27 March 2005

Pairwise Sequence Comparison Evaluation Tools

Copyright (c) 2002-2005
Ed Green <ed@compbio.berkeley.edu>,
Gavin Price <gavin@compbio.berkeley.edu>,
Gavin Crooks <gec@threeplusone.com>,
Steven Brenner <brenner@compbio.berkeley.edu>
University of California, Berkeley

Summary of Contents

I. Obtain

II. Install

III. Look

IV. Understand

V. Use (the impatient should start here)

VI. Complain

I. Obtain

The most recent version of this software can be obtained by visiting
http://compbio.berkeley.edu/people/ed

To make full use of this software, you will also need to get and install the following perl modules. You can get these from http://www.cpan.org

1. Math::Random::MT
2. Math::Round
3. Statistics::Descriptive
4. Bioperl

To make the plots described in the examples, you will need to install gnuplot. You can get that from http://www.gnuplot.info

You may also want to look at the plots. That will require a postscript viewer, like ghostscript, gv, or ggv.

II. Install

This package was developed and used under Perl-5.6.1 and Perl-5.8.2 and RedHat Linux 7.3 and RedHat Linux 9.0. I can't think of any reason why this code wouldn't work in other environments, but make no claims that it will.

Installation is very straigtforward. Just do this:

```
gzip -d psce.tar.gz
tar -xvf psce.tar
```

Then, add the `psce/bin/` directory to your `PATH`, so that you can call these programs from the command line without having to specify the complete path.

The modules that these programs use are in the `psce/pms/` subdirectory. You will need to make sure that the modules are in your perl `@INC` when you run these programs. One easy way to do that is to add the `psce/pms/` directory to your enviromental variable, `PERL5LIB`.

For example, if you unpacked this package in your home directory, `/home/me/`, and your shell is tcsh, you could do this:

```
setenv PERL5LIB /home/me/psce/pms:$PERL5LIB
```

For added convenience, add the above line to your `.tcshrc` file.

That's it!

The shabang lines of these scripts is

```
#!/usr/bin/env perl
```

This is a pretty good way of calling perl in various environments, but you may have a reason to change this. If so, go ahead.

Running any of the scripts without arguments describes their invocation syntax.

III. Look

This package contains many tools and example data for performing some of the analyses described in

> Green RE and Brenner SE, "Bootstrapping and normalization for enhanced evaluations of pairwise alignment methods", 2002, Proc IEEE.
>
> and
>
> Price GA, Crooks GA, Green RE, and Brenner SE, "Statistical Evaluation of Pairwise Sequence Comparison with the Bayesian Bootstrap" (in prep)

Programs in `psce/bin/`

More detailed usage information can be had by invoking any of these without parameters.

`astral_stats.pl`
Gives some basic information about the size and superfamily content of an astral database.

`concertedBootStrap.pl`
Takes as input one or more `dbvdbsum.pl` output files and generates a user-defined number of Bayesian bootstrap replications.

`cve_automator.pl`
Runs cve_maker.pl multiple times to generate CVE data on the bootstrap replicates generated from `concertedBootStrap.pl`

`cve_maker.pl`
Generates CVE data from a `dbvdbsum.pl` output file.

`dbvdbsum.pl`
Summarizes the results of a database versus database search.

`diffDist.pl`
Performs statistical tests of the difference between two Bayesian bootstrap distributions.

`e_finder.pl`
Generates a list of the hits from a database versus database search above a given errors per query rate.

`EPQvScore.pl`
Generates a comparison of the observed rate of false-positive generation and the e-value statistical scores of a given pairwise comparison method. Used to evaluate the statistical scoring routine of that paiwise comparison method.

`maxCov.pl`
Generates a list of coverages at a given error rate given a set of `cve_maker.pl` output files.

`seq_diff.pl`
Displays the number of hits unique to one pairwise comparison method relative to another pairwise comparison method. Requires the output of `e_finder.pl`.

`sfSizeDist.pl`
Generates a distribution of superfamily sizes within an input ASTRAL database.

`sfSizePerf.pl`
Generates data showing how well a given pairwise comparison method was able to find the relations within superfamilies of increasing size.

