGAGGGTGTGGCCGCTGCGTGCGCGCGCGCCGACCCG
GCGCTGTTTGAACCGGGCGGAGGCGGGGCTGGCGCCCGGTTGGGAGGGGGTTGGGGCCTGGCTTCCTGCCGCGCGCCGCG
GGGACGCCTCCGACCAGTGTTTGCCTTTTATGGTAATAACGCGGCCGGCCCGGCTTCCTTTGTCCCCAATCTGGGCGCGC
GCCGGCGCCCCCTGGCGGCCTAAGGACTCGGCGCGCCGGAAGTGGCCAGGGCGGGGGCGACCTCGGCTCACAGCGCGCCC
GGCTATTCTCGCAGCTCACC

>20|ENSG00000101361
GGGAGGAAAGGATGCACCGCCCACCCGGCGCCGGGGAGGCTCGTCCCCGAGTAGGGGACGGGAACGCCTGGGTCCCGGGG
ACGTGGGGCGCAGGGTCTGGGGCCCGAGGCCACGGACGAAGCGGGAAGCCGGCTAGGCAGCGGTCTCTGTCTGGGATGGA
CGCCGCGCCCCACGCGGACCTCCCGCCCCGACTTCACCCAGCTCTGCTGCCCGTAACTCATATGACAAGAACATCAGGAA
CCGGACTATTACCCCGAAAGGGCCTTCCCAAGTCGTTTCGCCGCCTGCAGTGCCCGACCGGGGGCCGCGCGTCCCCGGCA
ACCACGCCCGCCCGCGCAGCTGCAGAGGCCGGAAAAGGCCCGCTCGCCCGCGCTCGGCACCACCCAGCCCGCGCTCCGGG
CGATACCACCCTGGGCGGCCCCTCCAAAGGCCGGAGATGGTGTCGTCCCCGGCCTCCGATTGGTCGGGGGGGCGGGGGCG
TGGCCTCTGGAGCCTGGTTC
```

Figure 1.5: *Sequence data in Fasta format, after the Export step.*

For some microarrays (especially Affymetrix chips), BioMart provides direct mappings, so you can skip most of the previous steps. Instead, you can list all the genes on the chip directly from the Gene menu (Fig. 6), or choose a subset of them in the Filter step:.

GENE:

☑ with AFFY-HG-U133 ID(s) ◉ Only ○ Excluded

Figure 1.6: *Selecting all genes that are represented on AFFY-HG-U133.*

Such mappings with microarray contents have become more common in the genome sites, both at Ensembl and at UCSC, and the annotations provided by the manufacturers have improved, too. Therefore, retrieving the upstream sequences is less of a technical problem now, but the problem of correct transcription start sites remains a serious one, especially in the case of alternative promoters.

1.1.1 Final warnings regarding upstream data

- in some cases the genome assembly may be broken close to the start of your gene, so that you do not get a full 1000 (or whatever) nucleotides of sequence. In Ensembl this shows as a long string of N's in the sequence

- the start of a RefSeq mRNA may not always correspond to a true transcription start site. There are many mRNAs which give evidence of longer tran-

script variants than the current RefSeqs. UCSC genome browser relies directly to the RefSeq start sites in their upstream data sets.

- Ensembl tries to extend the 5'-end as far as possible, but in a few cases this leads erratically long sequences, due to seemingly mis-spliced mRNA versions

1.2 GeneSpring and promoter analysis

GeneSpring includes a promoter analysis tool, which can be used for finding novel common regulatory sequencies in a gene list, or to search for a known sequence. The tool can be invoked from *Tools->Find potential regulatory sequencies*. In order to search for potential regulatory sequencies, you need to have a whole genomic sequence of the organism under study. In principle, if only a partial genome of the organism is known, it is not possible to search for regulatory elements (GeneSpring forbids the use of the tool), because the statistical support and frequency values of the elements would be erraneous. However, there is a trick, which allows the analysis of partial genomes. For more information, see the tech note at `http://www.silicongenetics.com/cgi/TNgen.cgi/GeneSpring/GSnotes/Notes/how_contig`.

The tool opens a new window (Figure 1.7). First, you need to select a genelist you want to study, but do not use the "all genes" or "all genomic elements" list, because then you would compare the whole genome against itself, which is not a viable analysis. From the pull-down menu, select whether you want to search for new sequencies or for a specific sequence. You can also select the lenght of the sequence to be considered a promoter region, how long a regulatory element is being searched, and how many unknown bases are allowed. The longer the sequence, and the larger the number of unknown bases, the longer the analysis time. You have control over the probability statistics: The p-value cut-off for a significant pattern can be modified. Whether the sequence is relative to the sequence upstream of other genes or relative to the whole genomic sequence can also be modified. The first option is far more common.

After the analysis have completed, or you stop the search, the results are reported. They appear on right side of the toolbox. Potential regulatory sequencies, the number of genes they were detected in, and the detection p-value are reported. The best findings are reported first.

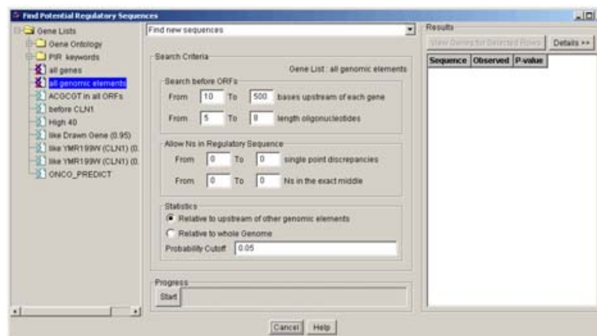

Figure 1.7: *Find potential regulatory elements -tool in GeneSpring.*

2 Web extra: Biological sequence annotations

Juha Saharinen

2.1 Using Ensembl server to batch process of annotations

Here are two small examples using these alternative interfaces to the Ensembl data. In example 1, 10 probe sets from Affymetrix U133A chip are queried from Ensembl *Homo sapiens* database and their Ensembl GeneStableIDs, and gene's sequence locations are returned. Example 2 shows a small Perl script, using EnsemblAPI, to retrieve the genomic sequence from a given organism, chromosome and physical position.

2.1.1 Example 1

