BUCKy

Bayesian Untangling of Concordance Knots
(applied to yeast and other organisms)

Version 1.1, 30 October 2006

Departments of Statistics and of Botany
University of Wisconsin - Madison
Medical Sciences Center,
1300 University Ave. Madison, WI 53706, USA.

Introduction

BUCKy is a program to analyze a multi-locus data sets with Bayesian Concordance Analysis (BCA), as described in Ané *et al.* (2006). This method accounts for biological processes like hybridization, incomplete lineage sorting or lateral gene transfer, which may result in different loci to have different genealogies. With BCA, each locus is assumed to have a unique genealogy, and different loci having different genealogies. The *a priori* level of discordance among loci is controlled by one parameter α.

BCA works in two steps: First, each locus is to be analyzed separately, with MrBayes for instance. Second, all these separate analyses are brought together to inform each other. BUCKy will perform this second step. BUCKy comes into two separate programs: `mbsum` and `bucky`. The first program `mbsum` summarizes the output produced by MrBayes from the analyses of a individual loci. The latter, `bucky`, takes the summary produced by `mbsum` and performs the second step of BCA. These two programs were kept separate because `mbsum` is typically run just once, while `bucky` might be run several times independently, with or without the same parameters.

Installation and Compilation

BUCKy is a command-line controlled program written is C++. It should be easily compiled and run on any Linux system or Mac OSX.

Installation (Mac OSX 10.4 users). Since Mac OSX 10.4 does not come with a C++ compiler, we can provide an executable file that compiled with OSX 10.4, upon request.

Installation (Linux or Mac OSX, version 10.3.9 or below). Pick a directory where you want the BUCKy code to be. This directory will be called `$BUCKY_HOME` in this documentation. Download the `bucky-1.1.tgz` file and put it in `$BUCKY_HOME`. To open the compressed tar file with the BUCKy source code and example data, do these commands:

```
cd $BUCKY_HOME
tar -xzvf bucky-1.1.tgz
```

This creates a directory named `BUCKy-1.1` with subdirectories `BUCKy-1.1/data` and `BUCKy-1.1/src`.

Compilation. If you have gcc installed, compile the software with these commands.

```
cd $BUCKY_HOME/BUCKy-1.1/src
make
```

This will compile programs `mbsum` and `bucky`. It is suggested that copies of `mbsum` and `bucky` be put in `~/bin` if this directory is in your path.

If you do not have gcc installed and the executable provided is not working on your system, you need to find the installer for gcc. On a Macintosh (version 10.3.9 or before), it may be in Applications/Installers/Developer Tools .

Running mbsum

Type these commands for a brief help message

```
mbsum --help
```

Purpose and Output. It is advised to have one directory containing the MrBayes output of all individual locus analyses. Typically, in this directory each file of the form `*.t` is a MrBayes output file from one single locus. Use `mbsum` to summarize all files from the same locus. You want `mbsum` to create a file `<filename>.in` for each locus. The extension `.in` just means input (for later analysis by `bucky`). Output files `*.in` from `mbsum` will typically look like the following, containing a list of tree topologies and a tally representing the trees' posterior probabilities from a given locus (as obtained in the first step of BCA).

```
(1,(2,(3,(4,(5,((6,7),8))))));  24239
(1,(2,(3,(4,(5,(6,(7,8)))))));  15000
(1,(2,(3,(4,(5,((6,8),7))))));  2983
(1,(2,(3,((4,5),((6,7),8)))));  2590
(1,(2,((3,((6,7),8)),(4,5))));  2537
(1,(2,((3,(6,(7,8))),(4,5))));  1097
(1,(2,(3,((4,5),(6,(7,8))))));  995
(1,(2,(3,((4,5),((6,8),7)))));  163
(1,(2,(3,((4,((6,7),8)),5))));  145
(1,(2,((3,((6,8),7)),(4,5))));  96
(1,(2,((3,(4,5)),((6,7),8))));  66
(1,(2,(3,((4,(6,(7,8))),5))));  51
(1,(2,((3,(4,5)),(6,(7,8)))));  22
(1,(2,(3,((4,((6,8),7)),5))));  15
(1,(2,((3,(4,5)),((6,8),7))));  1
```