`splitAstralByFold.pl`
Splits an ASTRAL database into two rougly equally sized databases. Useful for creating a training database to train parameters and a test database to test methods. This helps avoid over-fitting parameters to a particular dataset.

Modules in `psce/pms/`

More detailed information about each of these can be had using `perldoc`.

`BBNorm.pm`
Contains methods for doing the Bayesian bootstrap.

`DBVDB.pm`
Parses `dbvdbsum.pl` output and makes the data available through its methods.

`FamNorm.pm`
An module that takes an ASTRAL database as input and analyzes the superfamily level relations within it.

IV. Understand

Below is a general flow-chart of the steps in evaluating pairwise sequence comparison methods:

1. Get to know your datasets.

2. Do the database versus database search with whatever method, matrix, parameter set, etc. that you want to evaluate.
You're on your own for this part. The only requirements are that all sequences in the database be used, individually, as query to search the database and that each of these queries generates a discrete output file. Also, these output files must be accessible by recursively descending down from a single subdirectory, i.e., they can't be all over the place. The easiest way to do this is just write them all to a single directory.

3. Run `dbvdbsum.pl`.
This program summarizes the results of your database versus database search. For an example of its output, check out psce/ex/blast.sum

In short, it looks at the ASTRAL database you used and figures how many hits you should find. Then it goes through all your output files and writes a single line per hit. Each of these lines has the query sequence id, the id of the sequence that was hit, the e-value of the hit, a number between 0 and 3 indicating the level of SCOP relatedness between these two sequences, and then the weighting factors for normalization. The file formats supported for parsing are currently pretty limited, but it's easy enough to add more. Just add another entry to the % FORMATS at the beginning, then in processThisResult(), add another elsif clause for whatever output format your method writes and parse it accordingly. Alternatively, you can write your own code for generating this file. Its format is simple and the module, FamNorm.pm, will help you generate the information in the header lines.

4. Run `cve_maker.pl` and plot the ouput.
This program generates the CVE data. It also reports each false positive hit.

5. Bootstrap the output using `concertedBootstrap.pl`.
Perform both a Bayesian and standard bootstrap.

6. Make CVE files for each bootstrap file with `cve_automator.pl`.

7. Analyze the bootstrap distributions.

V. Use

NOTE: To generate the plots described in this section, you'll need gnuplot (http://www.gnuplot.info/). If you don't have gnuplot, go get it.

This section describes, by example, the analyses you can do with these tools. This purpose of this document is to show you how to use this software, not to explain why you would want to. If you would like an answer to that question, see

http://compbio.berkeley.edu/people/ed/SeqCompEval

Before we begin the tutorial, notice that I've included all of the output files that are generated by this tutorial. One way to use this tutorial is to just look at the commands and the output that is generated by each without actually executing the command. Or, you can follow along and type in the commands. Be aware, though, that if something goes wrong, you may over-write the given output with something else. No big deal, just something to think about.

To start, go to the `psce/db/` subdirectory and look for a file called `astral-1.61-40.fa`. This file comes from the ASTRAL database of sequences (http://astral.berkeley.edu).
This means that these sequences derive from solved structures and have been classified in SCOP (http://scop.mrc-lmb.cam.ac.uk/scop/).

What does all that mean? It means that the evolutionary relationships between all these sequences are known. If you take a look at this file, you'll see the first sequence, d1dlwa_, is followed by the SCOP classification string a.1.1.1. This string tells us that d1dlwa_ has been classified thusly:

class: (a) All alpha proteins
fold: (1) globin-like
superfamily: (1) globin-like
family: (1) truncated hemoglobin

The classification represented by this string lets us relate d1dlwa_ to all the other domain sequences in this database.

This database, `astral-1.61-40.fa` comes from version 1.61 of the ASTRAL database, filtered at 40% id. You could go to astral.berkeley.edu and download these sequences yourself, but I've already done it for you. The database was further processed with the `seg` program to remove low-complexity regions. Specifically, `seg` was called like this:

```
seg -w 12 -t 1.8 -e 2.0 -xT
```

Don't worry if you don't have the `seg` program. I've included the output of this

filtering: `astral-1.61-40.s.fa`.
Now, let's split that database into two non-overlapping databases with the program `splitAstralByFold.pl`

```
$ splitAstralByFold.pl -d astral-1.61-40.s.fa
```

This creates two non-overlapping databases, with the .A and .B suffixes. This is a good thing to do if you want to train some sequence comparison method on one database and then test it on another database to avoid over-fitting parameters. I did it for this example so we can work with smaller files.

