Appendix 1 CRM Database Application Code

SQL Files

schema.sql

```
--
-- Table structure for table `element`
--
DROP TABLE IF EXISTS `element`;
CREATE TABLE `element` (
  `element_id` int(11) NOT NULL default '0',
  `gene_id` int(11) default NULL,
  `element_type_id` int(11) default NULL,
  `name` text,
  `position_start` int(11) default NULL,
  `position_end` int(11) default NULL,
  `sequence` text,
  `regulator_id` int(11) default NULL,
  PRIMARY KEY  (`element_id`),
  KEY `gene_id` (`gene_id`),
  KEY `element_type_id` (`element_type_id`),
  KEY `regulator_id` (`regulator_id`),
  CONSTRAINT `element_geneid_fk`
      FOREIGN KEY (`gene_id`)
      REFERENCES `gene` (`gene_id`),
  CONSTRAINT `element_regulatorid_fk`
      FOREIGN KEY (`regulator_id`)
      REFERENCES `gene` (`gene_id`),
  CONSTRAINT `element_typeid_fk`
      FOREIGN KEY (`element_type_id`)
      REFERENCES `element_type` (`element_type_id`)
);



--
-- Table structure for table `element_type`
--
DROP TABLE IF EXISTS `element_type`;
CREATE TABLE `element_type` (
  `element_type_id` int(11) NOT NULL default '0',
  `name` text,
  PRIMARY KEY  (`element_type_id`)
);
```

```
--
-- Table structure for table `gene`
--
DROP TABLE IF EXISTS `gene`;
CREATE TABLE `gene` (
  `gene_id` int(11) NOT NULL default '0',
  `name` text,
  `sequence` longtext,
  `abbreviation` varchar(12) default NULL,
  `motifs` text,
  `img` varchar(255) default NULL,
  `img_small` varchar(255) default NULL,
  PRIMARY KEY  (`gene_id`)
);



--
-- Table structure for table `motif`
--
DROP TABLE IF EXISTS `motif`;
CREATE TABLE `motif` (
  `motif_id` int(11) NOT NULL default '0',
  `name` text,
  `abbreviation` varchar(12) default NULL,
  `motif` text,
  PRIMARY KEY  (`motif_id`)
);



--
-- Table structure for table `regulator`
--
DROP TABLE IF EXISTS `regulator`;
CREATE TABLE `regulator` (
  `element_id` int(11) NOT NULL default '0',
  `gene_id` int(11) NOT NULL default '0',
  PRIMARY KEY  (`element_id`,`gene_id`),
  CONSTRAINT `element_geneid_fk`
      FOREIGN KEY (`gene_id`)
      REFERENCES `gene` (`gene_id`),
  CONSTRAINT `element_fk`
      FOREIGN KEY (`element_id`)
      REFERENCES `element` (`element_id`)

);
```

element.cgi

```
#!/usr/local/bin/perl

# --
# -- HTML form for adding new elements (enhancers, promoters, exons)
# -- to a sequences
# --

use DBI;
use CGI;
use strict;
use lib "/home/zburke/bg/workspace/perl-lib";
use nazina;
use nazina_db;

my $dbh = nazina_db::dbh();

main();

sub main
{
    my $cgi = CGI->new();
    print $cgi->header();

    if ($ENV{'REQUEST_METHOD'} eq "GET") {
        do_get($cgi);
    } else {
        do_post($cgi);
    }
}

sub do_get
{
    my ($cgi) = @_;

    my $dao = nazina_db->new($dbh);
    my $gene_id = $cgi->param('gene_id');
    my $genes = $dao->genes();
    my $select .= "<select name='gene_id'>";
    for (sort { ${ $genes }{$a}->{'abbv'} cmp ${ $genes }{$b}->{'abbv'}
} keys %{ $genes }) {
        $select .= "<option value='${ $genes }{$_}->{'gene_id'}'>${
$genes }{$_}->{'name'}\n";
    }
    $select .= "</select>";

    print <<"EOT";
<html>
<body>
```

```
<form method=post>
gene id
$select
<br>
<table><tr>
    <td>start<br><input type="text" name="p_start"></td>
    <td>end<br><input type="text" name="p_end"></td>
    <td>type<br><select name="type_id"><option value=1>promoter<option
value=2>enh - early<option value=3>enh - late<option
value=4>exon</select></td>
    <td>name<br><input type="text" name="name"></td>
</tr></table>
<br>
<input type="submit" name="save">
</form>
EOT
}

sub do_post
{
    my ($cgi) = @_;

    my $idsth = $dbh->prepare("select max(element_id) as max_id from
element");
    $idsth->execute();
    my $element_id;
    while (my ($id) = $idsth->fetchrow_array) {
        $element_id = $id;
    }

    $element_id++;
    print STDERR "id is $element_id\n";

    my $gene_id = $cgi->param('gene_id');
    my $p_start = $cgi->param('p_start');
    my $p_end   = $cgi->param('p_end');
    my $type_id = $cgi->param('type_id');
    my $name    = $cgi->param('name');

    my $sth = $dbh->prepare("insert into element values (?, ?, ?, ?, ?,
?, '')");
    $sth->execute($element_id, $gene_id, $type_id, $name, $p_start,
$p_end);

    print STDERR "sth->execute($element_id, $gene_id, $type_id, $name,
$p_start, $p_end);\n";

    do_get($cgi);
}
```

gene.cgi

```
#!/usr/local/bin/perl

# --
# -- form for gene details (name, abbreviation, motifs, sequence)
# --

use DBI;
use CGI;
use strict;
use lib "/home/zburke/bg/workspace/perl-lib";
use nazina;
use nazina_db;

my $dao = nazina_db->new(); ;

main();

sub main
{
    my $cgi = CGI->new();

    my $gene_id = $cgi->param('gene_id');

    if ($ENV{'REQUEST_METHOD'} eq "GET" && $gene_id) {
        print $cgi->header();
        do_get($gene_id);
    } else {
        do_post($cgi);
    }
}



# --

# --
# -- do_get
# -- show gene editing form
sub do_get
{
    my ($gene_id) = @_;

    my $gene = $dao->gene($gene_id);

    print <<"EOT";
<html>
<body>
<form method=post>
<input type="hidden" name="gene_id" value="$gene_id">
<br>
<table>
<tr valign="top">
```

```
    <td>name</td><td><input type="text" name="name" value="$gene-
>{'name'}"></td>
</tr><tr valign="top">
    <td>abbreviation</td><td><input type="text" name="abbv"
value="$gene->{'abbv'}"></td>
</tr><tr valign="top">
    <td>motifs</td><td><textarea name="motifs" rows="20"
cols="20">$gene->{'motifs'}</textarea></td>
</tr><tr valign="top">
    <td>sequence</td><td><textarea name="sequence" rows="10"
cols="20"></textarea></td>
</tr><tr valign="top">
    <td><input type="submit"></td>
</tr>

</table>
<br>
</form>
EOT

}



# --

# --
# -- process form submission and return to index
sub do_post
{
    my ($cgi) = @_;

    my $gene_id = $cgi->param('gene_id');
    my $name    = $cgi->param('name');
    my $abbv    = $cgi->param('abbv');
    my $motifs  = $cgi->param('motifs');
    my $seq     = $cgi->param('sequence');

    my $sth = $dao->dbh()->prepare("update gene set name = ?,
abbreviation = ?, motifs = ? where gene_id = ?");
    $sth->execute($name, $abbv, $motifs, $gene_id);

    # only set sequence if a new one was submitted
    if ($seq) {
        my $sth = $dao->dbh()->prepare("update gene set sequence = ?
where gene_id = ?");
        $sth->execute($seq, $gene_id);
    }
    print "Location: index.cgi\n\n";
}
```

index.cgi

```
#!/usr/local/bin/perl

# --
# -- show all genes with sequence data,
# HTML form for putting new elements (enhancers, promoters, exons)
#


use DBI;
use CGI;
use strict;

use lib "/home/zburke/bg/workspace/perl-lib";
use nazina;
use nazina_db;

my $dao = nazina_db->new();

main();

sub main
{
    my ($cgi, $show, $gene_id, $content, $htmx);

    $cgi = CGI->new();
    $gene_id = $cgi->param('gene_id');
    $show    = $cgi->param('show');

    # show motifs for a gene
    if ($show eq "motifs" && $gene_id) {
        $content = motif_show($gene_id);
    }

    # show gene profile
    elsif ($show eq "gene" && $gene_id) {
        $content = gene_show($gene_id);
    }

    # show genes grouped by element regulators
    elsif ($show eq "regulation") {
        $content = regulator_show();
    }

    # DEFAULT: show profile list for genes with sequence data
    else {

        $content = gene_show_all();
    }



    $htmx = nazina::htmx();
```

```
    $htmx =~ s/~content~/$content/;
    print $cgi->header();
    print $htmx;

}



# --

# --
# -- gene_show_all
# -- show small picture and name for all genes with sequence data
# --
sub gene_show_all
{
    my $content = "";
    my $genes = $dao->genes();

    $content .= "<div><a href='index.cgi?show=regulation'>genes by
regulator element</a></div><br /><br />\n";

    $content .= "<table cellpadding='4'>\n";
    for (sort { ${ $genes }{$a}->{'abbv'} cmp ${ $genes }{$b}->{'abbv'}
} keys %{ $genes }) {
        my $g = ${ $genes }{$_};
        next unless $g->{'sequence'};
        $content .= <<EOT;
<tr>
    <td><a href="index.cgi?show=gene&gene_id=$g->{'gene_id'}"><img
src='images/small/$g->{'abbv'}s.gif' border='0'></a></td>
    <td><a href="index.cgi?show=gene&gene_id=$g->{'gene_id'}">$g-
>{'name'}</a>
(edit:
<a href="regulator.cgi?gene_id=$g->{'gene_id'}">regulators</a>
 | <a href="gene.cgi?gene_id=$g->{'gene_id'}">gene</a>
)</td>
</tr>

EOT
    }
    $content .= "</table>";

    $content .= gene_update_form($genes);
    return $content;
}



# --

# --
# -- gene_update_form
sub gene_update_form
{
```

```
    my ($genes) = @_;


    my $content = "";
    $content .= "
<div>
<div><b>update genes</b></div>
<form action='gene.cgi' method='get'>
<select name='gene_id' onchange='submit(this)'>
<option>edit gene";

    for (sort { ${ $genes }{$a}->{'abbv'} cmp ${ $genes }{$b}->{'abbv'}
} keys %{ $genes }) {
        my $g = ${ $genes }{$_};
        $content .= "<option value='$g->{'gene_id'}'>$g->{'abbv'} $g-
>{'name'}\n";
    }
    $content .= "</select></form></div>";

    return $content;
}

# --

# --
# -- gene_show
# -- display details about a gene
sub gene_show
{
    my ($gene_id) = @_;
    my ($content, $sth, $rsth, @elements);

    # get gene details
    $sth = $dao->dbh()->prepare("select name, abbreviation, sequence
from gene where gene_id = ?");
    $sth->execute($gene_id);
    my ($name, $abbv, $seq) = $sth->fetchrow_array();


    $content .= "<h3>$name ($abbv)</h3>\n";
    $content .= "<div style=\"margin: 2em;\"><img
src='images/$abbv.gif'></div>\n";
    $content .= "<table style=\"border-collapse: collapse;\"
border=\"1\" bordercolor=\"#111111\" cellpadding=\"4\"
cellspacing=\"0\">\n<tr><td>position</td><td>name</td><td>regulator</td
></tr>\n";

    # get elements
    $sth = $dao->dbh()->prepare("
        select element_id, element_type_id, name,
        position_start, position_end
        from element
        where gene_id = ?
        order by position_start");
```

```
    $rsth = $dao->dbh()->prepare("
        select g.abbreviation, g.gene_id
        from gene g, regulator r
        where r.element_id = ?
            and r.gene_id = g.gene_id");

    $sth->execute($gene_id);
    while (my ($element_id, $type_id, $name, $beg, $end) = $sth-
>fetchrow_array()) {
        push @elements, {
            'element_id' => $element_id,
            'type_id' => $type_id,
            'beg' => $beg,
            'end' => $end,
        };
        my @regulators = ();
        $rsth->execute($element_id);
        while (my ($regulator, $regulator_id) = $rsth-
>fetchrow_array()) {
            push @regulators, {
                'abbv' => $regulator,
                'gene_id' => $regulator_id,
            };
        }
        my $reg_links = regulator_links(\@regulators);
        $content .= <<"EOT";
<tr>
    <td>$beg-$end</td>
    <td><a href='#$element_id'>$name</a></td>
    <td>$reg_links</td>
</tr>
EOT
    }

    $content .= "</table>";
    $content .= "<h3>Sequence</h3>\n\n<pre
class='sequence'>".nazina::sequence_annotate(\$seq,
\@elements)."</pre>\n";

    return $content;
}



# --

# --
# -- motif_show
# -- return motifs for a given gene
sub motif_show
{
    my ($gene_id) = @_;

    return "<pre>".$dao->motifs($gene_id)."</pre>";
```

```
#    my $sth = $dbh->prepare("select motifs from gene where gene_id =
?");
#    $sth->execute($gene_id);
#    my ($motifs) = $sth->fetchrow_array();
#    return "<pre>$motifs</pre>";

}



# --

# --
# -- regulator_links
# -- given a list of regulatory elements, return links to display
# -- their motifs in a single string.
sub regulator_links
{
    my ($regulators) = @_;
    my @links = ();

    for ( sort { $a->{'abbv'} cmp $b->{'abbv'}  } @{ $regulators }) {

        push @links, "<a href='index.cgi?show=motifs&gene_id=$_-
>{'gene_id'}'>$_->{'abbv'}</a>";

    }
    return (join ", \n", @links);
}



# --
# -- regulator_show
# -- show genes grouped by regulator elemetns
# --
sub regulator_show
{
    my $content = "";
    my $genes = $dao->genes();

    $content .= "<table cellpadding='4'>\n";
    my $sth = $dao->dbh()->prepare("
        select distinct r.gene_id, g.name
        from regulator r, gene g
        where r.gene_id = g.gene_id
        order by g.name");
    $sth->execute();

    my $esth = $dao->dbh()->prepare("
        select e.gene_id, e.name e_name, g.name g_name, g.abbreviation
        from element e, gene g, regulator r
        where
            e.gene_id = g.gene_id
            and e.element_id = r.element_id
```

```
            and r.gene_id = ?
            order by g.name");

    # loop over regulator elements
    while (my ($r_id) = $sth->fetchrow_array()) {
        my $r = ${ $genes }{$r_id};
        $content .= "<tr><td colspan='2'><a
href='index.cgi?show=gene&gene_id=$r->{gene_id}'>$r->{name} ($r-
>{abbv})</a></td></tr>\n";
        $content .= "<tr><td>   </td><td>";

        # fetch elements regulated by this
        $esth->execute($r_id);
        while (my ($gene_id, $e_name, $g_name, $abbv) = $esth-
>fetchrow_array()) {
        $content .= "<a
href='index.cgi?show=gene&gene_id=$gene_id'>$g_name</a> ($abbv);
$e_name<br />\n";

        }
        $content .= "</tr>\n";

    }

    $content .= "</table>";

    return $content;
}
```

motifs.cgi

```
#!/usr/local/bin/perl

# --
# -- show motifs (TFBS?) for a given regulatory element
# --


use DBI;
use CGI;
use strict;

use lib "/home/zburke/bg/workspace/perl-lib";
use nazina;
use nazina_db;

my $dbh = nazina_db::dbh();

main();

sub main
{
    my ($cgi, $gene_id, $content, $htmx);

    $cgi = CGI->new();
    $gene_id = $cgi->param('gene_id');

    open HTMX, "template.htmx";
    while (<HTMX>) { $htmx .= $_; };
    close HTMX;

    if ($gene_id) {
        $content = gene_show($gene_id);
    } else {
        $content = gene_show_all();
    }


    $htmx =~ s/~content~/$content/;

    print $cgi->header();
    print $htmx;

}



# --

# --
# -- gene_show_all
# -- show small picture and name for all genes with sequence data
# --
```

```
sub gene_show_all
{
    my $content = "";
    my $genes = nazina::genes($dbh);

    $content .= "<form action='gene.cgi' method='get'><select
name='gene_id' onchange='submit(this)'>";
    for (sort { ${ $genes }{$a}->{'abbv'} cmp ${ $genes }{$b}->{'abbv'}
} keys %{ $genes }) {
        $content .= "<option value='${ $genes }{$_}->{'gene_id'}'>${
$genes }{$_}->{'name'}\n";
    }
    $content .= "</select></form>";

    $content .= "<table cellpadding='4'>\n";
    for (sort { ${ $genes }{$a}->{'abbv'} cmp ${ $genes }{$b}->{'abbv'}
} keys %{ $genes }) {
        next unless ${ $genes }{$_}->{'sequence'};
        $content .= <<EOT;
<tr>
    <td><a href="index.cgi?gene_id=${ $genes }{$_}->{'gene_id'}"><img
src='images/small/${ $genes }{$_}->{'abbv'}s.gif' border='0'></a></td>
    <td><a href="index.cgi?gene_id=${ $genes }{$_}->{'gene_id'}">${
$genes }{$_}->{'name'}</a>
(
<a href="regulator.cgi?gene_id=${ $genes }{$_}->{'gene_id'}">reg</a>
 | <a href="gene.cgi?gene_id=${ $genes }{$_}->{'gene_id'}">gene</a>
)</td>
</tr>

EOT
    }
    $content .= "</table>";

    return $content;
}



# --

# --
# -- gene_show
# -- display details about a gene
sub gene_show
{
    my ($gene_id) = @_;
    my ($content, $sth, $rsth, @elements);

    # get gene details
    $sth = $dbh->prepare("select name, abbreviation, sequence from gene
where gene_id = ?");
    $sth->execute($gene_id);
    my ($name, $abbv, $seq) = $sth->fetchrow_array();
```

```
    $content .= "<h3>$name ($abbv)</h3>\n";
    $content .= "<div style=\"margin: 2em;\"><img
src='images/$abbv.gif'></div>\n";
    $content .= "<table style=\"border-collapse: collapse;\"
border=\"1\" bordercolor=\"#111111\" cellpadding=\"4\"
cellspacing=\"0\">\n<tr><td>position</td><td>name</td><td>regulator</td
></tr>\n";

    # get elements
    $sth = $dbh->prepare("select element_id, element_type_id, name,
position_start, position_end from element where gene_id = ? order by
position_start");
    $rsth = $dbh->prepare("select g.abbreviation from gene g, regulator
r where r.element_id = ? and r.gene_id = g.gene_id");
    $sth->execute($gene_id);
    while (my ($element_id, $type_id, $name, $beg, $end) = $sth-
>fetchrow_array()) {
        push @elements, {
            'element_id' => $element_id,
            'type_id' => $type_id,
            'beg' => $beg,
            'end' => $end,
        };
        my @regulators = ();
        $rsth->execute($element_id);
        while (my ($regulator) = $rsth->fetchrow_array()) {
            push @regulators, $regulator;
        }
        @regulators = sort @regulators;
        $content .= <<"EOT";
<tr>
    <td>$beg-$end</td>
    <td><a href='#$element_id'>$name</a></td>
    <td>@regulators</td>
</tr>
EOT
    }

    $content .= "</table>";
    $content .=
"<h3>Sequence</h3>\n\n<pre>".nazina::sequence_annotate(\$seq,
\@elements)."</pre>\n";

    return $content;
}
```

Appendix 2 LWF Application and Analysis Code

Java Files

LwfException.java

```
package edu.dartmouth.bglab.crm.lwf;

/**
 * Simple exception class
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.2 $
 * @update   $Date: 2006/02/03 06:56:29 $
 *
 */
public class LwfException extends Exception
{
    public LwfException()
    {
        super();
    }

    public LwfException(String s)
    {
        super(s);
    }

    public LwfException(Exception e)
    {
        super(e);
    }
}
```

Model.java

```
package edu.dartmouth.bglab.crm.lwf;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;



/**
 * LWF helper class; basically a bit-bucket for model data.
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.7 $
 * @update:  $Date: 2005/05/16 02:33:15 $
 *
 */
public class Model
{
    /** word length */
    public long   length        = 0;

    /** how many mismatches are tolerated per word */
    public long   mismatchCount = 0;

    /** size of detection window */
    public long   window        = 0;
    public long   windowSmall   = 0;
    public long   windowFull    = 0;

    /** maximum number of channels to create */
    public long   channelCount  = 0;

    // TODO: what does expr represent? it's calculated as the binomial
    // distribution of mismatches in a window over the length of a
string
    public double   expr          = 0;

    /** string containing list of files composing this model */
    public String fileList      = null;

    /** list of files composing this model */
    public String trainsets[]   = null;

    /** */ // TODO: why is probability 0.75?
```

```
    public double probability   = 0.75;

    /** maximum frequency counts for each channel */
    public long[] channels      = null;

    /** L-scores for each window in each channel */
    public double[][] desmatrix = null;

    // nickname for this model
    private String name         = null;

    // filename containing model's params and data
    private String filename     = null;

    private long countmdllines  = 0;
    private long countrows      = 0;



    /**
     * default constructor does nothing interesting
     *
     */
    public Model()
    {

    }



    /**
     * Creates a Model named 'name' prepared to read model parameters
     * from the file 'filename'.
     *
     * @param name nickname for this model
     * @param filename name of a file containing this models params and
data
     *
     */
    public Model(String name, String filename)
    {
        this.name     = name;
        this.filename = filename;

        this.countmdllines = 0;
        this.countrows     = 0;

    }



    /**
     * Parse model's datafile into this object.
     *
```

```
     * @throws FileNotFoundException if the model's datafile cannot be
found
     * @throws IOException if the model's datafile cannot be read
     */
    public void fileParse()
    throws FileNotFoundException, IOException
    {
        BufferedReader br = null;
        String line       = null;
        int j             = 0;

        br = new BufferedReader(new FileReader(this.filename));
        while ((line = br.readLine()) != null)
        {
            if (line.trim().equals(""))
                continue;

            String[] words = line.split(" ");

            // word-length
            if (line.matches("Word:.*"))
                this.length = (new Long(words[1])).longValue();

            // mismatch count
            if (line.matches("Mism:.*"))
                this.mismatchCount = (new Long(words[1])).longValue();

            // sliding window size
            if (line.matches("Win:.*"))
            {
                this.window      = new Long(words[1]).longValue() / 2;
                this.windowSmall = this.window;
                this.windowFull  = this.windowSmall * 2;
            }

            // frequency limits for each channel
            if (line.matches("Limits:.*"))
            {
                this.channels = new long[words.length - 1];
                for (int i = 1; i < words.length; i++)
                    this.channels[i-1] = (new
Long(words[i])).longValue();

                this.channelCount = this.channels.length - 1;
            }

            // L-scores for a channel define strength of a channel as
an
            // predictor of membership in a training set
            if ( line.matches("DMatrix:.*"))
            {
                if (this.desmatrix == null)
                    this.desmatrix = new
double[(int)this.channelCount][];
```

```
                this.desmatrix[j] = new double[words.length - 1];
                for (int i = 1; i < words.length; i++)
                    this.desmatrix[j][i-1] = (new
Double(words[i])).doubleValue();

                j++;
            }

            this.countmdllines++;
        }
        this.countrows = j;

    }

}
```

Scan.java

```
package edu.dartmouth.bglab.crm.lwf;

import edu.dartmouth.bglab.crm.lwf.scan.ProgressBar;
import edu.dartmouth.bglab.crm.lwf.scan.ScanSequenceParams;
import edu.dartmouth.bglab.crm.lwf.train.TrainingSet;
import edu.dartmouth.bglab.crm.lwf.train.TrainSequenceParams;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.BufferedInputStream;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Hashtable;
import java.util.Date;
import java.util.Properties;




/**
 * Scans test sequence using statistical models generated by Lwf-train
and
 * generates a recognition profile
 *
 * Input:
 * - file containing input sequences in fasta format, *.fa (test set)
 * - at least two *.sts files generated by LWF-train and containing
statistics
 *   for the positive and the negative training sets. Both *.sts files
must
 *   contain the same input parameters for the scan (see *.sts file
headers).
 *
 * Options:
 * - equalize channels: low values will supress difference between
channels
 * - profile cutoff
 * - extraction option (fraction of sequence length)
 * - peak width cutoff
 * - smoothing window
 * - GNUPlot option - can display profile for sequences &lt; 1MB
```

```
 *
 * Output:
 * - a directory containing files:
 * - *.mdl - model for each pair of the input *.sts files
 * - *.dat - recognition profiles for each sequence from the input *.fa
file.
 * - *.xtr.dat - coordinates of the extracted sequences
 * - *.xtr.fa - extracted sequences
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.14 $
 * @update:  $Date: 2006/02/03 06:56:29 $
 *
 */
public class Scan
{

    public static void main(String[] args)
    {
        try
        {
            Scan s = null;
            // use properties file for runtime params
            if (args != null && args.length > 0)
                s = new Scan(args[0]);
            else
                s = new Scan();

            // get runtime parameters including the sequence to parse
            // and the models to scan against, and read the raw
sequence
            // data.
            s.train();

            // create an LWF profile of the sequence and calculate
            // the feature score for the windows starting at each
position
            // in the sequence. results are saved in a temp-scan file.
            s.scan();

            // parse temp-scan files to find windows with feature
scores
            // beyond the cutoff representing stronger resemblance to
the
            // positive, rather than the negative, training set, i.e.
            // sequence substrings classified as CRMs.
            s.classify();
```

```
            System.out.print("\n\n");
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        catch (LwfException e)
        {
            System.err.println("ERROR: "+e.getMessage());
            e.printStackTrace(System.err);
        }

    }


    // positive/negative training sequences used to parse this sequence
    private TrainingSet ts = null;

    // run-time params
    private ScanSequenceParams params = null;

    // lists putative CRM sequence subscripts
    private PrintWriter xtr = null;

    // lists putative CRM sequences in FASTA format
    private PrintWriter fa = null;

    // model properties file
    private Properties props = null;

    // for pretty printing
    private NumberFormat nf = null;

    // max length of sequence to use for GNUPlot scanning results
    private int GNUPLOT_MAX = 1000000;



    /**
     * Constructs a new CRM scanner and initializes the output files
     * @throws IOException
     */
    public Scan()
    throws IOException
    {
        init();
    }
```

```
    /**
     * Constructs a new CRM scanner and reads sequence and scanning
     * data from a properties file.
     *
     * @param pfile properties file to read
     */
    public Scan(String pfile)
    {
        try
        {
            init();

            Properties runtimeProps = new Properties();
            BufferedInputStream b   = null;
            b = new BufferedInputStream(new FileInputStream(pfile));
            if (b != null)
                runtimeProps.load(b);

            this.props = runtimeProps;
        }

        // error reading properties file; set it to null and continue
        // as if we never had a file to read.
        catch (IOException e)
        {
            this.props = null;
        }

    }



    /**
     * Initialize instance variables.
     *
     */
    private void init()
    {
        this.ts = new TrainingSet();

        this.params = new ScanSequenceParams();

        this.nf = NumberFormat.getInstance();
        if (this.nf instanceof DecimalFormat)
            ((DecimalFormat)
this.nf).applyPattern("#.##############################");

    }



    /**
     * Get training params and sequence filenames, then create models
     * from the training data.
     *
```

```
     * @throws FileNotFoundException if any input file cannot be found
     * @throws IOException if any input file cannot be read
     * @throws LwfException if there are problems parsing a training
set
     *   (e.g. parse params for positive and negative sets don't match)
     */
    private void train()
    throws FileNotFoundException, IOException, LwfException
    {
        // if we have a properties file, get sequence and model
filenames
        // from it; otherwise, collect params from user
        if (this.props != null)
        {
            this.params.trainSet(
                this.props.getProperty("sequence"),
                this.props.getProperty("train"),
                (new
Integer(this.props.getProperty("equalize"))).intValue());
        }
        else
        {
            this.params.trainGet();
        }

        System.out.print("\n\nCREATING MODELS...\n\n");
        trainParse();

        // get scanning params
        String propsScan = this.props.getProperty("scan");
        if (this.props != null && propsScan != null &&
propsScan.equals("defaults"))
        {
            this.params.scanSet(
                this.params.profileCutoffSuggest(),
                this.params.peakWidthSuggest(),
                this.params.smoothSuggest(),
                this.params.plotSuggest());
        }
        else if (this.props != null)
        {
            this.params.scanSet(
                (new Double(this.props.getProperty("profile-
cutoff"))).doubleValue(),
                (new Integer(this.props.getProperty("peak-
width"))).intValue(),
                (new
Integer(this.props.getProperty("smooth"))).intValue(),
                (new
Boolean(this.props.getProperty("plot"))).booleanValue());
        }
        else
        {
            this.params.scanGet();
        }
```

```
        // read sequence file
        this.ts.sequenceFileParse(this.params.seqfname, this.params);

    }



    /**
     * Iterate through pairs of positive and negative TrainingSet
files,
     * parsing each into a model which is written to a model profile
     * file (*.mdl).
     *
     * @throws FileNotFoundException if a training file cannot be found
     * @throws IOException if a training file cannot be read
     * @throws LwfException if a training set parse params don't match
     */
    private void trainParse()
    throws FileNotFoundException, IOException, LwfException
    {
        Iterator i = this.params.stsList.iterator();
        while (i.hasNext())
        {
            // get training set files
            Hashtable h = (Hashtable) i.next();
            String stsPos = (String) h.get("pos");
            String stsNeg = (String) h.get("neg");

            // create model file
            String[] nameList = stsPos.split("\\.");
            String mdlFile    = this.params.scanoutdir + "/" +
nameList[0] + ".mdl";
            PrintWriter mpw   = new PrintWriter(new BufferedWriter(new
FileWriter(mdlFile)));

            // parse training set into a model
            modelCreate(mpw, stsPos, stsNeg);

            this.params.modelNames.add(nameList[0]);

            mpw.flush();
            mpw.close();
            mpw = null;
        }

    }



    /**
     * Parse a positive and negative training-set pair into a
     * sequence model. Store the model data in a model profile
     * file (*.mdl).
     *
```

```
     * @param mpw where to write model data
     * @param stsPos filename of positive training set
     * @param stsNeg filename of negative training set
     *
     * @throws FileNotFoundException if a training set file cannot be
found
     * @throws IOException if a training set file cannot be read
     * @throws LwfException if a training set parse params don't match
     */
    private void modelCreate(PrintWriter mpw, String stsPos, String
stsNeg)
    throws FileNotFoundException, IOException, LwfException
    {
        // stores all statistics from both training sets
        ArrayList statistics = new ArrayList();

        // read training sets and make sure parse params match
        TrainSequenceParams tPos, tNeg = null;
        tPos = trainParsePos(statistics, stsPos, mpw);
        tNeg = trainParseNeg(statistics, stsNeg, mpw);

        if (tPos.channelCount != tNeg.channelCount || tPos.length !=
tNeg.length)
            throw new LwfException(stsPos+" and "+stsNeg+" matrix
dimensions don't match; model excluded\n");

        // weight cummulative E-values for feature scores within each
channel
        double[][] matrix = new double[(int)tPos.channelCount][];
        for (int j = 0; j < tPos.channelCount; j++)
        {
            double[] cPos = (double[]) statistics.get(j);
            // TODO: why grab a different channel for the negative
set????
            double[] cNeg = (double[]) statistics.get(j +
(int)tPos.channelCount );

            matrix[j] = trainWeightEValues(cPos, cNeg);
        }

        // write training model to an output file
        trainModelDistributionWrite(mpw, matrix);

    }



    /**
     * Read data from the positive training set and write it to the
     * model file.
     *
     * //TODO: rewrite with fieldnames instead of line numbers
     *
     * @param list collection of E-value distributions for each channel
     *    in the training set.
```

```
     * @param filename trainingset filename to read
     * @param mpw model file to write to
     *
     * @return a TrainSequenceParams object with its parse-parameters
     *   filled in
     *
     * @throws FileNotFoundException if the trainingset file cannot be
found
     * @throws IOException if the trainingset file cannot be read
     *
     */
    private TrainSequenceParams trainParsePos(ArrayList list, String
filename, PrintWriter mpw)
    throws FileNotFoundException, IOException
    {
        TrainSequenceParams tsp = new TrainSequenceParams();

        int i = 0;
        BufferedReader br = null;
        String line       = null;
        br = new BufferedReader(new FileReader(filename));
        while ((line = br.readLine()) != null)
        {
            line = line.trim();

            // collect Word, Mism, Win, Exp, Chann, Limits params
            if (i > 0 && i < 7)
                mpw.println(line);

            // add suppression (how much to supress extreme L-scores)
            if (i == 7)
            {
                mpw.println("Sigma: "+this.params.supression);
                mpw.println("");
            }

            String[] words = line.split(" ");

            // window size; "Win: "
            if (i == 3)
            {
                tsp.window = (new Long(words[1])).longValue();
                this.params.mdlwins.add(new Long(tsp.window));
            }
            // channel count; "Chann: "
            if (i == 5)
                tsp.channelCount = (new Long(words[1])).longValue();
            // derived detection window size
            if (i == 10)
                tsp.length = words.length;
            if (i > 9)
                list.add(trainModelTrainWrite(mpw, words, "TrainPos"));

            i++;
        }
```

```
        br.close();

        mpw.println("");
        mpw.flush();

        return tsp;
    }



    /**
     * Read data from the negative training set and write it to
     * the model file.
     *
     * // TODO: rewrite with fieldnames instead of line numbers
     *
     * @param list collection of E-value distributions for each channel
     *    in the training set.
     * @param filename trainingset filename to read
     * @param mpw model file PrintWriter
     *
     * @return a TrainSequenceParams object with its parse-parameters
     *   filled in
     *
     * @throws FileNotFoundException if the trainingset file cannot be
found
     * @throws IOException if the trainingset file cannot be read
     */
    private TrainSequenceParams trainParseNeg(ArrayList list, String
filename, PrintWriter mpw)
    throws FileNotFoundException, IOException
    {
        int i = 0;
        TrainSequenceParams tsp = new TrainSequenceParams();
        BufferedReader br = null;
        String line       = null;
        br = new BufferedReader(new FileReader(filename));
        while ((line = br.readLine()) != null)
        {
            line = line.trim();
            String[] words = line.split(" ");

            // channel count; Chann:
            if (line.matches("Chann:.*"))
                tsp.channelCount = (new Long(words[1])).longValue();

            // derived detection window size
            if (i == 10)
                tsp.length = words.length;

            // L-scores for each word in a window
            if (i > 9)
                list.add(trainModelTrainWrite(mpw, words, "TrainNeg"));
```

```
            i++;
        }

        br.close();

        mpw.println("");
        mpw.flush();

        return tsp;
    }



    /**
     * Write feature scores for a channel in a training set to the
     * model file.
     *
     * @param mpw model file PrintWriter
     * @param evalues String representations of feature scores for a
     *    channel, as read from a training file.
     * @param header field header indicating data-type on this line
     *
     * @return coverted feature scores (String -> double) for this
channel
     */
    private double[] trainModelTrainWrite(PrintWriter mpw, String[]
evalues, String header)
    {
        double[] numbers = new double[evalues.length];
        mpw.print(header +": ");
        for (int j = 0; j < evalues.length; j++)
        {
            //System.err.println("value is: "+evalues[j]);
            mpw.print(evalues[j] + " ");
            numbers[j] = ((Double) new
Double(evalues[j])).doubleValue();
        }
        mpw.println("");

        return numbers;
    }



    /**
     * Weight the distribution (E) of feature scores (S) within a
channel
     * (j) away from the value S-0 at which membership in either the
     * positive or negative training set is equally likely.
     *
     * From Nazina and Papatsenko regardging equations 2 and 3 and
error
     * correction:
     *
```

```
     * Given functional classes omega-1 and omega-2 representing
positive
     * and negative training sets, and given a feature score S for a
test
     * sequence, the classification problem is solved for each
frequency
     * channel (j) independently using a log-likelihood ratio:
     *
     * [Equation 2]
     *
     *    L-j == log (E [omega-1 | S-j] / E [omega-2 | S-j] )
     *
     *
     * where E is the distribution of S in each channel j. L-j is zero
     * where E [omega-1 | S-j] == E [omega-2 | S-j], i.e. where
membership
     * in either class is equally likely. As noted in the Methods
section,
     * this value is much closer to the expectation for the negative
     * training set than for the positive one. To account for this
error,
     * the cummulative E values are weighted away from S-0 as follows:
     *
     * [Equation 3]
     *
     *   L == log (E [omega-1 | S] / E [omega-2 | S] )
     *        +
     *        log ((sum(E [omega-1 | S]) * (1 - sum(E [omega-1 | S])))
     *             /
     *             (sum(E [omega-2 | S]) * (1 - sum(E [omega-2 | S])))
     *            )
     *
     *
     * @param cPos E-values for one channel the positive training set
     * @param cNeg E-values for one channel the negative training set
     *
     * @return weighted E-values for this channel
     *
     */
    private double[] trainWeightEValues(double[] cPos, double[] cNeg)
    {
        // weighted E-values for this channel
        double[] row = new double[(int)cPos.length];

        // TODO: figure out why the weighted E-values have this skew
attached
        double skew = 0.001;

        for (int i = 0; i < cPos.length; i++)
        {
            double sumPos = Util.sumArray(cPos, 0, i);
            double sumNeg = Util.sumArray(cNeg, 0, i);

            // log(a/b) + log ((sumA (1 - sumA + a)) / sumB (1 - sumB +
b))
```

```
            double value =
                Math.log( (cPos[i] + skew) / (cNeg[i] + skew) )
                +
                Math.log(
                    ((sumPos * (1 - sumPos + cPos[i] )) + skew)
                    /
                    ((sumNeg * (1 - sumNeg + cNeg[i] )) + skew)
                );

            row[i] = value * Util.normalDistribution(value, 0,
this.params.supression);

        }

        return row;
    }



    /**
     * Write weighted E-values for a channel in combined training set
to
     * the model file.
     *
     * @param mpw model file PrintWriter
     * @param evalues weighted E-values for a channel as determined by
     *    processing the positive and negative training sets.
     *
     * @see trainWeightEValues
     */
    private void trainModelDistributionWrite(PrintWriter mpw,
double[][] evalues)
    {
        for (int i = 0; i < evalues.length; i++)
        {
            mpw.print("DMatrix: ");

            for (int j = 0; j < evalues[i].length; j++)
                mpw.print(evalues[i][j] + " ");

            mpw.println("");
        }
    }



    /**
     * Scan the sequences using the models created from the training
sets.
     * Recognition data temporarily is stored in a file for later
parsing.
     * Each sequence is parsed with all available models.
     *
     * @throws FileNotFoundException
     * @throws IOException
```

```
     */
    private void scan()
    throws FileNotFoundException, IOException
    {
        System.out.print("\n\nPARSING MODELS...\n\n");

        Iterator i = this.params.modelNames.iterator();
        while (i.hasNext())
            modelParse((String) i.next());
    }



    /**
     * Parse each sequence with the named model.
     *
     * @param modelName name of model to use for parsing
     *
     * @throws FileNotFoundException if there is trouble finding a
model file
     * @throws IOException if there is trouble reading a model file
     */
    private void modelParse(String modelName)
    throws FileNotFoundException, IOException
    {
        // read model data from its source file
        String modelFile = this.params.scanoutdir + "/" + modelName +
".mdl";
        Model model = new Model(modelName, modelFile);
        model.fileParse();

        // process each sequence with this model
        Iterator iter = ts.sequences.iterator();
        while (iter.hasNext())
        {
            Sequence s = (Sequence) iter.next();

            // find beginning of last window in the sequence
            s.maxPosition   = s.length - model.windowSmall -
model.length + 1;

            String outfile  = this.params.scanoutdir + "/" + s.filename
+ "~" + modelName +".tmp";
            PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(outfile)));
            Date timeStart  = new Date();
            ProgressBar progress = new ProgressBar(s.data, s.length,
model.window, modelName);

            // build channel distribution matrix for a sequence.
            // the index of the start of the final sliding window is
returned
            // so word distributions for all positions beyond that
index can
            // be zeroed out.
```

```
            long pos = sequenceParse(model, s, out, progress);
            long print0 = pos - model.window + 1;
            while (print0++ < pos + model.window + 3)
                out.println("0");

            out.flush();
            out.close();
            out = null;

            Date timeEnd = new Date();
            System.out.println("");
            System.out.println(runtimeDisplay(timeStart, timeEnd, s));

        }
    }



    /**
     * Create a channel distribution matrix for the given sequence
     * with the given model. The results are written to a file.
     *
     * @param model model to use for parsing
     * @param s sequence to parse
     * @param out PrintWriter for temp sequence recognition file
     * @param progress progress meter
     *
     * @return index of the start of the final sliding window within
the
     *    sequence, beyond which no frequency-distribution calculations
     *    were performed.
     */
    private long sequenceParse(Model model, Sequence s, PrintWriter
out, ProgressBar progress)
    {
        List channels, channelWordCount;
        channels  = new ArrayList();
        channelWordCount = null;
        long pos  = model.window;
        int i     = 0;

        // TODO: why start at model.window instead of at 0?
        while (pos < s.maxPosition)
        {
            // detect channels for words in this window
            // i.e. list of word-counts, per channel
            channelWordCount = Util.channelDetect(s, pos, channels,
model);

            // feature score of a channel representing the number of
            // words in a window with a frequency in a given range
            double featureScore = 0;

            if (channelWordCount != null && channelWordCount.size() >
1)
```

```
            {
                // channel number
                int j = 0;

                // loop over channels
                Iterator iter = channelWordCount.iterator();
                while (iter.hasNext())
                {
                    long wordsInChannel =
((Long)iter.next()).longValue();
                    featureScore +=
model.desmatrix[j][(int)wordsInChannel];
                    j++;
                }
            }
            out.println(featureScore);

//            if (i == progress.progress)
//                progress.update(i);

            i++;
            pos++;

        }

        return pos;
    }



    /**
     * Classify windows as resembling the positive or negative training
     * set. Positive windows, i.e. putative CRMs, are extracted and
     * written to output files.
     *
     * @throws FileNotFoundException if the scan file cannot be found
     * @throws IOException if the scan file cannot be read
     */
    private void classify()
    throws FileNotFoundException, IOException
    {
        System.out.print("\n\nWRITING SEQUENCE PROFILE DATA...\n\n");

        // gathers sequence data for a GNUPlot file
        ArrayList plot = new ArrayList();

        // prepare output files:
        // xtr contains substring indexes of putative CRMs in the input
file
        // fa contains FASTA sequences of the putative CRMs in the
input file
        String xtrFile = this.params.scanoutdir+"/" +
this.params.seqfname + ".xtr.dat";
        this.xtr       = new PrintWriter(new BufferedWriter(new
FileWriter(xtrFile)));
```

```
        String faFile  = this.params.scanoutdir+"/" +
this.params.seqfname + ".xtr.fa";
        this.fa        = new PrintWriter(new BufferedWriter(new
FileWriter(faFile)));

        // parse sequence-profile temp files created during scanning to
find
        // sequences with sufficient similarity to the positive
training set.
        // write data to the xtr and fa files
        Iterator i = ts.sequences.iterator();
        while (i.hasNext())
            sequenceClassifyConfig((Sequence) i.next(), plot);

        xtr.flush(); xtr.close();
        fa.flush(); fa.close();

        // write GNUPlot data file for short-ish sequences
        if (ts.length < this.GNUPLOT_MAX)
            plot(plot);

        System.out.print("\n\n");

    }



    /**
     * Determine positive-classification score threshold, then classify
     * the windows. Data for positive windows is saved in a file.
     *
     * A window's resemblance to the positive or negative training set
     * is calculated as a simple log-likelihood ratio of predicted
     * membership in the positive set over predicted membership in the
     * negative set. Windows with scores beyond a certain threshold are
     * classified as positive.
     *
     * Before this score is calculated, the S-scores (used to calculate
     * the L-score) are smoothed to remove extreme values that would
     * otherwise dominate a channel's recognition profile.
     *
     * @param s the Sequence containing windows to classify
     * @param plotData bit bucket for plotting sequence profile with
GNUPlot
     *
     * @see classifyThreshold
     */
    private void sequenceClassifyConfig(Sequence s, List plotData)
    throws FileNotFoundException, IOException
    {
        String filename = this.params.scanoutdir + "/" + s.filename +
".dat";
        PrintWriter dat = new PrintWriter(new BufferedWriter(new
FileWriter(filename)));
```

```
        double[] profileTmp = spTmpDataParse(s);
        double[] profile    = Util.smooth(this.params.inputSmooth,
profileTmp);
        double threshold    = classifyThreshold(profile);

        sequenceClassify(threshold, profile, s, dat);

        if (s.length < GNUPLOT_MAX)
            plotData.add(profile);

        dat.flush();
        dat.close();
        dat = null;
    }



    /**
     * Retrieve Sequence's scan data (feature scores for each window in
     * the Sequence) from the tmp file.
     *
     * //TODO: rewrite this with file-filters
     *
     * @param s Sequence for which data will be retrieved
     *
     * @return scan profile data for the input sequences
     *
     * @throws FileNotFoundException if the Sequence's scan file cannot
be found
     * @throws IOException if the Sequence's scan file cannot be read
     */
    private double[] spTmpDataParse(Sequence s)
    throws FileNotFoundException, IOException
    {
        double[] profile = new double[(int)s.length];

        File dir = new File(this.params.scanoutdir);
        String[] files = dir.list();
        for (int i = 0; i < files.length; i++)
        {
            if (files[i].matches(s.data+"~.*"))
            {
                BufferedReader br = null;
                String line       = null;
                br = new BufferedReader(new
FileReader(this.params.scanoutdir + "/"+files[i]));

                int j = 0;
                while ((line = br.readLine()) != null)
                {
                    line = line.trim();

                    if (! line.equals(""))
```

```
                        profile[j] = (new Double(line)).doubleValue() +
profile[j];
                    j++;
                }
                br.close();

                File tmp = new File(this.params.scanoutdir +
"/"+files[i]);
                tmp.delete();
                tmp = null;

            }
        }
        return profile;
    }



    /**
     * Calculate the L-score threshold that serves as the border
between
     * positive or negative classification as a CRM.
     *
     * A window's resemblance to the positive or negative training set
     * is calculated as a simple log-likelihood ratio of predicted
     * membership in the positive set over predicted membership in the
     * negative set. Windows with scores beyond a certain threshold are
     * classified as positive.
     *
     * @param bins
     * @param profiles feature scores for each window
     *
     * @return threshold distinguishing positive and negative CRM
status
     */
    private double classifyThreshold(double[] profile)
    {
        // number of bins to distribute profile[] items (feature
scores) into
        int bins = 1000;

        Hashtable hash = distributeFeatureScores(bins, profile);
        double[] distribution = (double[]) hash.get("distribution");
        double max = ((Double) hash.get("max")).doubleValue();
        double min = ((Double) hash.get("min")).doubleValue();

        int count = distribution.length - 1;


        for (double i = 1; i > 1 - this.params.profileCutoff; count--)
            i = distribution[count];

        double cutoff = min + ( ( count * ( max - min ) ) / bins );

        return cutoff;
```

```
    }


    /**
     * Distribute feature scores for each channel across a number
     * of bins.
     *
     * @param bins number of bins into which scores are sorted
     * @param scores feature scores for each window
     *
     * @return a hash containing the max and min feature scores and
     *    an array of distributed feature scores
     */
    private Hashtable distributeFeatureScores(int bins, double[]
scores)
    {
        // copy profile so original isn't modified; sort copy
descending
        double[] profileSort = new double[scores.length];
        System.arraycopy(scores, 0, profileSort, 0, scores.length);
        Util.reverseSort(profileSort);

        double min = profileSort[profileSort.length - 1];
        double max = profileSort[0];

        // feature score histogram
        double[] distribution = distribute(profileSort, min, max,
bins);

        double sumArray       = Util.sumArray(distribution, 0,
distribution.length);

        double sum = 0.0;
        for (int i = 0; i < distribution.length; i++)
        {
            distribution[i] /= sumArray;
            sum += distribution[i];
            distribution[i] = sum;
        }

        Hashtable hash = new Hashtable();
        hash.put("max", new Double(max));
        hash.put("min", new Double(min));
        hash.put("distribution", distribution);

        return hash;

    }


    /**
     * Distribute feature scores for each channel across a number
     * of bins, i.e. create a histogram of feature scores.
```

```
     *
     * @param scores all feature scores
     * @param min smallest feature scores
     * @param max largest feature score
     * @param bins number of bins into which scores are sorted
     *
     * @return a feature score histogram
     */
    private double[] distribute(double[] scores, double min, double
max, int bins)
    {
        double[] distribution = new double[(int)bins + 1];
        Arrays.fill(distribution, 0);

        double binSize = (max - min) / bins;

        if (binSize <= 0)
            return distribution;

        for (int i = 0; i < scores.length; i++)
        {
            // TODO: what is the 0.000001 adjustment for?
            int current = (int) ((scores[i] - min - 0.000001) /
binSize);

            if (current <= bins)
                distribution[current]++;
        }

        return distribution;
    }



    /**
     * Find sequence windows with L-scores above the threshold
representing
     * classification as a putative CRM.
     *
     * Classification data for all windows is written to the dat file.
     * For positively-classified sequences, sequence data and substring
     * indexes of the window in the source sequence are written to the
     * fa and xtr files.
     *
     * @param threshold L-score boundary between classification as
positive
     *    or negative
     * @param scores feature scores for each window in this sequence
     * @param s Sequence to classify
     * @param dat PrintWriter for recognition profile of this Sequence
     */
    private void sequenceClassify(double threshold, double[] scores,
Sequence s, PrintWriter dat)
    {
        long pos = 0, posStart = 0;
```

```
        boolean inCRM = false;

        for (int i = 0; i < scores.length; i++)
        {
            pos++;
            dat.println(scores[i]);

            // found the end of a CRM; save it
            if (inCRM && scores[i] < threshold)
            {
                inCRM = false;
                spProfilePrint(posStart, pos, s);
            }

            // found the beginning of a CRM; save its start index
            if ((! inCRM) && scores[i] > threshold)
            {
                inCRM = true;
                posStart = pos;
            }
        }

        // print final data if we ended in a CRM
        if (inCRM)
        {
            spProfilePrint(posStart, pos, s);
        }

    }



    /**
     * write sequence data to output files
     *
     * @param end start of this window
     * @param begin end of this window
     * @param s sequence to write
     */
    private void spProfilePrint(long begin, long end, Sequence s)
    {
        // CRMs must be wider than params.peakWidth
        if (end - begin > this.params.peakWidth)
        {
            this.xtr.println(s.data +" "+ begin + "-"+ end);
            String fasta = Util.fasta(s.sequence.substring(((int)begin)
- 1,(int) end));
            this.fa.println(">"+ s.data +" "+ begin +"-"+ end +"\n"+
fasta);

        }
    }
```

```
    /**
     * Write GNUPlot config file for displaying a sequence profile.
     *
     * @param list
     */
    private void plot(List list)
    throws IOException
    {
        String plotFile = this.params.scanoutdir + "/"
+this.params.seqfname + ".plt";
        PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(plotFile)));

        double[] profile = null;
        Iterator i = list.iterator();
        while (i.hasNext())
        {
            profile = (double[]) i.next();
            for (int j = 0; j < profile.length; j++)
                out.print(profile[j] + " ");
            out.println();
        }
        out.flush();
        out.close();

        out = new PrintWriter(new BufferedWriter(new
FileWriter("gnuplot.ini")));
        out.println("plot '"+ plotFile +"' with lines");
        out.flush();
        out.close();

        if (this.params.plot)
        {
            // system("wgnuplot") if $p->{'plotit'};
        }

    }



    /**
     * Calculate running time and processing speed (KB/second).
     *
     * @param start when processing started
     * @param end when processing ended
     * @param s the Sequence that was processed.
     */
    private String runtimeDisplay(Date start, Date end, Sequence s)
    {
        // running time in seconds
        long time = (end.getTime() - start.getTime()) / 1000;

        int speed = (int) ((s.length / (10 * time)) / 100);

        int hour = (int) (time / (60 * 60));
```

```
        int min  = (int) ((time / 60) % 60);
        int sec  = (int) (time % 60);

        return ("runtime: "
                +(hour > 9 ? ""+hour : "0" + hour)+":"
                +(min  > 9 ? ""+min  : "0" + min)+":"
                +(sec  > 9 ? ""+sec  : "0" + sec)+" ("+speed+" KB/s)");

    }
}
```