Syntax and Options. To run `mbsum` on a single file, type:

```
mbsum [-h] [--help] [-n #] [-o filename] [--version] <inputfilename(s)>
```

For example, let's say an alignment `mygene.nex` was analyzed with MrBayes with two runs, and sampled trees are in files `mygene.run1.t` and `mygene.run2.t`. These two sample files include, say, 5000 burnin trees. To summarize these 2 runs use

```
mbsum -n 5000 -o mygene mygene.run1.t mygene.run2.t
```

or more generally

```
mbsum -n 5000 -o mygene mygene.run?.t
```

Note: the older version of `mbsum` could only take a single file. It was then necessary to have a single file for each locus, and to combine all independent MrBayes runs from the same locus into a single file. This is no longer necessary. With version 1.02b of `mbsum`, there can be several input files, such as several parallel runs from MrBayes. Here is a description of the available options.

`[-h]` or `[--help]`	prints a help message describing the options and then quits.
`[-n #]` or `[--skip #]`	This option allows the user to skips lines of trees before actually starting the tally tree topologies. The default is 0, i.e *no* tree is skipped. The same number of trees will be skipped in each input file.
`[-o filename]` or `[--out filename]`	sets the output file name. A single output file will be created even if there are multiple input files. The tally combines all trees (except skipped trees) found in all files.
`[--version]`	prints the program's name and version and then quits.

Since `mbsum` needs to be run on all tree files `*.t`, we provide here a way to do so very efficiently. For example, you can choose to run `mbsum` to all `*.t` files and remove the first 1000 trees of each for burnin, with:

```
for X in *.t; do mbsum -n 1000 $X; done
```

Warnings. `mbsum` will overwrite a file named `<filename>.in` if such a file exists. Most importantly, `mbsum` assumes that the same translation table applies to all files, i.e. that taxon 1 is the same taxon across all genes, and taxon 2 is the same taxon across all genes, etc. This is okay as long as taxa appear in the same order in all alignment files. But if not, the BCA would be screwed up with no warning. This shortcoming will be fixed in a later version of `mbsum`.

Running bucky

To run `bucky`, type

```
bucky [-options] <summary_files>
```

For example, after creating all `.in` files with `mbsum` in the same directory, you can run bucky with the default parameters by typing this:

```
bucky *.in
```

Options. Next, we describe the available options and the output of the program.

`[-o output-file-root]`	Use this option to change the names of output files. Default is `run1`.
`[-a alpha]`	α is the *a priori* level of discordance among loci. Default α is 1.
`[-n number-MCMC-updates]`	Use this option to increase the number of updates (default: 100,000). An extra number of updates will actually be performed for burnin. This number will be 10% of the desired number `n` of post-burning updates. The default, then, is to perform 10,000 updates for burnin, which will be discarded, and then 100,000 more updates.
`[-h]` or `[--help]`	Prints a help message describing options, and then quits.
`[-c number-chains]`	Use this option to run Metropolis coupled MCMC (or MCMCMC), whereby hot chains are run in addition to the standard (cold) chain. These chains occasionally swap states, so as to improve their mixing. The option sets the total number of chains, including the cold one. Default is 1, i.e. no heated chains.
`[-r MCMCMC-rate]`	When Metropolis coupled MCMC is used, this option controls the rate r with which chains try to swap states: a swap is proposed once every r updates. Default is 100.
`[-m alpha-multiplier]`	Warm and hot chains, in MCMCMC, use higher values of α than does the cold chain. The cold chain uses the α value given by the option `-a`. Warmer chains will use parameters $m\alpha, m^2\alpha, \ldots, m^{c-1}\alpha$. Default m is 10.
`[-s subsample-frequency]`	Use this option for thinning the sample. All post-burnin samples will be used for summarizing the posterior distribution of gene-to-tree maps, but you may choose to save just a subsample of these gene-to-tree maps. One sample will be saved every s updates. This option will have an effect only if option `--create-sample-file` is chosen. Default is 1, i.e. no thinning.
`[-s1 seed1]`	Default is 1234. FIXIT: Explain why 2 seeds??
`[-s2 seed2]`	Default is 5678.