Now, let's have a look at the two smaller databases we've created: `astral-1.63-40.s.fa.A` and `astral-1.63-40.s.fa.B`

Run the program `astral_stats.pl` on each one:
```
$ ../bin/astral_stats.pl -d astral-1.61-40.s.fa
For the astral database: astral-1.61-40.s.fa
There were 4774 sequences.
There were 4310 sequences in superfamilies with
      more than one member
There were 1109 superfamilies.
There were 98722 superfamily level relations

$ ../bin/astral_stats.pl -d astral-1.61-40.s.fa.A
For the astral database: astral-1.61-40.s.fa.A
There were 2592 sequences.
There were 2356 sequences in superfamilies with
      more than one member
There were 581 superfamilies.
There were 48864 superfamily level relations

$ ../bin/astral_stats.pl -d astral-1.61-40.s.fa.B
For the astral database: astral-1.61-40.s.fa.B
There were 2182 sequences.
There were 1954 sequences in superfamilies with
      more than one member
There were 528 superfamilies.
There were 49858 superfamily level relations
```

This gives us some general information about each database.

Now, let's look at the distribution of superfamily sizes within the `astral-1.61-40.s.fa.A` database using the `sfSizeDist.pl` program. Go to the `psce/ex/` subdirectory and run `sfSizeDist.pl` like this:
```
$ ../bin/sfSizeDist.pl -a ../db/astral-1.61-40.s.fa.A >
sfSizeDist/sfSizeDist.dat
```

This makes a data file with the distribution of superfamily sizes. I've written a gnuplot script that makes a nice graphical output of this data. If you have gnuplot installed, run the script like this:

```
$ gnuplot sfSizeDist/sfSizeDist.gs
```

This makes a nice plot, `sfSizeDist/sfSizeDist.eps`. Take a look at it with your favorite postscript viewer and see how most superfamilies have very few sequences in them. Also, notice that the most populous superfamily size is one. For our purposes, this is fairly important. The sequences in superfamilies of size one have no evolutionary relationship to any other sequence in this database. Therefore, the hits returned when we search the database with these sequences can only be false positives. They are, in a sense, decoys.

Now that we're familiar with our databases, let's do some evaluation of pairwse sequence comparison methods. In the `psce/ex/` directory there are two subdirectories `blastpgp-2.2.4_astral-1.61-40.s.fa.A` and `ssearch-3.4t22_astral-1.61-40.s.fa.A` that contain pairwise sequence comparison results that we'll evaluate.
These are the output files that were generated when I searched the `astral-1.63-40.s.fa.A` database against itself, using `blastpgp`, version 2.2.4 and `ssearch` version 3.4t22. That is, each of the 2782 sequences in that database was used as a query to search the same database using these two different programs. I should also point out that `blastpgp` was run with default options, whereas `ssearch` was run using a substitution matrix and gap parameters that I have previously found to be optimal. So this comparison could be described as asking how well we can detect remote homologs using optimized Smith-Waterman versus using blast off-the-shelf.

Each of these directories contains 2782 files, one per database search. These files serve as the input for the subsequent analyses. For your own analyses, you will be on your own for generating these. The requirements for these input sets are:

1. There must be an output file for each sequence in the database, even if there were no significant hits.
2. The output filename must begin with the 7 character ASTRAL identifier of the query sequence.
3. All the output files must be accessible by recursively searching down from a single subdirectory.

These example sets suit all these requirements. So, they are suitable input for `dbvdbsum.pl`, which summarizes the results of a database versus database search.