Sequence.java

```
package edu.dartmouth.bglab.crm.lwf;

import java.io.PrintStream;
import edu.dartmouth.bglab.crm.lwf.train.TrainSequenceParams;



/**
 * LWF helper class; basically a bit-bucket for sequence-related data.
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.3 $
 * @update:  $Date: 2006/02/07 23:10:23 $
 *
 */
public class Sequence
{
    /** length of sequence in chars */
    public long length         = 0;

    /** file header data */
    public String data         = null;

    /** */
    public String input        = null;

    /** the sequence characters */
    public String sequence     = null;

    /** */
    public long maxPosition    = 0;

    /** source filename */
    public String filename     = null;



    /**
     * simple constructor requires the filename this sequence was
     * created from
     *
     */
    public Sequence(String filename)
    {
        this.filename = filename;
```

```
        this.input    = new String();
        this.sequence = new String();
    }



    /**
     * simple constructor sets some params
     *
     * @param data
     * @param length
     * @param sequence
     * @param input
     */
    public Sequence(String data, long length, String sequence, String
input)
    {
        this.data     = data;
        this.length   = length;
        this.sequence = sequence;
        this.input    = input;
    }



    /**
     * print information about this seq to the specified stream
     * @param sequenceName
     * @param p
     * @param out
     */
    public void headerPrint(String sequenceName, TrainSequenceParams p,
PrintStream out)
    {
        out.println("Data: "   + sequenceName);
        out.println("Word: "   + p.length);
        out.println("Mism: "   + p.mismatchCount);
        out.println("Win: "    + p.window);
        out.println("Exp: "    + p.expr);
        out.println("Chann: "  + p.channelCount);
        out.println("Limits: " + p.channelsString());
        out.println("");
        out.println("Statistics: ");
        out.println("");
    }
}
```

Train.java

```
package edu.dartmouth.bglab.crm.lwf;


import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.PrintWriter;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;
import java.util.Date;
import java.util.Properties;
import java.text.NumberFormat;
import java.text.DecimalFormat;

import edu.dartmouth.bglab.crm.lwf.train.TrainProgressBar;
import edu.dartmouth.bglab.crm.lwf.train.TrainSequenceParams;
import edu.dartmouth.bglab.crm.lwf.train.TrainingSet;



/**
 * Generates statistical model for a training set based on word
 * frequency distributions.
 *
 * input
 *     * files containing input sequences in fasta format
 *
 * parameters
 *     * word length
 *     * string divergence (number of mismatches)
 *     * sliding window size
 *     * number of frequency channels
 *
 * output
 *     * .sts files containing corresponding word frequency statistics
 *        for each training set
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.9 $
 * @update:  $Date: 2005/05/16 01:20:05 $
```

```
 *
 */
public class Train
{
    /**
     * collect training parameters, then parse the input file into
     * a training set file.
     *
     * @param args
     */
    public static void main(String[] args)
    {
        try
        {
            Train t = null;
            if (args != null && args.length > 0)
                t = new Train(args[0]);
            else
                t = new Train();

            // get and display calculated params
            // user properties if availablet; otherwise prompt user
            if (t.props != null)
            {
                t.params.set(
                        (new Integer(t.props.getProperty("word-
length"))).intValue(),
                        (new Integer(t.props.getProperty("mismatch-
count"))).intValue(),
                        (new Integer(t.props.getProperty("window-
length"))).intValue(),
                        (new Integer(t.props.getProperty("channel-
count"))).intValue(),
                        t.props.getProperty("files")
                        );
            }
            else
            {
                t.params.get();
            }

            t.params.show();

            Date startTime = new Date();

            // parse the training files
            for (int i = 0; i < t.params.trainsets.length; i++)
                t.parseFile(t.params.trainsets[i]);

            Date endTime = new Date();

            long time = endTime.getTime() - startTime.getTime();
            System.out.println("runtime: "+(time/1000)+" seconds\n\n");

        }
```

```
    catch (FileNotFoundException e)
    {
        e.printStackTrace();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    catch (LwfException e)
    {
        System.err.println(e.getMessage());
    }

}



// runtime params
public TrainSequenceParams params = null;

// model properties file
public Properties props = null;

// for pretty printing
private NumberFormat nf = null;


/**
 * simple constructor initializes instance variables
 *
 */
public Train()
{
    init();
}



/**
 * Constructs a new Trainer with runtime parameters set in a
 * properties file.
 *
 * @param pfile name of the properties file
 *
 * @throws IOException if the properties file cannot be read
 */
public Train(String pfile)
throws IOException
{
    try
    {
        init();

        Properties runtimeProps = new Properties();
        BufferedInputStream b   = null;
```

```
            b = new BufferedInputStream(new FileInputStream(pfile));
            if (b != null)
                runtimeProps.load(b);
            else
                System.err.println("ERROR: could not read "+pfile);

            this.props = runtimeProps;

        }

        // error reading properties file; set it to null and continue
        // as if we never had a file to read.
        catch (IOException e)
        {
            System.err.println("couldn't read "+pfile);
            this.props = null;
        }
    }



    /**
     * Initialize instance variables.
     *
     */
    private void init()
    {
        this.params = new TrainSequenceParams();

        this.nf = NumberFormat.getInstance();
        if (this.nf instanceof DecimalFormat)
            ((DecimalFormat)
this.nf).applyPattern("#.##############################");

    }



    /**
     * parse a FASTA file into a training set file
     *
     * @param filename FASTA sequence file to read
     *
     * @throws FileNotFoundException if the sequence file cannot be
found
     * @throws IOException if the sequence file cannot be read
     */
    private void parseFile(String filename)
    throws FileNotFoundException, IOException
    {
        // distribution matrix of words in channels
        long[][] distribution = Util.matrix(this.params.windowFull,
this.params.channelCount, 0);
        int shiftRows = 0;
```

```
        // output filename + header
        String outfile = this.params.length
+"_"+this.params.mismatchCount+"_"+this.params.window+"_"+this.params.c
hannelCount+"_"+filename+".sts";

        // read sequence file, then prepare outfile
        TrainingSet ts  = new TrainingSet(filename, this.params);
        PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(outfile)));
        headerPrint(filename, out);

        TrainProgressBar progress = new TrainProgressBar(filename,
ts.length);

        // parse the sequences in the input file
        int count = 0;
        java.util.Iterator i = ts.sequences.iterator();
        while (i.hasNext())
            count = parseSequence(distribution, shiftRows, (Sequence)
i.next(), progress, count);

        shiftRows++;
        distributionMatrixWrite(distribution, out);
        out.flush();
        out.close();
        System.out.println("");
        System.out.println("");

    }



    /**
     * build distribution matrix for a sequence
     *
     * @param matrix
     * @param shiftrows
     * @param s sequence to parse
     * @param progress progress meeter
     *
     */
    private int parseSequence(long[][] matrix, long shiftrows, Sequence
s, TrainProgressBar progress, int i)
    {
        // TODO: why initialize to windowSmall instead of 0?
        long pos          = this.params.windowSmall;
        List detectChann  = new ArrayList();
        List channelNSum  = new ArrayList();

        while (pos < s.maxPosition)
        {
            // detect channels for words in this window
            channelNSum = Util.channelDetect(s, pos, detectChann,
this.params);
```

```
            if (channelNSum != null && channelNSum.size() > 1)
            {
//                System.out.println("channelNSum.size:
"+channelNSum.size());
                long countChanSum = 0;
                Iterator iter = channelNSum.iterator();
                while (iter.hasNext())
                {
                    int test1 = (int) (this.params.channelCount *
shiftrows + countChanSum);
                    int test2 = ((Long) iter.next()).intValue();

                    // TODO: figure out why test2 is ever successful
                    if (test1 >= matrix.length || test2 >=
matrix[test1].length)
                    {
                        System.out.println("DANGER: "+test1+" >= "
+matrix.length + " || "+test2 + " >= "+ matrix[test1].length);
                        continue;
                    }

//                      matrix[ new Long(this.params.channelCount *
shiftrows + countChanSum).intValue() ][ ((Long) iter.next()).intValue()
]++;
                    matrix[ new Long(this.params.channelCount *
shiftrows + countChanSum).intValue() ][ test2 ]++;
                    countChanSum++;

                }
            }

//              if (i == progress.progress)
//                  progress.update(i);

            i++;
            pos++;

        }
        return i;
    }



    /**
     * write the distribution matrix for each channel to the output
file
     *
     * @param channels 2D array of windows in each channel
     * @param out output file
     */
    private void distributionMatrixWrite(long[][] channels, PrintWriter
out)
    {
        int distrmrows = 0;
```

```
        for (int i = 0; i < channels.length; i++)
        {
            long[] current = channels[i];
            long unique    = current[0];

            int count0 = 0;
            for (int j = 1; j < current.length; j++)
            {
                if (0 == current[j])
                    count0++;
            }

            // smooth values and shift unique value back onto list
            double[] smoothMatrix = smoothed(current, count0, unique);

            double sumSmoothMatrix = Util.sum(smoothMatrix);
            for (int j = 0; j < smoothMatrix.length; j++)
                smoothMatrix[j] /= sumSmoothMatrix;

            // print it
            if (distrmrows++ == this.params.channelCount)
                out.println();
            for (int j = 0; j < smoothMatrix.length; j++)
                out.print(this.nf.format(smoothMatrix[j]) + " ");

            out.println();

        }

    }



    /**
     * Smooth values on the list, and return a list with the unique
     * value shifted back onto the start of th list. The original list
     * is not modfied.
     *
     * @param list list of values to smooth
     * @param count0
     * @param unique list value which is exempt from smoothing
     *
     * @return smoothed list
     */
    private double[] smoothed(long[] list, int count0, long unique)
    {
        long[] currentShift = new long[list.length - 1];

        System.arraycopy(list, 1, currentShift, 0,
currentShift.length);

        int smoothRate = StrictMath.round((this.params.windowFull -
count0) / 10);
        double[] smoothMatrixTemp = Util.smooth(smoothRate,
currentShift);
```

```
        double[] smoothMatrix = new double[smoothMatrixTemp.length +
1];
        smoothMatrix[0] = unique;
        System.arraycopy(smoothMatrixTemp, 0, smoothMatrix, 1,
smoothMatrixTemp.length);

        return smoothMatrix;

    }



    /**
     * print output file header
     *
     * @param sequenceName
     * @param out
     */
    public void headerPrint(String sequenceName, PrintWriter out)
    {
        out.print("Data: "  + sequenceName + "\n");
        out.print("Word: "  + this.params.length + "\n");
        out.print("Mism: "  + this.params.mismatchCount + "\n");
        out.print("Win: "   + this.params.window + "\n");
        out.print("Exp: "   + this.params.expr + "\n");
        out.print("Chann: " + this.params.channelCount + "\n");
        out.print("Limits: " + this.params.channelsString() +"\n\n");
        out.print("Statistics:\n\n");
    }
}
```

Util.java

```
package edu.dartmouth.bglab.crm.lwf;

import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;
import java.util.Collection;
import java.util.Arrays;

import edu.dartmouth.bglab.crm.lwf.Word;



/**
 * helper methods for train.pl, including input parsing and
mathematical
 * functions
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.12 $
 * @update:  $Date: 2005/05/16 02:33:15 $
 *
 */
public class Util
{

    /**
     * return sum of input values
     *
     * @param c list of values to sum
     *
     * @return sum of values on list
     *
     */
    public static double sum(Collection c)
    {
        long total = 0;
        Iterator i = c.iterator();
        while (i.hasNext())
            total += ((Long) i.next()).longValue();

        return total;
    }
```

```
    /**
     * return sum of values on list
     *
     * @param c list of values to sum
     *
     * @return sum of values on list
     */
    public static double sum(double c[])
    {
        double total = 0;
        for(int i = 0; i < c.length; i++)
            total += c[i];

        return total;
    }



    /**
     * compute probability distribution for a binomial distribution
     * allowing $mism mismatches over a sliding window of length
$winSize,
     *
     *
     * @param n - sample size (length of string)
     * @param mismatch - number of mismatches permitted
     * @param winSize - sliding window size
     *
     * @return
     */
    public static double binomial(long n, long mismatch, long winSize,
double p)
    {
        double prob = 0.0;

        for (long x = 0; x <= mismatch; x++)
        {
            prob += winSize * ( StrictMath.pow(p,x) ) * (
StrictMath.pow((1.0 - p),( n - x )) ) *
                (factorial(n) / (factorial(x) * factorial(n - x)));
        }

        prob = 1 + 2 * prob;

        return prob;
    }



    /**
     * compute factorial of a number
     *
     * @param f number to compute factorial of
     *
```

```
     * @return factorial of f
     *
     */
    public static double factorial(double f)
    {
        double value = 1;
        while (f > 0)
        {
            value *= f--;
        }
        return value;
    }



    /**
     * create frequency channels based on the number of desired
channels
     * (channelCount) and their expected normal distribution around
     * expr. The value for each item on the returned list is the
     * maximum frequency contained by that channel.
     *
     * @param channelCount number of channels
     * @param expr
     *
     * @return
     */
    public static long[] getChannels(long channelCount, double expr)
    {
        List channels       = new ArrayList();
        long limit_previous = 0;

        channels.add(new Long(0));
        for (int i = 1; i < channelCount; i++)
        {
            int temp = new Double(0.5 + channelCount / 2).intValue();
            double limChann  = StrictMath.pow((i *
(StrictMath.sqrt(expr) / temp)), 2);

            long limChannInt =
StrictMath.round(StrictMath.floor(limChann));
            if (limChannInt != limit_previous)
            {
                channels.add(new Long(limChannInt));
                limit_previous = StrictMath.round(limChann);
            }
        }

        // convert the List channels into an array
        long[] channelArray = new long[channels.size() + 1];
        int j = 0;
        Iterator i = channels.iterator();
        while (i.hasNext())
            channelArray[j++] =  ((Long) i.next()).longValue();
```

```
        return channelArray;
    }



    /**
     * create a 2D matrix
     *
     * @param cols
     * @param rows
     * @param value
     *
     * @return a new rows-by-cols matrix with value in every cell
     */
    public static long [][] matrix(long cols, long rows, long value)
    {
        long matrix[][] = new long[new Long(rows).intValue()][new
Long(cols).intValue()];

        for (int i = 0; i < rows; i++)
            Arrays.fill(matrix[i], value);

        return matrix;
    }



    /**
     * smooth values on a list
     *
     * @param rate how much to smooth values
     * @param list values to smooth
     *
     * @return a copy of the list with its values smoothed
     */
    public static double[] smooth(double rate, long list[])
    {
        double smoothList[] = new double[list.length];

        double item  = 0.00;       // current item
        double sum   = 0.00;       // current sum
        double count = 0;          // item count
        int intCount = 0;

        for (int i = 0; i < list.length; i++)
        {
            if (i < rate)
                item = 1 + 2 * i;
            else if (list.length - i - 1 < rate)
                item = 2 * (list.length - i) - 1;
            else
                item = 2 * rate + 1;

            sum = 0;
            count = (1 - item) / 2;
```

```
        while (count <= (item - 1)/2)
        {
            intCount = (int) count;
            sum += list[i + intCount];
            count++;
        }
        smoothList[i] = sum/item;
    }

    return smoothList;
}



/**
 * smooth values on a list
 *
 * @param rate how much to smooth values
 * @param list values to smooth
 *
 * @return a copy of the list with its values smoothed
 */
public static double[] smooth(double rate, double list[])
{
    double smoothList[] = new double[list.length];

    double item  = 0.00;       // current item
    double sum   = 0.00;       // current sum
    double count = 0;          // item count
    int intCount = 0;

    for (int i = 0; i < list.length; i++)
    {
        if (i < rate)
            item = 1 + 2 * i;
        else if (list.length - i - 1 < rate)
            item = 2 * (list.length - i) - 1;
        else
            item = 2 * rate + 1;

        sum = 0;
        count = (1 - item) / 2;
        while (count <= (item - 1)/2)
        {
            intCount = (int) count;
            sum += list[i + intCount];
            count++;
        }
        smoothList[i] = sum/item;
    }

    return smoothList;
}
```

```
/**
 * return the largest value on a list
 *
 * @param list
 * @return
 */
public static long max(long[] list)
{
    long max = list[0];
    for (int i = 0; i < list.length; i++)
        if (list[i] > max)
            max = list[i];

    return max;
}



/**
 * return the largest value on a list
 *
 * @param list
 * @return
 */
public static long max(List list)
{
    Long max = (Long) list.get(0);

    Iterator i = list.iterator();
    while (i.hasNext())
    {
        Long l = (Long) i.next();
        if (l.compareTo(max) > 0)
            max = l;
    }

    return max.longValue();
}



/**
 * return input in FASTA format
 *
 * @param s string to format
 * @return FASTA-formatted string
 */
public static String fasta(String s)
{
    int width          = 70;
    int l              = s.length();
    StringBuffer fasta = new StringBuffer();

    for (int i = 0; i < l; i += width)
```

```
        {
            if (i + width > l)
                fasta.append(s.substring(i) + "\n");
            else
                fasta.append(s.substring(i, i + width) + "\n");
        }

        return fasta.toString();

    }



    /**
     * in a window from 0 - p.window_full in sequence, count the
frequency
     * of word.
     *
     * @param p
     * @param sequence
     * @param globbackwin
     * @param word
     */
    public static void freqCount(Model m, String sequence, long
globbackwin, Word word)
    {
        int position = 0;

        while (position < m.windowFull + 1)
        {
            int offsetStart = (int)(position + globbackwin);
            int offsetEnd   = (int)(offsetStart + m.length);
            char[] string   = sequence.substring(offsetStart,
offsetEnd).toLowerCase().toCharArray();

            int[] mismatches = mismatchCount(string, word, m);

            if (mismatches[0] <= m.mismatchCount)
                word.frequency++;
            if (mismatches[1] <= m.mismatchCount)
                word.frequency++;

            position++;

        }
    }



    /**
     * count mismatches between two words, allowing p.mismatch_count
     * mismatches between the two strings.
     *
     * @param string
     * @param word
```

```
     * @param p
     * @return
     */
    public static int[] mismatchCount(char string[], Word word, Model
m)
    {
        int j   = 0;
        int[] mismatches = new int[2];

        for (int i = 0; i < string.length; i++)
        {

            if (string[i] != word.aWord[j])
                mismatches[0]++;
            if (string[i] != word.aWordR[j])
                mismatches[1]++;

            if (mismatches[0] > m.mismatchCount && mismatches[1] >
m.mismatchCount)
                break;

            j++;
        }

        return mismatches;

    }



    /**
     * loop over the list of channels in order of frequency and stop
when the
     * channel frequency is higher than the word frequency, i.e. stop
at the
     * channel that contains the current word based on its frequency.
return
     * this index int to the array.
     *
     * @param word - word with a frequency count
     * @param list - list of frequency channels
     *
     * @return array index of the frequency channel containing this
word
     */
    public static int limitCount(Word word, long[] list)
    {
        // TODO: throw an error here instead?
        if (list == null)
            return 0;


        for (int i = 0; i < list.length; i++)
        {
            if (word.frequency <= list[i])
```

```
                return i;
        }

        // TODO: throw an error here instead?
        System.err.println("word frequency "+word.frequency +" is not
contained in any channel");
        for (int i = 0; i < list.length; i++)
            System.err.print (list[i] + " ");
        System.err.println("");
        return 0;
    }



    /**
     * count the number of members that each frequency channel has in a
given
     * window. channels is a list of channels for each word in the
window. the
     * channels themselves are just values on an array, so an array
with the
     * same dimsensions (sum) is used to store the member-counts for
each channel.
     *
     * @param channels - list of channels for each word in a window
     * @param m.channelCount - how many channels to count
     *
     * @return
     */
    public static List channelMemberCount(List channels, Model m)
    {
        ArrayList sum = new ArrayList();
        int count = 1;

        Iterator i = null;
        while (count <= m.channelCount)
        {
            int members = 0;
            i = channels.iterator();
            while (i.hasNext())
            {
                if (((Long) i.next()).longValue() == count)
                    members++;
            }
//            System.err.println("members: "+members);
            sum.add(new Long(members));
            count++;
        }

//        System.err.println("");
        return sum;
    }
```

```
    /**
     * detect channels for words in a window.
     *
     * @param s - DNA string to parse
     * @param pos - current position in the string
     * @param channels - list of channels detected so far
     * @param m - model for this sequence
     *
     * @return list of member-counts, per-channel
     */
    public static List channelDetect(Sequence s, long pos, List
channels, Model m)
    {
        List sum = null;

        int offsetStart  = (int) pos;
        int offsetEnd    = offsetStart + (int) m.length;
        Word word        = new Word(s.sequence.substring(offsetStart,
offsetEnd));

        long globBackWin = pos - m.windowSmall;

        // count word's frequency in this sequence window.
        // store count in word.frequency
        Util.freqCount(m, s.sequence, globBackWin, word);

        // save the index of the channel containing this word
        int limitCount = Util.limitCount(word, m.channels);
        channels.add(new Long(limitCount));

        // we've determined channels for all words; now store each
channel's
        // member-count in sum.
        if (channels.size() == m.windowFull)
        {
            sum = Util.channelMemberCount(channels, m);
            channels.remove(0);
        }

        return sum;
    }



    /**
     * return sum of values in list between from and to, inclusive.
     *
     * @param list list of values to sum
     * @param from begin index
     * @param to end index
     * @return sum of values in list between from and to, inclusive.
     */
    public static double sumArray(double[] list, int from, int to)
    {
        // TODO: throw an exception here instead
```

```
        if (to >= list.length)
            to = list.length - 1;

        double total = 0;
        for (int i = from; i <= to; i++)
            total += list[i];

        return total;
    }



    /**
     * descending in-place sort of a list
     *
     * @param list to be reverse-sorted
     */
    public static void reverseSort(double [] list)
    {
        // ascending sort
        Arrays.sort(list);

        // reverse it
        int max = list.length / 2;
        double j;
        for (int i = 0; i < max; i++ )
        {
            j = list[i];
            list[i] = list[list.length - i - 1];
            list[list.length - i - 1] = j;
        }
    }



    /**
     * Plot outcome of a continuous random variable x.
     *
     * // TODO: figure out why sigma is outside the squareroot
     * compute a normal distribution (density function)
     *
     * @param x -
     * @param mu - mean
     * @param sigma - standard deviation
     *
     * @return
     */
    public static double normalDistribution(double x, double mu, double
sigma)
    {
//        return ((1 / (Math.sqrt(2 * Math.PI * sigma))) *
//                Math.pow(Math.exp(1), ((-1) * Math.pow((x - mu), 2) /
(2 * Math.pow(sigma, 2)))));

        return ((1 / (sigma * Math.sqrt(4 * Math.atan2(1, 0)))) *
```

```
                Math.pow(Math.exp(1), ((-1) * Math.pow((x - mu), 2) /
(2 * Math.pow(sigma, 2)))));
    }

}
```

Word.java

```
package edu.dartmouth.bglab.crm.lwf;

import java.lang.StringBuffer;



/**
 * LWF helper class stores information about a substring
 * (word) within a sequence
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.2 $
 * @update:  $Date: 2006/02/07 23:10:23 $
 *
 */
public class Word
{

    public String word      = null;
    public String wordR     = null;
    public char aWord[]     = null;
    public char aWordR[]    = null;
    public long frequency   = 0;

    public Word(String w)
    {
        this.word = w.toLowerCase();

        this.wordR  = new StringBuffer(this.word).reverse().toString();
        this.aWordR = this.wordR.toCharArray();
        for (int i = 0; i < this.aWordR.length; i++)
        {
            if (this.aWordR[i] == 'a')
                this.aWordR[i] = 't';
            else if (this.aWordR[i] == 't')
                this.aWordR[i] = 'a';
            else if (this.aWordR[i] == 'g')
                this.aWordR[i] = 'c';
            else if (this.aWordR[i] == 'c')
                this.aWordR[i] = 'g';
        }

        this.wordR = new String(this.aWordR);
```

```
            this.aWord  = this.word.toCharArray();
        }
    }
```

TrainProgressBar.java

```
package edu.dartmouth.bglab.crm.lwf.train;

import java.text.DecimalFormat;
import java.text.NumberFormat;

/**
 * LWF helper class has methods for displaying a progress
 * thermometer as parsing progresses.
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.3 $
 * @update:  $Date: 2006/02/11 01:37:34 $
 *
 */
public class TrainProgressBar
{
    private String seqName;
    private long seqLen;
    private long inc;
    public long progress;
    private StringBuffer bar;
    private NumberFormat nf;        // thermometer progress formatter


    // width of thermometer
    private final int WIDTH = 70;


    public TrainProgressBar()
    {
        this.bar = new StringBuffer();
    }

    public TrainProgressBar(String seqName, long length)
    {
        this.bar = new StringBuffer();
        init(seqName, length);
    }



    public void init(String seqName, long length)
    {
        this.seqName = seqName;
```

```
        this.seqLen = length;
        this.inc = length / 70;
        this.progress = this.inc;
        this.bar.append(".");

        this.nf = NumberFormat.getInstance();
        if (this.nf instanceof DecimalFormat)
            ((DecimalFormat) this.nf).applyPattern("##.##");


        System.out.println("");
        System.out.println("Building statistics for "+seqName+"...");
    }




    /**
     * update the progress thermometer
     *
     * print a thermometer that looks like this:
     *   4.25 [..........
]
     * the percentage and progress of the dots between the [ ] brackets
is
     * regularly updated by appending \r instead of \n to the end of
each
     * line
     *
     * @param i
     */
    public void update(long i)
    {
        this.bar.append(".");
        this.progress += this.inc;

        System.out.print("\r" + this.bar + "\r" + ( 100 * i /
this.seqLen ));

        StringBuffer pad = new StringBuffer();
        int max = this.WIDTH - this.bar.length();
        for (int j = 0; j <= max; j++)
            pad.append(" ");

        System.out.print( "\r       ["+this.bar + pad +"]\r
"+this.nf.format(100 * i / (double)this.seqLen ) +"\r");
        //                ^        ^                ^   ^
        //                ^        thermometer      ^   percent
complete so far
        //                return to|start of line       return
to|start of line



    }
```

```
}
```

