Rudimentary instructions on using the `intronLoss` package

Miklós Csűrös
e-mail:csuros AT iro.umontreal.ca

September 29, 2005

1 Introduction

The Java package `ca.umontreal.iro.evolution.introns.intronLoss.jar` contains a number of programs and supporting Java classes for modeling and analyzing intron evolution. The package implements the likelihood method of Csűrös (2005), the formulas of Scott Roy and Walter Gilbert (2005a, 2005b), and the Dollo parsimony (Farris 1977) method employed by Rogozin et al. (2003). Briefly, in the approach of Csűrös (2005), a probabilistic model is assumed in which every branch has a gain rate, loss rate, and length. These parameters are estimated from data on intron presence-absence in homologous positions. In addition, a number of all-0 "silent" sites are assumed: positions in which no introns are observed at the terminal taxa but may be sites for intron gain. The number of such sites is estimated by the programs or chosen by the user.

The programs may be useful for the phylogenetic analysis of any 0-1 data set where rates may vary on the branches. In that case it may or may not make sense to estimate the number of all-0 columns. (If 1-0 encodes presence-absence, you may have to, but if they encode, say, purines and pyrimidines, then there is no need of course.)

Please send me a note if you use the package. I am extending the model to include across-sites variation, and other features.

2 File types

The programs work with the following main file types.

⟨**tree file**⟩ describes a phylogeny in the Newick format (`http://evolution.genetics.washington.edu/phylip/newicktree.html`) used by Phylip

and other programs. It is good to keep it unrooted (with one inner node that has three descendants) because otherwise the root's position is not unique (can "slide" between its two children: it is impossible to determine the branch length to those two guys). Name the inner nodes, it helps deciphering the outputs (and is necessary for computing the Roy-Gilbert formulas). Tree branch lengths are not important for the optimization (since they will be optimized too).

⟨**table file**⟩ A file of aligned 0-1 sequences for intron presence-absence in homologous positions. Lines starting with `#` are skipped (supposed to be comments and such). Data lines have two fields, separated by TABs: taxon name (must match name in ⟨tree file⟩) and 0-1 string of intron presence/absence. (This is the format used by Rogozin et al. (2003).)

⟨**rate file**⟩ Gives, in a depth-first traversal, the gain and loss *probabilities* along each edge (from which gain and loss rates are computed), and then the intron presence probability at the root. Lines starting with `#` are ignored (supposed to be comments). This file is the output of the program `calculateRates` but can be edited manually.

3 Main programs

All programs are implemented in Java for portability. In order to run program X, you need to invoke it through the Java engine:

`java -cp intronLoss.jar ca.umontreal.iro.evolution.introns.`X ...

The programs write to the standard output, so you need to redirect or pipe it through something to save the result. The output typically begins with some information on who, when and how produced the text, written in comment lines that start with `#|`.

3.1 Program `calculateRates`

Produces a ⟨rate file⟩ by fitting the parameters (gain & loss rates, length on each branch, root's intron density, and number of all-0 sites) to the data to maximize likelihood.

(a) `calculateRates` [⟨switches⟩] ⟨table file⟩ ⟨tree file⟩. Optimize parameters starting from default initial values.

(b) `calculateRates` [⟨switches⟩] ⟨table file⟩ ⟨tree file⟩ ⟨rate file⟩. Optimize parameters with the starting values in the rate file (useful if restarting optimization).

(c) `calculateRates` [⟨switches⟩] ⟨table file⟩ ⟨tree file⟩ ⟨nzeros⟩. Optimize parameters assuming the given number of additional all-0 columns. If ⟨nzeros⟩ starts with the character `0` and is other than 0 itself, then it is also optimized from the given starting value: `calculateRates TBL TREE 035000`.

(d) `calculateRates` [⟨switches⟩] ⟨table file⟩ ⟨tree file⟩ ⟨rate file⟩ ⟨nzeros⟩. (b)+(c) together, but the number of all-0 columns is not optimized for even if ⟨nzeros⟩ starts with `0`.