You run `dbvdbsum.pl` on the files in these directory, like so:

```
$ ../bin/dbvdbsum.pl -q ../db/astral-1.61-40.s.fa.A -r bz2
-d blastpgp-2.2.4_astral-1.61-40.s.fa.A/ -a 1 -f B >
sum/blast.sum

$ ../bin/dbvdbsum.pl -q ../db/astral-1.61-40.s.fa.A -r bz2
-d ssearch-3.4t22_astral-1.61-40.s.fa.A/ -a 1 -f F >
sum/ssearch.sum
```

This takes about 10 minutes on my machine. This program, `dbvdbsum.pl`, reports on all the hits in all the results files in the directory specified by the -d flag. If you set the -a flag to a true value, then it also checks to make sure there is an output file for each sequence in the database and gives information on the level of SCOP relatedness of each hit, which is what we want. Take a look at the output files, `psce/ex/sum/blastp.sum` and `psce/ex/sum/ssearch.sum`
These files have a header region with some information about how it was run and how many relations there are to be found. After this, they have many lines describing all the hits found.

We can look at these output files to see if there are particular hits that we may be interested in. Even better, though, we can run `cve_maker.pl` using these files as input to generate coverage versus errors per query (CVE) data.

Run `cve_maker.pl` on these files, like so:

```
$ ../bin/cve_maker.pl -i sum/blast.sum -o cve/blast.sum.dat
-v 1 > fp/blast.fp
$ ../bin/cve_maker.pl -i sum/ssearch.sum -o
cve/ssearch.sum.dat -v 1 > fp/ssearch.fp
```

Each invocation of `cve_maker.pl` generates two files. One of these has the CVE data, the other has a list of the false positives. Take a look at `psce/ex/fp/blast.fp`

This file has the false positives that blast found, in order, from most significant to least significant. That is, it's a list of the sequences that blast says are similar, but SCOP tells us are not related.

The file has four columns of data:

query_id	hit_id	e-value	relatedness

where relatedness is either 0 (differenct SCOP class)
or 1 (same class, different fold).

We can use these data to see how good the stastical scores (e-values) are for these programs. We can plot the number of errors (false positives) at increasing e-values using the false-positives data, using the `EPQvScore.pl` program. Run it on the blast and ssearch data, like this:

```
$ ../bin/EPQvScore.pl -a ../db/astral-1.61-40.s.fa.A -f
fp/blast.fp > EPQvScore/epqvsc-blast.dat

$ ../bin/EPQvScore.pl -a ../db/astral-1.61-40.s.fa.A -f
fp/ssearch.fp > EPQvScore/epqvsc-ssearch.dat

$ gnuplot EPQvScore/epqvsc.gs
```

The resulting plot shows that ssearch does a very good job of estimating significance. Across the range of e-values shown, there were about the same number of false-positives as ssearch predicted there would be. The blast program, on the other hand, tends to under estimate significance; it generates fewer false positives at all error ranges than it believes it's generating.

Now, let's look at the CVE data. Take a look at
`psce/ex/cve/blast.sum.dat`
It has the sets of CVE data that you can use with gnuplot to make a nice CVE plot. Each line in this file contains the x and y coordinates of a single datapoint on the CVE plots. Datasets are separated by two blank lines -- gnuplot style. The first dataset is the unnormalized CVE data. The second is linearly normalized and the third is quadratically normalized.

Great. Now, let's make some CVE plots. I've written a script for gnuplot that will plot the CVE data we just generated. Take a look at, `psce/ex/cve/cve.gs`
then, run it like this:

```
$ gnuplot cve/cve.gs
```

This makes another postscript file, `psce/ex/cve/CVE.eps`
Take a look at it.

See how much more coverage optimized ssearch generates compared with off-the-shelf blast? To see exactly how much more coverage, at a given error rate, use the program `maxCov.pl`, like so:

```
$ ../bin/maxCov.pl -i cve/ -r dat
Unnormalized:
ssearch.sum.dat => 0.096267
blast.sum.dat => 0.079711
Linearly normalized:
ssearch.sum.dat => 0.235397
blast.sum.dat => 0.195182
Quadratically normalized:
```

```
ssearch.sum.dat => 0.367154
blast.sum.dat => 0.307481
```

Also, notice that for both blast and ssearch, coverage increases if the results are normalized. That is, if we down-weight the results from larger superfamilies, the amount of coverage increases. This tells us that the relations in larger superfamilies are more difficult to detect. Next, we'll look at that phenomenon directly.