TrainSequenceParams.java

```
package edu.dartmouth.bglab.crm.lwf.train;

import java.lang.StrictMath;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
import edu.dartmouth.bglab.crm.lwf.Util;
import edu.dartmouth.bglab.crm.lwf.Model;
import edu.dartmouth.bglab.crm.lwf.LwfException;



/**
 * LWF helper class; bit-bucket for runtime model-creation parameters.
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.10 $
 * @update:  $Date: 2006/02/03 06:56:28 $
 *
 */
public class TrainSequenceParams extends Model
{

    /**
     * default constructor
     *
     */
    public TrainSequenceParams()
    {

    }



    /**
     * constructor with some params
     * @param length
     * @param mmc
     * @param win
     * @param winSmall
     * @param cCount
     * @param files
     */
```

```
    public TrainSequenceParams(long length, long mmc, long win, long
winSmall, long cCount, String files)
    throws LwfException
    {
        set(length, mmc, win, cCount, files);

        this.windowSmall   = winSmall;
    }



    /**
     * show information about this run
     *
     */
    public void show()
    {
        System.out.println("");
        System.out.println("----------------------------------------
--------");
        System.out.println("Expected frequency: "+this.expr);
        System.out.println("Channels created:   "+this.channelCount);
        System.out.println("Channel limits:     "+channelsString());
        System.out.println("----------------------------------------
--------");
        System.out.println("");

    }



    /**
     * return channel list as a string
     *
     * @return
     */
    public String channelsString()
    {
        StringBuffer s = new StringBuffer();
        for (int i = 0; i < this.channels.length; i++)
            s.append(this.channels[i] + " ");

        return s.toString();
    }



    /**
     * gather input params
     *
     * @throws IOException if there is trouble reading params from
command line
     */
    public void get()
    throws IOException, LwfException
```

```
    {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String line = null;

        long length  = 0;
        long mmc     = 0;
        long win     = 0;
        long cCount  = 0;
        String files = null;

        while (files == null || files.equals(""))
        {
            // word length
            System.out.print("\nSelect string length (1-12) [5]: ");
            line = in.readLine().trim();
            length = line.equals("") ? 5 : new
Integer(line).intValue();

            // mismatch count
            long suggestmism = StrictMath.round(length / 3 - 0.1);
            System.out.print("\nSelect number of mismatches (0-4)
["+suggestmism+"]: ");
            line = in.readLine().trim();
            mmc = line.equals("") ? suggestmism : new
Integer(line).intValue();

            // window size
            long suggestwin  = StrictMath.round(64 *
StrictMath.pow(2.0, suggestmism));
            System.out.print("\nSelect detection window size (64-512)
["+suggestwin+"]: ");
            line = in.readLine().trim();
            win = line.equals("") ? suggestwin : new
Integer(line).intValue();

            // channel count
            long suggestccount = 10;
            System.out.print("\nSelect maximal number of channels (2-
24) ["+suggestccount+"]: ");
            line = in.readLine().trim();
            cCount = line.equals("") ? suggestccount : new
Integer(line).intValue();

            // sequence file names
            System.out.print("\nEnter training sequence filenames
(FASTA format): ");
            files = in.readLine().trim();
        }

        set(length, mmc, win, cCount, files);

    }
```

```
    public void set(long length, long mmc, long win, long cCount,
String files)
    throws LwfException
    {

        if (mmc >= length)
            throw new LwfException("mismatch count ("+mmc+") >= word-
length ("+length+")");

        // word length
        this.length = length;

        // mismatch count
        this.mismatchCount = mmc;

        // window size
        this.window = win;
        this.windowSmall =
StrictMath.round(StrictMath.floor(this.window / 2));
        this.windowFull  = this.windowSmall * 2;

        // channel count
        this.channelCount = cCount;

        // sequence file names
        this.fileList = files;


        // sequence filenames
        this.trainsets = this.fileList.split("\\s");

        this.expr      =  Util.binomial(this.length,
this.mismatchCount, this.window, this.probability);
        System.err.println("expr: "+expr + "based on len/mism/win/prob:
"
                + this.length +"/"+ this.mismatchCount+"/"+
this.window+"/"+ this.probability);
        // get channels
        this.channels     = Util.getChannels( this.channelCount,
this.expr );
        this.channelCount = this.channels.length - 1;

        // add a final channel wide enough for every word on each
        // strand to be unique.
        this.channels[this.channels.length - 1] = 2 * this.window;

    }
}
```

TrainingSet.java

```
package edu.dartmouth.bglab.crm.lwf.train;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;

import edu.dartmouth.bglab.crm.lwf.Sequence;
import edu.dartmouth.bglab.crm.lwf.Util;
import edu.dartmouth.bglab.crm.lwf.Model;



/**
 * Stores run-time parameters about a set of paired positive and
negative
 * training sequences that will later be used to scan an unknown
sequence
 * to determine whether it more closely resembles the positive or
negative
 * training set.
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.7 $
 * @update:  $Date: 2006/02/13 01:17:13 $
 *
 */
public class TrainingSet
{
    /** number of Sequences in this set */
    public long count     = 0;

    /** combined length of Sequences in this set */
    public long length    = 0;

    /** name of the file containing the raw sequence data */
    public String input   = null;

    /** length of each sequence */
    public List lengths   = null;
```

```
    /** Sequence objects in this training set */
    public List sequences = null;



    /**
     * initializes lists
     *
     */
    public TrainingSet()
    {
        init();
    }



    /**
     * initializes lists and reads a sequence file
     *
     * @param filename FASTA sequence file to read
     * @param p
     *
     * @throws FileNotFoundException if sequence file cannot be found
     * @throws IOException if sequence file cannot be read
     */
    public TrainingSet(String filename, Model m)
    throws FileNotFoundException, IOException
    {
        init();
        sequenceFileParse(filename, m);
    }



    /**
     * initialize instance variables
     *
     */
    private void init()
    {
        this.sequences = new LinkedList();
        this.lengths   = new LinkedList();
    }



    /**
     * print sequence info to requested stream
     *
     * @param sequenceName
     * @param p
     * @param out
     */
    public void headerPrint(String sequenceName, TrainSequenceParams p,
PrintStream out)
```

```
{
    out.print("Data: "  + sequenceName + "\n");
    out.print("Word: "  + p.length + "\n");
    out.print("Mism: "  + p.mismatchCount + "\n");
    out.print("Win: "   + p.window + "\n");
    out.print("Exp: "   + p.expr + "\n");
    out.print("Chann: " + p.channelCount + "\n");
    out.print("Limits: " + p.channelsString() +"\n\n");
    out.print("Statistics:\n\n");
}



/**
 * parse a file containing FASTA data into a training set
 *
 * @param filename
 * @param p
 *
 * @throws FileNotFoundException if filename cannot be found
 * @throws IOException if filename cannot be read
 */
public void sequenceFileParse(String filename, Model m)
throws FileNotFoundException, IOException
{
    BufferedReader br = null;
    String line       = null;

    int seqtotal = 0;
    Sequence sequence = new Sequence(filename);

    br = new BufferedReader(new FileReader(filename));
    while ((line = br.readLine()) != null)
    {
        if (line.trim().equals(""))
            continue;


        // get new seq's header data
        if (line.matches(">.*"))
        {
            if (sequence.length > 0)
            {
                this.lengths.add(new Long(sequence.length));
                this.sequences.add(sequence);
            }

            line.replaceAll(" ", "_");
            sequence = new Sequence(filename);
            sequence.data = line;
            seqtotal++;
        }
        else if (line.matches("\\^\\^.*"))
        {
            if (sequence.length > 0)
```

```
                {
                    this.lengths.add(new Long(sequence.length));
                    this.sequences.add(sequence);
                }
                sequence = new Sequence(filename);
            }
            else
            {
                sequence.sequence += line;
                sequence.length += line.length();
            }

        }

        if (sequence.data == null)
            sequence.data = filename;


        this.lengths.add(new Long(sequence.length));
        this.sequences.add(sequence);

        if (seqtotal == 0)
            seqtotal++;

        Sequence s = null;
        java.util.Iterator i = this.sequences.iterator();
        while (i.hasNext())
        {
            s = (Sequence) i.next();
            s.maxPosition = s.length - m.windowSmall - m.length + 1;
        }

        this.count  = seqtotal;
        this.input  = filename;
        this.length = StrictMath.round(Util.sum((Collection)
this.lengths));

    }
}
```

ProgressBar.java

```
package edu.dartmouth.bglab.crm.lwf.scan;

import java.text.DecimalFormat;
import java.text.NumberFormat;

/**
 * LWF helper class has methods for displaying a progress thermometer
 * as parsing progresses.
 *
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.6 $
 * @update:  $Date: 2005/04/12 12:55:21 $
 *
 */
public class ProgressBar
{
    /** */
    public long          progress;

    private long         seqLen;    // total length of sequence
    private long         win;       // size of window in sequence
    private long         inc;       // unit of progress (progress
proceeds in inc-sized increments)
    private StringBuffer bar;       // the thermometer
    private NumberFormat nf;        // thermometer progress formatter


    // width of thermometer
    private final int WIDTH = 70;



    /**
     * initialize the thermometer and display the sequence and model
being
     * used for parsing.
     *
     * @param seqName name of sequence to parse
     * @param length
     * @param win
     * @param modelName name of model to parse
     */
    public ProgressBar(String seqName, long length, long win, String
modelName)
```

```
    {
        this.bar = new StringBuffer();

        this.seqLen = length;
        this.win = win * 2;
        this.inc = (length - this.win) / this.WIDTH;
        this.progress = this.inc;
        this.bar.append(".");

        this.nf = NumberFormat.getInstance();
        if (this.nf instanceof DecimalFormat)
            ((DecimalFormat) this.nf).applyPattern("##.##");

        System.out.println("");
        System.out.println("        processing "+seqName+" using model
"+modelName+"...");
    }



    /**
     * update the progress thermometer
     *
     * print a thermometer that looks like this:
     *   4.25 [..........
]
     * the percentage and progress of the dots between the [ ] brackets
is
     * regularly updated by appending \r instead of \n to the end of
each
     * line
     *
     * @param i
     */
    public void update(long i)
    {
        this.bar.append(".");
        this.progress += this.inc;

        StringBuffer pad = new StringBuffer();
        int max = this.WIDTH - this.bar.length();
        for (int j = 0; j <= max; j++)
            pad.append(" ");

        System.out.print( "\r       ["+this.bar + pad +"]\r
"+this.nf.format(100 * i / ((double)this.seqLen - (double)this.win))
+"\r");
        //                 ^        ^                    ^   ^
        //                 ^        thermometer          ^   percent
complete so far
        //                 return to|start of line       return
to|start of line


    }
```

```
}
```

ScanSequenceParams.java

```
package edu.dartmouth.bglab.crm.lwf.scan;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.List;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.io.File;
import edu.dartmouth.bglab.crm.lwf.LwfException;
import edu.dartmouth.bglab.crm.lwf.Util;
import edu.dartmouth.bglab.crm.lwf.Model;



/**
 * LWF helper class; bit-bucket for runtime sequence scanning params.
 *
 * This code is based on Perl written by Dmitri Papatsenko. The source
 * paper is Nazina, A. and Papatsenko, D. (2003). Statistical
extraction
 * of Drosophila cis-regulatory modules using exhaustive assessment of
 * local word frequency, BMC Bioinformatics 4:65.
 *
 * @author:  $Author: zburke $
 * @version: $Revision: 1.11 $
 * @update:  $Date: 2006/02/03 06:56:29 $
 *
 */
public class ScanSequenceParams extends Model
{
    /** filename of sequence to scan */
    public String seqfname = null;

    /** output directory where parse files will be saved */
    public String scanoutdir = null;

    /** string containing list of model files */
//    public String fileList = null;

    /** list of +/- training model pairs */
    public List stsList = null;

    public long stsmaxcount = 0;

    public List mdlwins = null;

    public List models = null;
```

```
    public List modelNames = null;

    public List channels = null;

    /** whether to write GNU plot data for a sequence recognition
profile */
    public boolean plot = false;


    /**
     * run-time param, freely enterable regardless of model-data
     * range is 0-100; default is 10
     * passed as the standard-deviation to the normalDistribution
method
     */
    public long supression = 10;

    /**
     * threshold above which a window is classified positively.
     * Range: 0 - 1; 0 = stringent, 1 = relaxed
     */
    public double profileCutoff = 0.0;

    /**
     * model-based param
     * minimum width of a positively-classified span that will be
     * considered a CRM
     */
    public long peakWidth = 0;

    /**
     * model-based param
     * smoothing window
     */
    public long inputSmooth = 0;



    /**
     * default constructor initializes lists
     *
     */
    public ScanSequenceParams()
    {
        this.stsList = new ArrayList();

        this.mdlwins = new ArrayList();
        this.models = new ArrayList();
        this.modelNames = new ArrayList();
        this.channels = new ArrayList();

    }
```

```
    /**
     * collect runtime params from the user for model parsing
     *
     * @throws IOException
     */
    public void trainGet() throws IOException, LwfException
    {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String line = null;

        // sequence name
        System.out.print("\nEnter sequence name: ");
        String sequenceName = in.readLine().trim();

        // model names
        System.out.print("\nEnter +/- model pairs: ");
        String models = in.readLine().trim();

        // equalize channels
        System.out.print("\nEqualize channels (1-100) [10]: ");
        line = in.readLine().trim();
        int supression = line.equals("") ? 10 : new
Integer(line).intValue();

        trainSet(sequenceName, models, supression);

    }



    /**
     * save runtime
     * @param seqfname
     * @param fileList
     * @param supression
     * @throws LwfException
     */
    public void trainSet(String seqfname, String fileList, int
supression)
        throws LwfException
    {
        // sequence name
        this.seqfname = seqfname;

        // make sure we got pairs of models
        String[] stsList = fileList.split("\\s");
        if (stsList.length % 2 != 0)
            throw new LwfException("you must specify positive and
negative training model pairs");

        // create the output directory
        outDirCreate();
```

```
        for (int i = 0; i < stsList.length; i += 2)
        {
            Hashtable hash = new Hashtable();
            hash.put("pos", stsList[i]);
            hash.put("neg", stsList[i + 1]);

            this.stsList.add(hash);
        }

        // equalize channels
        this.supression = supression;

    }



    /**
     * collect runtime params for scanning
     *
     * @throws IOException
     */
    public void scanGet() throws IOException, LwfException
    {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String line = null;

        // profile cutoff
        double cutoffSuggest = profileCutoffSuggest();
        System.out.print("\nSelect profile cutoff (0-1) [" +
cutoffSuggest + "]: ");
        line = in.readLine().trim();
        double cutoff = line.equals("") ? cutoffSuggest : new
Double(line).doubleValue();

        // peak width cutoff (min CRM width)
        // suggestion is 1/2 size of biggest channel
        long peakSuggest = peakWidthSuggest();
        System.out.print("\nSelect peak width cutoff [" + peakSuggest +
"]: ");
        line = in.readLine().trim();
        long width = line.equals("") ? peakSuggest : new
Integer(line).intValue();

        // how much to smooth data
        // suggestion is 1/2 size of biggest channel
        long smoothSuggest = smoothSuggest();
        System.out.print("\nSelect smoothing window [" + smoothSuggest
+ "]: ");
        line = in.readLine().trim();
        long smooth = line.equals("") ? smoothSuggest : new
Integer(line).intValue();

        // whether to write GNUPlot files
```

```
        System.out.print("\nSelect plot option (y/n) [" + plotSuggest()
+ "]: ");
        line = in.readLine().trim();
        boolean gnuplot = line.equals("y") ? true : false;

        scanSet(cutoff, width, smooth, gnuplot);

    }



    public double profileCutoffSuggest()
    {
        return 0.3;
    }



    public long peakWidthSuggest()
    {
        return Util.max(this.mdlwins) / 2;
    }



    public long smoothSuggest()
    {
        return peakWidthSuggest();
    }



    public boolean plotSuggest()
    {
        return false;
    }



    /**
     * set run-time scanning parameters and stash a README file in the
     * output directory that records them.
     *
     * @param profile
     * @param peak
     * @param smooth
     * @param gnuplot
     *
     * @throws LwfException in case the README can't be written
     */
    public void scanSet(double profile, long peak, long smooth, boolean
gnuplot)
    throws LwfException
    {
        // profile cutoff
```

```
        this.profileCutoff = profile;

        // peak width cutoff (max channel width???)
        this.peakWidth = peak;

        // how much to smooth data
        this.inputSmooth = smooth;

        // whether to write GNUPlot files
        this.plot = gnuplot;

        // put a readme in the outdir that specifies the models
        outDirReadme();

    }



    /**
     * Create a uniquely named output directory in the pattern
     *     <sequence-file>_scan_<i>
     * e.g. eve.fa_scan_015
     *
     * @throws LwfException if directory creation fails
     */
    private void outDirCreate()
    throws LwfException
    {
        int i = 1;
        String outDir = this.seqfname + "_scan";
        String dir = null;

        boolean success = false;
        while (! success)
        {
            dir = outDir + "_" + (i > 99 ? "" + i : (i > 9 ? "0" +i :
"00" + i));
            success = (new File(dir.toString())).mkdir();
            if (! success)
                i++;
        }

        this.scanoutdir = dir;
    }



    /**
     * Create a simple README file that specifies some of the params
     * that were used during this scanning session.
     *
     */
    private void outDirReadme()
    throws LwfException
    {
```

```
        String filename = this.scanoutdir + "/README";
        try
        {
            PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(filename)));

            // threshold above which a window is classified positively
            // 0 = stringent, 1 = relaxed */
            pw.println("profile-cutoff: " + this.profileCutoff);

            // minimum width of a positively-classified span to be
            // considered a CRM
            pw.println("peak-width: " + this.peakWidth);

            // ?
            // suggestion is 1/2 width of biggest channel
            pw.println("inputSmooth: " + this.inputSmooth);

            // ?
            // default is 10
            pw.println("supression: " + this.supression);

            // training models
            Iterator i = this.stsList.iterator();
            Hashtable h = null;
            pw.println("Training models used for this scan:");
            while (i.hasNext())
            {
                h = (Hashtable) i.next();
                pw.println("\t" + h.get("pos") + " " + h.get("neg"));
            }


            pw.flush();
            pw.close();
        }
        catch (IOException e)
        {
            throw new LwfException(e);
        }

    }
}
```

build.xml

```
<?xml version="1.0"?>

<!--
build options for LWF, a Cis-Regulatory Module finder

This code is based on Perl written by Dmitri Papatsenko. The source
paper is Nazina, A. and Papatsenko, D. (2003). Statistical extraction
of Drosophila cis-regulatory modules using exhaustive assessment of
local word frequency, BMC Bioinformatics 4:65.


@author:  $Author: zburke $
@version: $Revision: 1.1 $
@update:  $Date: 2006/02/11 04:16:21 $

-->


<project name="LWF" default="build"  >
    <!-- name and location of the properties file. -->
    <property file="build.properties" />



    <target name="init" description="make sure the output directory
exists">
        <mkdir dir="${java.classes}" />
    </target>



    <target name="build" depends="init" description="compile modified
sources and create a jar archive">
        <javac destdir="${java.classes}" classpath="${build.classpath}"
debug="false">
            <src path="${java.src}"/>
            <classpath refid="build.classpath"/>
        </javac>

        <antcall target="build-jar" />

    </target>



    <target name="debug" depends="init" description="compile modified
sources and create a jar archive">
```

```
        <javac destdir="${java.classes}" classpath="${build.classpath}"
debug="true">
            <src path="${java.src}"/>
            <classpath refid="build.classpath"/>
        </javac>

        <antcall target="build-jar" />

    </target>



    <target name="clean" description="remove classes directory to force
full recompile">
        <delete dir="${java.classes}" quiet="true" />
    </target>



    <target name="build-jar" depends="init" description="archive
compiled classes in a jar file">
        <jar destfile="${lwf.jar}">
            <fileset dir="${java.classes}" excludes="*jar" />
        </jar>
    </target>


    <patternset id="nosources">
        <exclude name="**/src/*" />   <!-- Exclude standard source
directory -->
        <exclude name="**/*.java" />  <!-- Exclude stray .java files --
>
    </patternset>

    <!-- This task will print out Ant properties and values for this
build  -->
    <!-- script. This task can be helpful for troubleshooting your
build    -->
    <!-- configuration.
-->
    <target name="env" description="Report on the build properties.">
        <echo message="java version:    ${ant.java.version} "/>
        <echo message="java vendor:     ${java.vendor} " />
        <echo message="java source      ${java.src} " />
        <echo message="java classes     ${java.classes} " />
        <echo message="user home        ${user.home}" />
    </target>

</project>
```

build.properties

```
java.classes = bin
java.src = src
lwf.jar = ./bin/lwf.jar
lwf.train = edu.dartmouth.bglab.crm.lwf.Train
lwf.scan = edu.dartmouth.bglab.crm.lwf.Scan
```

props_train.pl

```
#!/usr/bin/perl

# --
# -- 2005-04-25, zak.burke@dartmouth.edu
# -- generate training parameter files for a variety of training
# -- sequence files with variable runtime parameters.
# --
# --

use strict;

main();

sub main
{
    # output directory
    die("usage: $0 <input-dir> <output-dir>\n\n")
        unless $ARGV[0] && $ARGV[1] && -d $ARGV[0];

    my $dir_in  = $ARGV[0];
    my $dir_out = $ARGV[1];

    if (! -d $dir_out) {
        mkdir $dir_out
            or die("could not mkdir $dir_out: $!\n");
    }

    opendir DIR, $dir_in;
    my @files = grep { /^train.*\.fa$/ } readdir DIR;
    closedir DIR;

    push @files, "neg_dm.fa";

    my $i       = 0;
    my $params = params();

    print STDERR "FILES: @files\n";
    print STDERR "Pcount: $#{$params}\n";

    for my $param (@{ $params }) {
        for my $file (@files) {
            my $name = "$dir_out/train_".($i > 999 ?
                $i : ($i > 99 ?
                "0$i" : ($i > 9 ?
                "00$i" : "000$i") )) . ".properties";
            open FILE, ">$name"
                or die("couldn't open $name: $!");
            print FILE <<EOT;
```

```
word-length $param->{'word_length'}
mismatch-count $param->{'mismatch'}
window-length $param->{'window'}
channel-count $param->{'channels'}
files $file
EOT
            close FILE;
            $i++;
        }
    }
}


sub params
{
    # length of words in a channel
    my @word_length = (3, 4, 5, 6);

    # number of channels to create
    my @channels = (9, 12, 15, 18, 21, 24);

    # size of scanning window
    my @window = (50, 100, 200, 400, 600, 800, 1000);

    # mismatches allowed in a word
#    my @mismatch = (0, 1, 2, 3, 4);
    my @mismatch = (0);

    my @list;
    for my $wl (@word_length) {
    for my $ch (@channels) {
    for my $wi (@window) {
    for my $mi (@mismatch) {
        push @list, {
            'word_length' => $wl,
            'channels' => $ch,
            'window' => $wi,
            'mismatch' => $mi,
            };
    } } } }


    return \@list;

}
```

props_scan.pl

```
#!/usr/bin/perl

# --
# -- 2005-05-29
# -- properties file generator for distinct or composite scanning with
# -- default parameters. see usage() for details.
# --
# -- if ! $flip do distinct; if $flip do composite. for distinct
scanning,
# -- a TS named train_<gene>.fa and its corresponding model named
# -- <a>_<b>_<c>_<d>_train_<gene>.fa.sts are based on the CRMs from
# -- <gene>. conversely, for composite scanning (i.e. when $flip == 1),
# -- that TS and model are named for the gene that was LEFT OUT,
# -- i.e. it should be used to scan <gene> because it is based on CRMs
# -- from every gene EXCEPT <gene>.
# --
# -- 2005-05-05, zak.burke@dartmouth.edu
# -- given a list of training (*.sts) files and a list of genes,
# -- for each gene and each set of distinct training parameters,
# -- create a scanning properties file with default scanning
# -- parameters.
# --
# --


use strict;
use Getopt::Long qw(GetOptions);

main();

sub main
{
    my ($model_dir, $dir_out, $flip, $group) = ("", "", 0, 0);

    my $result = GetOptions(
        "model_dir=s" => \$model_dir, # directory with model (.sts)
files
        "dir_out=s"   => \$dir_out,   # where to put output files
        "flip"        => \$flip,      # 0=TS has gene's CRMs; 1=TS has
lacks gene's CRMs
        "group=i"     => \$group,     # group-size for genes sharing TS
    );

    usage() unless $model_dir && $dir_out;

    if (! -d $dir_out) {
        mkdir $dir_out
            or die("could not mkdir $dir_out: $!\n");
    }


    my $gene_dir = "./data/gene";
```

```
    # all genes
    my $genes = genes($gene_dir);

    # all gene-gropus
    my $groups = group($genes, $group);

    # all trainsets, grouped by identical parameters
    my $ts = ts_by_param($model_dir);


    my $count = 1;

    for my $gene (@{ $genes }) {

        print "READING $gene...\n";
        for my $ts_key (sort keys %{ $ts } ) {
            my %param = (
                "gene"    => $gene,
                "genes"   => $genes,
                "dir_out" => $dir_out,
                "ts"      => ${ $ts }{$ts_key},
                "count"   => \$count,
                "flip"    => $flip,
                "groups"  => ${ $groups }{$gene},
            );

            props_default(%param);

        }
    }

}



# --

# --
# -- props_default
# --
sub props_default
{
    my %input = @_;

#    my ($gene, $genes, $dir_out, $ts, $count, $flip) = @_;

    # properties file name
    my $name = file_name_inc($input{'count'}, $input{'dir_out'});

    open FILE, ">$name"
        or die("could not open $name: $!\n");
    print FILE <<EOT;
sequence $input{'gene'}
equalize 10
```

```
scan defaults
EOT

    # get training sets which aren't based on this gene
    my $seq_list = ts_for_gene(%input);
    print FILE "train ".(join " ", @{ $seq_list }) ."\n";
    close FILE;

}



# --

# --
# -- file_name_inc
# -- get name of next properties file and increment the
# -- file_name counter
sub file_name_inc
{
    my ($count, $dir_out) = @_;

    my $name = "$dir_out/scan_".($$count > 999 ?
        $$count : ($$count > 99 ?
        "0$$count" : ($$count > 9
        ? "00$$count" : "000$$count"))) . ".properties";

    $$count++;

    return $name;

}



# --

# --
# -- ts_in_use
# -- i think all this does it throw out the neg_dm training sets
sub ts_in_use
{
    my ($pos, $genes) = @_;

    $pos =~ /.*train_([^\.]+).*/;
    my $ts = $1;

    return 1 if grep /$ts/, @{ $genes };
    return 0;
}



# --
```

```
# --
# -- ts_for_gene
# -- retrieve training sets to use for scanning this
# -- sequence, i.e. all TSes except those created from
# -- this gene group's CRMs
sub ts_for_gene
{
    my %input = @_;

    my @sequences;

    POS: for my $pos (@{ $input{'ts'} }) {

        # skip TS created with this gene
        if ($input{'flip'}) {
            next if $pos !~ /$input{'gene'}/;
        } else {

            for (@{ $input{'groups'} }) {

                next POS if $pos =~ /$_/
            }

#            print "\t$pos ($input{'gene'}): @{ $input{'groups'} }\n";

        }

        # skip training sets for unscanned genes
        next unless ts_in_use($pos, $input{'genes'});

        # get name of matching negative training set
        my $neg = $pos;
        $neg =~ s/train_[^\.]+/neg_dm/;

        push @sequences, "$pos $neg";
    }

    return \@sequences;

}



# --

# --
# -- genes
# -- get names of genes in $dir by lopping of the .fa extension.
# -- return an array ref.
sub genes
{
    my ($dir) = @_;

    opendir DIR, $dir;
    my @genes = grep { /fa$/ } readdir DIR;
```

```
    closedir DIR;

    return \@genes;
}



# --

# --
# -- ts_by_param
# -- return a hash of trainsets, grouped by common training params
sub ts_by_param
{
    my ($model_dir) = @_;


    # all trainsets, grouped by idential parameters
    opendir DIR, $model_dir;
    my @ts_files = grep { /.*train_.*\.fa\.sts$/ } readdir DIR;
    closedir DIR;

    my %list;
    for (@ts_files) {

        $_ =~ /([0-9]+_[0-9]+_[0-9]+_[0-9]+)_.*/;

        push @{ $list{$1} }, $_;

    }

    return \%list;
}



# --

# --
# -- params
# -- return a single list with hashes of all possible scan-time
# -- parameter combinations
sub params
{
    # standard deviation of normal distribution method
    my @equalize = (10, 30, 50);

    # threshold for recognizing a CRM 0 == stringent; 1 == lax
    my @profile_cutoff = (0.1, 0.3, 0.5);

    # peak width: size of smallest CRM window
    my @peak_width = (25, 50, 100, 150, 200);

    # how much to smooth data
    my @smooth = (25, 50, 100, 150, 200);
```

```
    my @list;

    for my $eq (@equalize) {
    for my $pr (@profile_cutoff) {
    for my $pw (@peak_width) {
    for my $sm (@smooth) {

        push @list, {'equalize' => $eq, 'cutoff' => $pr, 'peak_width'
=> $pw, 'smooth' => $sm};

    } } } }

    return \@list;

}



# --

# --
# -- group
# -- group the elements in @$genes into $n groups; return a
# -- hash with each gene as a key and its group members in an
# -- array-ref.
# -- NOTE: $genes contains <gene>.fa names; the returned hash
# --     strips the .fa.
sub group
{
    my ($genes, $n) = @_;

    # if $n is unspecified, each gene gets its own group
    $n = scalar @{ $genes } unless $n;

    # create the groups ...
    my (@list, $i);
    for (sort @{ $genes }) {

        # grab the gene name (eve) from the filename (eve.fa)
        (my $name = $_) =~ /(.*)\.fa$/;
        push @{ $list[$i++] }, $name;

        $i = 0 if $i == $n;
    }

    # ... and for simplicity's sake, return each group
    # as an array-ref on a gene-keyed hash, e.g.
    # $list{abd-A.fa} => [abd-A.fa eve.fa hb.fa]
    # $list{eve.fa}   => [abd-A.fa eve.fa hb.fa]
    # $list{hb.fa}    => [abd-A.fa eve.fa hb.fa]
    # $list{btd.fa}   => [btd.fa ftz.fa kni.fa]
    # ...
    my %hash;
    for my $group (@list) {
```

```
        for my $gene (@{ $group }) {
            $hash{$gene} = $group;
        }
    }

    return \%hash;
}



# --

# --
# -- usage
# -- explain how this script works, then QUIT
sub usage
{
    print <<EOT;

usage
    $0 --model_dir=<model-dir> --output_dir=<output-dir> [--flip] [--
group=i]

description
    generates scanning parameters for composite or distinct training
sets

params
    model_dir: location of training models (sts files)
    output_dir: where to put new parameter files
    flip: ifdef, CRMs from all genes EXCEPT that in the filename were
        used to generate this file
    group: ifdef, group genes into i groups and use the same set of
        training files to scan all the genes in each group.

EOT
    exit 0;
}
```

extract.pl

```
#!/usr/bin/perl

# --
# -- 2005-06-28, zak.burke@dartmouth.edu
# -- based on /lwf/analysis/hm_parse_best.pl, extract certain tuples
# -- from the output file created by hm_parse.pl.
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
}


use strict;

use lib $ENV{'BG_PERL_LIB'};
use scan_parse;
use extract;



main();


sub main
{
    usage() unless ($ARGV[0]);

    my @conditions = (
        {'word' => 3, 'window' => 200, 'channel' => 8},
        {'word' => 3, 'window' => 200, 'channel' => 13},
        {'word' => 3, 'window' => 200, 'channel' => 18},
        );

    # get lines matching conditions
    my $lines = extract::extract($ARGV[0], \@conditions);

    # extract LWF output-directory names and print 'em
    my @best;
    for my $line (@{ $lines }) {
        $line->{'file'} =~ /scans\/([^\.]+)\.fa_scan.*/;
        push @best, $line->{'file'};
    }
    print join "\n", @best;
}



# --

# --
```

```
# -- usage
# -- show how to call this program, then exit
sub usage
{
    print <<EOT;
usage
    $0 <hm_parse.pl-output-file>

summary
    prints each gene's top-scoring directory, according to harmonic
mean


EOT

    exit 1;
}
```

hm.pl

```
#!/usr/local/bin/perl

# --
# -- hm.pl (harmonic mean finder)
# --
# -- Given a DNA sequence which contains CRMs at known locations, and
# -- given a list of putative CRM locations in this sequence, run some
# -- statistical tests to figure out how well the pCRMs match the
# -- known CRMs.
# --
# --
# -- recall    (r): (true-positives) / (false-negatives + true
positives)
# --
# -- precision (p): (true-positives) / (false positives + true
positives)
# --
# -- harmonic mean: (2pr) / (p + r)
# --     the harmonic mean is useful for calculating an average of
rates,
# --     in this case the rates of false positives (precision) and
false
# --     negatives (recall).
# --
# -- phi-score: tp/(fp + fn)
# --     the phi-score has become a standard metric for describing the
# --     performance of motif finding programs. it was first described
in
# --     Pevzner, P.A. and Sze, S.H. (2000) Combinatorial approaches to
finding
# --     a subtle signals in DNA sequences. Proceedings of the
International
# --     Conference on Intelligent Systems in Molecular Biology 8:269-
78.
# --
# --
# -- zak.burke@dartmouth.edu
# --
# -- 2005-03-30, zak.burke@dartmouth.edu
# -- replaced hard-coded CRM data for even-skipped with a DB-based
method
# -- that pulls data for any gene in the DB
# --
# -- 2005-04-04, zak.burke@dartmouth.edu
# --
# -- $Author: zburke $
# -- $Revision: 1.5 $
# -- $Date: 2005/07/04 19:32:01 $
# --
```

```
use strict;
use DBI;

use lib "/home/zburke/bg/workspace/perl-lib";
use nazina;
use nazina_db;
use scan_parse;



use constant TRUE  => 1;
use constant FALSE => 0;

main();


sub main
{

    # make sure we were called correctly
    usage() unless $ARGV[0];

    # directory where scan files live
    my $dir_name = $ARGV[0];

    # how to format output
    my $parser_name = ($ARGV[1] ? $ARGV[1] : undef);

    # DAO for gene details
    my $dao = nazina_db->new();

    # directories containing scans
    opendir SCANS, $dir_name;
    my @scans = grep { /.*\.fa_scan_[0-9]+$/ } readdir SCANS;
    closedir SCANS;


    # find CRM data in each scan
    my $stats = stats($dir_name, \@scans, $dao);

    # print scans sorted by harmonic mean, desc
    my $line_parser = scan_parse::parser_by_name($parser_name);
    print "" . &{ $line_parser }("head") . "\n";
    for (sort { $b->{'hm'} <=> $a->{'hm'} } @{ $stats }) {

        print &{ $line_parser }("write", $_);

    }

}



# --
```

```
# --
# -- stats
# -- extract pCRM details from files
sub stats
{
    my ($dir_name, $scans, $dao) = @_;

    my @stats;

    # list of all genes, keyed by abbrevation
    my $genes = $dao->genes_by_abbv();

    # find CRM data in each scan
    for my $scan (@{ $scans }) {
        # gene abbv is the first element of the filename
        my @list = split '\.', $scan;
        my $gene = ${ $genes }{ $list[0] };

        #
        my $item = scan_file_parse("$dir_name/$scan", $dao->gene($gene-
>{'gene_id'}));
        push @stats, $item;
    }

    return \@stats;
}



# --

# --
# -- scan_file_parse
# -- compare a scan's putative CRMs to the true CRMs for that source
sub scan_file_parse
{
    my ($dir_name, $gene) = @_;

    my $source = $gene->enhancers();
    my $test   = scan_parse::pcrm_file_read($dir_name);

    # CRM file or pCRM file was empty (probably the latter,
    # but it never hurts to check).
    return unless ($source && $test);

    if (0 == scalar @{ $source } ) {
        print STDERR "WARNING: NO ENHANCER ELEMENTS FOR SOURCE $gene-
>{abbv}\n";
    }
    if (0 == scalar @{ $test }) {
        print STDERR "WARNING: NO ENHANCER ELEMENTS in dir
$dir_name\n";
    }
```

```
    # get scan-params from this directory's README file
    my $params = readme_file_read($dir_name);

    # there are no false positives beyond the last pCRM substring and
    # no false negatives beyond the last CRM substring. by finding the
    # max position between the CRM and pCRM datasets, we can figure
    # out where to stop counting.
    my $max = list_max($test, $source);

    # counters for true-positive, false-positive, false-negative
    my ($tp, $fp, $fn) = (0, 0, 0);

    # count tp, fp and fn for this scan
    for (my $i = 0; $i < $max; $i++) {

        $tp++ if (in_crm($source, $i) && in_crm($test, $i));
        $fp++ if (! in_crm($source, $i) && in_crm($test, $i));
        $fn++ if (in_crm($source, $i) && ! in_crm($test, $i));

    }

    my $recall = recall($tp, $fn);
    my $precision = precision($tp, $fp);

    # return results for this scan
    return {
        'file'      => $dir_name,
        'tp'        => $tp,
        'fp'        => $fp,
        'fn'        => $fn,
        'recall'    => $recall,
        'precision' => $precision,
        'hm'        => harmonic_mean($recall, $precision),
        'phi_score' => phi_score($tp, $fp, $fn),
        's_to_n'    => $gene->s2n(),
        %{ $params },
        };
}



# --

# --
# -- readme_file_read
# -- extract run-time params from a scan-directory's README file.
sub readme_file_read
{
    my ($dir_name) = @_;
    my %list;

    eval {
    open FILE, "<$dir_name/README"; # or die("can't open
$dir_name/README: $!\n");
    while (<FILE>) {
```

```
        # trim trailing whitespace, including damn windows line breaks
        chomp;
        s/\s+$//g;

        # training params
        if (! $list{'word'} && /neg_dm\.fa\.sts$/) {
            s/^\s+//;
            my @words = split /\s/;
            ($list{'word'}, $list{'window'}, $list{'channel'}) =
($words[0] =~ /^([0-9]+)_0_([0-9]+)_([0-9]+)_train_[^\.]+\.fa.*/);

            next;
        }

        # scanning params
        my @line = split "\\s";
        if ($line[0] =~ /profile-cutoff/) { $list{'profile-cutoff'} =
$line[1]; next; }
        if ($line[0] =~ /peak-width/)     { $list{'peak-width'}     =
$line[1]; next; }
        if ($line[0] =~ /inputSmooth/)    { $list{'input-smooth'}   =
$line[1]; next; }
        if ($line[0] =~ /supression/)     { $list{'supression'}     =
$line[1]; next; }

    }
    close FILE;
    };
    if ($@) {
        print STDERR "couldn't find $dir_name/README: $@";
    }
    return \%list;
}



# --

# --
# -- in_crm
# -- return TRUE if $i is within a defined CRM region in @$list;
# -- return FALSE otherwise.
sub in_crm
{
    my ($list, $i) = @_;


    for (@{ $list }) {

        return TRUE if ($i >= $_->{'beg'} && $i <= $_->{'end'})

    }

    return FALSE;
}
```

```
# --

# --
# -- crm_display
# -- show CRMs in the given list
sub crm_display
{
    my ($list) = @_;

    my $string = "\t" . (join "\n\t", map { "$_->{beg} - $_->{end}" }
@{ $list } );

    return $string;
}



# --

# --
# -- list_max
# -- given two ascending-sorted lists where each element has a
# -- hash-ref with an "end" key, return the max "end" value.
sub list_max
{
    my ($list_a, $list_b) = @_;

    # capture index of last CRM. in case no CRMs are specified
    # count will be -1 and we don't want to dereference that i
    # array position
    my $i_a = $#{ $list_a };
    my $i_b = $#{ $list_b };

    my $end_a = $i_a > -1 ? ${ $list_a }[$i_a]->{'end'} : 0;
    my $end_b = $i_b > -1 ? ${ $list_b }[$i_b]->{'end'} : 0;

    return ($end_a > $end_b ? $end_a : $end_b);

}



# --

# --
# -- recall
# -- given the count of true-positives and false-negatives, calculate
# -- and return the recall rate.
# --
# -- recall is calculated as follows:
# --
# --     R == (true-positives) / (false-negatives + true positives)
```

```
# --
sub recall
{
    my ($tp, $fn) = @_;

    my $sum = $fn + $tp;

    return $sum ? ($tp / $sum) : 0;
}



# --

# --
# -- precision
# -- given the count of true-positives and false-positives, calculate
# -- and return the precision rate.
# --
# -- precision is calculated as follows:
# --
# --     P == (true-positives) / (false positives + true positives)
# --
sub precision
{
    my ($tp, $fp) = @_;

    return ($tp / ($fp + $tp));
}



# --

# --
# -- harmonic_mean
# -- return the harmonic mean of the 2 given rates. The harmonic mean
is useful
# -- for calculating an average of rates.
# --
# -- the harmonic mean is calculated as follows:
# --
# --     H == n / ( (1/a[1]) + (1/a[2]) + ... + (1/a[n]) ).
# --
# -- for a simple two-variable equation, this reduces to:
# --
# --     H == (2ab) / (a + b)
# --
sub harmonic_mean($$)
{
    my ($recall, $precision) = @_;

    return 0 if ($precision + $recall) == 0;

    return ((2 * $precision * $recall) / ($precision + $recall));
```

```
}



# --

# --
# -- phi_score
# -- calculate a phi_score: true-positives / [false-pos + false-neg]
sub phi_score
{
    my ($tp, $fp, $fn) = @_;

    return ($tp / ($fp + $fn));
}



# --

# --
# -- usage
# -- show how to call this script, then exit
sub usage
{
    print <<EOT;

summary
    $0 <directory-name> [parser-name]

    scans a directory for pCRM files and generates statistical
    profiles for the different CRM-finder parameters

usage

    $0 <directory-name> [parser-name]

    where directory-name contains the results of scanning,
    i.e. a bunch of *.sts files, and each *.sts directory
    contains a *.xtr.dat file containing the substring
    indices of the putative CRMs recognized during this scan.

    optional: parser-name is the name of a CRM-scan parser
    that specifies an output format.
    default: hm phi_score recall precision file
    format2: hm phi_score recall precision s_to_n file
    format3: hm phi_score recall precision s_to_n profile-cutoff
        peak-width input-smooth supression file

description

    given a list of locations of CRMs in a sequence, scan other lists
    of locations which may contain the CRM's coordinates (true
positives),
```

```
    incorrect locations (false positives) or which may omit the CRM's
    coordinates (false negatives).

    recall    (r): (true-positives) / (false-negatives + true
positives)
    precision (p): (true-positives) / (false positives + true
positives)
    harmonic mean: (2pr) / (p + r)
    phi-score    :
    s_to_n       : signal-to-noise ratio (crm-length/sequence-length)
of sequence



EOT
    exit 0;
}
```