`[--create-sample-file]`	Use this option for saving samples of gene-to-tree maps. Default is to NOT use this option: samples are not saved, although a file `.sample` is created. Saving all samples can slow down the program.
`[--use-independence-prior]`	Use this option if you want to assume *a priori* that loci choose their trees independently of each other. This is equivalent to setting $\alpha = \infty$. Default is to NOT use this option.
`[--calculate-pairs]`	Use this option if you want to calculate the posterior probability that pairs of loci share the same tree. Default is to NOT use this option.
`[--use-update-groups]`	Use this option if you want to permit all loci in a group to be updated to another tree. Default is to use this option, as it improves mixing.
`[--no-use-update-groups]`	Use this option to disable the update that changes the tree of all loci in a group in a single update. Default is to NOT use this option. If both options `--use-update-groups` and `--no-use-update-groups` are used, only the last one is applied. No warning is given, but the file `rootname.out` indicates whether group updates were enabled or disabled.

Output. Running `bucky` will create a bunch of output files. With defaults parameters, these output files will have names of the form `run1.*`, but you can choose you own root file name. The following output files describe the input data, input parameters, and progress history.

`run1.out`	Gives the date, version (1.1), input file names, parameters used, running time and progress history. If MCMCMC is used, this file will also indicate the acceptance history of swaps between chains.
`run1.input`	Gives the list of input files. There should be one file per locus.
`run1.single`	Gives a table with tree topologies in rows and loci in columns. The entries in the table are posterior probabilities of trees from the separate locus analyses. The is a one-file summary of the first step of BCA.
`run1.gene`	This file provides almost the same information as `run1.single`, but differently. For each locus, topologies supported by the locus are listed along with their posterior probabilities. In this file, topologies indices are indicated, instead of parenthetical representations.
`run1.top`	Gives a table with tree topologies in rows. Entries are indices of bipartitions (or splits) that are present in the tree topologies.
`run1.splits`	Splits key giving the correspondence between bipartitions (splits) and their indices used in the file `run1.top`.

The following files give the full results as well as various summaries of the results.

`run1.sample`	Gives the list of gene-to-tree maps sampled by `bucky`. With n post-burnin updates and subsampling every s steps, this file contains n/s lines, one for each saved sample. Each line contains the number of accepted updates (to be compared to the number of genes), the number of clusters in the gene-to-tree map (loci mapped to the same tree topology are in the same cluster), the log-posterior probability of the gene-to-tree map (up to an additive constant ??FIXIT), followed by the gene-to-tree map. If there are k loci, this map is just a list of k trees. In file `run1.sample` trees are given by their indices. This file will be created but empty if the option `--create-sample-file` is not set to `true`.
`run1.concordance`	Gives the posterior distribution of concordance factors of clades (bipartitions). The sample-wide concordance factor of a clade is the proportion of loci in the sample who have the clade. However, in this file, concordance factors are expressed in number of loci (instead of proportion of loci). The file `run1.concordance` starts with a list of all clades in the concordance tree (clades with concordance factor greater than 50% and possibly other clades). Then, clades are listed along with their sample-wide concordance factor's posterior distribution and credibility intervals. Clades are sorted by their posterior mean concordance factor.
`run1.cluster`	Gives the posterior distribution of the number of clusters, as well as credibility intervals. A cluster is a group of loci sharing the same tree topology, and loci in different clusters have different tree topologies.
`run1.joint`	Gives a table with topologies in rows and loci in columns. This file is similar to file `run1.single` although topologies are named by their indices rather than with the parenthetical description. Entries are frequencies with which each locus was mapped to each topology.
`run1.genepost`	This file is similar to file `run1.gene`, but contains more information. If gives the list of loci, and for each locus it lists all topologies supported by this locus (topology index and parenthetical description). For each topology is indicated the posterior probability of this topology from the individual gene analysis (like in `run1.gene`) as well as the posterior probability that the locus has this tree given all data from all loci (like in `run1.joint`).
`run1.topologies`	Gives the list of all supported topologies, the arithmetic average across loci of their locus-specific posterior probability, from the individual analyses as well as from the concordance analysis. The interpretation of these numbers is not clear, so we do not recommend using them.
`run1.pairs`	Gives a k by k matrix where k is the number of loci. Entries are the posterior probability that two given loci share the same tree. This file is created only if option `--calculate-pairs` is used.