Let's see how easy or difficult is was for blast to find superfamily level relations in superfamilies of increasing size. The script `sfSizePerf.pl` does this. Run it like this:

```
$ ../bin/sfSizePerf.pl -a ../db/astral-1.61-40.s.fa.A -s
sum/blast.sum > sfSizePerf/sfSizePerf-blast.dat
```

Again, the output file, `psce/ex/sfSizePerf/sfSizePerf-blast.dat`, is suitable as a datafile for gnuplot. If we look at this file, we see the first section lists all superfamilies, by size, with the number of correctly identified relations, incorrectly identified relations, and the number that were undetermined. Undetermined means that the two sequences involved were in the same fold, but not the same superfamily. The next section has the total coverage for superfamiles of each size. This is the data we'll plot. To view these data graphically, run gnuplot with the provided script:

```
$ gnuplot sfSizePerf/sfSizePerf.gs
```

Taking a look at the output file, `psce/ex/sfSizePerf/sfSizePerf-blast.eps`, you can see that there is a general negative correlation between superfamily size and ability to detect pairwise relations. That is, the pairwise relations in large superfamilies are more difficult to detect than those in smaller superfamilies. Why? I don't know. You may also notice that there are several superfamily sizes that are not in line with the general trend. Size 22, for example, has superfamilies whose relations are relatively easy to detect. Looking at the input file, `psce/ex/sfSizePerf/sfSizePerf-blast.dat`, we see that there is a single superfamily of that size: d.144.1, which is the PK-like superfamily. Why are the sequences in this superfamily relatively easy to detect? Again, I don't know.

We can look at the CVE plots and see that ssearch is finding more remote homologs than blast. We can use `maxCov.pl` to see how much more coverage at a given error-rate. But how do we know if the difference is significant? One way to determine this is to bootstrap the results. This gives us a measure of the variance of each result. We can then do a statistical test to see if the difference in coverage at a given error rate is significant.

We'll use the `dbvdbsum.pl` output files to generate bootstrapped data sets with `concertedBootStrap.pl`. This program, which replaces the `bootStrap.pl` program from previous versions of this software, bootstraps the results of a database versus database search. It does this by randomly assigning weights to each sequence in that database and then making a `dbvdbsum.pl` style output file of the results. This Bayesian bootstrap technique replaces the previously described standard bootstrap technique, which did simple random sampling with replacement of sequences in the database. The standard bootstrap was described in the manuscript, "Bootstrapping and normalization for enhanced pairwise sequence comparison." As mentioned there, there is an unfortunate bias in bootstrapping this way: some smaller superfamiles, by chance, do not get sampled at all. Because of the aforementioned tendency for the easier to detect relations to occur in smaller superfamilies, this creates a bias in the standard bootstrap method. The effect of this bias can be seen by comparing the mean of the bootstrapped coverage distribution with the coverage of the original data. The bootstrapped mean is less because some of the easier relations fall out. To address this, we implemented the Bayesian bootstrap. Because all sequences are present in the Bayesian bootstrap (just with different weights) there is no such bias. See Price et al. for a more thorough description of this.

Efron, the Father of the Bootstrap, gives a rule of thumb of 200 bootstrap replicates to get a reliable distribution. For the sake of keeping this tutorial down to a managable size, however, I am only going to run 10 for each database versus database search. If you run this yourself and do 200 replicates, beware that it generates A LOT of output. Run `concertedBootStrap.pl` like this:

```
$ ../bin/concertedBootStrap.pl -a ../db/astral-1.61-
40.s.fa.A -n 10 -s 35 sum/blast.sum sum/ssearch.sum
```

This will generate ten `dbvdbsum.pl` style files in the same directory as the original `dbvdbsum.pl` file. They will have the same name as the original and the suffix `.sum.bbs.X`, where `X` is the bootstrap replicate number. Take a look at one and you'll see that it's identical in format to a standard `dbvdbsum.pl` output file.

Next we'll use `cve_automator.pl` to make CVE files for each bootstrap file. Again, a new output cve data file is made for each bootstrap replicate. So, if you have done a lot of these, you're going to get a lot more data here. `cve_automator.pl` just runs `cve_maker.pl` multiple times so the user doesn't have to run it for each bootstrap file.