hm_parse.pl

```
#!/usr/local/bin/perl

# --
# -- hm_parse.pl
# -- parse the output of hm.pl (which provides details about how a
# -- collection of scan parameters scores) to extract details about
# -- how the individual scan parameters word-length, window-length
# -- and channel count score.
# --
# -- the output is oriented toward cutting and pasting into MSExcel
# -- for a stock plot, which you can trick into doing bar-whisker
# -- plots by pasting the median, q1, min, max and q3 values in the
# -- fields normally used for volume, open, high, low and close.
# --
# -- zak.burke@dartmouth.edu
# --
# -- 2005-03-30, zak.burke@dartmouth.edu
# -- added group-by options to group on gene-name and training-set-id
# -- in addition to grouping scores on the field params word-length,
# -- window-length and channel-count.
# --
# -- 2005-04-13, zak.burke@dartmouth.edu
# -- added box_plot so we don't have to bother with MSExcel to
# -- visualize the data anymore. hooray, although I really should
# -- learn Matlab and R anyway.
# --
# -- TODO:
# -- remove lines corresponding to empty dat files, i.e. where
# -- no data whatsoever is available.
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.5 $
# -- $Date: 2005/07/04 19:30:59 $
# --

use Statistics::Descriptive;
use GD::Graph::boxplot;
use lib "/home/zburke/bg/workspace/perl-lib";
use scan_parse;



main();

sub main
{
    usage() unless ($ARGV[0] && $ARGV[1]);

    my $field = $ARGV[0];
    my $file  = $ARGV[1];
```

```
    print "source: $file\n\n";

    my (@lines, $stats);

    # parser for fields embedded in filename
    my $field_parser = scan_parse::field_parser($ARGV[2]);

    # parse lines into records; grab first line to determine format
    open FILE, $file or die("couldn't open $file: $!");
    my $line = <FILE>;
    my $line_parser  = scan_parse::parser($line);
    while (<FILE>) {
        chomp;
        # skip header lines
        next unless /^[0-9]/;
        push @lines,  scan_parse::field_parse($_, $line_parser,
$field_parser);
    }
    close FILE;

    # field headers
    my (@stat_fields) = qw(statistic median q1 min max q3);

    # calculate stats for each of these record fields
    my (@scores)      = qw(hm phi_score recall precision s_to_n);

    # correlate groups the data by field value
    # stats builds statistical profiles of that data
    for my $score (@scores) {

        # only score fields parsed from the header
        if (! grep /$score/,  (split "\t", &{ $line_parser }("head")))
{
            print STDERR "$score is not a valid field\n";
            print STDERR "fields include ".(&{ $line_parser
}("head"))."\n";
            next;
        }

        $stats = stats(correlate($score, $field, \@lines), $score,
$field);
        print "\n\n$score\n";
        for (@stat_fields) {

            print "$_\t". (join "\t", @{ ${ $stats }{$_} }) ."\n";

        }

        my $s = Statistics::Descriptive::Full->new();
        $s->add_data(@{ ${ $stats }{'median'} });

        print "median\t" . $s->median(). "\n";
        print "stddev\t" . $s->standard_deviation() ."\n";
    }
```

```
}



# --

# --
# -- stats
# -- given a list of data buckets, generate statistical profiles
# -- for each bucket.
sub stats
{
    my ($list, $score, $field_name) = @_;

    my %stats;
    my ($value_list, $field_list, $max) = ([],[],0);

    # get key-sorter subroutine-ref
    my $sort_sub = stats_sort_set($list);

    # calculate stats for each distinct field-value,
    # e.g. if field is word-length, keys will be 3, 4, 5
    for my $field (sort $sort_sub keys %{ $list }) {

        my $s = Statistics::Descriptive::Full->new();

        # remove 0s from value list because they'll throw
        # off some calculations
        my $values = zero_zap(${ $list }{ $field });

        $s->add_data(@{ $values });

        my ($q1, $junk1) = $s->percentile(25);
        my ($q3, $junk2) = $s->percentile(75);

        push @{ $stats{'statistic'} }, "field-$field";
        push @{ $stats{'median'} }, $s->median;
        push @{ $stats{'q1'} }, $q1;
        push @{ $stats{'min'} }, $s->min;
        push @{ $stats{'max'} }, $s->max;
        push @{ $stats{'q3'} }, $q3;

        # graph info
        push @{$field_list}, $field;
        push @{$value_list}, $values;
        my $list_max = list_max($values);
        $max = $list_max if ($list_max > $max);

    }

    # generate box plots
    box_plot($field_list, $value_list, $max, $score, \%stats,
$field_name);

    return \%stats;
```

```
}



# --

# --
# -- stats_sort_set
# -- given a hash-ref, determine if the keys are alphabetic or
# -- numeric and return a subroutine-ref to an appropriate sort
# -- function
sub stats_sort_set
{
    my ($list) = @_;

    my @temp = keys %{ $list };
    if ($temp[0] !~ /[0-9]/) {
        return \&sort_alpha;
    }

    return \&sort_num;

}

sub sort_alpha
{
    $a cmp $b;
}
sub sort_num
{
    $a <=> $b;
}


# --

# --
# -- box_plot
# -- output boxplots of statistical data as PNG images
sub box_plot
{
    my ($field_list, $value_list, $max, $score, $stats, $field_name) =
@_;

    my $s = Statistics::Descriptive::Full->new();

    for (@{ $value_list }) {
        $s->add_data(@{ $_ });
    }

    #$s->add_data(@{ ${ $stats }{'median'} });

    my $my_graph = GD::Graph::boxplot->new(600,300);
    $my_graph->set(
```

```
        title           => "$field_name - $score",
        x_label         => "median: ".$s->median()."; stddev: ".$s-
>standard_deviation(),
        do_stats        => 1,
        box_spacing     => 35,
        symbolc         => 'black',
        y_min_value     => 0,
        y_max_value     => 1,
        warnings        => 1,
        long_ticks      => 1,
        x_ticks         => 0,
    );
        #y_max_value    => (sprintf("%.1f", $max + .1)),

    # Output the graph
    my @data = ($field_list, $value_list);
    my $gd = $my_graph->plot(\@data);
    open(IMG, ">$field_name.$score.png") or die $!;
    binmode IMG;
    print IMG $gd->png;


}



# --

# --
# -- zero_zap
# -- return a copy of @$list with entries == 0 removed. the original
list
# -- is not affected.
sub zero_zap
{
    my ($list) = @_;

    my @values;
    for (@{ $list }) {
        next if ! $_;
        push @values, $_;
    }
    return \@values;
}



# --

# --
# -- correlate
# -- group scores for the given metric into bins for each
# -- distinct value of the field. for example, say the
# -- $metric is 'phi' and $field is 'word-length'. the
# -- returned hash will have keys 3, 4 and 5 (the range
```

```
# -- of possible values for the word-length field) and
# -- the values of the hash will be array-refs of
# -- phi-scores. $hash{3}->[] will contain scores for all
# -- the records with word-length 3; $hash{4}->[] will
# -- contain scores for records with word-length 4, etc.
# --
# -- params
# --     metric - the score to calculate
# --     field - the field for which to calculate the score
# --     list - list of records
# --
# -- return
# --    hash-ref of metric values, keyed by field value
sub correlate
{
    my ($metric, $field, $list) = @_;

    # hash ref with distinct field values as keys and
    # array-refs as values
    my $fields = fields_from_list($field, $list);

    # sort the metric-value for each record into the correct bin
    # for the given field.
    for ( @{ $list } ) {

        push @{ ${ $fields }{ $_->{ $field } } }, $_->{$metric};

    }

    return $fields;

}



# --

# --
# -- fields_from_list
# -- extract distinct values for $field from @$list and return a hash-
ref
# -- with those values as keys. e.g. get distinct window-lengths,
# -- channel-counts or word-lengths.
sub fields_from_list
{
    my ($field, $list) = @_;

    my (%fields);

    # find distinct fields
    for (@{ $list }) {

        $fields{ $_->{ $field } } = [];

    }
```

```
    return \%fields;
}


#
# list_max
# return the biggest item on a list. the list is sorted
# in the process.
sub list_max
{
    my ($list) = @_;

    my @sorted = sort {$b <=> $a} @{ $list };

    return $sorted[0];
}



#
# usage
# explain how to call this script, then EXIT
sub usage
{
    print <<EOT;
decription
    parse scan results into bins for distinct values of a field,
    reports statistics for each bin and draws bar-whisker graphs.

usage
    $0 [word|window|channel|abbv] hm-output-file <ts|gene>

EOT

    exit 0;

}
```

hm_parse_best.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-05-10, zak.burke@dartmouth.edu
# -- given an output file from hm_parse.pl, which is sorted by
# -- harmonic mean, find each gene's top scoring directory

use Statistics::Descriptive;
use GD::Graph::boxplot;
use lib "/home/zburke/bg/workspace/perl-lib";
use scan_parse;

main();

sub main
{
    usage() unless ($ARGV[0]);

    my $file  = $ARGV[0];

    my (@lines);

    # parser for fields embedded in filename
    my $field_parser = scan_parse::field_parser('ts');

    # parse lines into records; grab first line to determine format
    open FILE, $file or die("couldn't open $file: $!");
    my $line = <FILE>;
    my $line_parser  = scan_parse::parser($line);
    while (<FILE>) {
        chomp;
        # skip header lines
        next unless /^[0-9]/;
        push @lines,  scan_parse::field_parse($_, $line_parser,
$field_parser);
    }
    close FILE;

    my @best;
    for my $line (@lines) {
        $line->{'file'} =~ /scans\/([^\.]+)\.fa_scan.*/;
        my $gene = $1;
        push (@best, $line->{'file'}) unless grep /$gene/, @best;
    }
    print join "\n", @best;
}


# --

# --
# -- usage
```

```
# -- show how to call this program, then exit
sub usage
{
    print <<EOT;
usage
    $0 <hm_parse.pl-output-file>

summary
    prints each gene's top-scoring directory, according to harmonic
mean


EOT

    exit 1;
}
```

hm_parse_by_ts.pl

```
#!/usr/local/bin/perl

# --
# -- hm_parse_by_ts.pl
# -- 2005-12-08, zak.burke@dartmouth.edu
# -- parse the output of hm.pl (which provides details about how a
# -- collection of scan parameters scores) to extract details about
# -- how different test-sets perform.
# --
# -- this script is nearly identical to hm_parse.pl with the
# -- exception of how the data are aggregated. here, they are grouped
# -- by membership in a test-set (i.e. a word-length, window-length,
# -- channel-count tuple) rather than by individual parameters. thus,
# -- the plots generated here do not suffer the problem hm_parse.pl's
# -- plots have of collapsing the unselected parameters (e.g. word-
length
# -- and window-length in a plot of channel-count) in the background.
# --
# -- here, the only variation is among training sets; all other
# -- parameters are constant within each bar.
# --
# -- $Author: zburke $
# -- $Revision: 1.1 $
# -- $Date: 2005/12/08 18:54:36 $
# --

use Statistics::Descriptive;
use GD::Graph::boxplot;
use Getopt::Long;
use strict;

use lib "/home/zburke/bg/workspace/perl-lib";
use scan_parse;


my $params = {
    "file" => "",       # file (from hm.pl) to parse
    "fp"   => "",       # field parser for TS name
    "sort" => "median", # how to sort results; median|mea
};



main();

sub main
{
    GetOptions($params, 'file=s', 'fp=s', 'sort=s');

    usage() unless ($params->{'file'} && $params->{'fp'});

    print "source: $params->{'file'}\n\n";
```

```
    my (@lines, $stats);

    # parser for fields embedded in filename
    my $field_parser = scan_parse::field_parser($params->{'fp'});

    # parse lines into records; grab first line to determine format
    open FILE, $params->{'file'} or die("couldn't open $params-
>{'file'}: $!");
    my $line = <FILE>;
    my $line_parser  = scan_parse::parser($line);
    while (<FILE>) {
        chomp;
        # skip header lines
        next unless /^[0-9]/;

        my $l = scan_parse::field_parse($_, $line_parser,
$field_parser);
        next if 0 == $l->{'hm'};
        push @lines,  scan_parse::field_parse($_, $line_parser,
$field_parser);
    }
    close FILE;

    # field headers
    my (@stat_fields) = qw(statistic median q1 min max q3);

    # calculate stats for each of these record fields
    my (@scores)      = qw(hm phi_score recall precision s_to_n);

    # correlate groups the data by field value
    # stats builds statistical profiles of that data
    for my $score (@scores) {

        # only score fields parsed from the header
        if (! grep /$score/,  (split "\t", &{ $line_parser }("head")))
{
            print STDERR "$score is not a valid field\n";
            print STDERR "fields include ".(&{ $line_parser
}("head"))."\n";
            next;
        }

        $stats = stats(correlate($score, \@lines), $score);
        print "\n\n$score\n";
        for (@stat_fields) {

            print "$_\t". (join "\t", @{ ${ $stats }{$_} }) ."\n";

        }

        my $s = Statistics::Descriptive::Full->new();
        $s->add_data(@{ ${ $stats }{'median'} });

        print "median\t" . $s->median(). "\n";
```

```
        print "stddev\t" . $s->standard_deviation() ."\n";
    }

}



# --

# --
# -- stats
# -- given a list of data buckets, generate statistical profiles
# -- for each bucket.
sub stats
{
    my ($list, $score) = @_;

    my %stats;
    my ($value_list, $field_list, $max) = ([],[],0);

    # calculate stats for each distinct test set
    for my $field (sort {$a cmp $b } keys %{ $list }) {

        next if (scalar @{ $list->{$field} } ) < 10;

        print STDERR "" . (scalar @{ $list->{$field} }) . " points in
the $field test-set.\n";
        my $s = Statistics::Descriptive::Full->new();
        $s->add_data(@{ $list->{$field} });

        my ($q1, $junk1) = $s->percentile(25);
        my ($q3, $junk2) = $s->percentile(75);

        push @{ $stats{'statistic'} }, "field-$field";
        push @{ $stats{'median'} }, $s->median;
        push @{ $stats{'mean'} }, $s->mean;
        push @{ $stats{'q1'} }, $q1;
        push @{ $stats{'min'} }, $s->min;
        push @{ $stats{'max'} }, $s->max;
        push @{ $stats{'q3'} }, $q3;

        # graph info
        push @{$field_list}, $field;
        push @{$value_list}, $list->{$field};

        my $list_max = list_max($list->{$field});
        $max = $list_max if ($list_max > $max);

    }

    my @items;
    for (my $i = 0; $i <= $#{ $field_list }; $i++) {
        my $item;
        $item->{'field'}  = $field_list->[$i];
        $item->{'median'} = $stats{'median'}->[$i];
```

```
        $item->{'mean'}   = $stats{'mean'}->[$i];
        $item->{'list'}   = $value_list->[$i];
        push @items, $item;
    }

    my ($values_sort, $fields_sort);
    for (sort { $b->{$params->{'sort'}} <=> $a->{$params->{'sort'}} }
@items) {
        push @$values_sort, $_->{'list'};
        push @$fields_sort, $_->{'field'};
    }


    # generate box plots
    box_plot($fields_sort, $values_sort, $max, $score, \%stats);

    return \%stats;

}



# --

# --
# -- box_plot
# -- output boxplots of statistical data as PNG images
sub box_plot
{
    my ($field_list, $value_list, $max, $score, $stats) = @_;

    my $s = Statistics::Descriptive::Full->new();

    for (@{ $value_list }) {
        $s->add_data(@{ $_ });
    }

    my $my_graph = GD::Graph::boxplot->new(1000,300);
    $my_graph->set(
        title          => "$score - $params->{'sort'}",
        x_label        => "median: ".$s->median()."; stddev: ".$s-
>standard_deviation(),
        do_stats       => 1,
        box_spacing    => 2,
        symbolc        => 'black',
        y_min_value    => 0,
        y_max_value    => 1,
        warnings       => 1,
        long_ticks     => 1,
        x_ticks        => 0,
        x_labels_vertical => 1,
    );

    # Output the graph
    my @data = ($field_list, $value_list);
```

```
    my $gd = $my_graph->plot(\@data);
    open(IMG, ">$score.$params->{'sort'}.png") or die $!;
    binmode IMG;
    print IMG $gd->png;


}



# --

# --
# -- correlate
# -- group scores for the given metric into bins for each
# -- distinct test set.
# -- $metric is 'phi' and $field is 'word-length'. the
# -- returned hash will have keys 3, 4 and 5 (the range
# -- of possible values for the word-length field) and
# -- the values of the hash will be array-refs of
# -- phi-scores. $hash{3}->[] will contain scores for all
# -- the records with word-length 3; $hash{4}->[] will
# -- contain scores for records with word-length 4, etc.
# --
# -- params
# --     metric - the score to calculate
# --     list - list of records
# --
# -- return
# --    hash-ref of metric values, keyed by field value
sub correlate
{
    my ($metric, $list) = @_;

    my $hash = {};
    # sort the metric-value for each record into the correct bin
    # for the given field.
    for ( @{ $list } ) {
        # for better sorting of field names
        my $word    = sprintf("%04d", $_->{'word'});
        my $window  = sprintf("%04d", $_->{'window'});
        my $channel = sprintf("%04d", $_->{'channel'});

        push @{ $hash->{ "$word:$window:$channel" } }, $_->{$metric};

    }

    return $hash;

}



# list_max
# return the biggest item on a list. the list is sorted
```

```
# in the process.
sub list_max
{
    my ($list) = @_;

    my @sorted = sort {$b <=> $a} @{ $list };

    return $sorted[0];
}



#
# usage
# explain how to call this script, then EXIT
sub usage
{
    print <<EOT;
decription
    parse scan results into bins for distinct test-sets.
    reports statistics for each bin and draws bar-whisker graphs.

usage
    $0 --file=hm-output-file --fp=<ts|gene> --sort=<median|mean>

EOT

    exit 0;

}
```

overlap.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-07-04, zak.burke@dartmouth.edu
# -- based on plot_crm2.pl 2005-07-04, this script does a three-way
# -- comparison between multiple LWF scans of the same region and drops
# -- pCRM regions that don't show up in all three sequences.
# --
# -- the selected pCRMs, dropped pCRMs and true CRMs are all plotted to
# -- a png file; the selected pCRM sequences are printed to STDOUT in
# -- FASTA format. note that the LWF score plotted on the graph is
# -- merely that of the last profile read for a given sequence.
# --
# -- 2005-12-06, zak.burke@dartmouth.edu
# -- added --dir param to set output dir for plots
# --
# -- parameters
# --   file: output file from hm_parse.pl
# --   genes: comma-separated list of genes to scan; null list means
use 'em all
# --   plot[noplot]: whether to generate plots, a time-consuming
process
# --   fasta[nofasta]: write selected sequence to STDOUT
# --   dir: where to write plot files; default is $CWD
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
}


use strict;
use Getopt::Long;
use GD::Graph;
use GD::Graph::mixed;

use lib $ENV{'BG_PERL_LIB'};
use scan_parse;
use nazina_db;
use extract;
use gene;
use util;
use lwf_plot;



use constant TRUE  => 1;
use constant FALSE => 0;
use constant FASTA_LINE_LENGTH => 60;

# file containing PIP data
```

```
use constant PIP_FILE =>
"/home/zburke/bg/workspace/data/nazina/profiles.txt";


main();


sub main
{
    my $params = {
        "file"  => "",
        "genes" => [],
        "plot"  => TRUE,
        "fasta" => TRUE,
        "help"  => FALSE,
        "dir"   => "",
        };

    GetOptions(
        $params,
        'file=s',
        'genes=s@',
        'plot!',
        'fasta!',
        'help!',
        'dir=s',
        );
    $params->{'genes'} = [split ",", (join ",", @{ $params->{'genes'} }
)];

    usage() if ($params->{'help'} || ! $params->{'file'});

    # data from input file...
    my $genes = input_parse($params);

    for my $abbv (sort keys %{ $genes }) {

        # directories with raw pCRM data
        my $pcrm_dirs = ${ $genes }{$abbv};

        print STDERR "plotting $abbv (" . (join ", ", @{ $pcrm_dirs })
. ")\n";

        # true CRMs
        my $gene = gene($abbv);

        my $blocks;
        $blocks->{'crm'} = $gene->enhancers();
        $blocks->{'exon'} = $gene->exons();
        $blocks->{'promoter'} = $gene->promoters();


        # extract pCRM positions from all test-sets
        my ($test, $plot_file, $plot_files) = pcrm_extract($pcrm_dirs);
```

```
        # separte pCRMs occuring in all test-sets from those which
don't
        my ($pcrm_blocks, $omit_blocks) = overlap_extract($test);

        # translate from lists of region boundaries (e.g. 100-150, 250-
400)
        # to arrays with elements for each position in the entire
sequence
        my $plot_data = profile($blocks, $pcrm_blocks, $omit_blocks,
$plot_file, $plot_files);
        $plot_data->{'abbv'} = $abbv;

        # create <gene>.profile.png
        plot($plot_data, $params->{'dir'}) if $params->{'plot'};

        # write FASTA sequence containing all pCRM blocks
        fasta($gene, $pcrm_blocks) if $params->{'fasta'};

    }


}



# --

# --
# -- input_parse
# -- parse LWF output file, extracting only datasets matching the
# -- chosen conditions and genes
sub input_parse
{
    my ($params) = @_;

    # result sets matching conditions
    my $lines = extract::extract($params->{'file'}, conditions());

    # group result sets by gene, e.g. $genes{'eve'}->[dir_1..dir_n]
    my (%genes);
    for my $line (@{ $lines }) {
        push @{ $genes{$line->{'abbv'}} }, $line->{'file'};
    }

    # filter result sets -- remove gene not specified on CLI
    my $genes = gene_select(\%genes, $params);

    return $genes;

}



# --
```

```
# --
# -- conditions
# -- parameter conditions to extract
sub conditions
{
    my $conditions = [
        {'word' => 3, 'window' => 200, 'channel' => 8},
        {'word' => 3, 'window' => 200, 'channel' => 13},
        {'word' => 3, 'window' => 200, 'channel' => 18},
        ];

    return $conditions;

}



# --

# --
# -- gene_select
# -- return list of genes to plot, as specified on the command line
sub gene_select
{
    my ($genes, $params) = @_;

    # no genes specified so scan 'em all ...
    return $genes if ($#{$params->{'genes'}} < 0);

    # ... or scan named genes only
    my %list;
    for (my $i = 0; $i < scalar @{ $params->{'genes'} }; $i++) {
        $list{${ $params->{'genes'} }[$i]} = ${ $genes }{${ $params-
>{'genes'} }[$i]};
    }

    return \%list;
}



# --

# --
# -- pcrm_extract
# -- extract the pCRM positions from a list of pCRM directories
sub pcrm_extract
{
    my ($pcrm_dirs) = @_;

    my (@test, $plot_file, @plot_files);

    for my $pcrm_dir (@{ $pcrm_dirs } ) {

        opendir DIR, $pcrm_dir;
```

```
        my @files = grep /plt$/, readdir DIR;
        close DIR;

        if (! scalar @files) {
            print STDERR "no .plt file in $pcrm_dir";
            next;
        }

        $plot_file = "$pcrm_dir/$files[0]";
        push @plot_files, $plot_file;

        # pCRM positions
        push @test, scan_parse::pcrm_file_read($pcrm_dir);
    }

    return (\@test, $plot_file, \@plot_files);

}


# --

# --
# -- overlap_extract
# -- given multiple sets, each set containing multiple regions, return
# -- only those regions which have some degree of overlap among every
# -- set.
# --
# -- This a little convoluted in order to make sure that the counting
# -- is accurate. Given sets A, B and C each with a long list of pCRM
# -- regions, it's possible that A will have 1 long pCRM that overlaps
# -- 2 short pCRMs in B but 0 pCRMs in C. A's pCRM should NOT be
counted
# -- because, although it overlaps 2 pCRMs it only overlaps 1 set and
# -- we want overlap across all sets.
sub overlap_extract
{
    my ($pcrms) = @_;

    my (@selected, @omitted);

    # how many overlaps are required?
    my $required = $#{ $pcrms };

    for (my $i = 0; $i <= $#{ $pcrms }; $i++) {

        # @temp holds everything but the current set
        my @temp;
        for (my $j = 0; $j <= $#{ $pcrms }; $j++) {
            push @temp, ${ $pcrms }[$j] unless $i == $j;
        }

        # loop over pCRMs in current set
        for my $pcrm (@{ ${ $pcrms }[$i] }) {
            # print "\t$pcrm->{'beg'} - $pcrm->{'end'} \n";
```

```
            my $count = 0;

            # loop over sets ...
            SET:
            for my $set (@temp) {

                # loop over pCRMs in curent set
                for (@{ $set }) {

                    # find the following overlaps:
                    # AAAA   |   AAAA |  AA  | AAAA
                    #   BBBB | BBBB   | BBBB |  BB
                    # 4 is special case of 1 and is therefore omitted
                    #
                    # once we've found an overlap in this set, we can
                    # skip to the next one
                    if (   ($_->{'beg'} >= $pcrm->{'beg'} && $_-
>{'beg'} <= $pcrm->{'end'})
                        || ($_->{'end'} >= $pcrm->{'beg'} && $_-
>{'end'} <= $pcrm->{'end'})
                        || ($_->{'beg'} <= $pcrm->{'beg'} && $_-
>{'end'} >= $pcrm->{'end'})
                        ) {
                        $count++;
                        #print "\t\tadding $_->{'beg'} - $_-
>{'end'}\n";
                        next SET;
                    }

                }
            }

            push @selected, $pcrm if $count == $required;
            push @omitted, $pcrm if $count < $required;

        }

    }

    # collapse overlapping regions into continous blocks
    # this will simplify creation of FASTA files later
    my $deduped = overlap_de_dup(\@selected);

    return ($deduped, \@omitted);

}



# --

# --
# -- overlap_de_dup
# -- given a sorted list with overlapping regions, e.g. 10-20, 15-25,
30-90,
```

```
# -- 100-120, 110-130, return a list with overlapping regions
collapsed,
# -- e.g. 10-25, 30-90, 100-130.
sub overlap_de_dup
{
    my ($overlaps) = @_;

    my (@list, $current);

    $current->{'beg'} = -1;
    $current->{'end'} = -1;

    for my $item (sort { $a->{'beg'} <=> $b->{'beg'} }  @{ $overlaps })
{

        # item is shorter than current; ignore it
        next if $item->{'end'} < $current->{'end'};

        # item starts after current ends:
        # add current to the list and reset current to item
        if ($item->{'beg'} > $current->{'end'}) {

            push @list, $current if $current->{'beg'} > 0;
            $current = $item;

            next;

        }
        # item starts in the middle of current and continues past its
end
        # reset current's end to item's end
        elsif ($item->{'beg'} <= $current->{'end'} && $item->{'end'} >=
$current->{'end'}) {

            $current->{'end'} = $item->{'end'};

        }
    }

    push @list, $current if $current->{'beg'} > 0;

    return (\@list);
}



# --

# --
# -- fasta
# -- extract given list of regions from a sequence
sub fasta
{
    my ($gene, $blocks) = @_;
```

```
    for my $block (@{ $blocks }) {

        # correct substr
        my $sequence = lc substr($gene->{'sequence'}, $block->{'beg'},
(($block->{'end'} + 1) - $block->{'beg'}));

        # wrong substr, but great results...
###        my $sequence = lc substr($gene->{'sequence'}, $block-
>{'beg'}, $block->{'end'});

        print ">$gene->{'abbv'} $block->{'beg'} - $block->{'end'}\n";

        for (my $i = 0; $i < length($sequence); $i+= FASTA_LINE_LENGTH)
        {
            print substr($sequence, $i, FASTA_LINE_LENGTH). "\n";
            last if (substr($sequence, $i, FASTA_LINE_LENGTH) eq "");
        }
        print "\n";

    }

}



# --

# --
# -- plot
# -- create a graph
sub plot
{
    # fields: array-ref, each value is a sequence-offset
    # crms: array-ref, non-zero indicates CRM
    # pcrms: array-ref, non-zero indicates pCRM
    # omits: array-ref, non-zero indicates non-overlapping pCRM region
    # abbv: string, name of gene to plot
    # stats: hash-ref, tp, fp, tn, fn stats about pCRM sequences
    my ($input, $dir) = @_;

    my (@data, $types, $dclrs);
    my $pip = lwf_plot::pip($input->{'abbv'}, PIP_FILE);

    @data  = ($input->{'fields'}, $input->{'exons'}, $input-
>{'promoters'}, $input->{'crms'}, $input->{'pcrms'}, $input-
>{'omits'});

    #            exons   promos  CRMs    pCRMs   omits
    $types = [qw(bars    bars    bars    bars    bars  )];
    $dclrs = [qw(marine  blue    dblue   lgray   yellow )];

    # plot LWF feature scores for each test-set
    for (@{ $input->{'thresholds'} }) {
        my $threshold = lwf_plot::threshold_normalize(
            $_,
```

```
            $input->{'threshold_min'},
            $input->{'threshold_max'});
        push @data, $threshold;
        push @{$types}, "lines";
        push @{$dclrs}, "black";
    }

    # use pip data if available
    if ($pip) {
        push @data, $pip;
        push @{ $types }, "lines";
        push @{ $dclrs }, "green";
    }

    eval
    {
        my $filename = ($dir ? "$dir/$input->{'abbv'}.profile.png" :
"$input->{'abbv'}.profile.png");

        my $graph = GD::Graph::mixed->new(800, 100) or die GD::Graph-
>error;
        $graph->set(
            title    => "$input->{'abbv'}",
            types    => $types,
            dclrs    => $dclrs,
            x_ticks  => FALSE,
            x_label_skip  => 1000,
            x_label       => lwf_plot::stats_calculate($input-
>{'stats'}),
            y_max_value   => 1,
            y_min_value   => 0,
            correct_width => FALSE,

        ) or die $graph->error;
        my $gd = $graph->plot(\@data) or die $graph->error;
        #open(IMG, ">$input->{'abbv'}.profile.png") or die $!;
        open(IMG, ">$filename") or die $!;
        binmode IMG;
        print IMG $gd->png();
        close IMG;
    };
    if ($@) {
        print STDERR "ERROR: $@";
    }
}



# --

# --
# -- profile
# -- build plot-data arrays from CRM, pCRM sequence data.
sub profile
{
```

```
    my ($blocks, $pcrm_blocks, $omit_blocks, $file, $files) = @_;

    # plot file is space-delimited scores all on one line
    open FILE, $file;
    my $line = <FILE>;
    close FILE;
    my @list = split " ", $line;

    my (@fields, @exons, @promoters, @crms, @pcrms, @omits, @threshold,
$t_min, $t_max);

    my ($in_crm, $in_pcrm, $in_omit, $fp, $fn, $tp, $tn) = (0, 0, 0, 0,
0, 0, 0);

    # read the plot data
    for (my $i = 0; $i <= $#list; $i++) {

        $in_pcrm = in_crm($pcrm_blocks, $i);     # in pCRM?
        $in_omit = in_crm($omit_blocks, $i);     # in omitted pCRM?
        $in_crm  = in_crm($blocks->{'crm'}, $i); # in CRM?

        if    ($in_pcrm   && $in_crm)   { $tp++; }
        elsif ($in_pcrm   && ! $in_crm) { $fp++; }
        elsif (! $in_pcrm && ! $in_crm) { $tn++; }
        elsif (! $in_pcrm && $in_crm)   { $fn++; }

        push @fields, "-$i-";
        push @threshold, $list[$i];
        push @pcrms, $in_pcrm ? .75 : 0;
        push @omits, $in_omit ? .75 : 0;
        push @crms,  $in_crm  ? 1   : 0;
        push @exons, in_crm($blocks->{'exon'}, $i) ? 1 : 0;
        push @promoters, in_crm($blocks->{'promoter'}, $i) ? 1 : 0;

        $t_min = $list[$i] if $list[$i] < $t_min;
        $t_max = $list[$i] if $list[$i] > $t_max;
    }

    my $stats = { 'tp' => $tp, 'fp' => $fp, 'tn' => $tn, 'fn' => $fn};

    my @profiles;
    for my $filename (@{ $files }) {
        push @profiles, profile_plotfile_reader($filename);
    }

    return {
        "fields"    => \@fields,
        "threshold" => \@threshold,
        "threshold_min" => $t_min,
        "threshold_max" => $t_max,
        "thresholds"    => \@profiles,
        "exons"         => \@exons,
        "promoters"     => \@promoters,
        "crms"      => \@crms,
        "pcrms"     => \@pcrms,
```

```
        "omits"     => \@omits,
        "stats"     => $stats,
        };
}



# --

# --
# -- profile_plotfile_reader
# --
sub profile_plotfile_reader
{
    my ($filename) = @_;

    open FILE, $filename;
    my $line = <FILE>;
    close FILE;
    return [split " ", $line];
}



# --

# --
# -- gene
# -- return a gene object, given its abbreviation
sub gene
{
    my ($abbv) = @_;
    my $dao = nazina_db->new();
    my $list = $dao->genes_by_abbv();

    return $dao->gene(${ $list }{$abbv}->{'gene_id'});

}



# --

# --
# -- in_crm
# -- return TRUE if $i is within a defined CRM region in @$list;
# -- return FALSE otherwise.
sub in_crm
{
        my ($list, $i) = @_;


        for (@{ $list }) {

                return TRUE if ($i >= $_->{'beg'} && $i <= $_->{'end'})
```

```
        }

        return FALSE;
}



# --

# --
# -- usage
# -- show how to use program, then exit
sub usage
{
    print <<EOT;
usage
    $0 --file <hm.pl-output-file> [--genes gene-1,..,gene-n] [--noplot]
[--nofasta]
    e.g. $0 --file hm.output.txt
    e.g. $0 --file hm.output.txt --genes btd,eve --noplot --nofasta

description
    plots pCRM data over true CRM data. pCRM segments are compiled from
    multiple LWF scans; only pCRMs with some degree of overlap in every
    scan are plotted. write FASTA-formatted pCRM sequences to stdout.

arguments
    file: hm.output.txt -- output file from hm.pl
    genes: gene-1 -- list of one or more genes to plot; omit to plot
all genes
    plot/noplot: plot pCRMs to a png file (default: true)
    fasta/nofasta: write FASTA-formatted pCRMs to stdout (default:
true)
    dir: where to create output plots
EOT


    exit 0;
}
```

plot_crm.pl

```
#!/usr/local/bin/perl

# --
# -- Plot pCRM recognition profiles over true CRMs to provide a simple
# -- picture of how well a CRM-finding tool's profiles track to true
CRMs.
# -- The plot format is CRM bars in the background with a continuous
line
# -- in the foreground denoting the pCRM profile score.
# --
# -- Input is a list of pCRM files named like "<gene>.fa.plt" where
<gene>
# -- is an abbreviation for a gene which can be looked up in a
CRM/sequence
# -- DB which to extract the true CRM positions.
# --
# -- 2005-04-18, zak.burke@dartmouth.edu
# --
# -- 2005-05-07, zak.burke@dartmouth.edu
# -- The output isn't very useful. Withouth knowing the pCRM
recognition
# -- threshold and other scanning params such as the minimum pCRM
width,
# -- these graphs won't clearly show pCRM regions. See plot_crm2.pl
# -- for better output.
# --

use strict;
use GD::Graph;
use GD::Graph::mixed;
use lib "/home/zburke/bg/workspace/perl-lib";
use nazina_db;
use gene;

use constant TRUE  => 1;
use constant FALSE => 0;

main();


sub main
{
    usage() unless scalar @ARGV;

    # plot each input file
    for my $file (@ARGV) {

        # abbreviation of gene to plot
        my @list = split "\\.", $file;
        my $abbv = $list[0];

        # retrieve CRMs for this gene
```

```
        my $enhancers = gene($abbv)->enhancers();
        print "plotting $abbv from $file\n";
        crm_plot(input($enhancers, $file), $abbv);
    }
}



# --

# --
# -- plot
# -- create a graph
sub crm_plot
{
    my ($fields, $crms, $pcrms, $abbv) = @_;

    my @data = ($fields, $crms, $pcrms);

    my $graph = GD::Graph::mixed->new(800, 300) or die GD::Graph-
>error;

    # bars for the CRMs; lines for the pCRM profile
    $graph->set(
        types   => ['bars', 'lines'],
        x_ticks => FALSE,
        x_label_skip => 1000,
        correct_width => FALSE,

    ) or die $graph->error;
    my $gd = $graph->plot(\@data) or die $graph->error;
    open(IMG, ">$abbv.profile.png") or die $!;
        binmode IMG;
    print IMG $gd->png();
    close IMG;
}



# --

# --
# -- input
# -- build plot-data arrays from CRM, pCRM sequence data.
sub input
{
    my ($enhancers, $file) = @_;

    # plot file is space-delimited scores all on one line
    open FILE, $file;
    my $line = <FILE>;
    close FILE;
    my @list = split " ", $line;

    # plot arrays: x-values, plot-1, plot-2
```

```
    my (@fields, @crms, @pcrms);

    # read the plot data
    for (my $i = 0; $i <= $#list; $i++) {
        push @fields, "-$i-";
        push @pcrms,  $list[$i];
        push @crms,   (in_crm($enhancers, $i) ? 1 : 0);
    }

    return (\@fields, \@crms, \@pcrms);
}



# --

# --
# -- gene
# -- return a gene object, given its abbreviation
sub gene
{
    my ($abbv) = @_;
    my $dao = nazina_db->new();
    my $list = $dao->genes_by_abbv();

    return $dao->gene(${ $list }{$abbv}->{'gene_id'});

}



# --

# --
# -- in_crm
# -- return TRUE if $i is within a defined CRM region in @$list;
# -- return FALSE otherwise.
sub in_crm
{
        my ($list, $i) = @_;


        for (@{ $list }) {

                return TRUE if ($i >= $_->{'beg'} && $i <= $_->{'end'})

        }

        return FALSE;
}



# --
```

```
# --
# -- usage
# -- show how to use program, then exit
sub usage
{
    print <<EOT;
usage
    $0 <plot-data-file-1> .. <plot-data-file-n>
    e.g. $0 eve.fa.plt

description
    plots pCRM data from plot-data-file over true CRM data
    from Nazina and Papatsenko's CRM DB. .plt files are
    generated by N&P's LWF CRM finder for sequences shorter
    than 100,000 bases long.

EOT


    exit 0;
}
```

plot_crm2.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-04-18, zak.burke@dartmouth.edu
# -- plot CRMs, pCRMs, pCRM profile scores and percent-identity profile
# -- scores on a single graph to see how CRM-finding tools are doing.
# --
# -- The plot format is CRM bars in the background, shorter pCRM bars
in
# -- front of the CRMs, and lines denoting the pCRM recognition profile
# -- and PIP are in the foreground.
# --
# -- 2005-05-09, zak.burke@dartmouth.edu
# -- retooled input interface to accept a list of directories on
# -- STDIN to make it easy to pipe output from a script that lists
# -- profile directories straight into this script.
# --
# -- 2005-06-28, zak.burke@dartmouth.edu
# -- include scan-dirname in image name so that if multiple scans
# -- of the same gene are in the input file the output filenames
# -- won't overlap
# --
# -- INPUT
# -- pCRM directory: name of a directory containing a <gene>.fa.xtr.dat
file
# --     which contains positions of pCRMs and a <gene>.fa.plt file
which
# --     contains LWF's sequence recognition profile
# --
# -- PIP profiles are read from the file named by the constant PIP_FILE
# --
# --

use strict;
use GD::Graph;
use GD::Graph::mixed;
use lib "/home/zburke/bg/workspace/perl-lib";
use scan_parse;
use nazina_db;
use gene;




use constant TRUE  => 1;
use constant FALSE => 0;