```
bbu-juhaad@Opteronix:~> mysql -h ensembldb.ensembl.org -u anonymous homo_sapiens_core_27_35a
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1217078 to server version: 4.0.18-standard-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> select xref.dbprimary_acc, gene_stable_id.stable_id, seq_region.name, gene.seq_region_start, gene.seq_region_end,
gene.seq_region_strand
from xref
join external_db on external_db.external_db_id = xref.external_db_id
join object_xref on object_xref.xref_id = xref.xref_id
join translation on translation.translation_id = object_xref.ensembl_id
join transcript on transcript.transcript_id = translation.transcript_id
join transcript_stable_id on transcript_stable_id.transcript_id = transcript.transcript_id
join gene on gene.gene_id = transcript.gene_id
join gene_stable_id on gene_stable_id.gene_id = gene.gene_id
join seq_region on seq_region.seq_region_id = gene.seq_region_id
where (external_db.db_name = 'AFFY_HG_U133A')
limit 10;
+-------------------------+-----------------+------+------------------+----------------+-------------------+
| dbprimary_acc           | stable_id       | name | seq_region_start | seq_region_end | seq_region_strand |
+-------------------------+-----------------+------+------------------+----------------+-------------------+
| 209137_s_at             | ENSG00000102034 | X    |        128924393 |      128970188 |                -1 |
| 209136_s_at             | ENSG00000102034 | X    |        128924393 |      128970188 |                -1 |
| 202264_s_at             | ENSG00000196586 | 6    |         76515703 |       76682665 |                 1 |
| 202264_s_at             | ENSG00000118260 | 2    |        208220192 |      208289067 |                 1 |
| AFFX-HSAC07/X00351_5_at | ENSG00000196586 | 6    |         76515703 |       76682665 |                 1 |
| AFFX-HSAC07/X00351_5_at | ENSG00000166261 | 11   |        123100699 |      123117573 |                -1 |
| 217991_x_at             | ENSG00000166261 | 11   |        123100699 |      123117573 |                -1 |
| 209137_s_at             | ENSG00000125447 | 17   |         70744290 |       70769299 |                -1 |
```