Invoke `cve_automator.pl` thusly:

```
$ ../bin/cve_automator.pl -i blast.sum -n 10 -d sum/ -c cve/
$ ../bin/cve_automator.pl -i ssearch.sum -n 10 -d sum/ -c
cve/
```

The output files can be seen in the `psce/ex/cve` subdirecoty. They are identical (format-wise) to the file generated earlier by `cve_maker.pl`.

Now let's plot the whole mess with the custom gnuplot script provided.
gnuplot bootstrap.gnuscript

```
$ gnuplot cve/bootstrap.gs
```

Take a look at the output plot, `cve/bbs.eps`. It shows the CVE lines generated by the original data plus all CVE data from the bootstrap replicates. The bootstrap distribution falls in a fairly narrow band around the orignal CVE lines. These distributions can now be used to statistically compare these two methods.

Use `diffDist.pl` to compare the bootstrap distributions from two bootstrap distributions. This program goes through all the bootstrap CVE (`.bbs.X.dat`) files for two bootstrap distributions and does some statistical tests. The goal of these tests is to determine if the distributions are significantly different. The details of this method can be found in the manuscripts mentioned above. In brief, 95% confidence intervals are generated, by several methods. If the value 0 is within the confidence interval, then you can not reject the null hypothesis that these two distributions are actually the same. Run `diffDist.pl` like this:

```
$ ../bin/diffDist.pl -i ssearch.sum -I cve/ -j blast.sum -J
cve/ -e 0.01 -g dd.gs -b 10 -s .dat -h 0.025
Expected value for coverage difference at 0.01 EPQ is
0.0417932
Standard error for coverage difference at 0.01 EPQ is
0.0039175565740792
Original value of CVE line at 0.01 EPQ is 0.040215

Data min/max: [0.03713 , 0.048241]
Parameterized confidence interval: [0.0341147891148048 ,
0.0494716108851952]
CI walking from mean: [0.03713 , 0.048241]
CI walking from CVE line: [ , 0.047794]
CI walking in from max/min data: [0.03713 , 0.048241]
```

You can see that by none of the 95% confidence interval generating methods is the value 0 in the confidence interval. Therefore, we can reject the null hypothesis and conclude that these are two separate distributions. In other words, at 0.01 errors per query, optimized ssearch finds a statistically significant greater fraction of remote homologs than does off-the-shelf blast. You can plot the histogram of coverage differences at 0.01 errors per query, in `dd.dat` by calling gnuplot on the gnuplot script `dd.gs`.

Now, let's look at one final way to compare the results of these two pairwise search methods. Let's extract e-values, hits, and false positives at a particular error rate for both and compare the results. Run `e_finder.pl` like this:

```
$ e_finder.pl -i sum/blast.sum -v -l -h -e 0.01 >
e_finder/blast.ef
$ e_finder.pl -i sum/ssearch.sum -v -l -h -e 0.01 >
e_finder/ssearch.ef
```

This makes two files that have each discrete E-Value at each error per query rate, up to 0.01, and all the hits and false positives at that e-value.

The false positives can be saved by replacing the -h flag with the -v flag. To list only the unique hits in each set, use `seq_diff.pl`.

```
$ seq_diff.pl -i e_finder/blast.ef -j e_finder/ssearch.ef
```

With a little UNIX magic, you can easily count the number of unique hits from each file:

```
$ seq_diff.pl -i e_finder/blast.ef -j e_finder/ssearch.ef |
grep blast | wc -l
    102
$ seq_diff.pl -i e_finder/blast.ef -j e_finder/ssearch.ef |
grep ssearch | wc -l
    911
```

This shows us that the blast dbvdbsum file contains 102 hits not contained in the ssearch file and the ssearch file contains 911 unique hits.
One final note: Some of these tools require that their input filenames have certain features. If you stick closely to the examples described above, you should not run into problems with this.

VI. Complain

You can contact me at ed@compbio.berkeley.edu or edgreen@ugaalum.uga.edu

You can contact Gavin (the author of the Bayesian bootstrap stuff) at gavin@compbio.berkeley.edu. He's a nasty little troll, though, so don't be surprised if his response consists only of 'RTFM.'

There is no manual.

I'm very interested in results generated by these or related tools or suggestions for improvement or new features.