# file containing PIP data
use constant PIP_FILE =>
"/home/zburke/bg/workspace/data/nazina/profiles.txt";
```

```
main();


sub main
{
    my $pcrm_dir = "";
    my $success = FALSE;

    if ($ARGV[0]) {
        $success = TRUE;
        for (@ARGV) {
            chomp;
            prepare($_);

        }
    } else {
        while (<>) {
            $success = TRUE;
            chomp;
            prepare($_);
        }
    }


    usage() unless $success;
}



# --

# --
# -- prepare
# -- given a pCRM directory, find the pCRM profile file and gene-name
# -- and hand these off to input() for further processing
sub prepare
{
    my ($pcrm_dir) = @_;

    opendir DIR, $pcrm_dir;
    my @files = grep /plt$/, readdir DIR;
    close DIR;

    die("no .plt file in $pcrm_dir") unless scalar @files;

    my $plot_file = "$pcrm_dir/$files[0]";

    # abbreviation of gene to plot
    $plot_file =~ /.*\/([^\.]+)\.fa\.*/;
    my $abbv = $1;
    my @dir = split "_", $pcrm_dir;
    my $filename = "$abbv" . "_" . $dir[$#dir];

    my $crms = gene($abbv)->enhancers();
    print "plotting $abbv from $plot_file\n";
```

```
    crm_plot(input($crms, $pcrm_dir, $plot_file), $abbv, $filename);
}



# --

# --
# -- plot
# -- create a graph
sub crm_plot
{
    my ($fields, $crms, $pcrms, $threshold, $abbv, $filename) = @_;

    my (@data, $types, $dclrs);
    my $pip = pip($abbv);

    # use pip data if available
    if ($pip) {

        @data  = ($fields, $crms, $pcrms, $pip);
        $types = [qw(bars  bars  lines)];
        $dclrs = [qw(dblue lgray green)],

    } else {

        @data  = ($fields, $crms, $pcrms);
        $types = [qw(bars  bars)];
        $dclrs = [qw(dblue lgray)],

    }

    eval
    {

        my $graph = GD::Graph::mixed->new(800, 100) or die GD::Graph-
>error;
        $graph->set(
            title    => "$abbv",
            types    => $types,
            dclrs    => $dclrs,
            x_ticks  => FALSE,
            x_label_skip  => 1000,
            x_label       => "",
            y_max_value   => 1,
            y_min_value   => 0,
            correct_width => FALSE,

        ) or die $graph->error;
        my $gd = $graph->plot(\@data) or die $graph->error;
        open(IMG, ">$filename.profile.png") or die $!;
            binmode IMG;
        print IMG $gd->png();
        close IMG;
    };
```

```
    if ($@) {
        print STDERR "ERROR: $@";
    }
}



# --

# --
# -- crm_plot_threshold
# -- create a graph including LWF's cutoff value
sub crm_plot_threshold
{
    my ($fields, $crms, $pcrms, $threshold, $abbv, $filename) = @_;

    my (@data, $types, $dclrs);
    my $pip = pip($abbv);

    # use pip data if available
    if ($pip) {

        @data  = ($fields, $crms, $pcrms, $pip, $threshold);
        $types = [qw(bars  bars  lines lines)];
        $dclrs = [qw(dblue lgray green black)],

    } else {

        @data  = ($fields, $crms, $pcrms, $threshold);
        $types = [qw(bars  bars  lines)];
        $dclrs = [qw(dblue lgray black)],

    }

    eval
    {

        my $graph = GD::Graph::mixed->new(800, 300) or die GD::Graph-
>error;
        $graph->set(
            title    => "$abbv",
            types    => $types,
            dclrs    => $dclrs,
            x_ticks  => FALSE,
            x_label_skip  => 1000,
            x_label       => "",
            y_max_value   => 1,
            y_min_value   => -4,
            correct_width => FALSE,

        ) or die $graph->error;
        my $gd = $graph->plot(\@data) or die $graph->error;
        open(IMG, ">$filename.profile.png") or die $!;
            binmode IMG;
        print IMG $gd->png();
```

```
        close IMG;
    };
    if ($@) {
        print STDERR "ERROR: $@";
    }
}



# --

# --
# -- input
# -- build plot-data arrays from CRM, pCRM sequence data.
sub input
{
    my ($crms, $pcrm_dir, $file) = @_;

    my $test = scan_parse::pcrm_file_read($pcrm_dir);

    # plot file is space-delimited scores all on one line
    open FILE, $file;
    my $line = <FILE>;
    close FILE;
    my @list = split " ", $line;

    # plot arrays: x-values, plot-1, plot-2
    my (@fields, @crms, @pcrms, @threshold);

    # read the plot data
    for (my $i = 0; $i <= $#list; $i++) {
        push @fields, "-$i-";
        push @threshold, $list[$i];
        push @pcrms, (in_crm($test, $i) ? .75 : 0);
        push @crms,  (in_crm($crms, $i) ? 1 : 0);
    }

    return (\@fields, \@crms, \@pcrms, \@threshold);
}



# --

# --
# -- gene
# -- return a gene object, given its abbreviation
sub gene
{
    my ($abbv) = @_;
    my $dao = nazina_db->new();
    my $list = $dao->genes_by_abbv();

    return $dao->gene(${ $list }{$abbv}->{'gene_id'});
```

```
}



# --

# --
# -- in_crm
# -- return TRUE if $i is within a defined CRM region in @$list;
# -- return FALSE otherwise.
sub in_crm
{
        my ($list, $i) = @_;


        for (@{ $list }) {

                return TRUE if ($i >= $_->{'beg'} && $i <= $_->{'end'})

        }

        return FALSE;
}



# --

# --
# -- pip
# -- extract percent-identity-profile for D. Melangaster and D.
Pseudoobscura
# --
# -- return
# --     array-ref of percent identity for each position from 1-16000
# -- params
# --     $abbv - abbreviation of gene to extract profile for
# --
sub pip
{
    my ($abbv) = @_;

    open FILE, PIP_FILE;
    my $line = <FILE>;
    chomp $line;
    my @genes = split /\t/, $line;
    my @list = ();
    my $pos = 0;
    while (<FILE>) {
        chomp;
        my @scores = split /\t/;
        for (my $i = 0; $i < scalar @scores; $i++) {
            push @{ $list[$i] }, ($scores[$i] / 100);
        }
        last if $pos++ > 16000;
```

```
    }
    close FILE;

    for (my $i = 0; $i < scalar @genes; $i++) {
        next unless $genes[$i] eq $abbv;

        return $list[$i];
    }

}



# --

# --
# -- usage
# -- show how to use program, then exit
sub usage
{
    print <<EOT;
usage
    $0 <pcrm-dir>
    e.g. $0 ./scans/eve.fa_scan_225
    e.g. cat list-of-dirs | $0

description
    plots pCRM data from plot-data-file over true CRM data
    from Nazina and Papatsenko's CRM DB. .plt files are
    generated by N&P's LWF CRM finder for sequences shorter
    than 100,000 bases long.

EOT


    exit 0;
}
```

plot_crm2_xargs.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-04-18, zak.burke@dartmouth.edu
# -- plot CRMs, pCRMs, pCRM profile scores and percent-identity profile
# -- scores on a single graph to see how CRM-finding tools are doing.
# --
# -- The plot format is CRM bars in the background, shorter pCRM bars
in
# -- front of the CRMs, and lines denoting the pCRM recognition profile
# -- and PIP are in the foreground.
# --
# -- 2005-05-09, zak.burke@dartmouth.edu
# -- retooled input interface to accept a list of directories on
# -- STDIN to make it easy to pipe output from a script that lists
# -- profile directories straight into this script.
# --
# -- 2005-06-28, zak.burke@dartmouth.edu
# -- include scan-dirname in image name so that if multiple scans
# -- of the same gene are in the input file the output filenames
# -- won't overlap
# --
# -- 2005-12-06, zak.burke@dartmouth.edu
# -- accept --dir=/path/to/output parameter to stash output files
# -- wherever you please. works well with xargs, e.g.
# -- ./extract.pl hm.ouput.txt | xargs $0 --dir=/tmp/
# --
# -- INPUT
# -- pCRM directory: name of a directory containing a <gene>.fa.xtr.dat
file
# --     which contains positions of pCRMs and a <gene>.fa.plt file
which
# --     contains LWF's sequence recognition profile
# --
# -- PIP profiles are read from the file named by the constant PIP_FILE
# --
# --

use strict;
use GD::Graph;
use GD::Graph::mixed;
use lib "/home/zburke/bg/workspace/perl-lib";
use scan_parse;
use nazina_db;
use gene;
use lwf_plot;



use constant TRUE  => 1;
```

```
use constant FALSE => 0;

# file containing PIP data
use constant PIP_FILE =>
"/home/zburke/bg/workspace/data/nazina/profiles.txt";


my $dao = undef;
main();


sub main
{
    my $pcrm_dir = "";
    my $success = FALSE;

    if ($ARGV[0]) {
        my $dir_out = "";
        $success = TRUE;
        parse($_, \$dir_out) for (@ARGV);
    } else {
        my $dir_out = "";
        while (<>) {
            $success = TRUE;
            parse($_, \$dir_out);
        }
    }


    usage() unless $success;
}



# --

# --
# -- dir_out
# -- given a param like --foo=bar, return bar.
sub dir_out
{
    my ($param) = @_;

    my ($junk, $dir) = split /=/, $param;
    return $dir;
}



# --

# --
# -- parse
# -- parse a single command-line parameter
sub parse
```

```
{
    my ($dir, $dir_out) = @_;

    if ($dir =~ /^--dir/) {
        $$dir_out = dir_out($dir);
        return;
    }
    chomp $dir;
    prepare($dir, $$dir_out);
}



# --

# --
# -- prepare
# -- given a pCRM directory, find the pCRM profile file and gene-name
# -- and hand these off to input() for further processing
sub prepare
{
    my ($pcrm_dir, $dir_out) = @_;

    opendir DIR, $pcrm_dir;
    my @files = grep /plt$/, readdir DIR;
    close DIR;

    die("no .plt file in $pcrm_dir") unless scalar @files;

    my $plot_file = "$pcrm_dir/$files[0]";

    # abbreviation of gene to plot
    $plot_file =~ /.*\/([^\.]+)\.fa\.*/;
    my $abbv = $1;
    my @dir = split "_", $pcrm_dir;
    my $filename = "$abbv" . "_" . $dir[$#dir];
    $filename = "$dir_out/$filename" if $dir_out;

    my $blocks;
    $blocks->{'crms'}      = gene($abbv)->enhancers();
    $blocks->{'exons'}     = gene($abbv)->exons();
    $blocks->{'promoters'} = gene($abbv)->promoters();

    my $plot_data = profile($blocks, $pcrm_dir, $plot_file);

    print "plotting $abbv from $plot_file\n";
    crm_plot($plot_data, $abbv, $filename);
    #crm_plot(input($crms, $pcrm_dir, $plot_file), $abbv, $filename);
}



# --

# --
```

```
# -- crm_plot
# -- create a graph
sub crm_plot
{
    my ($input, $abbv, $filename) = @_;

    my (@data, $types, $dclrs);
    my $pip = lwf_plot::pip($abbv, PIP_FILE);
    my $t_normal = lwf_plot::threshold_normalize(
        $input->{'threshold'},
        $input->{'threshold_min'},
        $input->{'threshold_max'});

    @data  = ($input->{'fields'}, $input->{'exons'}, $input-
>{'promoters'}, $input->{'crms'}, $input->{'pcrms'}, $t_normal);

    #           exons   promos  CRMs    pCRMs  feature-score
    $types = [qw(bars    bars    bars    bars   lines)];
    $dclrs = [qw(marine  blue    dblue   lgray  black)];

    # use pip data if available
    if ($pip) {
        push @data, $pip;
        push @{ $types }, "lines";
        push @{ $dclrs }, "green";
    }


    eval
    {

        my $graph = GD::Graph::mixed->new(800, 100) or die GD::Graph-
>error;
        $graph->set(
            title    => "$abbv",
            types    => $types,
            dclrs    => $dclrs,
            x_ticks  => FALSE,
            x_label_skip  => 1000,
            x_label       => lwf_plot::stats_calculate($input-
>{'stats'}),
            y_max_value   => 1,
            y_min_value   => 0,
            correct_width => FALSE,

        ) or die $graph->error;
        my $gd = $graph->plot(\@data) or die $graph->error;
        open(IMG, ">$filename.profile.png") or die $!;
            binmode IMG;
        print IMG $gd->png();
        close IMG;
    };
    if ($@) {
        print STDERR "ERROR: $@";
    }
```

```
}



# --

# --
# -- profile
# -- build plot-data arrays from CRM, pCRM sequence data.
sub profile
{
    my ($blocks, $pcrm_dir, $file) = @_;

    $blocks->{'pcrms'} = scan_parse::pcrm_file_read($pcrm_dir);

    # plot file is space-delimited scores all on one line
    open FILE, $file;
    my $line = <FILE>;
    close FILE;
    my @list = split " ", $line;

    # plot arrays: x-values, plot-1, plot-2
    my (@fields, @exons, @promoters, @crms, @pcrms, @threshold, $t_min,
$t_max);
    my ($in_crm, $in_pcrm, $fp, $fn, $tp, $tn) = (0, 0, 0, 0, 0, 0, 0);

    # read the plot data
    for (my $i = 0; $i <= $#list; $i++) {

        $in_pcrm = in_crm($blocks->{'pcrms'}, $i); # in pCRM?
        $in_crm  = in_crm($blocks->{'crms'},  $i); # in CRM?

        if    ($in_pcrm   && $in_crm)   { $tp++; }
        elsif ($in_pcrm   && ! $in_crm) { $fp++; }
        elsif (! $in_pcrm && ! $in_crm) { $tn++; }
        elsif (! $in_pcrm && $in_crm)   { $fn++; }

        push @fields, "-$i-";
        push @threshold, $list[$i];
        push @pcrms, $in_pcrm ? .75 : 0;
        push @crms,  $in_crm  ? 1   : 0;
        push @exons,     in_crm($blocks->{'exons'}, $i) ? 1 : 0;
        push @promoters, in_crm($blocks->{'promoters'}, $i) ? 1 : 0;

        $t_min = $list[$i] if $list[$i] < $t_min;
        $t_max = $list[$i] if $list[$i] > $t_max;
    }

    my $stats = { 'tp' => $tp, 'fp' => $fp, 'tn' => $tn, 'fn' => $fn};

    return {
        "fields"    => \@fields,
        "threshold" => \@threshold,
        "threshold_min" => $t_min,
        "threshold_max" => $t_max,
```

```
        "exons"         => \@exons,
        "promoters"     => \@promoters,
        "crms"      => \@crms,
        "pcrms"     => \@pcrms,
        "stats"     => $stats,
        };
}




# --

# --
# -- gene
# -- return a gene object, given its abbreviation
sub gene
{
    my ($abbv) = @_;
    $dao = nazina_db->new() unless $dao;
    my $list = $dao->genes_by_abbv();

    return $dao->gene(${ $list }{$abbv}->{'gene_id'});

}



# --

# --
# -- in_crm
# -- return TRUE if $i is within a defined CRM region in @$list;
# -- return FALSE otherwise.
sub in_crm
{
        my ($list, $i) = @_;


        for (@{ $list }) {

                return TRUE if ($i >= $_->{'beg'} && $i <= $_->{'end'})

        }

        return FALSE;
}



# --

# --
# -- usage
# -- show how to use program, then exit
```

```
sub usage
{
    print <<EOT;
usage
    $0 <pcrm-dir>
    e.g. $0 ./scans/eve.fa_scan_225
    e.g. cat list-of-dirs | $0

description
    plots pCRM data from plot-data-file over true CRM data
    from Nazina and Papatsenko's CRM DB. .plt files are
    generated by N&P's LWF CRM finder for sequences shorter
    than 100,000 bases long.

EOT


    exit 0;
}
```

ston.pl

```
#!/usr/local/bin/perl

# --
# -- show signal-to-noise ration for genes with sequence and
# -- CRM data
# --
# -- 2005-04-18, zak.burke@dartmouth.edu
# --

use strict;
use lib "/home/zburke/bg/workspace/perl-lib";
use nazina;
use nazina_db;



main();

sub main
{
    # how to sort output? default is gene abbreviation
    # see gene.pm for other params
    my $sort = $ARGV[0] || "abbv";

    my $dao = nazina_db->new();
    my $genes = $dao->genes();

    my @list = ();
    for ( sort { ${ $genes }{$a}->{'abbv'} cmp ${ $genes }{$b}-
>{'abbv'} } keys %{ $genes }) {
        my $gene = $dao->gene(${ $genes }{$_}->{'gene_id'});
        $gene->{'s2n'} = $gene->s2n();
        next unless $gene->{'s2n'};
        push @list, $gene;
    }

    for my $gene (sort { $a->{$sort} cmp $b->{$sort} } @list) {

        print "$gene->{'name'} ($gene->{'abbv'})\n\t" . $gene-
>s2n()."\n";

    }
}
```

Appendix 3 Corregulation Filtering Application Code

Perl Files

compare.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-07-19, zak.burke@dartmouth.edu
# -- given directories with motif-density profiles for different motif
counts,
# -- generate HTML files that include the same gene at all different
counts
# -- in order to see how different counts compare.
# --
# -- this should helps us determine how many of the top-N motifs from
SCOPE we
# -- should use when trying to rank the pCRM sequences that were fed to
SCOPE.
# --

use strict;
use Getopt::Long;

use constant TRUE  => 1;
use constant FALSE => 0;


main();

sub main
{
    my $params = {
        "genes" => [],
        "numbers" => [],
        "help"  => FALSE,
        };

    GetOptions(
        $params,
        'genes=s@',
        'numbers=s@',
        'help!',
        );
```

```
    $params->{'genes'}   = [split ",", (join ",", @{ $params->{'genes'}
} )];
    $params->{'numbers'} = [split ",", (join ",", @{ $params-
>{'numbers'} } )];

    for my $gene (@{ $params->{'genes'} }) {
        mkdir $gene;

        my $dir_name = "$gene.$params->{'numbers'}->[1]";
        opendir DIR, $dir_name or die("couldn't open $dir_name: $!");

        my @regulators = grep /png$/, readdir DIR;
        closedir DIR;

        open INDEX, ">$gene/index.html";
        for my $reg (sort @regulators) {
            $reg =~ /(.*)\.motif_profile\.png/;
            my $name = $1;

            open FILE, ">$gene/$name.html";
            print FILE "<table>\n";
            for my $number (@{ $params->{'numbers'} }) {
                print FILE "<tr><td><h3>$number</h3></td><td><img
src='../$gene.$number/$reg'></td></tr>\n";
            }
            print FILE "</table>\n";
            close FILE;

            print INDEX "<li><a href=\"$name.html\">$name</li>\n";
        }
        close INDEX;
    }

}
```

compare2.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-07-19, zak.burke@dartmouth.edu
# -- given directories with motif-density profiles for different motif
counts,
# -- generate HTML files that include the same gene at all different
counts
# -- in order to see how different counts compare.
# --
# -- this should helps us determine how many of the top-N motifs from
SCOPE we
# -- should use when trying to rank the pCRM sequences that were fed to
SCOPE.
# --

use strict;
use Getopt::Long;

use constant TRUE  => 1;
use constant FALSE => 0;


main();

sub main
{
    my $params = {
        "genes" => [],
        "numbers" => [],
        "rounds" => [],
        "help"  => FALSE,
        };

    GetOptions(
        $params,
        'genes=s@',
        'numbers=s@',
        'rounds=s@',
        'help!',
        );
    $params->{'genes'}   = [split ",", (join ",", @{ $params->{'genes'}
} )];
    $params->{'numbers'} = [split ",", (join ",", @{ $params-
>{'numbers'} } )];
    $params->{'rounds'}  = [split ",", (join ",", @{ $params-
>{'rounds'} } )];

    for my $gene (@{ $params->{'genes'} }) {
        mkdir $gene unless -d $gene;

        my $dir_name = "round1/$gene.$params->{'numbers'}->[0]";
```

```
        opendir DIR, $dir_name or die("couldn't open $dir_name: $!");

        my @regulators = grep /png$/, readdir DIR;
        closedir DIR;

        open INDEX, ">$gene/index.html";
        for my $reg (sort @regulators) {
            $reg =~ /(.*)\.motif_profile\.png/;
            my $name = $1;

            open FILE, ">$gene/compare.$name.html";
            print FILE "<table>\n";
            for my $number (@{ $params->{'numbers'} }) {
                print FILE "<tr>\n";
                print FILE "<td rowspan=2><h1>$number</h1></td>\n";
                print FILE "<td><img
src='../round1/$gene.$number/$reg'></td></tr>\n";
                print FILE "<tr><td><img
src='../round2/$gene.$number/$reg'></td></tr>\n";
                print FILE "<tr><td colspan=2><hr></td></tr>\n";
            }
            print FILE "</table>\n";
            close FILE;

            print INDEX "<li>$name | <a
href=\"compare.$name.html\">compare</a></li>\n";
        }
        close INDEX;
    }

}
```

motif_rank.pl

```
#!/usr/bin/perl -w

# --
# -- 2005-07-12, zak.burke@dartmouth.edu
# --
# -- given a list of sequences and a list of motifs, draw a heat map of
# -- motif density for each sequence.
# --
# -- graphing routines based on overlap.pl
# --
# --

BEGIN {
    # output FASTA filename
    use constant OUTPUT_FASTA_FILE => "update.fa";

    # output phi-score filename
    use constant OUTPUT_PHI_SCORE_FILE => "../phi_score.txt";


    $ENV{'BG_PERL_LIB'}   = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'HOME_PERL_LIB'} = "." unless $ENV{'HOME_PERL_LIB'};

    $time_beg = time();

}

END {
    print STDERR util::elapsed_time(time() - $time_beg);
}


# --
# -- nothing to configure below
# --

use strict;

use Getopt::Long;
use Data::Dumper;

use lib $ENV{'BG_PERL_LIB'};
use scan_parse;
use nazina_db;
use extract;
use gene;
use motif_rank;
use motif_rank_plot;
use util;
```

```
# height of a true CRM
use constant PLOT_HEIGHT => 0.4;



# unbuffer stdout
$|++;

main();


sub main
{
    my $time_beg = time();

    # parse command line options, then show help if necessary
    my $params = init();
    usage() if $params->{'help'};
    usage() unless ($params->{'fasta'} && $params->{'motifs'});

    # read input files
    my $pcrms  = motif_rank::fasta_read($params);
    my $motifs = motif_rank::motif_read($params);

    my %summary = ();

    my $block_count = 0;

    # parse motifs in distinct genes
    for my $abbv (sort keys %{ $pcrms }) {
        print STDERR "locating motifs in $abbv...\n";

        # pCRM blocks for this gene
        my $blocks = ${ $pcrms }{$abbv};

        $block_count += scalar @{ $blocks };

        $summary{$abbv} = sequence_motif_parse($abbv, $blocks, $motifs,
$params);

    }

    # rank motifs based on SCOPE's phi_score and coverage of pCRM
blocks in
    # the input sequence
    print STDERR "re-ranking SCOPE's motifs...\n";
    $motifs = motifs_rank($motifs, $block_count, scalar keys %{ $pcrms
}, $params);
    motifs_write($motifs);

    # splice out extra motifs
    plottable_motifs($motifs, $params);
```

```
    # plot pCRMs for each distinct gene
    for my $abbv (sort keys %{ $pcrms }) {
        print STDERR "plotting $abbv...\n";

        # pCRM blocks for this gene
        my $blocks = ${ $pcrms }{$abbv};

        $summary{$abbv} = sequence_plot($summary{$abbv}, $motifs,
$params);

    }


    motif_rank::phi_write(\%summary);

    # plot frequency distributions across all genes, e.g. if there are
    # 5 sequences with 10 pCRMs each, rank pCRMs from 1-50 instead of
    # 1-10 within each sequence.
    motif_rank::summary_plot(\%summary) if $params->{'summary'};
}



# --

# --
# -- motifs_write
# -- write re-ranked motifs to motifs.reranked.txt
sub motifs_write
{
    my ($motifs) = @_;

    my $filename = "motifs.rerank.txt";

    open RERANK, ">", $filename or die("couldn't create '$filename':
$!");


    printf RERANK "\%20s     \%8s     \%8s     \%8s     \%8s\n",
"motif", "sig", "p-cover", "g-cover", "pipeline";
    for (@{ $motifs }) {

        printf RERANK "\%20s     \%6f     \%6f     \%6f     \%6f\n",
            $_->{'motif'}, $_->{'sig_score'}, $_-
>pcrm_block_coverage_score, $_->pcrm_gene_coverage_score, $_-
>pipeline_score;

    }
    close RERANK;

}



# --
```

```
# --
# -- motifs_rank
# -- rank motifs by SCOPE's sig_score and their coverage score
sub motifs_rank
{
    my ($motifs, $block_count, $gene_count, $params) = @_;

    # set motif scores ...
    for my $motif (@{ $motifs }) {

        # how well does this motif cover pCRMs in the set?
        $motif->pcrm_block_coverage_score($motif->pcrm_block_count() /
$block_count);

        # how well does this motif cover genes in the set?
        $motif->pcrm_gene_coverage_score($motif->pcrm_gene_count() /
$gene_count);

        # combine sig and gene-coverage scores
        $motif->pipeline_score($params->{'score_m_sig'}, $params-
>{'score_m_pcrm'});
    }

    # filter out motifs without full gene coverage
    my @list = grep { $_->pcrm_gene_coverage_score() == 1 } @{ $motifs
};

    # ... and sort by combined (pipeline) score, high (best) to low
(worst)
    #@list = sort { $b->pipeline_score() <=> $a->pipeline_score() }
@list;

    return \@list;

}



# --

# -- sequence_motif_parse
# -- find motif positions in sequences
sub sequence_motif_parse
{
    my ($abbv, $blocks, $motifs, $params) = @_ ;

    my $seq = {};
    $seq->{'abbv'} = $abbv;
    $seq->{'blocks'} = $blocks;

    # initialize motif positions for this gene
    for my $motif (@{ $motifs }) { $motif->{'blocks'} = []; }

    # CRM blocks for this gene
```

```
    $seq->{'gene'} = motif_rank::gene($abbv);
    $seq->{'crm_blocks'} = $seq->{'gene'}-
>enhancers_by_regulator($params->{'regulator'});


    # count motif density in each block
    my $count_distribution = motif_rank::motif_count($seq->{'blocks'},
$motifs, $seq->{'gene'});

    for my $motif (@{ $motifs }) {

        $seq->{'motif_blocks'}{$motif->{'motif'}} = $motif->{'blocks'};

    }

    return $seq;

}



# --

# --
# -- sequence_plot
# -- plot pCRMs and motif positions in a sequence
sub sequence_plot
{
    my ($seq, $motifs, $params) = @_;

    print STDERR "parsing $seq->{'abbv'}...\n";

    my $count_distribution = plot_motif_count($seq->{'blocks'},
$motifs, $params);

    my $plot_fields = {};
    $plot_fields->{'abbv'}   = $seq->{'abbv'};
    $plot_fields->{'fields'} = [0..($seq->{'gene'}->length() - 1)];

    # prepare plots for CRMs, promoters and exons
    $plot_fields->{'crms'}      = motif_rank::profile($seq->{'gene'}-
>enhancers(), PLOT_HEIGHT, $seq->{'gene'}->length());
    $plot_fields->{'promoters'} = motif_rank::profile($seq->{'gene'}-
>promoters(), PLOT_HEIGHT, $seq->{'gene'}->length());
    $plot_fields->{'exons'}     = motif_rank::profile($seq->{'gene'}-
>exons(), PLOT_HEIGHT, $seq->{'gene'}->length());

    # for each block, get score-per-sequence-position from blocks
    $plot_fields->{'pcrms'}  = motif_rank::blocks2bins($seq-
>{'blocks'}, $count_distribution, $seq->{'gene'});

    # motif positions
    $plot_fields->{'y_min'} = 0; # this is reset by reference on the
next line
```

```
    $plot_fields->{'motifs'} = motif_rank::motif_blocks2bins($motifs,
$seq->{'gene'}, \$plot_fields->{'y_min'});
    $plot_fields->{'y_max'} = PLOT_HEIGHT;

    # density values
    $plot_fields->{'density_values'} = $count_distribution;

    # plot it
    print STDERR "\tplotting\n";
    motif_rank_plot::plot($plot_fields) if $params->{'plot'};

    # new FASTA file keeping only $params->{keep}% of the pCRMs
    print STDERR "\twriting new FASTA\n";
    motif_rank::fasta_write($seq->{'blocks'}, $seq->{'gene'}, $params)
if $params->{'keep'} > 0;

    my $scores = motif_rank::phi_score($seq->{'blocks'}, $seq-
>{'gene'});
    my $block_scores = motif_rank::phi_score_block($seq->{'blocks'},
$seq->{'gene'});

    # collect block density, profile information for summary plots
    return {
        'blocks'      => $seq->{'blocks'},
        'plot_fields' => $plot_fields,

        'phi_score'   => $scores->{'phi'}, # position-based phi-score
        'tp_score'    => $scores->{'tp'},  # position-based true
positives
        'fp_score'    => $scores->{'fp'},  # position-based false
positives

        'tp_score_block' => $block_scores->{'tp'}, # block-based crm-
pcrm overlaps
        'fp_score_block' => $block_scores->{'fp'}, # block-based crm-
pcrm misses

        };
}



# --

# --
# -- plot_motif_count
# -- new version of motif_count only counts $params->{'top'} motifs
sub plot_motif_count
{
    my ($blocks, $motifs, $params) = @_;

    my @counts;

    for my $block (@{ $blocks }) {
```

```
        $block->{'motif_count'} = 0;
        for my $motif (@{ $motifs }) {
            if ($block->{'motifs'}{$motif->{'motif'}}) {
                $block->{'motif_count'} += $block->{'motifs'}{$motif-
>{'motif'}};
            }
        }

        my $max = length $block->{'sequence'};

        # calculate density score
        $block->{'density'} = $block->{'motif_count'} / $max;

        print "$block->{'abbv'} $block->{'beg'} - $block->{'end'}\n";
        print "\ttotal motif-count: $block->{'motif_count'} (density
score: $block->{'density'})\n";
        print "\tdistinct motifs: ". scalar (keys %{ $block->{'motifs'}
}) . " of " . (scalar @{ $motifs }). "\n";

        push @counts, $block->{'density'};
    }

    return \@counts;

}







# --

# --
# -- plottable_motifs
# -- remove low-scoring motifs from a ranked list.
sub plottable_motifs
{
    my ($motifs, $params) = @_;


    # return top N motifs only, if N > 0
    if ($params->{'top'} > 0) {

        splice @{ $motifs }, $params->{'top'};

    }

    return $motifs;

}
```

```
# --

# --
# -- init
# -- parse command line options and truncate the output files.
sub init
{
    my $params = {
        "fasta"   => "",     # file containing pCRM sequences
        "motifs"  => "",     # file containing motifs found in $params-
>{fasta}
        "top"     => -1,     # how many motifs from $params->{motifs}
to use
        "keep"    => -1,     # percent of pCRM block to write to output
FASTA file
        "plot"    => util::TRUE,    # TRUE to generate pCRM plots,
ranked by density within each sequence
        "summary" => util::TRUE,    # TRUE to generate pCRM plots,
ranked by density with the set
        "raw"     => util::FALSE,   # TRUE to dump the plot data
structure to a file for fast printing later
        "block_rm_0"   => util::TRUE, # TRUE to always remove all pCRM
blocks with 0-density scores
        "block_rm_1"   => util::TRUE, # TRUE to always remove at least
one pCRM block
        "score_m_sig"  => 1,          # motif sig-score multiplier
        "score_m_pcrm" => 1,          # motif pcrm-coverage-score
multiplier
        "regulator"    => "",         # abbv of gene regulating this
set
        "help"         => util::FALSE,# TRUE to show help and quit
        };

    GetOptions(
        $params,
        'fasta=s',
        'motifs=s',
        'top=i',
        'keep=i',
        'plot!',
        'summary!',
        'raw!',
        'block_rm_0!',
        'block_rm_1!',
        'score_m_sig=f',
        'score_m_pcrm=f',
        'regulator=s',
        'help!',
        );


    # truncate OUTPUT_FASTA_FILE
    if ($params->{'keep'} > 0) {
        open FILE, '>', OUTPUT_FASTA_FILE;
        close FILE;
```

```
    }

    return $params;
}



sub raw_write
{
    my ($abbv, $plot_fields) = @_;

    open FILE, ">$abbv.raw";
    print FILE Dumper([$plot_fields]);
    close FILE;

}



# --

# --
# -- usage
# -- show how to use program, then exit
sub usage
{
    print <<EOT;
usage
    $0 --fasta <file> --motifs <file> [--top 1-9]

description
    calculate motif density in CRM blocks; plot as a heat map

ouput
    graphs displaying motifs density per block as a heat map
    also spits out .raw files with Data::Dumper for quick plotting
    later on; see motif_rank_raw.pl.

arguments
    fasta: file containing FASTA sequence blocks
    motifs: file containing motifs, one per line

optional arguments; default booleans in ALL CAPS
    top: number of motifs to use; default is all
    keep: % of pCRM blocks to write to FASTA file; default -1 (none)
    [no]plot: generate pCRM plots, ranked by density within each
sequence
    [no]summary: generate pCRM plots, ranked by density within the set
    [NO]raw: dump the plot data structure to a file for fast printing
later
    [no]block_rm_0: remove all pCRM blocks with 0-density scores
    [no]block_rm_1: remove at least one pCRM block

EOT
```

```
    exit 0;
}
```

motif_rank_raw.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-07-19, zak.burke@dartmouth.edu
# -- plots .raw files found in directory; defaults to current
directory.
# -- see motif_rank.pl for raw file data structure details.
# --

use strict;
use Getopt::Long;
use GD::Graph;
use GD::Graph::mixed;
use Statistics::Descriptive;
use Data::Dumper;
use lib "/home/zburke/bg/workspace/perl-lib";
use scan_parse;
use nazina_db;
use extract;
use gene;
use motif_rank_plot;


use constant TRUE  => 1;
use constant FALSE => 0;
use constant FASTA_LINE_LENGTH => 60;
use constant MOTIF_COUNT_BINS => 11;



main();

sub main
{
    my $params = {
        "dir" => ".",
        "help"  => FALSE,
        };

    GetOptions(
        $params,
        'dir=s',
        'help!',
        );

    usage() if $params->{'help'};

    # find *.raw files in requested directory ...
    opendir DIR, $params->{'dir'} or die("couldn't open dir $params-
>{'dir'}: $!");
    my @list = grep /raw$/, readdir DIR;
    closedir DIR;
```

```
    # ... and plot them
    for (@list) {

        motif_rank_plot::plot(raw_read("$params->{'dir'}/$_"));

    }

}



# --

# --
# -- raw_read
# -- read a file written by Data::Dumper back into a datastructure and
# -- return it.
sub raw_read
{
    my ($filepath) = @_;

    print "reading     $filepath...\n";

    open FILE, "<$filepath" or die("couldn't open $filepath: $!");
    my @lines = <FILE>;
    close FILE;

    my $VAR1;
    eval (join "", @lines);

    return $VAR1->[0];

}



# --

# --
# -- usage
# -- describe how this program works, then quit.
sub usage
{

    print <<EOT;

usage
    $0 --dir <directory>

description
    plots .raw files found in directory; defaults to current directory.
    see motif_rank.pl for raw file data structure details.

ouput
```

```
    graphs displaying motifs density per block as a heat map.

arguments
    directory: where to look for raw files; defaults to current
directory.