The parameter optimization is not very sophisticated: one parameter is optimized at a time, using Brent's method of line optimization (Press et al. 1997). The optimization first estimates the rates using the default number of unobserved intron sites. This may go through many hundreds of rate optimization rounds. After that, the number of unobserved intron sites ("zeros") are guessed, in a few rounds per estimated number of zeros, but many guesses may be evaluated (few thousand for the Rogozin data set). To learn about the switches that affect the optimization process, launch `calculateRates -h`.

3.2 Program `calculateNodePosteriors`

Outputs mean values for intron counts at the tree nodes in the likelihood model.

> `calculateNodePosteriors` [⟨switches⟩] ⟨table file⟩ ⟨tree file⟩ ⟨rate file⟩ [⟨nzeros⟩]. Parameters as in calculateRates but there is no more optimization performed.

3.3 Program `calculateEdgePosteriors`

Outputs mean values for intron losses/gains on edges in the likelihood model.

> `calculateEdgePosteriors` [⟨switches⟩] ⟨table file⟩ ⟨tree file⟩ ⟨rate file⟩ [⟨nzeros⟩]. Parameters as in calculateRates but there is no more optimization performed. The output is a table with columns named `n00` (number of intron sites that stayed unused), `01` (number of introns gained), `10` (number of introns lost), `11` (number of introns inherited). The numbers are for the edge that leads to each row's node.

3.4 Program `simulateEvolution`

Simulates intron evolution in the model.

`simulateEvolution` [⟨switches⟩] ⟨tree file⟩ ⟨rate file⟩ n. Produces a random ⟨table file⟩ with n non-zero columns by simulating intron evolution along the tree.

In comment lines starting with `##count`, the true numbers are also listed for each node: introns present at the node, introns gained on the edge leading to the node, and introns lost on the edge leading to the node.

3.5 Program `RoyGilbert`

Computes the Roy-Gilbert formulas for intron gain and loss on an edge or intron presence at a node.

(a) `RoyGilbert` ⟨table file⟩ ⟨tree file⟩ s_1 s_2. Computes gain and loss values for an outer edge leading to taxon s_1. The sibling clade is s_2. This formula first appears in Roy and Gilbert (2005a).

(b) `RoyGilbert` ⟨table file⟩ ⟨tree file⟩ s_1 s_2 s_3. Computes gain and loss values on an inner edge leading from the MRCA of s_1, s_2, s_3 to the MRCA of s_1, s_2. This formula first appears in Roy and Gilbert (2005b).

3.6 Program `DolloParsimony`

`DolloParsimony` ⟨table file⟩ ⟨tree file⟩. Computes intron counts, gains and losses using Dollo parsimony Farris (1977), i.e., with the assumption that each intron arose only once in the course of evolution. The format of the output is the same as that of `simulateEvolution`: in comment lines starting with `##count`, the numbers are listed for each node: introns present at the node, introns gained on the edge leading to the node, and introns lost on the edge leading to the node.

3.7 Debug messages

All the programs respond to the switch `-v` which allows (`-v true`) or disables (`-v false`) detailed debug messages. Debug messages are disabled by default. Usually, the message has the form `#**X.aBC...` where `X` identifies the Java class and `aBC` identifies the method, by initials of its name in the Java source code. For instance, `#**RG.aOI` messages are output by the method `allOtherIntrons()` in the class `RoyGilbert`. You can filter the messages through various pipes. Without filtering, don't be surprised to get an output of tens of thousand lines, taking up 10-20 Mbytes. Filtering out the messages, the output is very small. (Maybe a few hundred lines.) Or don't use the `-v true` switch.

4 An example session

This session illustrates how to use the programs in conjunction with the data of Rogozin et al. (2003).