Examples

The example data provided with the program is organized as follows: directory `$BUCKY_HOME/BUCKy-1.1/data/example1/` contains 10 folders named `ex0` to `ex9`, one for each locus. These 10 folders contain a single file each, named `ex.in`, which was created by `mbsum`. For analyzing these data, one can use the default parameters and type

```
bucky $BUCKY_HOME/BUCKy-1.1/data/example1/ex?/ex.in
```

The question mark will match any character (any digit 0 to 9 in particular), so that all 10 locus files will be used for the analysis. The following commands will run `bucky` twice, with $\alpha = 5$, no MCMCMC, group updates disabled, and one million updates. To have independent runs, seeds are changed. (keep each of these two commands on a single line).

```
bucky -n 1000000 -a 5 -s1 745203 -s2 905423 --no-use-update-groups
                                  $BUCKY_HOME/BUCKy-1.1/data/example1/ex?/ex.in
bucky -n 1000000 -a 5 --no-use-update-groups -s1 4948537 -s2 8764223
                                  $BUCKY_HOME/BUCKy-1.1/data/example1/ex?/ex.in
```

After the first run, a look at the file `run1.genepost` shows that all genes give a 100% estimated posterior probability to tree 1 and 0% estimated posterior probability to other trees. However, after the second run we see that tree 3 receives 100% estimated posterior probability by all loci, instead of tree 1. So the two run are in very strong disagreement. A look at the `run1.concordance` files would have shown this disagreement too. The poor mixing of either previous run is fixed by using the option `--use-update-groups` (or by not using the `--no-use-update-groups` option!).

The yeast data analyzed in Ané *et al.* (2006) is provided with the program and organized as follows. The directory `$BUCKY_HOME/BUCKy-1.1/data/yeast/` contains 106 folders named `y000` to `y105`, one for each gene. In each of these folders, a file created by `mbsum` and named `run2.nex.in` contains the data from one gene. For analyzing these data with $\alpha = 2.5$, $n = 1,000,000$ updates, $c = 4$ chains (one cold and 3 hot chains), and for saving samples once every 1000 updates, one would type (on a single line)

```
bucky -a 2.5 -n 1000000 -c 4 --create-sample-file
                            $BUCKY_HOME/BUCKy-1.1/data/yeast/y???/run2.nex.in
```

General notes

Choosing the *a priori* level of discordance α. To select a value based on biological relevance, the number of taxa and number of genes need to be considered. For example, the user might have an *a priori* for the proportion of loci sharing the same genealogy. One can turn this information into a value of α since the probability that two randomly chosen loci share the same tree is about $1/(1+\alpha)$ if α is small compared to the total number of possible tree topologies. Also, the value of α sets the prior distribution on the number of distinct locus genealogies in the sample. Go to the website `www.stat.wisc.edu/~larget/bucky.html` for an interactive display of this distribution, which can serve as a tool for the choice of α.

First step of BCA: individual locus analysis. Any model of sequence evolution can be selected for any locus: there need not be one model common to all loci. Some loci can be protein alignments, others DNA alignments, and other morphological characters.

Missing sequences. If some loci have missing sequences, i.e. missing taxa, then rows of missing data (????) need to be inserted in place of the missing taxon's sequence. However, a more efficient way to deal with missing sequences will be implemented in a future version of `bucky`.