EOT

    exit 0;
}
```

phi_score_reader.pl

```
#!/usr/bin/perl -w

# --
# -- 2005-08-24, zak.burke@dartmouth.edu
# -- given a formatted phi-score file, pull out and print the
# -- score for the requested gene/round combination. prints -1
# -- if the requested tuple is not found.
# --
# -- fileformat:
#
#round    1
#    btd    0.214855286473715
#    eve    0.688316606703276
#    hb    0.197399783315276
#    kni    0.105945545471384
#    kr    0.417505924170616
#    otd    0.260181268882175
#round     2
#    btd    0.1875
#    eve    0.60302049622438
#    hb    0.209136822773186
#    kni    0.10976779888697
#    kr    0.42992222052768
#    otd    0.267719472767968
# ...
# round n
#     ...
# --
# --

use strict;


my ($filename, $round, $abbv);

($filename, $round, $abbv) = @ARGV;

my (%rounds, $i, $gene, $junk);

open FILE, "<", $filename or die ("couldn't open $filename: $!");
while (<FILE>) {
    chomp;

    if (/^round/) {
        ($junk, $i) = split /\s+/, $_;
        next;
    }

    # trim leading and trailing whitespace
    s/^\s+//g; s/\s+$//g;

    my ($gene, $phi) = split /\s+/, $_;
```

```
    if ($gene eq $abbv && $round eq "round_$i") {
        print "$phi";
        exit;
    }
}

print "-1";
```

pipeline.pl

```
#!/usr/bin/perl -w

# --
# -- 2005-08-14, zak.burke@dartmouth.edu
# -- iterate through SCOPE and motif_rank to winnow pCRMs
# --
# -- $Author: zburke $
# -- $Revision: 1.4 $
# -- $Date: 2005/10/21 03:13:38 $
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'BG_PERL_BIN'} = "/home/zburke/bg/workspace/pipeline" unless
$ENV{'BG_PERL_BIN'};

}

# point to script handling sequence parsing, motif-reranking and pCRM
ranking
use constant MOTIF_RANK => $ENV{'BG_PERL_BIN'} ."/motif_rank.pl";

# how many times to run SCOPE?
use constant ITERATION_COUNT => 5;

# default: number of motifs to use in ranking pCRM blocks
use constant MOTIF_COUNT => 4;

# default: % of pCRM blocks to keep after each round
use constant PCRM_THRESHOLD => 75;


# --
# -- nothing to configure below
# --


use strict;
use Getopt::Long;

use lib $ENV{'BG_PERL_LIB'};
use scan_parse;
use scope;
use util;


my $params = {
    "dir"  => "",             # directory containing seed FASTA files
from LWF
    "top"  => MOTIF_COUNT,    # number of motifs to use in ranking pCRM
blocks
```

```
    "keep" => PCRM_THRESHOLD, # % of pCRM blocks to keep
    "plot" => util::TRUE,     # whether to draw plots
    "score_m_sig"  => 1,      # motif sig-score multiplier
    "score_m_pcrm" => 1,      # motif pcrm-coverage-score multiplier
    "regulator"    => "",     # abbv of gene regulating a given set
                              # (derived from seed sequence name)
    "help" => util::FALSE,    # whether to show usage details
     };



main();


sub main
{
    GetOptions(
        $params, 'dir=s', 'top=i', 'keep=i',
        'score_m_sig=f', 'score_m_pcrm=f',
        'plot!',
        'help!',
        );

    $params->{'dir'} = "$ENV{'PWD'}/$params->{'dir'}" unless $params-
>{'dir'} =~ /^\//;

    # show help and exit, if requested
    usage() if $params->{'help'};

    # extract seed *.fa files from $params->{dir}, then iterate on them
    my $seeds = seed_read($params->{'dir'});
    for my $seed (@{ $seeds }) {
        $seed =~ /^([^\.]+)/;

        # abbreviation of gene regulating this set
        $params->{'regulator'} = $1;
        do_rounds($seed);
    }
}



# --

# --
# -- seed_read
# -- get *.fa files in given dir; dies on error.
sub seed_read
{
    my ($dir) = @_;

    opendir DIR, $dir or die("couldn't open $dir: $!");
    my @list = grep /\.fa$/, readdir DIR;
    closedir DIR;
```

```
    return \@list;
}



# --

# --
# -- do_rounds
# -- (SCOPE <=> motif_rank) loop for a given sequence
sub do_rounds
{
    my ($seed) = @_;

    mkdir $seed;
    chdir $seed;

    mkdir "round_0";
    `cp $params->{'dir'}/$seed round_0/sequence.fasta`;

    my $scope = scope->new();
    for my $i (1..ITERATION_COUNT) {

        `echo "round \t$i" >> phi_score.txt`;

        my $previous = "round_" . ($i-1);
        my $dir = "round_$i";
        mkdir $dir;
        chdir $dir;

        my $cwd = "$ENV{'PWD'}/$seed";
        my $fasta = "$cwd/$previous/sequence.fasta";

        # path to SCOPE's output file
        $scope->dir_out("$cwd/$dir/testResults/unknown_SCOPE.txt");
        $scope->run($fasta);

        motifs_write($scope);


        my $motif_rank = MOTIF_RANK . " " . (join " ", @{
motif_rank_flags($fasta, $params) });
        `$motif_rank > density_profile.txt`;
        `mv update.fa sequence.fasta`;

        chdir "..";

    }

    chdir "..";
}



# --
```

```
# --
# -- motifs_write
# -- write a list of words to the output file ./motifs.txt
sub motifs_write
{
    my ($scope) = @_;

    my $filename = "motifs.txt";

    open FILE, ">", $filename or die("couldn't create '$filename':
$!");
    for (@{ $scope->results() }) {
        print FILE scan_parse::line_write($_,
scope::motif_line_format());
    }
    close FILE;

}



# --

# --
# -- motif_rank_flags
# -- run-time params for motif_rank.pl
sub motif_rank_flags
{
    my ($fasta, $params) = @_;

    my @flags;

    push @flags,
        "--fasta=$fasta",
          "--motifs=motifs.txt",
        "--top=$params->{'top'}",
        "--keep=$params->{'keep'}",
        "--nosummary",
        "--noplot",
        "--score_m_sig=$params->{'score_m_sig'}",
        "--score_m_pcrm=$params->{'score_m_pcrm'}",
        "--regulator=$params->{'regulator'}",
        ;

    return \@flags;

}



# --

# --
# -- usage
```

```
# -- show how to call, then quit
sub usage
{
    print <<EOT;
description
    wrapper around motif_rank.pl for all FASTA files in a directory

usage
    $0 --dir=<dir>

arguments
    dir: directory containing seed FASTA files from LWF

optional arguments
             top: number of motifs to use in ranking pCRM blocks [4]
            keep: % of pCRM blocks to keep [75]
        [no]plot: whether to draw plots
     score_m_sig: motif sig-score multiplier [1]
    score_m_pcrm: motif pcrm-coverage-score multiplier [1]
        [NO]help: show this help

EOT

    exit 0;
}
```

pipeline_compare.pl

```
#!/usr/bin/perl  -w

# --
# -- parse pipeline output files into a summary HTML page.
# --
# -- within $param->{dir}, uses phi_score.txt and other dirs which
contain
# -- *.png and motifs.rerank.txt files which are parsed to generate
# -- round-based and summary statistics. all output is written to
STDOUT
# -- in HTML.
# --
# -- $Author: zburke $
# -- $Revision: 1.1 $
# -- $Date: 2005/10/23 00:47:47 $
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'BG_PERL_BIN'} = "/home/zburke/bg/workspace/pipeline" unless
$ENV{'BG_PERL_BIN'};
}


use strict;
use Getopt::Long;
use GD::Graph::lines;

use lib $ENV{'BG_PERL_LIB'};
use pipeline_util;


main();

sub main
{
    # name of directory to read
    my $params = {
        "dir" => "",
        "motif_count" => 7,
        "title" => $ENV{'PWD'},
        };

    GetOptions($params, 'dir=s', 'motif_count=i', 'title=s');

    # read all non-hidden files in the dir
    opendir DIR, $params->{'dir'} or die("couldn't open $params-
>{'dir'}");
    my @pairs = grep /90/, readdir DIR;
#    my @pairs = grep !/^\./, readdir DIR;
    closedir DIR;
```

```
    my $summary = {};
    my @regs;

    for my $pair_dir (@pairs) {
        next unless -d $pair_dir;

        $pair_dir =~ /([^-]+)-to-([^-]+)__*/;
        my $sig_multiplier = $1;
        my $coverage_multiplier = $2;

        opendir DIR, "$params->{'dir'}/$pair_dir" or die("couldn't open
$params->{'dir'}/$pair_dir");
        my @genes = grep /\.fa$/, readdir DIR;
        closedir DIR;
        for my $gene_dir (@genes) {

            push @regs, $gene_dir;
            print "$params->{'dir'}/$pair_dir/$gene_dir\n";

            my $run_summary =
pipeline_util::phi_score_reader_summary("$params-
>{'dir'}/$pair_dir/$gene_dir/phi_score.txt");
            $summary->{$pair_dir}->{$gene_dir} =
summarize($run_summary, "$params->{'dir'}/$pair_dir/$gene_dir");

            print "" . ("=" x 60) . "\n";
        }

    }

    for my $gene (@regs) {
        my $plot = summary_plot_prepare($summary, $gene);
        pipeline_util::summary_graph_bars($summary, $plot, "$gene.png",
"rho score");

    }
}



# --

# --
# -- summarize
# -- summarize results for a run
sub summarize
{
    my ($summary, $dirname) = @_;

    my ($phi_list, $rho_list, $rho2_list) = (undef, undef, undef);

    my $values = {};

    for my $round (sort keys %{ $summary }) {
```

```
        my ($rho_sum, $rho_avg, $rho2_sum, $rho2_avg, $phi_sum,
$phi_avg) = (0, 0, 0, 0);

        $rho_sum += $_ for map { "$summary->{$round}->{$_}->{rho}" }
sort keys %{ $summary->{$round} };
        $rho2_sum += $_ for map { "$summary->{$round}->{$_}->{rho2}" }
sort keys %{ $summary->{$round} };
        $phi_sum += $_ for map { "$summary->{$round}->{$_}->{phi}" }
sort keys %{ $summary->{$round} };

        $rho_avg = $rho_sum / (scalar keys %{ $summary->{$round} });
        push @{ $rho_list }, $rho_avg;

        $rho2_avg = $rho2_sum / (scalar keys %{ $summary->{$round} });
        push @{ $rho2_list }, $rho2_avg * 1000;

        $phi_avg = $phi_sum / (scalar keys %{ $summary->{$round} });
        push @{ $phi_list }, $phi_avg;

        print "$round\tphi: $phi_avg\trho: $rho_avg\trho2:
$rho2_avg\n";

        $values->{$round} = $rho_avg;

    }

    my $params = {
        "rho"  => $rho_list,
        "rho2" => $rho2_list,
        "phi"  => $phi_list,
    };

    my $plot = run_plot_prepare($params);
    pipeline_util::summary_graph($params, $plot,
"$dirname/trends_summary.png");

    return $rho_list;

}



# --

# --
# -- run_plot_prepare
# -- helper for summary_graph. mungs data into GD::graph compatible
format
# --
sub run_plot_prepare
{
    my ($params) = @_;

    my (@data, @types, @legend);
```

```
    for (keys %{ $params }) {
        push @data, $params->{$_};
        push @types, "lines";
        push @legend, $_;
    }

    # labels for rounds; must be first @data element
    my @rounds;
    for (my $i = 1; $i <= scalar @{$data[0]}; $i++) {
        push @rounds, $i;
    }
    unshift @data, \@rounds;

    return {
        "legend" => \@legend,
        "data"   => \@data,
        "types"  => \@types,
        };
}



# --

# --
# -- summary_plot_prepare
# -- $params->{$pair}->{$gene}->{$round} = score
sub summary_plot_prepare
{
    my ($params, $gene) = @_;

    my (@data, @types, @legend, @rounds);

    for (sort keys %{ $params }) {

        if ($params->{$_}->{$gene}) {
            for (my $i = 0; $i < (scalar @{ $params->{$_}->{$gene} });
$i++) {
                $data[$i] = [] unless $data[$i];
                push @{ $data[$i] }, ${ $params->{$_}->{$gene} }[$i];
            }

            push @types, "bars";
            push @rounds, $_;
        }
    }

    # labels for rounds; must be first @data element
    unshift @data, \@rounds;

    return {
        "data"   => \@data,
        "types"  => \@types,
        };
```

```
}
```

pipeline_pictures.pl

```
#!/usr/bin/perl  -w

# --
# -- parse pipeline output files into a summary HTML page.
# --
# -- within $param->{dir}, uses phi_score.txt and other dirs which
contain
# -- *.png and motifs.rerank.txt files which are parsed to generate
# -- round-based and summary statistics. all output is written to
STDOUT
# -- in HTML.
# --
# -- $Author: zburke $
# -- $Revision: 1.5 $
# -- $Date: 2005/10/23 00:47:47 $
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'BG_PERL_BIN'} = "/home/zburke/bg/workspace/pipeline" unless
$ENV{'BG_PERL_BIN'};
}


use strict;
use Getopt::Long;

use lib $ENV{'BG_PERL_LIB'};
use pipeline_util;


main();

sub main
{
    # name of directory to read
    my $params = {
        "dir" => "",
        "motif_count" => 7,
        "title" => $ENV{'PWD'},
        };

    GetOptions($params, 'dir=s', 'motif_count=i', 'title=s');

    # read all non-hidden files in the dir
    opendir DIR, $params->{'dir'} or die("couldn't open $params-
>{'dir'}");
    my @rounds = grep !/^\./, readdir DIR;
    closedir DIR;

    # print a table with a row for each round; each row contains a cell
```

```
    # for each gene
    print "<html><head><title>$params-
>{'title'}</title></head><body><table border='1'>\n";
    for my $round (@rounds) {
        print round($round, $params);
    }
    print "</table></body></html>\n";

    gene_print($params);

    my $plot = pipeline_util::summary_graph_data($params);
    pipeline_util::summary_graph($params, $plot, "trends.png");
}



# --

# --
# -- round
# -- gather information about each gene in the given round into a <tr>
# -- and return it
sub round
{
    my ($round, $params) = @_;
    my $content = "";

    # skip non-directory items, i.e. everything but round_0..round_n
    return unless -d "$params->{'dir'}/$round";

    # holder for calculating avg phi-score for this round
    my @phi_total;
    my @scores;

    $content .= "<tr valign='top'>\n";

    # find all image files in this round
    opendir DIR, "$params->{'dir'}/$round";
    my @files = grep /.*motif_profile.png$/, readdir DIR;
    closedir DIR;
    for my $file (@files) {
        my @words = split /\./, $file;
        my $abbv = $words[0];

        # phi-score for this round-gene pair
        my $score = pipeline_util::phi_score_reader("$params-
>{'dir'}/phi_score.txt", $round, $abbv);
        push @phi_total, $score->{'phi'};
        push @scores, $score;

        my $filename = "$params->{'dir'}/$round/motifs.rerank.txt";
        my $motifs = join ", ", @{
pipeline_util::motifs_reader($filename, $params) };

        $content .= <<EOT;
```

```
<td align='center'>
    <h3>phi: $score->{'phi'}</h3>
    <h3>tp/fp: $score->{'tp'}/$score->{'fp'} ($score->{'tp_fp'})</h3>
    <h3>rho: $score->{'rho'}</h3>
    <h4>$motifs</h4>
    <img src='$round/$abbv.legend.png'>   <img
src='$round/$file' />
</td>
EOT

        my $details = {};
        $details->{'phi'}    = $score->{'phi'};
        $details->{'motifs'} = join "<br />", split ", ", $motifs;
        $details->{'img'}    = "$round/$file";
        $details->{'legend'} = "$round/$abbv.legend.png";
        $details->{'score'}  = $score;

        $params->{'genes'}->{$abbv}->{$round} = $details;

    }

    # print phi-score summary for this round
    my $phi_avg = 0.0;
    $content .= score_summary(\@scores, \$phi_avg);
    if ($phi_avg) {
        $params->{'rounds'}->{$round} = $phi_avg;
    }

    $content .= "</tr>\n";

    return $content;
}



# --

# --
# -- phi_summary
# -- return phi score summary in a TD, and store it in $$phi_avg as a
side-effect
sub score_summary
{
    my ($scores, $phi_avg) = @_;

    my $content = "";
    my ($tp_total, $fp_total) = (0, 0);

    $$phi_avg = pipeline_util::score_average($scores, "phi");
    $tp_total += $_ for map { $_->{'tp'} }  @{ $scores };
    $fp_total += $_ for map { $_->{'fp'} }  @{ $scores };

    my $ratio = 0.0;
    $ratio = ($tp_total / $fp_total) if ($fp_total > 0);
```

```
    my $rho_ratio = 0.0;
    $rho_ratio = ($tp_total / ($tp_total + $fp_total)) if ($fp_total >
0);


    $content .= "<td>\n";
    $content .= "<h2>avg. phi score: $$phi_avg</h2>\n";
    $content .= "<h2>tp/fp: $tp_total / $fp_total ($ratio)</h2>\n";
    $content .= "<h2>rho: $rho_ratio</h2>\n";
    $content .= "</td>\n";

    return $content;
}



# --

# --
# -- gene_print
# -- print rounds for each specific gene into a file named $gene.html
sub gene_print
{
    my ($params) = @_;

    for (sort keys %{ $params->{'genes'} }) {
        my $details = $params->{'genes'}->{$_};

        open GENE, ">", "$_.html";
        print GENE "$params->{'title'}";
        print GENE "<table>\n";
        for my $round_key (sort keys %{ $details } ) {
            my $round = $details->{$round_key};
            print GENE <<EOT;
<tr valign='top'>
    <td><img src="$round->{'legend'}" /></td>
    <td><img src="$round->{'img'}" /></td>
    <td>phi: $round->{'phi'}<br />$round->{'motifs'}</td>
</tr>
EOT
        }
        print GENE "</table>\n";
        close GENE;
    }

}
```

regulators.pl

```
#!/usr/bin/perl

# --
# -- for each regulatory element, find all the elements it regulates
# -- and write them to a FASTA file named for the regulator.
# --
# -- $Author: zburke $
# -- $Revision: 1.5 $
# -- $Date: 2005/10/23 00:47:47 $
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'BG_PERL_BIN'} = "/home/zburke/bg/workspace/pipeline" unless
$ENV{'BG_PERL_BIN'};
}


use strict;
use Getopt::Long;

use lib $ENV{'BG_PERL_LIB'};
use nazina_db;
use util;

main();

sub main
{
    my $dao = nazina_db->new();

    my $list = $dao->regulatory_elements();

    for my $id (sort {$list->{$a}->{'name'} cmp $list->{$b}->{'name'} }
keys %{ $list }) {
        my $r = $list->{$id};
        my $filename = "$r->{'abbv'}.fa";
        open FILE, ">$filename" or die("couldn't create $filename:
$!");
        print "$r->{'name'}\n";
        for my $element (sort { $a->{gene}->{abbv} cmp $b->{gene}-
>{abbv} } @{ $r->{'elements'} }) {
            print "\t$element->{gene}->{name} ($element->{name})\n";
            print FILE util::fasta(
                $element->{'sequence'},
                "$element->{'gene'}->{'name'} $element-
>{'position_start'}-$element->{'position_end'}"
                );
        }
    }
}
```

summary.pl

```
#!/usr/local/bin/perl

# --
# -- 2005-08-07, zak.burke@dartmouth.edu
# -- given directories with summary motif density plots (i.e. motif
density
# -- is colored based on rank within all pCRMs in the set, not just
within
# -- the sequence), create an HTML page that shows all these summaries.
# --
# --

use strict;
use Getopt::Long;

use constant TRUE  => 1;
use constant FALSE => 0;


main();

sub main
{
    my $params = {
        "genes" => [],
        "numbers" => [],
        "help"  => FALSE,
        };

    GetOptions(
        $params,
        'genes=s@',
        'numbers=s@',
        'help!',
        );
    $params->{'genes'}   = [split ",", (join ",", @{ $params->{'genes'}
} )];
    $params->{'numbers'} = [split ",", (join ",", @{ $params-
>{'numbers'} } )];

    for my $gene (@{ $params->{'genes'} }) {


        my $dir_name = "$gene.$params->{'numbers'}->[1]";
        opendir DIR, $dir_name or die("couldn't open $dir_name: $!");
        my @regulators = grep /summary\.motif_profile\.png$/, readdir
DIR;
        closedir DIR;

        my %summary;
        for my $number (@{ $params->{'numbers'} }) {
            open FILE, ">$gene.$number/summary.html";
```

```
            for my $reg (sort @regulators) {
                print FILE "<img src='$reg'><br />\n";
                push @{ $summary{$reg} }, "<img
src='$gene.$number/$reg'>";
            }
            close FILE;
        }

        open FILE, ">$gene.summary.html";
        print FILE "<table>\n";
        for my $row (sort keys %summary) {
            print FILE "<tr>\n\t<td>". (join "</td>\n\t<td>", @{
$summary{$row} } ) . "</td>\n</tr>\n";

        }
        print FILE "</table>";
        close FILE;
    }
}
```

Appendix 4 Library Code

Perl Files

extract.pm

```
package extract;

# --
# -- 2005-06-28, zak.burke@dartmouth.edu
# -- based on /lwf/analysis/hm_parse_best.pl, extract certain tuples
# -- from the output file created by hm_parse.pl.
# --

BEGIN
{
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
}

use strict;

use lib $ENV{'BG_PERL_LIB'};
use scan_parse;

use constant TRUE  => 1;
use constant FALSE => 0;


# --

# --
# -- extract
# -- parse an output file from hm.pl
sub extract
{
    my ($file, $conditions) = @_;

    my @lines = ();

    # parser for fields embedded in filename
    my $field_parser = scan_parse::field_parser('gene');

    # parse lines into records; grab first line to determine format
    open FILE, $file or die("couldn't open $file: $!");
    my $header = <FILE>;
```

```
    my $line_parser  = scan_parse::parser($header);
    while (<FILE>) {
        chomp;
        # skip header lines
        next unless /^[0-9]/;

        my $line = scan_parse::field_parse($_, $line_parser,
$field_parser);

        # keep lines satisfying a condition-set
        push @lines, $line if condition_matches($conditions, $line);

    }
    close FILE;

    return \@lines;

}



# --

# --
# -- condition_matches
# -- returns TRUE if the given line matches at least one of the
# -- given condition-sets
sub condition_matches
{
    # conditions: array-ref of condition-sets
    # line: list of values to match against a condition-set
    my ($conditions, $line) = @_;

    # assume there is no match;
    my $match = 0;

    for my $condition (@{ $conditions }) {

        # how many conditions are there in this condition-set?
        my $match_required = keys %{ $condition };

        # how many conditions matched so far?
        my $match_count = 0;

        for my $field (keys %{ $condition }) {

            $match_count++ if ($condition->{$field} == $line-
>{$field});

        }

        # matched all parameters in a condition-set;
        return TRUE if ($match_count == $match_required);
```

```
    }

    # line failed to match a condition-set
    return FALSE;

}

return 1;
```

gene.pm

```
package gene;

# --
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.4 $
# -- $Date: 2005/10/20 05:19:04 $
# --



sub new
{
    my $class = shift;


    # capture gene_id, name, abbv, sequence
    my %input = @_;
    my $self = \%input;


    bless $self, $class;


    return $self;
}



# --

# --
# -- return ratio of cumulative CRM length to total sequence length
sub s2n
{
    my $self = shift;

    my $seq_len = length($self->{'sequence'});

    # can't be any CRMs if we haven't any sequence
    return 0 unless $seq_len;

    if (! $self->{'enhancers'}) {

        $self->{'enhancers'} = $self->enhancers();

    }

    my $crm_len = 0;
    for (@{ $self->{'enhancers'} }) {
```

```
        $crm_len += ($_->{'end'} - $_->{'beg'}) + 1;

    }

    return ($crm_len / $seq_len);

}



# --

# --
# -- enhancers
# -- return list of enhancer elements for this gene
sub enhancers
{
    my $self = shift;

    return $self->{'enhancers'} if $self->{'enhancers'};

    $self->{'enhancers'} = $self->{'dao'}->gene_enhancers($self-
>{'gene_id'});

    return $self->{'enhancers'};

}



# --

# --
# -- enhancers_by_regulator
# -- return the enhancer elements governed by the given regulator
sub enhancers_by_regulator
{
    my $self = shift;
    my ($regulator) = @_;

    my @list;
    for (@{ $self->enhancers() }) {
        for my $reg (@{ $_->{'regulators'} }) {
            if ($reg->{'abbv'} eq $regulator) {
                push @list, $_;
            }
        }
    }

    return \@list;

}
```

```
# --

# --
# -- promoters
# -- return list of promoter elements for this gene
sub promoters
{
    my $self = shift;

    return $self->{'promoters'} if $self->{'promoters'};

    $self->{'promoters'} = $self->{'dao'}->gene_promoters($self-
>{'gene_id'});

    return $self->{'promoters'};

}



# --

# --
# -- exons
# -- return list of exon elements for this gene
sub exons
{
    my $self = shift;

    return $self->{'exons'} if $self->{'exons'};

    $self->{'exons'} = $self->{'dao'}->gene_exons($self->{'gene_id'});

    return $self->{'exons'};

}



# --

# --
# -- length
# -- length of the sequence
sub length
{
    my $self = shift;
    if (! $self->{'length'}) {
        $self->{'length'} = length $self->{'sequence'};
    }
    return $self->{'length'}
}

1;
```

lwf_plot.pm

```
package lwf_plot;

# --
# -- 2005-12-14, zak.burke@dartmouth.edu
# -- helper routines for LWF's CRM-pCRM plot scripts,
# -- e.g. lwf/analysis/plot_crm2.pl, overlap.pl
# --

BEGIN {
        $ENV{'BG_PERL_LIB'}   = "/home/zburke/bg/workspace/perl-lib"
unless $ENV{'BG_PERL_LIB'};
        $ENV{'HOME_PERL_LIB'} = "." unless $ENV{'HOME_PERL_LIB'};
}

use strict;

use lib $ENV{'BG_PERL_LIB'};
use gene;
use util;


# --

# --
# -- stats_calculate
# -- given true-positive, false-positive, true-negative and false-
negative
# -- counts, return a string indicating the pCRM coverage of the entire
# -- sequence and the pCRM coverage of true CRMs.
sub stats_calculate
{
    my ($stats) = @_;

    my $size = $stats->{'tp'} + $stats->{'fp'} + $stats->{'tn'} +
$stats->{'fn'};

    my $total_coverage = ($stats->{'tp'} + $stats->{'fp'}) / $size;
    my $crm_coverage   = $stats->{'tp'} / ($stats->{'tp'} + $stats-
>{'fn'});
    my $phi_score      = util::phi_score($stats->{'tp'}, $stats-
>{'fp'}, $stats->{'fn'});
    my $hm             = util::harmonic_mean(
                util::recall($stats->{'tp'}, $stats->{'fn'}),
                util::precision($stats->{'tp'}, $stats->{'fp'}));

    return sprintf("t-cover %0.3f; crm-cover: %0.3f; hm: %0.3f; phi:
%0.3f", $total_coverage, $crm_coverage, $hm, $phi_score);

}
```

```
# --

# --
# -- threshold_normalize
# -- normalize an array such that 0 <= values <= 1
sub threshold_normalize
{
    my ($list, $min, $max) = @_;

    my (@threshold, $adjust);

    # how much to bump up scores to get them all in the range 0 - $max
    $adjust = 0 - $min;
    $max += $adjust;

    for (@{ $list }) {

        push @threshold, (($_ + $adjust) / $max);

    }


    return \@threshold;
}



# --

# --
# -- pip
# -- extract percent-identity-profile for D. Melangaster and D.
Pseudoobscura
# --
# -- return
# --     array-ref of percent identity for each position from 1-16000
# -- params
# --     $abbv - abbreviation of gene to extract profile for
# --
sub pip
{
    my ($abbv, $pip_file) = @_;

    open FILE, $pip_file;
    my $line = <FILE>;
    chomp $line;
    my @genes = split /\t/, $line;
    my @list = ();
    my $pos = 0;
    while (<FILE>) {
        chomp;
        my @scores = split /\t/;
        for (my $i = 0; $i < scalar @scores; $i++) {
            push @{ $list[$i] }, ($scores[$i] / 100);
        }
```

```
        last if $pos++ > 16000;
    }
    close FILE;

    for (my $i = 0; $i < scalar @genes; $i++) {
        next unless $genes[$i] eq $abbv;

        return $list[$i];
    }

}

1;
```

motif.pm

```
package motif;

# --
# -- motif data bit bucket
# --
# --
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.4 $
# -- $Date: 2005/10/23 01:10:06 $
# --


use strict;

sub new
{
    my $class = shift;

    my $self = {};

    bless($self, $class);


    my %input = @_;

    # letters in the motif
    $self->{'motif'}     = $input{'motif'} || "";

    # blocks where this motif occurs anywhere in a sequence, i.e. not
    # necessarily pCRM blocks. in this case, a block is just an item
with
    # begin and end positions indicating the subsequence this motif
matches.
    $self->{'blocks'}    = $input{'blocks'} || [];

    # count of pCRM blocks where this motif occurs
    $self->{'pcrm_block_count'} = $input{'pcrm_block_count'} || 0;

    # SCOPE's sig score for this motif
    $self->{'sig_score'} = $input{'sig_score'} || 0;



    # derived properties
    if ($self->{'motif'}) {

        # number of letters in $self->{'motif'}
        $self->{'length'}    = length $self->{'motif'};

        # regular expression matching $self->{'letters'}
```

```
        $self->re();

    }

    return $self;
}



# -- static regular expression building blocks for expanding motifs
# -- defined in the IUPAC alphabet
my %iupac = (
    'a' => "[a]",
    'c' => "[c]",
    'g' => "[g]",
    't' => "[t]",
    'm' => "[ac]",
    'r' => "[ag]",
    'w' => "[at]",
    's' => "[cg]",
    'y' => "[ct]",
    'k' => "[gt]",
    'v' => "[acg]",
    'h' => "[act]",
    'd' => "[agt]",
    'b' => "[cgt]",
    'n' => "[acgt]",
);



# --

# --
# -- re
# -- generate a regular expression matching the expanded iupac alphabet
# -- used to represent the motif
sub re
{
    my $self = shift;

    if (! $self->{'re'}) {

        my @letters = ();
        my $re = "";

        @letters = split "", $self->{'motif'};
        for (@letters) {

            $re .= $iupac{$_};

        }

        $self->{'re'} = $re;
    }
```

```
    return $self->{'re'};

}



# --

# --
# -- pipeline_score
# --
sub pipeline_score
{
    my $self = shift;

    my ($sig_multiplier, $coverage_multiplier) = @_;

    # force recalculation if we have new multipliers
    if ($sig_multiplier || $coverage_multiplier) {
        $self->{'sig_multiplier'} = $sig_multiplier || $self-
>{'sig_multiplier'} || 1;
        $self->{'coverage_multiplier'} = $coverage_multiplier || $self-
>{'coverage_multiplier'} || 1;

        $self->{'pipeline_score'} = undef;
    }

    return $self->{'pipeline_score'} if $self->{'pipeline_score'};

    $self->{'pipeline_score'} = ($self->{'sig_multiplier'} * $self-
>{'sig_score'})
        + ($self->{'coverage_multiplier'} * $self-
>pcrm_gene_coverage_score());

    return $self->{'pipeline_score'}
}



# --

# --
# -- pcrm_block_coverage_score
# -- set (if necessary) and return the block coverage score. this is
# -- calculated as (pCRM blocks containing this motif / total pCRM
blocks)
sub pcrm_block_coverage_score
{
    my $self = shift;

    my $score = shift;
    $self->{'pcrm_block_coverage_score'} = $score if $score;
    return $self->{'pcrm_block_coverage_score'};
}
```

```
# --

# --
# -- pcrm_gene_coverage_score
# -- set (if necessary) and return the gene coverage score. this is
# -- calculated as (sequences containing this motif / total sequence
count)
sub pcrm_gene_coverage_score
{
    my $self = shift;

    my $score = shift;
    $self->{'pcrm_gene_coverage_score'} = $score if $score;
    return $self->{'pcrm_gene_coverage_score'};
}



# --

# --
# -- pcrm_block_count
# -- how many pCRM blocks contain this motif?
sub pcrm_block_count
{
    my $self = shift;

    if (! $self->{'pcrm_block_count'}) {

        $self->{'pcrm_block_count'} = 0;
        for my $abbv (keys %{ $self->{'pcrm_blocks'} }) {

            $self->{'pcrm_block_count'} += scalar (keys %{ $self-
>{'pcrm_blocks'}->{$abbv} } );

        }
    }

    return $self->{'pcrm_block_count'};
}



# --

# --
# -- pcrm_gene_count
# -- how many gene sequences conain this motif?
sub pcrm_gene_count
{
    my $self = shift;
```

```
    if (! $self->{'pcrm_gene_count'}) {

        $self->{'pcrm_gene_count'} = 0;
        for my $abbv (keys %{ $self->{'pcrm_blocks'} }) {

            $self->{'pcrm_gene_count'}++ if scalar (keys %{ $self-
>{'pcrm_blocks'}->{$abbv} } );

        }
    }

    return $self->{'pcrm_gene_count'};

}

1;

__END__
```

motif_rank.pm

```
package motif_rank;

# --
# -- given a list of sequences and a list of motifs, draw a heat map of
# -- motif density for each sequence.
# --
# -- graphing routines based on overlap.pl
# --
# --
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.2 $
# -- $Date: 2005/10/20 05:22:44 $
# --


BEGIN {
    # output FASTA filename
    use constant OUTPUT_FASTA_FILE => "update.fa";

    # output phi-score filename
    use constant OUTPUT_PHI_SCORE_FILE => "../phi_score.txt";


    $ENV{'BG_PERL_LIB'}   = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'HOME_PERL_LIB'} = "." unless $ENV{'HOME_PERL_LIB'};
}


# --
# -- nothing to configure below
# --

use strict;

use Getopt::Long;
use Data::Dumper;

use lib $ENV{'HOME_PERL_LIB'};
use GD::Graph;
use GD::Graph::mixed;
use Statistics::Descriptive;

use lib $ENV{'BG_PERL_LIB'};
use scan_parse;
use nazina_db;
use extract;
use gene;
use scope;
use pcrm_block;
```

```
use motif;
use motif_rank_plot;
use util;



# characters per line in output FASTA file
use constant FASTA_LINE_LENGTH => 60;

# DEPRECATED; distribute pCRMs by density in this many groups
use constant MOTIF_COUNT_BINS => 6;

# height of a true CRM
use constant PLOT_HEIGHT => 0.4;

# distance between motif-position plots
use constant PLOT_INCREMENT => 0.1;




# --

# --
# -- summary_plot
# -- plot pCRMs ranked by density across all sequences in the set, e.g.
if
# -- there are 5 sequences with 10 pCRMs each, rank pCRMs from 1-50
instead
# -- of 1-10 within each sequence.
sub summary_plot
{
    my ($summary) = @_;

    # get array containing all density-profiles
    my $list = summary_plot_prepare($summary);

    for my $abbv (sort keys %{ $summary }) {
        print STDERR "plotting $abbv summary\n";
        my $set = ${ $summary }{$abbv};

        # initialize plot_fields to zap some of the arrays set above
        $set->{'plot_fields'}->{'abbv'} = "$abbv.summary";
        $set->{'plot_fields'}->{'pcrms'}  = [];

        my @empty_pcrm;
        for (0..($#{$set->{'plot_fields'}->{'fields'}})) {
            push @empty_pcrm, 0;
        }

        for my $block ( sort {$b->{'density'} <=> $a->{'density'}} @{
$list } ) {

            if ($block->{'abbv'} eq $abbv) {
```

```
                push @{ $set->{'plot_fields'}->{'pcrms'} }, $block-
>{'plot'};
            } else {
                push @{ $set->{'plot_fields'}->{'pcrms'} },
[@empty_pcrm];
            }

        }

        # plot it
        motif_rank_plot::plot($set->{'plot_fields'});

    }

    summary_plot_legend($list);
}



# --

# --
# -- summary_plot_prepare
# -- create an array of pCRMs across all sequences in a set
sub summary_plot_prepare
{
    my ($summary) = @_;

    my @list;
    for my $abbv (keys %{ $summary }) {

        my $set = ${ $summary }{$abbv};

        for my $block (sort {$b->{'density'} <=> $a->{'density'} } @{
$set->{'blocks'} }) {
            push @list, {
                'abbv' => $abbv,
                'density' => $block->{'density'},
                'plot' => $block->{'plot'},
                };
        }
    }

    return \@list;

}



# --

# --
# -- summary_plot_legend
# -- plot an HTML table that displays a density legend for all pCRMs in
the set
```

```
sub summary_plot_legend
{
    my ($list) = @_;

    my $divisor = $#{ $list } || 1;
    my $increment = int(255 / $divisor);

    open FILE, ">legend.html" or die ("couldn't open legend.html");
    print FILE "<table cellpadding='3' cellspacing='1' border='0'>";

    my $i = 0;
    for my $block ( sort {$b->{'density'} <=> $a->{'density'}} @{ $list
} ) {
        my $color = GD::Graph::colour::rgb2hex(255, $i * $increment, $i
* $increment);
        print FILE "<tr><td bgcolor='$color'><span style='font-size:
10px;'>$block->{'density'}</span></td></tr>\n";

        $i++;
    }

    print FILE "</table>\n";
    close FILE;

}



# --

# --
# -- blocks2bins
# -- convert pCRM blocks into plottable bins.
sub blocks2bins
{
    my ($blocks, $count_distribution, $gene) = @_;

    my $plot_height = PLOT_HEIGHT / 2;
    my $sequence_length = length $gene->{'sequence'};

    my (@pcrm_blocks, @zero_blocks);
    for my $block (sort {$b->{'density'} <=> $a->{'density'} } @{
$blocks }) {

        # handle all blocks with density == 0 in one unit
        if ($block->{'density'} == 0) {
            push @zero_blocks, $block;
            next;
        }

        my $profile = profile([$block], $plot_height,
$sequence_length);

        $block->{'plot'} = $profile;
```

```
        push @pcrm_blocks, $profile;

    }

    # 0-density handler
    if (scalar @zero_blocks) {

        my $profile = profile(\@zero_blocks, $plot_height,
$sequence_length);
        for my $block (@zero_blocks) {
            $block->{'plot'} = $profile;
        }
        push @pcrm_blocks, $profile;

    }

    return \@pcrm_blocks;

}



# --

# --
# -- motif_blocks2bins
# -- convert a list of motifs and positions where they occur into
plottable
# -- objects
sub motif_blocks2bins
{
    my ($motifs, $gene, $y_min) = @_;

    my @plotable;

    my $height = -0.1;

    for my $motif (@{ $motifs }) {

        push @plotable, profile($motif->{'blocks' }, $height , length
$gene->{'sequence'});

        $height -= PLOT_INCREMENT;

    }

    ${ $y_min } = $height;

    return \@plotable;
}
```

```
# --

# --
# -- fasta_read
# -- read fasta data into an array of hash-refs: { abbv, beg, end,
sequence }
sub fasta_read
{
    my ($params) = @_;

    my (@blocks, $block, @lines);
    open FILE, $params->{'fasta'} or die("couldn't open $params-
>{'fasta'}: $!");
    while (<FILE>) {
        chomp;
        if (/^>/) {
            if ($block) {
                $block->{'sequence'} = lc join "", @lines;
                push @blocks, $block;
            }
            $block = {};
            @lines = ();

            my ($abbv, $beg, $end) = ($_ =~ />([^\s]+) ([0-9]+) - ([0-
9]+)/);
            $block = {
                'abbv' => $abbv,
                'beg'  => $beg,
                'end'  => $end};

            next;
        }

        push @lines, $_;

    }
    if ($block) {
        $block->{'sequence'} = join "", @lines;
        push @blocks, $block;
    }

    close FILE;

    my %genes;
    for my $block (@blocks) {
        push @{ $genes{$block->{'abbv'}} }, $block;
    }

    return \%genes;
}



# --
```

```
# --
# -- fasta_write
# -- slice off the pCRM blocks we choose not to keep, then print FASTA
# -- sequences for the remaining blocks to the file OUTPUT_FASTA_FILE.
# -  Because one gene may have multiple regulators, update.fa is opened
# -- for APPENDING. This file was truncated in init();
sub fasta_write
{
    my ($blocks, $gene, $params) = @_;

    my $sorted = fasta_write_trim($blocks, $params);