```
| 209136_s_at              | ENSG00000125447 | 17   |         70744290 |       70769299 |                 -1 |
| 217988_at                | ENSG00000163251 | 2    |        208456223 |      208459624 |                 -1 |
+--------------------------+-----------------+------+------------------+----------------+--------------------+
10 rows in set (0.11 sec)

mysql>
mysql> quit
Bye
```

2.1.2 Example 2

```
#!/usr/bin/perl

use Bio::EnsEMBL::DBSQL::DBAdaptor;

sub main
{
  $db = Bio::EnsEMBL::DBSQL::DBAdaptor->new
    (-host   => 'ensembldb.ensembl.org',
     -dbname => $ARGV[0],
     -user   => 'anonymous');

  # get the slice adaptor and fetch a slice on a given region
  $slice_adaptor = $db->get_SliceAdaptor();
  $slice = $slice_adaptor->fetch_by_region('chromosome', $ARGV[1], $ARGV[2], $ARGV[3]);

  # print out the sequence from this region
  print $slice->seq();
}

#start program
{
  if ($#ARGV != 3)  {
    print "\nGenoFetcher - Juha Saharinen, Biomedicum Bioinformatics Unit & KTL, 2004\n\n";
    print "Use of GenoFecther is \n";
    print "genofetcher <database> <chromosome> <chromosome_start> <chromosome_end>\n\n";
    print "Like genofetcher homo_sapiens_core_27_35a 12 1000000 1001000\n\n";
    print "This will return slice of human chromosome 12 from 1.000.000 to 1.001.000\n\n";
  }
  else {
    main;
  }
}
```

3 Web extra: Software issues

Jarno Tuimala and Teemu Toivanen

3.1 Programming

When the capacities of the spreadsheet program are not sufficient or many similar files need processing, some programming tools can be used instead for the datafile management purposes or for data analyses. There are actually several programming languages that are especially suited for the data file formatting and other text file manipulations.

3.1.1 Perl

One of the most common languages is Perl, which has very powerful and easy to use text manipulation tools. Perl is available for free for UNIX, Linux `http://www.perl.org` and PC machines `http://www.activeperl.com`. Perl is a programming language, which means that getting to know it takes some time and effort. However, if data conversion tools are needed everyday, it would definitely be worthwhile to befriend Perl.

To illustrate how easily text can be manipulated using Perl, we present a short example. The next code produces a complementary DNA sequence from the original sequence, which has been stored into the text string $sequence. Both the original sequence and its complement are printed on the computer screen.

The program starts with a line that tells the computer where The Perl software can be found. The next four lines contain the actual commands that manipulate the DNA sequences. Note that every command line has to end with a semicolon. Function `tr` makes a complementary sequence from the original one, and function `print` outputs the result to the screen.

```
#!/usr/bin/perl
$sequence="aaattcgagtaggtcaggcat";
print "Original:            $sequence\n";
$sequence=~ tr/acgtACGT/tgcaTGCA/;
print "Complementary:       $sequence\n";
```

You can use pico editor on CSC's Cedar server to create a similar file and test

the example yourself. Perl programs are started in Cedar with a command `perl filename`.

3.1.2 Awk

Awk is a standard UNIX and Linux tool, which is available on CSC's servers. With Awk, individual columns can be easily extracted from tab-delimited text files. Using other standard UNIX tools, these individual columns can be saved into a new text file. For example, the next script takes the first column from a specified datafile and saves it into a new file.

```
Awk '{print$1}' datafile > newfile
```

Two columns can be "awked" into a new file next to each other separated with a space:

```
awk '{print$1, $2}' datafile > newfile
```

or one after another:

```
awk '{print$1}{print$2}' datafile > newfile
```

3.1.3 R

R is a free statistical analysis tool and a self-sufficient programming language. It is available for UNIX, Linux, Macintosh and PC platforms. In R, scripts for analyses and data file manipulations can be easily constructed. R has many add-on packages for cluster analysis, self-organizing maps, and neural networks, among others. There are also many packages available that have been specifically tailored for DNA microarray data analysis: Bioconductor project develops and updates many of these pacakges.

Here is an example on how to read the tab-delimited datafile to R and how to process it into a new table, which is then written out to a new file. The function `read.table` reads in the specified file with the headers. The table is then saved in a variable data. Two columns are extracted from the data, and saved into new variable (x and y). The new variables are used for the creation of a new table (dataout), which is then written to a new text file (filenameout.txt). Such a script can easily be automated using R, and the analyses can simultaneously be intergrated with the datafile conversions.