Genome-wide concordance factors. The output file `run1.concordance` provides the posterior distribution of the *sample-wide* concordance factor for each clade. For example, the information pertaining to clade $\{1, 2, 3\}$ might look like this:

```
{1,2,3|4,5,6,7,8}
#Genes      count probability cumulative
    90          2    0.000002   0.000002
    91         11    0.000011   0.000013
    92         96    0.000096   0.000109
    93        883    0.000883   0.000992
    94       4354    0.004354   0.005346
    95      17375    0.017375   0.022721
    96      52301    0.052301   0.075022
    97     124923    0.124923   0.199945
    98     211516    0.211516   0.411461
    99     260611    0.260611   0.672072
   100     204057    0.204057   0.876129
   101     100355    0.100355   0.976484
   102      22995    0.022995   0.999479
   103        521    0.000521   1.000000

mean CF = 98.760
99% CI for CF = (94,102)
95% CI for CF = (96,101)
90% CI for CF = (96,101)
```

However, the sample contained 106 loci only and there is extra uncertainty on the genome-wide number of loci having clade $\{1, 2, 3\}$. The 95% credibility interval for the genome-wide concordance factor of this clade must be wider than $(96/106, 101/106)$. Ané *et al.* (2006) describe how one can get the genome-wide posterior distribution from the sample-wide posterior distribution. For now, this is implemented in a separate program, (a set of R functions). In the later version of `bucky`, these programs will be unified.

Download the file `concordance_genomewide.r`. Open an R session (to download R, go to `http://cran.us.r-project.org`) and set R's working directory to the directory where you have `concordance_genomewide.r`. Have R read the file with

```
> source("concordance_genomewide.r")
```

You are now ready to use the program. The first thing to do is to read the results from `bucky` regarding sample-wide concordance factors. In the example above, clade `123|45678` is considered. Results can be read from a file or just given to R with

```
> sampleCF = c(0.000002,0.000011,0.000096,0.000883,0.004354,0.017375,0.052301,
+              0.124923,0.211516,0.260611,0.204057,0.100355,0.022995,0.000521)
```

Since these posterior probabilities correspond to concordance factors of 90 genes out of 106 through 103 genes out of 106, probabilities of 0 need to be added. A plot is suggested for checking.

```
> sampleCF = c(rep(0,90), sampleCF, 0,0,0)
> plot(sampleCF, type="h")
```

To get credibility intervals for the genome-wide concordance factor, use

```
> genomewide(sampleCF,alpha=2.5,N=6000,n=106,Ntax=8,Nclade=3)
> genomewide(sampleCF,alpha=2.5,N=6000,n=106,Ntax=8,Nclade=3,conf.level=.99)
```

Parameters: The level of discordance α was set to 2.5 because it was the value used in `bucky`. There were `n`=106 genes in the sample, and it is thought that there are a total of about `N`=6,000 genes in the yeast genome. The bipartition has `Nclade`=3 taxa on one side and 5 on the other side for a total of `Ntax`=8 taxa. The function `genomewide` may take some time (here a couple seconds). If you want to use this function many times for many different bipartitions, it is suggested to use a 2-step procedure, which we now explain. The posterior distribution of the genome-wide CF is obtained by multiplying the posterior distribution of the sample-wide CF (named `sampleCF` in the example) with some matrix. This matrix depends on the size of the bipartition, but all bipartitions with the same size will use the same matrix, which can then be re-used. The function `dpPosteriorWeights` will calculate this matrix. In the example above use

```
> mat = dpPosteriorWeights(alpha=2.5,N=6000,n=106,Ntax=8,Nclade=3)
```

and finally get the posterior distribution of the genome-wide CF for this clade, or for any clade with separates the 8 taxa into a group of 3 and a group of 5 taxa.

```
> genomeCF = mat %*% sampleCF
> distribution.summary(genomeCF)
```

References

Ané, C., B. Larget, D. A. Baum, S. D. Smith, and A. Rokas. 2006. Bayesian estimation of concordance among gene trees. Molecular Biology and Evolution .