#    # sort blocks so we can pull off the top N percent
#    my @sorted = sort {$a->{'density'} <=> $b->{'density'} } @{
$blocks };

#    # $params->{'keep'} specifies the % of blocks to keep
#    # always remove at least one block
#    my $rm_count = (scalar @sorted / (100 / (100 - $params-
>{'keep'})));
#    $rm_count = ($rm_count < 1 ? 1 : $rm_count);
#    splice @sorted, 0, $rm_count;

    open FILE, '>>', OUTPUT_FASTA_FILE;
    for my $block (sort {$a->{'beg'} <=> $b->{'beg'} } @${ sorted }) {

        my $sequence = lc substr($gene->{'sequence'}, $block->{'beg'},
($block->{'end'} - $block->{'beg'}) + 1);
        print FILE ">$gene->{'abbv'} $block->{'beg'} - $block-
>{'end'}\n";

        for (my $i = 0; $i < length($sequence); $i+= FASTA_LINE_LENGTH)
        {
            print FILE substr($sequence, $i, FASTA_LINE_LENGTH). "\n";
            last if (substr($sequence, $i, FASTA_LINE_LENGTH) eq "");
        }
        print FILE "\n";

    }
    close FILE;
}


sub fasta_write_trim
{
    my ($blocks, $params) = @_;


    # sort blocks so we can pull off the top N percent
    my @sorted = sort {$a->{'density'} <=> $b->{'density'} } @{ $blocks
};

    # $params->{'keep'} specifies the % of blocks to keep; convert that
    # into a number of blocks to remove.
```

```
    my $rm_count = (scalar @sorted / (100 / (100 - $params-
>{'keep'})));

    # always remove at least one block
    if ($params->{'block_rm_1'}) {
        $rm_count = ($rm_count < 1 ? 1 : $rm_count);
    }

    # count of blocks removed so far
    my $removed = 0;

    # if there are blocks with 0-density, remove *all* of them, then
see
    # what is left to remove
    if ($params->{'block_rm_0'}) {
        while (@sorted) {
            if (0 == $sorted[0]->{'density'}) {
                shift @sorted;
                $removed++;
            } else {
                last;
            }
        }
    }

    if ($removed < $rm_count) {
        splice @sorted, 0, ($rm_count - $removed);
    }

    return \@sorted;

}



# --

# --
# -- gene_select
# -- return list of genes to plot, as specified on the command line
sub gene_select
{
    my (%genes) = @_;

    # no genes specified so scan 'em all ...
    return %genes if ($#ARGV == 0);

    # ... or scan named genes only
    my %list;
    for (my $i = 1; $i < scalar @ARGV; $i++) {
        $list{$ARGV[$i]} = $genes{$ARGV[$i]};
    }

    return %list;
}
```

```
# --

# --
# -- motif_read
# -- read motifs from a file, one per line
sub motif_read
{
    my ($params) = @_;

    my (@motifs);

    open FILE, "<", $params->{'motifs'} or die("couldn't open $params-
>{'motifs'}: $!");
    while (<FILE>) {
        chomp;

        my $motif = motif->new(%{ scan_parse::line_read($_,
scope::motif_line_format()) });

        push @motifs, $motif;

    }
    close FILE;

#    # return top N motifs only, if N > 0
#    if ($params->{'top'} > 0) {
#
#        splice @motifs, $params->{'top'};
#
#    }

    return \@motifs;
}



# --

# --
# -- motif_count
# -- count the number of motifs in each pCRM block, then calculate
density
# -- scores (count / block-length) in order to normalize the block-
scores.
# -- without normalization, we would inadvertantly reward a long block
with
# -- low density and could penalize a short block with high density.
# --
# -- the blocks array is passed by reference so density scores are
added to
# -- the block objects as a side effect. for convenience, an array of
the
```

```
# -- density scores is also returned. obviously, we could generate this
# -- later by looping over the blocks, but it's convenient to create
here.
sub motif_count
{
    my ($blocks, $motifs, $gene) = @_;

    my (@counts);

    my $i_max = length $gene->{'sequence'};

    for (my $i = 0; $i < $i_max; $i++) {

        for my $motif (@{ $motifs }) {

            my $word = substr $gene->{'sequence'}, $i, $motif-
>{'length'};

            my $re = $motif->{'re'};
            if ($word =~ /^$re$/ ) {

                # count motif in block
                for my $block (@{ $blocks }) {
                    if ($i >= $block->{'beg'} && ($i + $motif-
>{'length'}) <= $block->{'end'}) {
                        $block->{'motifs'}{$motif->{'motif'}}++;
                        $block->{'motif_count'}++;

                        $motif->{'pcrm_blocks'}->{$gene->{'abbv'}}-
>{$block->{'beg'}} = 1;

                        last;  # can't be in more than one block at a
time
                    }
                }

                # log position of this motif, which may not be within a
pCRM
                push @{ $motif->{'blocks'} }, {'beg' => $i, 'end' =>
($i + $motif->{'length'})};
            }

        }

    }


    for my $block (@{ $blocks }) {
        $block->{'motif_count'} = $block->{'motif_count'} || 0;
        my $max = length $block->{'sequence'};

        # calculate density score
        $block->{'density'} = $block->{'motif_count'} / $max;

        print "$block->{'abbv'} $block->{'beg'} - $block->{'end'}\n";
```

```
        print "\ttotal motif-count: $block->{'motif_count'} (density
score: $block->{'density'})\n";
        print "\tdistinct motifs: ". scalar (keys %{ $block->{'motifs'}
}) . " of " . (scalar @{ $motifs }). "\n";

        push @counts, $block->{'density'};
    }

    return \@counts;
}



# --

# --
# -- gene
# -- return a gene object, given its abbreviation
sub gene
{
    my ($abbv) = @_;
    my $dao = nazina_db->new();
    my $list = $dao->genes_by_abbv();

    return $dao->gene($list->{$abbv}->{'gene_id'});

}



# --

# --
# -- threshold_normalize
# -- normalize an array such that 0 <= values <= 1
sub threshold_normalize
{
    my ($list, $min, $max) = @_;

    my (@threshold, $adjust);

    # how much to bump up scores to get them all in the range 0 - $max
    $adjust = 0 - $min;
    $max += $adjust;

    for (@{ $list }) {

        push @threshold, (($_ + $adjust) / $max);
    }


    return \@threshold;
}
```

```
# --

# --
# -- profile
# -- build plot-data arrays from CRM, pCRM sequence data.
sub profile
{
    my ($blocks, $height, $max) = @_;

    my @list;

    for my $block (@{ $blocks }) {

        for (my $i = $block->{'beg'}; $i < $block->{'end'}; $i++) {

            $list[$i] = $height;
        }

    }


    $list[$max] = 0 unless $list[$max];

    return \@list;

}



# --

# --
# -- phi_score
# -- calculate a phi_score given the pCRMs and CRMs in this sequence
sub phi_score
{
    my ($pcrm_blocks, $gene) = @_;

    my ($tp, $fp, $fn, $in_pcrm, $in_crm) = (0, 0, 0);

    my $crm_blocks = $gene->enhancers();
    my $max = length $gene->{'sequence'};

    for (my $i = 0; $i < $max; $i++) {

        $in_pcrm = util::in_crm($pcrm_blocks, $i); # in pCRM?
        $in_crm  = util::in_crm($crm_blocks,  $i); # in CRM?

        if    ($in_pcrm   && $in_crm)   { $tp++; }
        elsif ($in_pcrm   && ! $in_crm) { $fp++; }
        elsif (! $in_pcrm && $in_crm)   { $fn++; }

    }
```

```
    my $phi = util::phi_score($tp, $fp, $fn);

    return {
        'phi' => $phi,
        'tp'  => $tp,
        'fp'  => $fp,
        'fn'  => $fn,
    };

}



sub phi_write
{
    my ($summary) = @_;

#        $summary{$abbv} = {
#            'blocks'      => $blocks,
#            'plot_fields' => $plot_fields,
#            'phi_score'   => phi_score($blocks, $gene),

    open FILE, ">>", OUTPUT_PHI_SCORE_FILE;

    for (sort keys %{ $summary }) {

        print FILE "\t$_\t$summary->{$_}->{'phi_score'}\t$summary-
>{$_}->{'tp_score'}\t$summary->{$_}->{'fp_score'}\n";

    }

    close FILE;

}



# --

# --
# --
# -- cribbed from overlap.pl
sub phi_score_block
{
    my ($blocks, $gene) = @_;

    my ($tp, $fp, $fn) = (0, 0, 0);

    PCRM:
    for my $pcrm (@{ $blocks }) {

        CRM:
        for my $crm (@{$gene->enhancers()}) {
```

```
            # find the following overlaps:
            #         1         2      3      4
            # pcrm: AAAA   |   AAAA |  AA  | AAAA
            #  crm:   BBBB | BBBB   | BBBB |  BB
            # 4 is special case of 1 and is therefore omitted
            # skip to the next one
            if (   ($crm->{'beg'} >= $pcrm->{'beg'} && $crm->{'beg'} <=
$pcrm->{'end'})
                || ($crm->{'end'} >= $pcrm->{'beg'} && $crm->{'end'} <=
$pcrm->{'end'})
                || ($crm->{'beg'} <= $pcrm->{'beg'} && $crm->{'end'} >=
$pcrm->{'end'})
                ) {

                $tp++;
                next PCRM;

            }

        }

        # didn't match any CRMs
        $fp++;
    }

    # how many CRMs did we miss?
    $fn = (scalar @{$gene->enhancers()}) - $tp;

    my $phi = util::phi_score($tp, $fp, $fn);

    return {
        'phi' => $phi,
        'tp'  => $tp,
        'fp'  => $fp,
        'fn'  => $fn,
    };


}

1;
```

motif_rank_plot.pm

```
package motif_rank_plot;

# --
# -- 2005-07-12, zak.burke@dartmouth.edu
# --
# -- given FASTA-formatted sequences and a list of motifs, rank the
# -- sequences by motif-count.
# --
# -- graphing routines based on overlap.pl
# --
# --
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.10 $
# -- $Date: 2005/10/20 05:21:43 $
# --



BEGIN {
    $ENV{'HOME_PERL_LIB'} = "" unless $ENV{'HOME_PERL_LIB'};
}

use strict;

use lib $ENV{'HOME_PERL_LIB'};
use GD::Graph;
use GD::Graph::mixed;
use GD::Graph::hbars;

use constant TRUE  => 1;
use constant FALSE => 0;
use constant PLOT_BARS   => "bars";
use constant PLOT_LINES  => "lines";
use constant PLOT_POINTS => "points";

use constant PLOT_WIDTH  => 800;
use constant PLOT_HEIGHT => 125;

use constant PLOT_FGCLRBOX => "white";
use constant PLOT_FGCLR => "black";

use constant CLR_R_MAX => 204;
use constant CLR_MAX   => 255;

use constant MOTIF_COLORS => qw(dgray dblue dgreen dred dpurple);



# --
```

```
# --
# -- plot
# -- prepare a plot, then create it
sub plot
{
    # $params->fields: array-ref, each value is a sequence-offset
    # $params->crms: array-ref, non-zero indicates CRM
    # $params->pcrms: array-ref of array-refs, non-zero indicates pCRM
    # $params->abbv: string, name of gene to plot
    my ($params) = @_;

    # configure fields, exons, promoters, CRMs
    plot_prepare($params);

    # prepare pCRM regions
    pcrm_color_by_density($params);

    # prepare motif regions
    motif_lines($params);

    # create the graph
    graph($params);

}



# --

# --
# -- plot_prepare
# -- configure a plot's fields, exons, promoters and CRMs
sub plot_prepare
{
    my ($params) = @_;


    # prepare fields (x axis values)
    push @{ $params->{'data'} }, $params->{'fields'};

    # prepare CRM regions, promoters and exons
    plot_bars($params, "exons",     "gold",   PLOT_BARS);
    plot_bars($params, "promoters", "dgreen", PLOT_BARS);
    plot_bars($params, "crms",      "dblue",  PLOT_BARS);


}



# --

# --
# -- graph
# -- write a plot to a .png file
```

```
sub graph
{
    my ($params) = @_;
    eval
    {
        my $graph = GD::Graph::mixed->new(PLOT_WIDTH, ($params-
>{'motifs'} ? PLOT_HEIGHT * 2 : PLOT_HEIGHT)) or die GD::Graph->error;
        $graph->set(
            title    => $params->{'abbv'},   # headline
            types    => $params->{'types'},  # type of each plot
(bars|lines|points)
            dclrs    => $params->{'dclrs'},  # color of each plot
            boxclr        => PLOT_FGCLRBOX,  # background color of plot
field
            labelclr      => PLOT_FGCLR,     # color of axis labels
            axislabelclr  => PLOT_FGCLR,     # color of axis-position
labels
            fgclr         => PLOT_FGCLR,     # color of axes
            markers       => [10],           # for points, use only the
| glyph
            x_label_skip  => 1000,           # skip n ticks and tick-
labels along x axis
            y_number_format => "",           # no labels along y axis
            zero_axis_only => 1,             # print axis y = 0 and
print x tick-labels there
            x_label       => "sequence position",
            y_max_value   => $params->{'y_max'} || 1,
            y_min_value   => $params->{'y_min'} || ($params->{'motifs'}
? -2 : 0),
            correct_width => FALSE,

        ) or die $graph->error;

        # legend labels must match the order of fields in $params-
>{'data'}
        $graph->set_legend(qw(Exon Promoter CRM pCRM));

        my $gd = $graph->plot($params->{'data'}) or die $graph->error;
        open(IMG, ">$params->{'abbv'}.motif_profile.png") or die $!;
        binmode IMG;
        print IMG $gd->png();
        close IMG;
    };
    if ($@) {
        print STDERR "ERROR: $@";
    }
}



# --

# --
# -- pcrm_color_by_density
```

```
# -- pCRM regions are colored from red->pink->white as motif density
sinks
# -- this assumes the @{ $params->{pcrms} } array is already in order
from
# -- dense to sparse
sub pcrm_color_by_density
{
    my ($params) = @_;

    # steps between red and white
    my $divisor = $#{ $params->{'pcrms'} } || 1;
    my $increment = int(CLR_MAX / int(($divisor+1) / 2 ) );
    my (@dcolors, @colors, @roy, @gee, @biv);

    for (my $i = 0; $i <= int(($divisor + 1) / 2); $i++) {
        push @colors, {'r' => CLR_R_MAX, 'g' => 0, 'b' => $i *
$increment};
        push @biv,    {'r' => CLR_R_MAX, 'g' => $i * $increment, 'b' =>
CLR_MAX};
    }
    shift @biv;
    push @colors, @biv;

    for (my $i = 0; $i <= $#{ $params->{'pcrms'} }; $i++) {

        my $color = GD::Graph::colour::rgb2hex(
            $colors[$i]->{'r'},
            $colors[$i]->{'g'},
            $colors[$i]->{'b'}
            );
        push @dcolors, $color;
        GD::Graph::colour::add_colour($color);

        push @{ $params->{'dclrs'} }, $color;

        push @{ $params->{'types'} }, PLOT_BARS;

        push @{ $params->{'data'} }, ${ $params->{'pcrms'} }[$i];

    }

    # plot a legend
    plot_legend(\@dcolors, $params);

}



# --

# --
# -- motif_points
# -- color bars for motifs; rotating rainbow
sub motif_points
{
```

```
    my ($params) = @_;

    return unless $params->{'motifs'};

    #my @colors = qw(lgreen lblue lyellow lpurple cyan lorange);
    my @colors = MOTIF_COLORS; # qw(lgreen lblue lyellow lpurple cyan
lorange);

    for (my $i = 0; $i <= $#{ $params->{'motifs'} }; $i++) {
        # plot is too cluttered with > 5 motif pictures
        last if $i > 12;

        my $color = $colors[$i % (scalar @colors)];

        push @{ $params->{'dclrs'} }, $color;

        push @{ $params->{'types'} }, PLOT_POINTS;

        push @{ $params->{'data'} }, ${ $params->{'motifs'} }[$i];

        # debugging output
        #motif_location_display(${ $params->{'motifs'} }[$i]);

    }

}



# --

# --
# -- motif_lines
# -- plot motif locations along a staff of lines
sub motif_lines
{
    my ($params) = @_;

    return unless $params->{'motifs'};

    my $x_max = scalar @{ ${ $params->{'motifs'} }[0] };

    #my @colors = qw(lgreen lblue lyellow lpurple cyan lorange);
    my @colors = MOTIF_COLORS; #qw(lgreen lblue lyellow lpurple cyan
lorange);

    for (my $i = 0; $i <= $#{ $params->{'motifs'} }; $i++) {
        my $color = $colors[$i % (scalar @colors)];

        # determine & configure y-intercept for staff-line
        my $height = 0;
        for (@{ ${ $params->{'motifs'} }[$i] }) {
            if ($height = $_) {
                push @{ $params->{'dclrs'} }, $color;
                push @{ $params->{'types'} }, PLOT_LINES;
```

```
                push @{ $params->{'data'} }, plot_line_at($x_max,
$height);

                last;
            }
        }

        # configure motif positions
        push @{ $params->{'dclrs'} }, $color;
        push @{ $params->{'types'} }, PLOT_POINTS;
        push @{ $params->{'data'} }, ${ $params->{'motifs'} }[$i];

    }

}



# --

# --
# -- motif_set_background
# -- paint a big white block from x = 0 to x = sequence-length, y = 0
to y = -1
# -- then we can plot motifs atop it
sub motif_set_background
{
    my ($params) = @_;

    my (@block, $max);

    # number of points in the plot, i.e. length of the sequence
    $max = scalar @{ ${ $params->{'motifs'} }[0] };

    push @{ $params->{'dclrs'} }, "white";

    push @{ $params->{'types'} }, PLOT_BARS;

    for (0..$max) {
        push @block, -1;
    }
    push @{ $params->{'data'} }, \@block;

}



# --

# --
# -- plot_line_at
# -- prepare the points to plot a line from 0..x at height y
sub plot_line_at
{
    my ($x, $y) = @_;
```

```
    my (@blocks);

    for (0..$x) {
        push @blocks, $y;
    }

    return \@blocks;

}




# --

# --
# -- motif_location_display
# -- prints locations of motifs in a block to STDOUT
sub motif_location_display
{
    my ($list) = @_;

    my $max = $#{ $list } + 1;

    for (my $i = 0; $i < $max; $i++) {

        next unless $list->[$i];
        print "$i\n";
    }

}



# --

# --
# -- plot_bars
# -- configure a particular param field to be plotted as bars
sub plot_bars
{
    my ($params, $field, $color, $type) = @_;

    push @{ $params->{'data'} }, $params->{$field};
    push @{ $params->{'dclrs'} }, $color;
    push @{ $params->{'types'} }, $type;

}



# --

# --
```

```
# -- plot_legend
# -- create an HTML graph legend mapping density => color
sub plot_legend
{
    my ($colors, $params) = @_;

    my (@data, @dcolors, @types);

    my $i = 0;
    for (sort {$b <=> $a} @{ $params->{'density_values'} }) {
        my $color = shift @{ $colors };

        push @types, "bars";
        push @{ $data[0] }, $i;
        my @row;
        $row[$#{ $params->{'density_values'} }] = 0;
        $row[$i] = 1;
        push @data, \@row;
        push @dcolors, $color;

        $i++;
    }

        eval
        {

                my $graph = GD::Graph::mixed->new(50, (scalar @{
$data[0]})*20) or die GD::Graph->error;
                $graph->set(
                        types    => \@types,
                        dclrs    => \@dcolors,
                        x_ticks  => FALSE,
                        x_label       => "",
                        y_tick_number => 0,
                        y_max_value   => 1,
                        y_min_value   => 0,
                        y_number_format => "",
                        x_number_format => "",
                        boxclr        => "white",
                        rotate_chart => TRUE,

                ) or die $graph->error;
                my $gd = $graph->plot(\@data) or die $graph->error;
                open(IMG, ">$params->{'abbv'}.legend.png") or die $!;
                binmode IMG;
                print IMG $gd->png();
                close IMG;
        };
        if ($@) {
                print STDERR "ERROR: $@";
        }


}
```

```
1;
```

nazina.pm

```
package nazina;

# --
# -- nazina
# --
# --
# -- zak.burke@dartmouth.edu, 2005-03-27
# --
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.2 $
# -- $Date: 2005/10/20 05:19:03 $
# --


use DBI;
use CGI;
use strict;

sub sequence_annotate
{
    my ($seq, $elements) = @_;
    my ($string, @sequence, $len, $i, $j) = ("", (), 0, 0, 0);
    my %colors = (
        1 => '#6699cc', # promoter
        2 => '#ff0000', # enhancer - early
        3 => '#ff0000', # enhancer - late
        4 => '#ffff99', # exon
        5 => '#ff0000', # enhancer
     );

    @sequence = split '', $$seq;

    # sequence element counts are 1-based so we add an element
    # to the beginning of the list to make counting easier
    # likewise the for loop is 1.. <= length instead of 0..
    unshift @sequence, 'x';
    $len = length($$seq);
    for ($i = 1; $i < $len; $i++) {

        # tag element start
        for (@{ $elements }) {
            my $color = $colors{$_->{'type_id'}};
            $string .= "<a name=\"$_->{'element_id'}\"><span
class='sequence' style=\"background-color: $color; \">" if ($i == $_-
>{'beg'});
        }

        # current position
        $string .= $sequence[$i];
```

```
        # tag element end
        for (@{ $elements }) {
            my $color = $colors{$_->{'type_id'}};
            $string .= "</span></a>" if $i == $_->{'end'};
        }

        # wrap long lines
        if ($j++ > 70) {
            $string .= "\n";
            $j = 0;
        }
    }

    return $string;
}



# --

# --
# -- htmx
# -- read template from file; return it as a string
sub htmx
{
    my $htmx = "";
    open HTMX, "template.htmx";
    while (<HTMX>) { $htmx .= $_; };
    close HTMX;

    return $htmx;
}



1;
```

nazina_db.pm

```
package nazina_db;

# --
# -- nazina_db
# -- database connection and fetch routines to assist with
# -- with pulling data from the gene reguation database.
# --
# -- $Author: zburke $
# -- $Revision: 1.7 $
# -- $Date: 2005/10/20 05:19:03 $
# --

BEGIN {
    $ENV{'BG_PERL_LIB'}   = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'HOME_PERL_LIB'} = "." unless $ENV{'HOME_PERL_LIB'};
}



use DBI;
use strict;

use lib $ENV{'BG_PERL_LIB'};
use gene;

use constant USERNAME => "nazina";
use constant PASSWORD => "nazina";


sub new
{
    # class name
    my $class = shift;

    # named input parameters
    my %input = @_;

    my $self  = {};

    bless ($self, $class);

    # use dbh if it's available, or create a new one.
    $self->{'dbh'} = $input{'dbh'} || $self->dbh();

    return $self;
}


# --

# --
```

```
# -- genes
# -- retrieve all genes from the DB
sub genes
{
    my $self = shift;

    # return cached copy, if available
    return $self->{'genes'} if $self->{'genes'};

    my %list;
    my $sth = $self->{dbh}->prepare("select gene_id, name,
abbreviation, sequence from gene order by abbreviation");
    $sth->execute();

    while (my ($gene_id, $name, $abbv, $sequence) = $sth-
>fetchrow_array()) {
        $list{$gene_id} = {
            'gene_id' => $gene_id,
            'name' => $name,
            'abbv' => $abbv,
            'sequence' => $sequence,
            'dao' => $self,
        };
    }

    # cache this collection for fast retrieval later on
    $self->{'genes'} = \%list;

    return $self->{'genes'};

}



# --

# --
# -- genes_by_abbv
# -- list of genes keyed by their abbreviated names
sub genes_by_abbv
{
    my $self = shift;
    my $genes = $self->genes();
    my %list;
    for (keys %{ $genes }) {
        $list{ ${ $genes }{$_}->{'abbv'} } = ${ $genes }{$_};
    }

    return \%list;
}



# --
```

```
# --
# -- gene
# -- retrieve gene by id
sub gene
{
    my $self = shift;
    my ($gene_id) = @_;
    my %list;
    my $sth = $self->{dbh}->prepare("select name, abbreviation,
sequence, motifs from gene where gene_id = ?");
    $sth->execute($gene_id);
    my ($name, $abbreviation, $sequence, $motifs) = $sth-
>fetchrow_array();

    my $ref = {
            'gene_id'  => $gene_id,
            'name'     => $name,
            'abbv'     => $abbreviation,
            'sequence' => $sequence,
            'motifs'   => $motifs,
            'dao'      => $self,
        };

    my $gene = gene->new(%{ $ref });

    return $gene;

}



# --

# --
# -- gene_elements
sub gene_elements
{
    my $self = shift;
    my ($gene_id) = @_;

    my @elements;
    my $sth = $self->{dbh}->prepare("
        select element_id, element_type_id, name,
        position_start, position_end
        from element
        where gene_id = ?
        order by position_start");

    $sth->execute($gene_id);
    while (my ($element_id, $type_id, $name, $beg, $end) = $sth-
>fetchrow_array()) {
        my $regulators = $self->regulators($element_id);
        my $element = {
            'element_id' => $element_id,
            'type_id' => $type_id,
```

```
            'name' => $name,
            'beg' => $beg,
            'end' => $end,
            'regulators' => $regulators,
        };
        push @elements, $element;
    }

    return \@elements;
}



# --

# --
# -- gene_enhancers
# -- return enhancer elements for a gene
sub gene_enhancers
{
    my $self = shift;
    my ($gene_id) = @_;

    my @list;
    my $elements = $self->gene_elements($gene_id);
    for (@{ $elements }) {
        next unless $_->{'type_id'} == 2 || $_->{'type_id'} == 3;
        push @list, $_;
    }

    return \@list;
}



# --

# --
# -- gene_enhancers_by_regulator
# -- get enhancers for a gene which are known to be regulated by the
# -- given gene_id
sub gene_enhancers_by_regulator
{
    my $self = shift;
    my ($gene_id, $regulator_id) = @_;

    my @list = ();
    my $enhancers = $self->gene_enhancers($gene_id);
    for my $enhancer (@{ $enhancers }) {
        for my $reg (@{ $enhancer->{'regulators'} }) {
            if ($reg->{'gene_id'} == $regulator_id) {
                push @list, $reg;
            }
        }
    }
```

```
    return \@list;


}



# --

# --
# -- gene_promoters
# -- return promoter elements for a gene
sub gene_promoters
{
    my $self = shift;
    my ($gene_id) = @_;

    my @list;
    my $elements = $self->gene_elements($gene_id);
    for (@{ $elements }) {
        next unless $_->{'type_id'} == 1;
        push @list, $_;
    }

    return \@list;
}



# --

# --
# -- gene_exons
# -- return exon elements for a gene
sub gene_exons
{
    my $self = shift;
    my ($gene_id) = @_;

    my @list;
    my $elements = $self->gene_elements($gene_id);
    for (@{ $elements }) {
        next unless $_->{'type_id'} == 4;
        push @list, $_;
    }

    return \@list;
}



# --

# --
# -- regulators
```

```
# -- regulators for a given element
sub regulators
{
    my $self = shift;
    my ($element_id) = @_;

    my @list;
    my $sth = $self->{dbh}->prepare("
        select g.gene_id, g.name, g.abbreviation
        from gene g, regulator r
        where r.element_id = ?
            and g.gene_id = r.gene_id
        ");
    $sth->execute($element_id);
    while (my ($r_id, $name, $abbv) = $sth->fetchrow_array()) {
        push @list, {
            'gene_id' => $r_id,
            'name' => $name,
            'abbv' => $abbv,
             };
    }

    return \@list;

}



# --

# --
# -- motifs
# -- return motifs for a given gene
sub motifs
{
    my $self = shift;
    my ($gene_id) = @_;

    my $sth = $self->{dbh}->prepare("select motifs from gene where
gene_id = ?");
    $sth->execute($gene_id);
    my ($motifs) = $sth->fetchrow_array();

    return $motifs;

}



# --

# --
# -- regulatory_elements
# -- retreive all genes that are known to regulate other genes
sub regulatory_elements
```

```
{
    my $self = shift;

    my $genes = $self->genes();

    my $list = undef;
    my $sth = $self->{dbh}->prepare("select gene_id, element_id from
regulator");
    $sth->execute();
    while (my ($gene_id, $element_id) = $sth->fetchrow_array()) {

        $list->{$gene_id} = $genes->{$gene_id};

        for my $element (@{ $self->elements() }) {

            if ($element->{'element_id'} == $element_id) {
                $element->{'gene'} = $genes->{ $element->{'gene_id'} };
                push @{ $list->{$gene_id}->{'elements'} }, $element;
            }
        }

    }

    return $list;
}



# --

# --
# -- elements
# -- all elements
sub elements
{
    my $self = shift;

    return $self->{'elements'} if ($self->{'elements'});

    my @elements;
    my $sth = $self->{dbh}->prepare("
        select element_id, gene_id, element_type_id, name,
        position_start, position_end, sequence
        from element");

    $sth->execute();
    while (my $element = $sth->fetchrow_hashref()) {
        $element->{'regulators'} = $self->regulators($element-
>{'element_id'});
        push @elements, $element;
    }

    $self->{'elements'} = \@elements;

    return $self->{'elements'};
```

```
}


# --

# --
# -- dbh
# -- new DB handle
sub dbh
{
    my $self = shift;

    return $self->{dbh} if $self->{dbh};

    my $dsn = "DBI:mysql:database=nazina;host=localhost";
    $self->{dbh} = DBI->connect($dsn, USERNAME, PASSWORD);

    return $self->{dbh};

}

1;
```

pcrm_block.pm

```
package pcrm_block;

# --
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.2 $
# -- $Date: 2005/10/20 05:19:03 $
# --

use strict;

sub new
{
    my $class = shift;
    my $self  = {};

    bless($self, $class);

    my %input = @_;

    $self->{'abbv'} = $input{'abbv'} || "";
    $self->{'beg'}  = $input{'beg'} || -1;
    $self->{'end'}  = $input{'end'} || -1;
    $self->{'motif_count'} = $input{'motif_count'} || 0;
    $self->{'density'}     = $input{'density'} || 0;
    $self->{'sequence'}    = $input{'sequence'} || "";
    $self->{'motifs'}      = $input{'motifs'} || {};


    return $self;

}



# --

# --
# -- score
sub score
{

}


1;

__END__
```

pipeline_util.pm

```
package pipeline_util;

# --
# -- utility methods for pipeline output files
# --
# -- $Author: zburke $
# -- $Revision: 1.2 $
# -- $Date: 2005/10/23 00:53:09 $
# --

BEGIN {
    $ENV{'BG_PERL_LIB'} = "/home/zburke/bg/workspace/perl-lib" unless
$ENV{'BG_PERL_LIB'};
    $ENV{'BG_PERL_BIN'} = "/home/zburke/bg/workspace/pipeline" unless
$ENV{'BG_PERL_BIN'};
}

use strict;
use warnings;
use GD::Graph::lines;
use GD::Graph::bars;
use GD::Graph::hbars;


# --

# --
# -- motifs_reader
# -- given a formatted motifs file, pull out the top motifs
# --
# -- file format:
# --
#               motif          sig      p-cover      g-cover
pipeline
#              grncbs     1.000000     0.857143     1.000000
2.200000
#               cgatc     0.244534     0.396825     1.000000
1.444534
#               agggg     0.129743     0.380952     1.000000
1.329743
#
sub motifs_reader
{
    my ($filename, $params) = @_;

    my @motifs;
    my @lines = map { chomp; $_;} `head -$params->{'motif_count'}
$filename`;

    # ignore the header line
    shift @lines;
    for (@lines) {
```

```
        # zap leading/trailing whitespace
        s/^\s+//g; s/\s+$//g;

        my ($motif, $sig, $pcover, $gcover, $pipeline) = split /\s/;
        push @motifs, $motif;
    }

    return \@motifs;

}



# --

# --
# -- phi_score_reader
# -- given a formatted phi-score file, pull out and return the
# -- score for the requested gene/round combination. return -1
# -- if the requested tuple is not found.
# --
# -- fields: gene, phi-score, tp-positions, fp-positions
# -- fileformat:
#
#round     1
#    btd    0.208278867102397    956    4117
#    eve    0.688316606703276    3635    2186
#    hb    0.197399783315276    911    4104
#    kni    0.105945545471384    572    4943
#    kr    0.417505924170616    2819    2381
#    otd    0.260181268882175    2153    6024
#round     2
#    btd    0.2333984375    956    3623
#    eve    0.210315759394199    1472    1741
#    hb    0.209136822773186    911    3845
#    kni    0.122065727699531    572    4230
#    kr    0.42992222052768    2819    2186
#    otd    0.267719472767968    2153    5791# ...
# round n
#     ...
sub phi_score_reader
{
    my ($filename, $round, $abbv) = @_;

    my ($i, $gene, $junk, %rounds) = (0, "", "", ());
    my $value = {};

    open PHI_FILE, "<", $filename or die ("couldn't open $filename:
$!");
    while (<PHI_FILE>) {
        chomp;

        if (/^round/) {
            ($junk, $i) = split /\s+/, $_;
```

```
            next;
        }

        # trim leading and trailing whitespace
        s/^\s+//g; s/\s+$//g;

        my ($gene, $phi, $tp, $fp) = split /\s+/, $_;
        if ($gene eq $abbv && $round eq "round_$i") {
            $value = {
                'phi' => $phi,
                'tp'  => $tp,
                'fp'  => $fp,
                'tp_fp' => 0.0,
                'rho' => 0.0,
            };
            last;
        }
    }
    close PHI_FILE;

    if ($value->{'fp'}) {
        $value->{'tp_fp'} = $value->{'tp'} / $value->{'fp'};
        $value->{'rho'} = $value->{'tp'} / ($value->{'tp'} + $value-
>{'fp'});
    }

    return $value;

}



sub phi_score_reader_summary
{
    my ($filename) = @_;

    my ($i, $gene, $junk, $summary) = (0, "", "", {});

    open PHI_FILE, "<", $filename or die ("couldn't open $filename:
$!");
    while (<PHI_FILE>) {
        chomp;

        if (/^round/) {
            ($junk, $i) = split /\s+/, $_;
            next;
        }

        # trim leading and trailing whitespace
        s/^\s+//g; s/\s+$//g;

        my ($gene, $phi, $tp, $fp) = split /\s+/, $_;
        $summary->{$i}->{$gene}->{'phi'}   = $phi;
        $summary->{$i}->{$gene}->{'tp'}    = $tp;
        $summary->{$i}->{$gene}->{'fp'}    = $fp;
```

```
        $summary->{$i}->{$gene}->{'tp_fp'} = 0.0;
        $summary->{$i}->{$gene}->{'rho'}   = 0.0;
        $summary->{$i}->{$gene}->{'rho2'}  = 0.0;

        # calculate tp/fp and tp/(tp + fp) only if fp is non-zero
        if ($summary->{$i}->{$gene}->{'fp'}) {
            $summary->{$i}->{$gene}->{'tp_fp'} = $summary->{$i}-
>{$gene}->{'tp'} / $summary->{$i}->{$gene}->{'fp'};
            $summary->{$i}->{$gene}->{'rho'}  = $summary->{$i}-
>{$gene}->{'tp'} / ( $summary->{$i}->{$gene}->{'tp'} + $summary->{$i}-
>{$gene}->{'fp'});
            $summary->{$i}->{$gene}->{'rho2'} = $summary->{$i}-
>{$gene}->{'tp'}
                / ( $summary->{$i}->{$gene}->{'tp'}
                    + $summary->{$i}->{$gene}->{'fp'}
                    * $summary->{$i}->{$gene}->{'fp'}
                  );
        }

    }
    close PHI_FILE;

    return $summary;


}



# --

# --
# -- summary_graph
# -- plot phi-scores for each gene, and average across all genes, for
each round.
sub summary_graph
{
    my ($params, $plot, $filename) = @_;

    eval
    {
        my $graph = GD::Graph::lines->new(600, 600) or die GD::Graph-
>error;
        $graph->set(
            title      => "phi scores",   # headline
            types      => $plot->{'types'},  # type of each plot
(bars|lines|points)
            x_label    => "round number",
            y_label    => "phi score",
#            line_width => 4
        ) or die $graph->error;

        # legend labels must match the order of fields in $params-
>{'data'}
        $graph->set_legend(@{ $plot->{'legend'} });
```

```
        my $gd = $graph->plot($plot->{'data'}) or die $graph->error;
        open(IMG, ">$filename") or die $!;
        binmode IMG;
        print IMG $gd->png();
        close IMG;
    };
    if ($@) {
        print STDERR "ERROR: $@";
    }

}


# --

# --
# -- summary_graph
# -- plot phi-scores for each gene, and average across all genes, for
each round.
sub summary_graph_bars
{
    my ($params, $plot, $filename, $title) = @_;

    eval
    {
        my $graph = GD::Graph::hbars->new(400, 1000) or die GD::Graph-
>error;
        $graph->set(
            title      => $title || "phi score",   # headline
            types      => $plot->{'types'},  # type of each plot
(bars|lines|points)
            x_label    => "",
            y_label    => $title || "phi score",
#            x_labels_vertical => 1,
            show_values => 1,
            values_format => "%.8f",
#            values_vertical => 1,
        ) or die $graph->error;

        # legend labels must match the order of fields in $params-
>{'data'}
        $graph->set_legend(@{ $plot->{'legend'} }) if $plot-
>{'legend'};

        my $gd = $graph->plot($plot->{'data'}) or die $graph->error;
        open(IMG, ">$filename") or die $!;
        binmode IMG;
        print IMG $gd->png();
        close IMG;
    };
    if ($@) {
        print STDERR "ERROR: $@";
    }

}
```

```
# --

# --
# -- summary_graph_data
# -- helper for summary_graph. mungs data into GD::graph compatible
format
# --
# -- $params->{'genes'}->{$abbv}->{$round}->{'phi'} = x.y;
# -- $params->{'rounds'} = [a.b, c.d, ...];
sub summary_graph_data
{
    my ($params) = @_;

    my (@data, @types, @legend);

    # prepare scores for each gene
    for (sort keys %{ $params->{'genes'} }) {
        my $details = $params->{'genes'}->{$_};
        push @legend, $_;

        my @row = ();

        # for this gene, prepare score for each round
        for my $round_key (sort keys %{ $details } ) {
            my $round = $details->{$round_key};

            push @types, "lines";
            push @row, $round->{'phi'};

        }

        push @data, \@row;
    }

    # prepare rounds' average scores
    my @row = ();
    for (sort keys %{ $params->{'rounds'} }) {

        push @row, $params->{'rounds'}->{$_};

    }
    push @legend, "average";
    push @types, "lines";
    push @data, \@row;

    # labels for rounds; must be first @data element
    my @rounds;
    for (my $i = 1; $i <= scalar @row; $i++) {
        push @rounds, $i;
    }
    unshift @data, \@rounds;
```

```
    return {
        "legend" => \@legend,
        "data"   => \@data,
        "types"  => \@types,
        };


}



# --

# --
# -- score_average
# -- return average score for a given field in a list of score hash-
refs
sub score_average
{
    my ($scores, $field) = @_;

    my $avg = 0.0;
    if (scalar @{ $scores }) {
        my $sum = 0;
        $sum += $_ for map { $_->{$field} }  @{ $scores };

        $avg = $sum / (scalar @{ $scores });
    }

    return $avg;
}



1;

__END__
```

scan_parse.pm

```
package scan_parse;

# --
# -- package scan_parse
# -- CRM scan results parsers and formatters
# --
# -- 2005-04-04, zak.burke@dartmouth.edu
# --
# -- $Author: zburke $
# -- $Revision: 1.5 $
# -- $Date: 2005/07/12 03:22:11 $
# --


use strict;