```
data<-read.table("filename.txt", header=T)
x<-data$greenintensity
y<-data$redintensity
dataout<-cbind(x,y)
sink("filenameout.txt")
dataout
```

```
sink()
```

Many images included in chapters 5–8 have been produced by R using real DNA microarray datasets.

3.2 Common code examples

This section covers common problems when manipulating text formatted datafiles that most programs export and import. Problems is that those fi le formats are program specific and using them for analysis is hard without a lot of editing. These are on Linux systems, but same code should work on all UNIX flavours and even in windows with minor changes.

For the actual code examples, see online material at `http: www.csc.fi/oppaat/siru/`.

3.2.1 Read tabular data

This is the basis for rest of the examples (copy/paste this first and then add what you want from the latter examples). Reads tabular files (change variables for sectioned tabular data). Please note that examples are in both Perl and Python languages and cannot be intermixed.

Perl

```
 1  %%
 2  %% This is file ‘.tex’,
 3  %% generated with the docstrip utility.
 4  %%
 5  %% The original source files were:
 6  %%
 7  %% fileerr.dtx  (with options: ‘return’)
 8  %%
 9  %% This is a generated file.
10  %%
11  %% Copyright 1993 1994 1995 1996 1997 1998 1999
12  %% The LaTeX3 Project and any individual authors listed elsewhere
13  %% in this file.
14  %%
15  %% This file was generated from file(s) of the Standard LaTeX ‘Tools Bundle’.
16  %% --------------------------------------------------------------------------
17  %%
18  %% It may be distributed and/or modified under the
19  %% conditions of the LaTeX Project Public License, either version 1.2
20  %% of this license or (at your option) any later version.
21  %% The latest version of this license is in
22  %%    http://www.latex-project.org/lppl.txt
23  %% and version 1.2 or later is part of all distributions of LaTeX
24  %% version 1999/12/01 or later.
25  %%
26  %% This file may only be distributed together with a copy of the LaTeX
```

```
27  %% ‘Tools Bundle’. You may however distribute the LaTeX ‘Tools Bundle’
28  %% without such generated files.
29  %%
30  %% The list of all files belonging to the LaTeX ‘Tools Bundle’ is
31  %% given in the file ‘manifest.txt’.
32  %%
33   \message{File ignored}
34  \endinput
35  %%
36  %% End of file ‘.tex’.
```

```
#!/usr/bin/perl -w
#filename to read, get from commandline
my $filename=shift;
#open file (F is the filehandle)
open(F, $filename);
#specify data delimiter \t = tab, \; = semicolon
$delim="\t";
#this will have the data matrix
@data=();
#put this to 1 if first row is a header
$hasHeader=1;
#this includes header
@headerData=();
#use filter to specify when data section starts and ends
#not needed, for normal tabular data
$useFilter=0;
#search for this string in the beginning of the line to start data section
$startFilter="BEGIN DATA";
#search for this string in the beginning of the line to end data section
$endFilter="END DATA";
#internal variable so that we know when we are at data section
$doInsert=0;
#loop while there’s data
while($line=<F>) {
    chomp($line);
    if ($useFilter==1) { # go here if we are using filters
        if ($doInsert==0) { # not in data section
            if ($line =~ /$startFilter/) { #search for beginning of data
                $doInsert=1;  #go to data phase
            }
            next;
        } else { # we are in data phase
            if ($line =~ /$endFilter/) { #search for the end of data
                $doInsert=0;  #exit data phase
                next;
            }
        }
    }
    #data insertion section
    if ($hasHeader==1) {
        $hasHeader=-1; # we come here only once
        #split header row with $delim to a vector
        @headerData=split($delim, $line);
```

```
        next; #read next line
    }
    #add new row to matix data, that has $line split with $delim
    #to a vector (which is that row of the matrix)
    push(@data, [split($delim, $line)]);
}
#now we have data in @data matrix and possible header info in @haderData
```

Python