1. Download the data file `ftp://ftp.ncbi.nlm.nih.gov/pub/koonin/intron_evolution/rogozin_introns_table`. For simplicity, name it `table.txt`. The data is for 7236 homologous intron positions in 684 orthologous gene sets, across 8 species: *D. melanogaster* (Dr), *A. gambiae* (An), *C. elegans* (Ca), *H. sapiens* (Ho), *S. ecervisiae* (Sa), *S. pombe* (Sc), *A. thaliana* (Ar), and *P. falciparum* (Pl). This will be your ⟨table file⟩.

2. Choose your phylogeny. Here are two possibilities:

```
(((((Dr, An) "Diptera", Ca) "Ecdysozoa", Ho) "Bilateria",
(Sc, Sa) "Ascomycota") "Opisthokont", Ar, Pl) "Crown";
[ tree with ecdysozoan clade (nematoda-arthropods)]
```

   or

```
(((((Dr, An) "Diptera", Ho) "Coelomata", Ca) "Bilateria",
(Sc, Sa) "Ascomycota") "Opisthokont", Ar, Pl) "Crown";
[ tree with coelomate clade (vertebrates-arthropods)]
```

   Save your phylogeny in a file called `tree.tre`, this will be your ⟨tree file⟩.

3. Optimize the rate parameters.

```
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.calculateRates \
   table.txt tree.tre 00 > rates.txt &
```

   The execution will take some time (depending on optimization parameters and your computer's speed, a couple of minutes or a couple of hours). If you want to see what is going on, use instead the `-v true` switch and pipe through `tee rates.txt`. The file `rates.txt` lists the probabilities for gain and loss for each node (on the edge that leads to it), and from the comments you can decypher the actual gain and loss rates and branch lengths — the rates and lengths are scaled on each edge that the loss and gain rates add up to 1. You should also find a comment line that starts with `#z` that tells you what the probability is for an all-0 column, and how many all-0 columns should be added to your data to correspond to that value. This will be your n in the posterior computations. Let's suppose you have $n = 10000$.

4. Check the posterior counts.

```
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.calculateNodePosteriors
   table.txt tree.tre rates.txt 10000 > nodes-data.txt
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.calculateEdgePosteriors \
   table.txt tree.tre rates.txt 10000 > edges-data.txt
```

5. Compare with Dollo parsimony estimates.

```
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.DolloParsimony \
   table.txt tree.tre > dollo-data.txt
```

6. Compare with Roy-Gilert estimates.

```
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.RoyGilbert \
   table.txt tree.tre Ecdysozoa Ho Ascomycota
```

   You need to do this for every edge, choosing appropriate clades in the analysis.

7. Do a couple of rounds using simulated data to assess accuracy.

```
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.simulateEvolution \
   tree.tre rates.txt 7236 > table-rnd.txt
java -cp intronLoss.jar \
   ca.umontreal.iro.evolution.introns.calculateRates \
   table-rnd.txt tree.tre 00 > rates-rnd.txt
...
```

References

Csűrös, M. (2005). Likely scenarios of intron evolution. In A. McLysaght and D. H. Huson (Eds.), *Comparative Genomics*, Volume 3678 of *LNBI*, Heidelberg, pp. 47–60. Springer-Verlag.

Farris, J. S. (1977). Phylogenetic analysis under Dollo's law. *Systematic Zoology 26*(1), 77–88.

Press, W. H., S. A. Teukolsky, W. V. Vetterling, and B. P. Flannery (1997). *Numerical Recipes in C: The Art of Scientific Computing* (Second ed.). Cambridge UniversIty Press.

Rogozin, I. B., Y. I. Wolf, A. V. Sorokin, B. G. Mirkin, and E. V. Koonin (2003). Remarkable interkingdom conservation of intron positions and massive, lineage-specific intron loss and gain in eukaryotic evolution. *Current Biology 13*, 1512–1517.

Roy, S. W. and W. Gilbert (2005a). Complex early genes. *Proceedings of the National Academy of Sciences of the USA 102*(6), 1986–1991.

Roy, S. W. and W. Gilbert (2005b). Rates of intron loss and gain: Implications for early eukaryotic evolution. *Proceedings of the National Academy of Sciences of the USA 102*(16), 5773–5778.