# --

# --
# -- parser
# -- return the parser whose head method matches the given line
sub parser
{
    my ($line) = @_;

    my $parsers = formats();

    for (keys %{ $parsers }) {
        my $head = &{ ${ $parsers }{$_} }("head");
        return ${ $parsers }{$_} if $line =~ /^$head$/;
    }

    print STDERR "parser $line is not available; returning default
parser\n";
    return ${ $parsers }{'format_default'};
}



# --

# --
# -- parser_by_name
# -- select a parser by name. the default is returned if the requested
# -- parser is not available.
sub parser_by_name
{
    my ($name) = @_;

    my $parsers = formats();
    return ${ $parsers }{'format_default'} unless $name;
```

```
    return ${ $parsers }{$name} ? ${ $parsers }{$name} : ${ $parsers
}{'format_default'};

}



# --

# --
# -- formats
# -- return available parsers with names as keys and function-refs
# -- as values
sub formats
{
    my %list = (
        'format_default' => \&format_default,
        'format2'        => \&format2,
        'format3'        => \&format3,
        'format4'        => \&format4,
        'format5'        => \&format5,
    );

    return \%list;
}



# --

# --
# -- format_default
# -- default parser
sub format_default
{
    my ($what, $item) = @_;

    # FIELD NAMES
    #        hm: harmonic mean score
    #       phi: phi score
    #    recall: true-positives / (false-negatives + true-positives)
    # precision: true-positives / (false-positives + true-positives)
    #      file: path to file these stats are derived from
    my @fields = qw(hm phi_score recall precision file);

    if ($what eq "head") {

        return join "\t", @fields;

    } elsif ($what eq "read") {

        return line_read($item, \@fields);

    } elsif ($what eq "write") {
```

```
        return line_write($item, \@fields);

    }
}



# --

# --
# -- format2
# -- formatter with CRM metadata
sub format2
{
    my ($what, $item) = @_;

    my @fields = qw(hm phi_score recall precision s_to_n file);

    if ($what eq "head") {

        return join "\t", @fields;

    } elsif ($what eq "read") {

        return line_read($item, \@fields);

    } elsif ($what eq "write") {

        return line_write($item, \@fields);

    }
}



# --

# --
# -- format3
# -- formatter with CRM metadata and detail scan params
sub format3
{
    my ($what, $item) = @_;

    my @fields = qw(hm phi_score recall precision s_to_n profile-cutoff
peak-width input-smooth supression file);

    if ($what eq "head") {

        return join "\t", @fields;

    } elsif ($what eq "read") {

        return line_read($item, \@fields);
```

```
    } elsif ($what eq "write") {

        return line_write($item, \@fields);

    }
}



# --

# --
# -- format4
# -- formatter with CRM metadata, train params and scan params
sub format4
{
    my ($what, $item) = @_;

    my @fields = qw(hm phi_score recall precision s_to_n word window
channel profile-cutoff peak-width input-smooth supression file);

    if ($what eq "head") {

        return join "\t", @fields;

    } elsif ($what eq "read") {

        return line_read($item, \@fields);

    } elsif ($what eq "write") {

        return line_write($item, \@fields);

    }
}



# --

# --
# -- format5
# -- formatter with CRM metadata, train params and scan params
sub format5
{
    my ($what, $item) = @_;

    my @fields = qw(hm phi_score recall precision specificity s_to_n
word window channel profile-cutoff peak-width input-smooth supression
file);

    if ($what eq "head") {

        return join "\t", @fields;
```

```
    } elsif ($what eq "read") {

        return line_read($item, \@fields);

    } elsif ($what eq "write") {

        return line_write($item, \@fields);

    }
}



# --

# --
# -- line_read
# -- split on tabs, then push fields into hash with keys from
# -- @$fields array. return a hash-ref.
# --
# -- ARGUMENTS
# --     $item -- tab-delimited data
# --     $fields -- array-ref of field-names
sub line_read
{
    my ($item, $fields) = @_;

    my %list;
    my @cols = split "\t", $item;
    for (my $i = 0; $i < (scalar @{ $fields} ); $i++) {

        $list{${ $fields }[$i]} = $cols[$i];

    }

    return \%list;
}



# --

# --
# -- line_write
# -- format %$items values into a tab-delimited line of fields from
# -- @$fields
# --
# -- ARGUMENTS
# --     $item - hash ref of data
# --     $fields - ordered list of fields to pluck from $item
sub line_write
{
    my ($item, $fields) = @_;

    return "" . (join "\t", map { $item->{$_} } @{ $fields }) . "\n";
```

```
}



# --

# --
# -- field_parser
# -- return the requested file field-parser
sub field_parser
{
    my ($name) = @_;

    return $name ? ${ field_parsers() }{$name} : ${ field_parsers()
}{'default'}
}



# --

# --
# -- field_parsers
# -- return a hash of line parsers. keys are parser names; values are
# -- function refs.
sub field_parsers
{
    my %list;

    $list{'ts'}      = \&field_parse_ts;
    $list{'gene'}    = \&field_parse_gene;
    $list{'default'} = \&field_parse_gene;


    return \%list;
}



# --

# --
# -- field_parse
# -- parse line into a list of fields
# --
# -- ARGUMENTS
# --     $line - line of data to read
# --     $line_parser - function-ref; maps line values to key-value
pairs
# --     $file_field_parser - function-ref; extracts values encoded in
# --         line's file field
# --
sub field_parse
{
```

```
    my ($line, $line_parser, $file_field_parser) = @_;

    # parse line into fields
    my $list = &{ $line_parser }("read", $line);

    # extract fields from the file name ...
    my $fields = &{ $file_field_parser }(${ $list }{'file'});

    # ... and add them to the fields to be returned.
    # note that some filenames include fields and some omit them.
    # in the case that the filename OMITS the field, leave the value
    # extracted from the line-parsing in place, rather than replacing
it
    # with the (possibly empty) value extracted from the filename.
    for (keys %{ $fields }) {
        ${ $list }{$_} = ${ $fields }{$_} unless ${ $list }{$_};
        ${ $list }{$_} = ${ $fields }{$_} if ${ $fields }{$_};
    }

    return $list;
}



# --

# --
# -- field_parse_ts
# -- split a training-set filename into its constituent parts
# --
# -- ARGUMENTS
# --     $file - filename with embedded metadata
sub field_parse_ts
{
    my ($file) = @_;

    # fields embedded in filename are:
    #      abbv: abbreviated gene-name
    #      word: word-length
    #    window: window-length
    #   channel: number of channels
    #        ts: training set name

    # extract fields from the filename; throw out leading directories
    my @file = split "\/", $file;
    $file = $file[$#file];
    my ($abbv, $word, $mismatch, $window, $channel, $ts) = ($file =~
/([^\/]+)\.fa_([0-9]+)_([0-9]+)_([0-9]+)_([0-9]+)_(train_[0-9]+).*/);

    return {
        'abbv'      => $abbv,
        'word'      => $word,
        'mismatch'  => $mismatch,
        'window'    => $window,
        'channel'   => $channel,
```

```
        'ts'        => $ts,
        };

}



# --

# --
# -- field_parse_gene
# -- split a training-set filename into its constituent parts
# --
# -- ARGUMENTS
# --     $file - filename with embedded metadata
sub field_parse_gene
{
    my ($file) = @_;

    # fields embedded in filename are:
    #      abbv: abbreviated gene-name

    # extract fields from the filename; throw out leading directories
    my @file = split "\/", $file;
    $file = $file[$#file];
    my ($abbv) = ($file =~ /([^\/]+)\.fa_scan.*/);

    return {
        'abbv'      => $abbv,
        };
}



# --

# --
# -- pcrm_file_read
# -- read pCRM position information from the FASTA file in $dir_name.
# -- if the file is missing or contains no data, return nothing.
# -- otherwise return an array of hash-refs with position details.
# --
# -- ARGUMENTS
# --     $dir_name - path to directory with LWF scan result files
# --
sub pcrm_file_read
{
    my ($dir_name) = @_;

    # find the file with the pCRM substrings
    opendir SCANDIR, $dir_name;
    my @scan = grep { /.*\.fa\.xtr\.dat$/ } readdir SCANDIR;
    closedir SCANDIR;

    if (! $scan[0]) {
```

```
        print STDERR "no .fa.xtr.dat file in $dir_name\n";
        return;
    }

    # read the pCRM substrings
    open SCAN, "<$dir_name/$scan[0]";
    my @lines = <SCAN>;
    close SCAN;

    if (! scalar @lines) {
        print STDERR "$scan[0] has no content\n";
        return ;
    }

    # get pCRM positions
    my @test;
    for (@lines) {
        # trim trailing whitespace, including damn windows line breaks
        chomp;
        s/\s+$//g;

        my @words = split / /;
        my ($beg, $end) = split "-", $words[1];
        push @test, {'beg' => $beg, 'end' => $end};
    }

    return \@test;
}

1;
```

scope.pm

```
package scope;

# --
# -- routines helpful for calling SCOPE and parsing its output
# --
# --
# -- $Author: zburke $
# -- $Revision: 1.4 $
# -- $Date: 2005/10/20 05:19:56 $
# --

use strict;


#use constant CLASS_LIB =>  "/home/zburke/bg/workspace/scopegenie/lib";
use constant CLASS_LIB =>  "/home/zburke/bg/workspace/scope/lib";
use constant CLASS_PATH =>
CLASS_LIB."/cos.jar:".CLASS_LIB."/acme.jar:".CLASS_LIB."/scope.jar";

# only motifs with sig scores >= SIG_THRESHOLD will be used for
anything
use constant SIG_THRESHOLD => 3.0;



sub new
{
    my $class = shift;
    my $self  = {};

    bless($self, $class);

    my %input = @_;

    return $self;
}



# --

# --
# -- dir_out
# -- get/set the output file to read. the setter doesn't tell SCOPE
where
# -- to write its output, rather it tells this module where to find the
# -- output from its run of SCOPE.
sub dir_out
{
    my $self = shift;
    $self->{'dir_out'} = shift if @_;
    return $self->{'dir_out'};
```

```
}



# --

# --
# -- parse
# -- parse the output file and cache the results
sub parse
{
    my $self = shift;

    my $max_score = 0;

    open FILE, "<", $self->dir_out() or die("couldn't open $self-
>{'dir_out'}: $!");
    while (<FILE>) {
        chomp;
        next unless $_;
        my ($sig, $word) = split /\s+/, $_;

        last if $sig < SIG_THRESHOLD;

        # found a new maximum score
        $max_score = $sig if $sig > $max_score;

        my ($motif, $side) = split ",", $word;

        push @{ $self->{results} }, {
            'motif' => $motif,
            'sig_score' => $sig,
            'side'  => $side,
            };
    }

    close FILE;


    # normalize sig scores into the range 0-1
    $self->sig_normalize($max_score);

}



# --

# --
# -- sig_normalize
# -- normalize sig scores into the range 0-1
sub sig_normalize
{
    my $self = shift;
    my ($max) = @_;
```

```
    for (@{ $self->{'results'} }) {

        $_->{'sig_score'} = $_->{'sig_score'} / $max;

    }
}



# --

# -- field
# -- return output values for $field
sub field
{
    my $self = shift;
    my ($field) = @_;

    $self->parse() unless $self->{'results'};

    my (@list);

    for (@{ $self->{'results'} }) {
        push @list, $_->{$field};
    }

    return \@list;
}



# --

# --
# -- motifs
# -- return motifs SCOPE found
sub motifs
{
    my $self = shift;
    return $self->field("motif");
}



# --

# --
# -- scores
# -- return sig-scores SCOPE found
sub scores
{
    my $self = shift;
    return $self->field("sig_score");
```

```
}



# --

# --
# -- results
# -- return sig-scores, motifs and side on which SCOPE found
# -- each motif
sub results
{
    my $self = shift;
    $self->parse() unless $self->{'results'};
    return $self->{'results'};
}



# --

# --
# -- run
# -- run SCOPE
sub run
{
    my $self = shift;
    my ($filename) = @_;

    $self->{'results'} = undef;

    die("couldn't find $filename") unless -f $filename;

    my $flags = join " ", @{ $self->run_flags($filename) };
    system("java $flags");

}



# --

# --
# -- run_flags
sub run_flags
{
    my $self = shift;
    my ($filename) = @_;

    my (@flags);

    push @flags, "-classpath ".CLASS_PATH;
    push @flags, "-mx1000M";
#    push @flags, "-Dscope.root=/home/zburke/bg/workspace/scopegenie/";
#    push @flags, "-Djava.path=/usr/bin/java";
```

```
    push @flags, "edu.dartmouth.bglab.beam.Scope";
    push @flags, "-pf D_melanogaster_enhancers.param";
    push @flags, "-sgs fastaFile";
    push @flags, "-sgi $filename";

    return \@flags;
}



# --

# --
# -- motif_line_format
# -- ordered list of fields for printing motif information
# -- NOTE: THIS IS A STATIC METHOD
sub motif_line_format
{
    my @fields = qw(motif sig_score side);
    return \@fields;
}


return 1;


__END__
```

util.pm

```
package util;

# --
# -- utility methods
# --
# -- $Author: zburke $
# -- $Revision: 1.2 $
# -- $Date: 2005/09/22 03:32:32 $
# --

use strict;

use constant TRUE  => 1;
use constant FALSE => 0;

use constant FASTA_LINE_LENGTH => 60;

# --

# --
# -- recall
# -- given the count of true-positives and false-negatives, calculate
# -- and return the recall rate.
# --
# -- recall is calculated as follows:
# --
# --     R == (true-positives) / (false-negatives + true positives)
# --
sub recall
{
    my ($tp, $fn) = @_;

    my $sum = $fn + $tp;

    return $sum ? ($tp / $sum) : 0;
}



# --

# --
# -- precision
# -- given the count of true-positives and false-positives, calculate
# -- and return the precision rate.
# --
# -- precision is calculated as follows:
# --
# --     P == (true-positives) / (false positives + true positives)
# --
sub precision
{
```

```
    my ($tp, $fp) = @_;

    return ($tp / ($fp + $tp));
}



# --

# --
# -- harmonic_mean
# -- return the harmonic mean of the 2 given rates. The harmonic mean
is useful
# -- for calculating an average of rates.
# --
# -- the harmonic mean is calculated as follows:
# --
# --     H == n / ( (1/a[1]) + (1/a[2]) + ... + (1/a[n]) ).
# --
# -- for a simple two-variable equation, this reduces to:
# --
# --     H == (2ab) / (a + b)
# --
sub harmonic_mean($$)
{
    my ($recall, $precision) = @_;

    return 0 if ($precision + $recall) == 0;

    return ((2 * $precision * $recall) / ($precision + $recall));

}



# --

# --
# -- phi_score
# -- calculate a phi_score: true-positives / [false-pos + false-neg]
# --
# -- the phi-score has become a standard metric for describing the
# -- performance of motif finding programs. it was first described in
# -- Pevzner, P.A. and Sze, S.H. (2000) Combinatorial approaches to
finding
# -- subtle signals in DNA sequences. Proceedings of the International
# -- Conference on Intelligent Systems in Molecular Biology 8:269-78.
sub phi_score
{
    my ($tp, $fp, $fn) = @_;

    return ($tp / ($fp + $fn));
}
```

```
# --

# --
# -- in_crm
# -- return TRUE if $i is within a defined CRM region in @$list;
# -- return FALSE otherwise.
sub in_crm
{
    my ($list, $i) = @_;

    for (@{ $list }) {

        return TRUE if ($i >= $_->{'beg'} && $i <= $_->{'end'})

    }

    return FALSE;
}



# --

# --
# -- fpe
# -- floating-point-equality test hack
sub fpe
{
    my ($X, $Y, $POINTS) = @_;

    $POINTS = 10 unless $POINTS;

    my ($tX, $tY);
    $tX = sprintf("%.${POINTS}g", $X);
    $tY = sprintf("%.${POINTS}g", $Y);
    return $tX eq $tY;
}


# --

# --
# -- fple
# -- floating-point-less-than-or-equal test hack
sub fple
{
    my ($X, $Y, $POINTS) = @_;

    return util::TRUE if ($X < $Y);

    return fpe($X, $Y, $POINTS);

}
```

```
# --

# --
# -- elapsed_time
# -- return the given time in seconds formatted as hh:mm:ss
sub elapsed_time
{
    my ($timer) = @_;

    my ($hour, $min, $sec) = (0, 0, 0);
    $hour = int($timer / (60 * 60) );
    $min = int (($timer / 60) % 60);
    $sec = $timer % 60;

    return sprintf("\nruntime: %2s:%2s:%2s\n\n",
        ($hour > 9 ? $hour : "0$hour"),
        ($min  > 9 ? $min  : "0$min"),
        ($sec  > 9 ? $sec  : "0$sec"));

}


# --

# --
# -- fasta
# -- return a fasta formatted sequence
sub fasta
{
    my ($sequence, $comment) = @_;

    my $content;

    $content .= ">$comment\n" if $comment;

    for (my $i = 0; $i < length($sequence); $i+= FASTA_LINE_LENGTH) {
        $content .= substr($sequence, $i, FASTA_LINE_LENGTH). "\n";
        last if (substr($sequence, $i, FASTA_LINE_LENGTH) eq "");
    }

    $content .= "\n";

    return $content;
}

1;
```

Appendix 5 Genbank Parser Application Code

Java Files

BglabCommandLine.java

```
package edu.dartmouth.bglab.pipeline.util;

import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.MissingArgumentException;

/**
 *
 * @author  $Author: zburke $
 * @version $Revision: 1.2 $
 * update   $Date: 2005/09/22 03:21:34 $
 *
 */
public class BglabCommandLine
{
    private CommandLine cl = null;

    public BglabCommandLine(CommandLine c)
    {
        this.cl = c;
    }



    /**
     * Return true if the requested option has been set
     *
     * @param opt short name of option to retrieve
     * @return true if the option has been set; false otherwise.
     */
    public boolean hasOption(String opt)
    {
        return this.cl.hasOption(opt);
    }



    /**
     * Retrieve an Option's value.
     *
     * @param opt short name of option to retrieve
```

```
     * @return string value of requested option
     * @throws MissingArgumentException if the requested Option is
undefined
     */
    public String value(String opt)
    throws MissingArgumentException
    {
        String value = null;
        if (this.cl.hasOption(opt))
            value = this.cl.getOptionValue(opt);

        if (value == null)
            throw new RuntimeException("The option " + opt + " has no
value.");

        return value;
    }



    /**
     * Retrieve the int value of an option.
     *
     * @param opt short name of option to retrieve
     * @return the integer value of opt
     * @throws MissingArgumentException if the requested Option is
undefined
     */
    public int intValue(String opt)
    throws MissingArgumentException
    {
        String value = null;
        if (this.cl.hasOption(opt))
            value = this.cl.getOptionValue(opt);

        if (value == null)
            throw new RuntimeException("The option " + opt + " has no
value.");

        return Integer.parseInt(value);
    }



    /**
     * Return true if an Option is defined, false otherwise.
     *
     * @param opt short name of option to retrieve
     * @return boolean value of the Option
     */
    public boolean booleanValue(String opt)
    {
        return (this.cl.hasOption(opt) ? true : false);
    }
}
```

Genbank.java

```
package edu.dartmouth.bglab.pipeline.util;

import org.biojava.bio.BioException;
import org.biojava.bio.seq.impl.RevCompSequence;
import org.biojava.bio.seq.io.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.PrintStream;
import java.io.FileNotFoundException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;


/**
 * Genbank file utility methods.
 *
 * A sample file is available here:
 * http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html
 *
 * 2005-08-25, zak.burke@dartmouth.edu
 *
 */
public class Genbank
{
    /** display distinct features in each sequence */
    public boolean logFeatures = false;

    /** bucket for distinct feature labels */
    private HashSet features = null;

    /** display gene locations in each sequence */
    public boolean logLocations = false;

    /** length of a line in a FASTA-formatted sequence */
    public int fastaLineLength = 60;

    /** show gene features only */
    public boolean filterGenic = false;

    /** show non-gene features only */
    public boolean filterBg = false;

    /** show gene and non-gene features */
    public boolean filterAll = false;
```

```
    /** show features upstream of genes */
    public boolean filterUpstream = false;

    /** default upstream -sequence length to extract */
    public int filterUpstreamLength = 1500;

    /** remove {@link #FEATURE_REPEAT} regions from intergenic regions
*/
    public boolean removeRepeats = false;

    /** Genbank file region label */
    public static String FEATURE_GENE = "gene";

    /** Genbank file region label */
    public static String FEATURE_REPEAT = "repeat_region";

    public static int STRAND_POS = 1;
    public static int STRAND_NEG = -1;



    /**
     * default constructor does nothing at all.
     *
     */
    public Genbank()
    {}



    /**
     * Converts the Genbank file named by filename into FASTA and
prints
     * it to ps.
     *
     * @param filename Genbank-formatted file to read from
     * @param ps printstream to write to
     * @throws Exception
     */
    public void toFasta(String filename, PrintStream ps)
    throws FileNotFoundException, BioException
    {
        BufferedReader br = new BufferedReader(new
FileReader(filename));

        // parse the Genbank file. damn that's easy.
        System.err.println("reading "+filename);
        SequenceIterator stream = SeqIOTools.readGenbank(br);

        System.err.println("parsing "+filename);
        while (stream.hasNext())
            toFasta(stream.nextSequence(), ps);

    }
```

```
    /**
     * Converts a {@link Sequence} into FASTA and prints it to ps.
     *
     * @param filename Genbank-formatted file to read from
     * @param ps printstream to write to
     */
    public void toFasta(Sequence seq, PrintStream ps)
    throws IllegalAlphabetException
    {
        // log features in this sequences
        if (this.logFeatures)
            featureRecord(seq);

        List genes   = filter(seq, Genbank.FEATURE_GENE);

        List repeats = new ArrayList();

        // TODO: implement repeat removal
        //if (this.removeRepeats)
        //    repeats = filter(seq, Genbank.FEATURE_REPEAT);

        if (this.filterGenic)
            displayGenic(seq, genes, ps);

        else if (this.filterAll)
            displayAll(seq, genes, repeats, ps);

        else if (this.filterBg)
            displayIntergenic(seq, genes, repeats, ps);

        else if (this.filterUpstream)
            displayUpstream(seq, genes, ps);

        if (this.logLocations)
            locationsDisplay(genes, System.err);
    }



    /**
     * Filter certain features from the given {@link Sequence} and
return
     * them in an ordered list.
     *
     * @param seq sequence containing {@link Feature}s
     * @param filter name of features to extract from seq
     * @return Features sorted by location in seq
     */
    public List filter(Sequence seq, String filter)
    {
        ArrayList list = new ArrayList();

        // make a Filter for "gene" features and retrieve them
```

```
        // features from D. Melanogaster CHR_2 include:
        // CDS gene mRNA misc_RNA repeat_region rRNA snoRNA source tRNA
        FeatureFilter ff = new FeatureFilter.ByType(filter);
        FeatureHolder fh = seq.filter(ff);

        // store all Features in fh so we can sort & print 'em later
        for (Iterator i = fh.features(); i.hasNext(); )
            list.add(i.next());

        // sort the list to make it easier to parse out non-gene pieces
        System.err.println("sorting " + list.size() + " " + filter + "
features...");
        Collections.sort(list, featureLocationComparator());

        return list;
    }



    /**
     * Extract genic sequences of this sequence. Gene features can
occur
     * on either strand although their positions are always recorded
     * based on their offset from the 5' end of the + strand. Genes on
the -
     * strand are reported as the reverse complement of their reported
     * positions, that is, as the bases on the - strand in the 5' to 3'
     * direction.
     *
     * @param seq sequence containing list of gene Features
     * @param list gene Features in this sequence
     * @param ps stream to print to
     */
    public void displayGenic(Sequence seq, List list, PrintStream ps)
    {
        StrandedFeature f = null;      // currently iterated feature
        String letters = null; // subsequence of current feature
        for (Iterator i = list.iterator(); i.hasNext();)
        {
            f = (StrandedFeature) i.next();
            letters = f.getSymbols().seqString();

            ps.println(">"+f.getAnnotation().getProperty("gene"));
            string2fasta(letters, ps);
        }
    }



    /**
     * Extract intergenic sequences of this sequence. Gene features can
occur
     * on either strand, but this algorithm treats all features as
though
```

```
     * they occur on the same (+) strand and only extracts the
intergenic
     * regions from this strand.
     *
     * @param seq sequence containing list of gene Features
     * @param list gene Features in this sequence
     * @param repeats repeat_region Features in this sequence
     * @param ps stream to print to
     */
    public void displayIntergenic(Sequence seq, List list, List
repeats, PrintStream ps)
    {
        // intial location has NOTHING -- bio sequences use 1-based
counting
        Location previous = LocationTools.makeLocation(0, 0);
        Location current = null;

        // temp variable store offsets of current intergenic region
        int beg = 0, end = 0;

        for (Iterator i = list.iterator(); i.hasNext();)
        {
            current = ((Feature) i.next()).getLocation();

            // start at 1 past last sub-sequence; end at 1 before
current one
            beg = previous.getMax() + 1;
            end = current.getMin() - 1;

            displayIntergenicHelper(beg, end, seq, ps, repeats);

            // reset previous if this sequence ends after the previous
one
            if (current.getMax() > previous.getMax())
                previous = current;
        }
    }



    /**
     * Extract genic and intergenic sequences of this sequence and
display
     * them. Gene features can occur
     * on either strand although their positions are always recorded
     * based on their offset from the 5' end of the + strand. Genes on
the -
     * strand are reported as the reverse complement of their reported
     * positions, that is, as the bases on the - strand in the 5' to 3'
     * direction.
     *
     * @param seq sequence containing list of gene Features
     * @param list gene Features in this sequence
     * @param repeats repeat_region Features in this sequence
     * @param ps stream to print to
```

```
     */
    public void displayAll(Sequence seq, List list, List repeats,
PrintStream ps)
    {
        // intial location has NOTHING -- bio sequences use 1-based
counting
        Location previous = LocationTools.makeLocation(0, 0);
        Location current = null;

        // temp variable store offsets of current intergenic region
        int beg = 0, end = 0;

        Feature f = null;
        for (Iterator i = list.iterator(); i.hasNext();)
        {
            f = (Feature) i.next();
            current = f.getLocation();

            // start at 1 past last sub-sequence; end at 1 before
current one
            beg = previous.getMax() + 1;
            end = current.getMin() - 1;

            // intergenic sequence
            displayIntergenicHelper(beg, end, seq, ps, repeats);

            // genic sequence
            ps.println(">"+f.getAnnotation().getProperty("gene"));
            string2fasta(f.getSymbols().seqString(), ps);

            // reset previous if this sequence ends after the previous
one
            if (current.getMax() > previous.getMax())
                previous = current;
        }
    }



    /**
     * format the region between beg and end (inclusive) as FASTA.
     *
     * @param beg first intergenic position
     * @param end last intergenic position
     * @param seq sequence containing this span
     * @param ps stream to write to
     * @param repeats list of {@link Feature}s labeled as repeat_region
     */
    private void displayIntergenicHelper(int beg, int end, Sequence
seq, PrintStream ps, List repeats)
    {

        /*
        if (this.removeRepeats)
        {
```

```
            Location l = LocationTools.makeLocation(beg, end);
            List list = recordRepeats(l, repeats);
            Location repeat = null;
            for (Iterator i = list.iterator(); i.hasNext();)
            {
                repeat = (Location) i.next();

                // handle:    rrrrrrr
                //         llllll
                if (l.getMin() < repeat.getMin())


                // handle:    rrrrrrr
                //               llllll
                if (l.getMax() > repeat.getMax())
            }

        }

        */

        if (beg < end)
        {
            ps.println(">intergenic-" + beg + "-" + end);

            // note: seq.subStr is 1-based and spans beg-end INCLUSIVE
            string2fasta(seq.subStr(beg, end), ps);
        }
        else
        {
            System.err.println("intergenic overlap from "+beg+" to
"+end);
        }
    }



    /**
     * Find Locations on a list that intersect the given location.
     *
     * @param l location which may have overlaps
     * @param repeats ordered list of locations of repeat regions
     *
     * @return repeat locations overlapping l
     */
    private List recordRepeats(Location l, List repeats)
    {
        ArrayList list = new ArrayList();
        Location repeat = null;
        for (Iterator i = repeats.iterator(); i.hasNext();)
        {
            repeat = (Location) i.next();
            if (l.overlaps(repeat))
                list.add(repeat);
```

```
            if (repeat.getMin() > l.getMin())
                break;
        }

        return list;
    }



    /**
     * Extract regions upstream of the genic features in this sequence.
     * Extractions are
     *  Gene features can occur
     * on either strand although their positions are always recorded
     * based on their offset from the 5' end of the + strand. Genes on
the -
     * strand are reported as the reverse complement of their reported
     * positions, that is, as the bases on the - strand in the 5' to 3'
     * direction.
     *
     * @param seq sequence containing list of gene Features
     * @param list gene Features in this sequence
     * @param ps stream to print to
     */
    public void displayUpstream(Sequence seq, List list, PrintStream
ps)
    throws IllegalAlphabetException
    {
        RevCompSequence rcSeq = new RevCompSequence(seq);

        // note the funny math here:
        // because bio-string positions are 1-based but String.length()
is
        // 0-based, extracting sequences in rcSeq based on their
offsets in
        // seq is easy. given begin and end offsets on the + strand,
the
        // corresponding offsets on the - strand are (length - end) and
        // (length - beg) for the beginning and end, respectively.
        int seqLen = rcSeq.seqString().length();

        StrandedFeature f = null;      // currently iterated feature
        String letters = null; // subsequence of current feature
        for (Iterator i = list.iterator(); i.hasNext();)
        {
            f = (StrandedFeature) i.next();
            int beg = f.getLocation().getMin() -
this.filterUpstreamLength;
            if (beg < 1)
                beg = 1;
            int end = f.getLocation().getMin() - 1;

            if (f.getStrand().getValue() == Genbank.STRAND_POS)
                letters = seq.subStr(beg, end);
            else if (f.getStrand().getValue() == Genbank.STRAND_NEG)
```

```
                letters = seq.subStr(seqLen - end, seqLen - beg);

            ps.println(">"+f.getAnnotation().getProperty("gene"));
            string2fasta(letters, ps);
        }
    }



    /**
     * Display location of feature on list.
     *
     * @param list gene Features in this sequence
     * @param ps stream to print to
     */
    public void locationsDisplay(List list, PrintStream ps)
    {
        // currently iterated feature
        Feature f = null;
        for (Iterator i = list.iterator(); i.hasNext();)
        {
            f = (Feature) i.next();
            ps.println(">"+f.getAnnotation().getProperty("gene") + " "
+ f.getLocation());
        }
    }



    /**
     * Compare Location minimums.
     *
     * @return -1 if a is < b; 0 if a == b; 1 if a > b.
     */
    private Comparator locationComparator()
    {
        return new Comparator() {
            public int compare(Object o1, Object o2)
            {
                int l1 = ((Location) o1).getMin();
                int l2 = ((Location) o2).getMin();

                if (l1 > l2)
                    return 1;
                else if (l1 == l2)
                    return 0;
                else
                    return -1;
            }
        };

    }
```

```
    /**
     * Compare Features' Location minimums.
     *
     * @return -1 if a is < b; 0 if a == b; 1 if a > b.
     */
    private Comparator featureLocationComparator()
    {

        return new Comparator() {
            public int compare(Object o1, Object o2)
            {
                Comparator c = locationComparator();
                return c.compare(
                    ((Feature) o1).getLocation(),
                    ((Feature) o2).getLocation()
                );
            }
        };

    }



    /**
     * Pull feature types from this sequence so we can find out what
types
     * we can use as filters.
     *
     * @param s a Sequence to filter
     */
    private void featureRecord(Sequence s)
    {
        if (this.features == null)
            this.features = new HashSet();

        for (Iterator i = s.features(); i.hasNext(); )
        {
            Feature f = (Feature) i.next();
            this.features.add(f.getType());
        }
    }



    /**
     * FASTA format a string.
     *
     * @param s string to format
     * @param ps stream to write to
     */
    public void string2fasta(String s, PrintStream ps)
    {
        // entire sequence on one line
        if (this.fastaLineLength < 1)
        {
```

```
            ps.println(s);
            return;
        }

        int max = s.length();
        for (int j = 0; j < max; j += this.fastaLineLength)
        {
            int eol = j + this.fastaLineLength;  // end of current line
            if (eol > max) eol = max;            // end of last line
            ps.println(s.substring(j, eol));     // print it
        }

    }



    /**
     * Display features which were captured with {@link #featureRecord}
     *
     * @param ps stream to write to
     */
    public void featuresDisplay(PrintStream ps)
    {
        if (this.features == null)
            return;

        for (Iterator i = this.features.iterator(); i.hasNext();)
            ps.println((String) i.next());
    }
}
```

GenbankDriver.java

```
package edu.dartmouth.bglab.pipeline.util;

import org.apache.commons.cli.*;


/**
 * Genbank file utility methods.
 *
 * A sample file is available here:
 * http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html
 *
 * 2005-08-25, zak.burke@dartmouth.edu
 *
 */
public class GenbankDriver
{
    /**  */
    public BglabCommandLine cli = null;

    /** command line options: input filename */
    public static String INPUT     = "i";

    /** command line options: return all sequence substrings */
    public static String FILTER_ALL   = "p";

    /** command line options: return only substrings of genes */
    public static String FILTER_GENIC = "q";

    /** command line options: return only non-gene substrings */
    public static String FILTER_BG    = "r";

    /** command line options: return only non-gene substrings */
    public static String FILTER_UP    = "u";

    /** command line options: length of sequence to extract upstream of
gene features */
    public static String UPSTREAM_LENGTH = "v";

    /** command line options: skip repeats in FILTER_ALL and FILTER_BG
*/
    public static String SKIP         = "s";

    /** command line options: reset fasta line length */
    public static String LINE_LENGTH  = "z";

    /** command line options: display distinct features from input
sequence */
    public static String FEATURES  = "f";

    /** command line options: display genic locations from input
sequence */
    public static String LOCATIONS = "l";
```

```
/** command line options: show help */
public static String HELP      = "h";



public static void main(String[] args)
{
    GenbankDriver gd = new GenbankDriver();
    Genbank g = new Genbank();

    // available command line options
    Options options = gd.optionsInit();
    try
    {
        gd.init(args, options, g);

        // input sequence filename
        String file = gd.cli.value(GenbankDriver.INPUT);

        g.toFasta(file, System.out);

        // show features if requested
        if (gd.cli.booleanValue(GenbankDriver.FEATURES))
            g.featuresDisplay(System.err);

    }
    // command line parse error; show help and EXIT.
    catch (ParseException e)
    {
        gd.usage(options, "ERROR: " + e.getMessage() + "\n", 1);
    }

    // file parsing error
    catch (Exception e)
    {
        System.err.println(e.getMessage());
        e.printStackTrace();
    }

}



/**
 * default constructor does nothing at all.
 *
 */
public GenbankDriver()
{}



/**
 * Parse command line options.
```

```
     *
     * @param args command line argument array
     * @param options available options
     *
     * @return options as parse from the command line
     */
    private void init(String[] args, Options options, Genbank g)
    throws ParseException
    {
        // options as parsed from the command line
        // throws error on missing option and missing option value
        CommandLine cmd = new PosixParser().parse(options, args);


        // ERROR: we can never get to this code if one of the required
options
        // is missing, which means that calling this application with
just "-h"
        // will always exit with a non-zero error code. I can't figure
out
        // how to short-circuit the parsing algorithm to check for help
first,
        // though it seems like there ought to be a way to do that.

        // user needs help; show it and exit.
        if (cmd.hasOption("h"))
            usage(options, null, 0);

        this.cli = new BglabCommandLine(cmd);

        // not set options on t
        try
        {
            if (this.cli.booleanValue(GenbankDriver.LINE_LENGTH))
                g.fastaLineLength =
this.cli.intValue(GenbankDriver.LINE_LENGTH);
        }
        catch (Exception e)
        {
            System.err.println("Couldn't parse " +
GenbankDriver.LINE_LENGTH + " value; using default.");
        }


        // record distinct features
        if (this.cli.booleanValue(GenbankDriver.FEATURES))
            g.logFeatures = true;

        // record distinct features
        if (this.cli.booleanValue(GenbankDriver.LOCATIONS))
            g.logLocations = true;

        // select features to display
        if (this.cli.booleanValue(GenbankDriver.FILTER_GENIC))
            g.filterGenic = true;
```

```
        else if (this.cli.booleanValue(GenbankDriver.FILTER_ALL))
            g.filterAll = true;
        else if (this.cli.booleanValue(GenbankDriver.FILTER_BG))
            g.filterBg = true;
        else if (this.cli.booleanValue(GenbankDriver.FILTER_UP))
            g.filterUpstream = true;

        // if FILTER_UP, check
        if (this.cli.booleanValue(GenbankDriver.FILTER_UP))
            if (this.cli.hasOption(GenbankDriver.UPSTREAM_LENGTH))
                g.filterUpstreamLength =
this.cli.intValue(GenbankDriver.UPSTREAM_LENGTH);

    }



    /**
     * Explain how to use this program, then exit.
     *
     * @param options list of command line options
     * @param message error message, if any
     * @param exitCode error code in case we got here from an Exception
     */
    protected void usage(Options options, String message, int exitCode)
    {
        if (message != null)
            System.err.println(message);

        HelpFormatter formatter = new HelpFormatter();
        formatter.printHelp( "Genbank", options );

        System.exit(exitCode);

    }



    /**
     * Configure options which can be passed in on the command line.
     *
     * @return list of options
     */
    protected Options optionsInit()
    {
        Option o = null;
        Options options = new Options();

        // input: name of genbank file to read
        o = new Option(GenbankDriver.INPUT, "input", true, "name of
file to read");
        o.setArgName("gbk-file");
        o.setRequired(true);
        options.addOption(o);
```

```
        // filter what features to display
        OptionGroup g = new OptionGroup();
        g.setRequired(true);
        g.addOption(new Option(GenbankDriver.FILTER_ALL,   "filter-
all",   false, "return all sequence subsets"));
        g.addOption(new Option(GenbankDriver.FILTER_GENIC, "filter-
genic", false, "return gene features only"));
        g.addOption(new Option(GenbankDriver.FILTER_BG,    "filter-bg",
false, "return non-gene features only"));
        g.addOption(new Option(GenbankDriver.FILTER_UP,    "filter-up",
false, "return sequences upsream of gene features only"));
        options.addOptionGroup(g);


        // locations: whether to display genic locations in input
sequence
        options.addOption(GenbankDriver.LOCATIONS, "locations", false,
"display genic locations in input sequence");

        // output line length
        o = new Option(GenbankDriver.LINE_LENGTH, "line-length", true,
"line length of output; 0 for single line");
        o.setArgName("line-length");
        options.addOption(o);

        // output line length
        o = new Option(GenbankDriver.UPSTREAM_LENGTH, "upstream-
length", true, "length of sequence to extract upstream of gene
features");
        o.setArgName("line-length");
        options.addOption(o);

        // features: whether to display the distinct features in input
sequence
        options.addOption(GenbankDriver.FEATURES, "features", false,
"display distinct features in gbk file");

        // locations: whether to display genic locations in input
sequence
        options.addOption(GenbankDriver.LOCATIONS, "locations", false,
"display genic locations in input sequence");

        // help: whether to display options and quit
        options.addOption(GenbankDriver.HELP, "help", false, "display
these instructions");

        return options;
    }

}
```