```
#!/usr/bin/python
from sys import argv #for filename
import re,string # reqular expressions and strings
#filename to read, get from commandline
filename=argv[1]
#open file (f is the filehandle)
f = open(filename, 'r')
#specify data delimiter \t = tab, \; = semicolon
delim="\t"
#this will have the data matrix
data=[]
#put this to 1 if first row is a header
hasHeader=1
#this includes header
headerData=[]
#use filter to specify when data section starts and ends
#not needed, for normal tabular data
useFilter=0
#search for this string in the beginning of the line to start data section
startFilter="BEGIN DATA"
#search for this string in the beginning of the line to end data section
endFilter="END DATA"
#internal variable so that we know when we are at data section
doInsert=0
#loop while there's data
line=f.readline()
while len(line) != 0:
    next=0
    line=re.sub("\n|\r","",line)
    if (useFilter==1):# go here if we are using filters
        next=1
        if (doInsert==0):  # not in data section
            if (re.search(startFilter, line) != None):  #search for beginning of data
                doInsert=1  #go to data phase
                next=1 #do not append data
        else:  # we are in data phase
            next=0
            if (re.search(endFilter, line) != None):  #search for the end of data
                doInsert=0  #exit data phase
                next=1 #do not append data
    #data insertion section
    if (hasHeader==1):
```

```
        hasHeader=-1 # we come here only once
        #split header row with $delim to a vector
        headerData=string.split(line, delim)
        next=1 #do not append data

    #add new row to matix data, that has $line split with $delim
    #to a vector (which is that row of the matrix)
    if next != 1:
        data.append(string.split(line, delim))
    line = f.readline()

#now we have data in a matrix and possible header info in haderData
```

3.2.2 Filtering/Rows

Select what column to filter (column) and what to search (search).

Perl

```
#column number to use for filtering (columns nubered from 0,1,2...)
$colnum=0; # first column
#what to search
$search="1";
#check each data row
@filteredData=();
for my $row (@data) {
    #exact match 'eq', numric match '==', numeric greater or eaual >=
    # =~ /$search/ for regexp match
    if ($row->[$colnum] eq $search) {
        push(@filteredData, $row);
    }
}
#replace @data with filtered data @ret
@data=@filteredData;
```

Python

```
#column number to use for filtering (columns nubered from 0,1,2...)
colnum=0 # first column
#what to search
search="1"
#check each data row
filteredData=[]
for row in (data):
    #exact match '==', numric match '==', numeric greater or eaual >=
    # re.search(pattern, string for regexp match
    if (row[colnum] == search):
        filteredData.append(row)
#replace data with filtered data ret
data=filteredData
```

Filtering/Columns

Perl

Change "cols"list to reflect wanted columns

```
#column number in a list (rest is filtered out, columns nubered from 0,1,2...)
@cols=(0,2); # first and third column
@filteredData=();
#header cols
if (scalar(@headerData)>0) { #check that header is not empty
    @headerData=@headerData[@cols];
}
for my $r (@data) {
    my @row=@{$r};
    my @tmp;
    @tmp=@row[@cols];
    push(@filteredData, \@tmp);

}
#replace @data with filtered data @ret
@data=@filteredData;
```

Python

```
#column number in a list (rest is filtered out, columns nubered from 0,1,2...)
cols=[0,2] # first and third column
filteredData=[]
#header cols
if (len(headerData)>0): #check that header is not empty
    tmp=[]
    for x in cols: # append all columns
        tmp.append(headerData[x])
    headerData=tmp
# for each row
for row in (data):
    tmp=[]
    for x in cols:
        tmp.append(row[x]) # append all columns
    filteredData.append(tmp)
#replace data with filtered data ret
data=filteredData
```

3.2.3 File convertions/print/write

First print header (if exists) and then rest of the data with selected delimiter. Use > 'filename' to print to a file.

Perl

```
# open(FILEOUT, "> outfile.txt"); #remove leading hash for file writing
# also change STDOUT to FILEOUT later on this file
#output delim
```

```
$outDelim=";";
if (scalar(@headerData)>0) { #check that header is not empty
    #print header with $outDelim as delimiter
    print STDOUT join($outDelim,@headerData);
    print STDOUT "\n"; #new line
}
#for each row
for my $row (@data) {
    #print row vector with $outDelim as delimiter
    print STDOUT join($outDelim,@{$row});
    print STDOUT "\n"; #new line
}
# close(FILETOUT); #see open(..) comments
```

Python

```
#output delim
outDelim=";"
if (len(headerData)>0):  #check that header is not empty
    #print header with outDelim as delimiter
    print string.join(headerData,outDelim)
#for each row
for row  in data:
    #print row vector with outDelim as delimiter
    print string.join(row,outDelim)
```