Abstract

1. Growing numbers of large phylogenetic syntheses are being published. Sometimes as part of a hypothesis testing framework, sometimes to present novel methods of phylogenetic inference, and sometimes as a snapshot of the diversity within a database. Commonly used methods to reuse these trees in scripting environments have their limitations.

2. I present a toolkit that transforms data presented in the most commonly used format for such trees into a database schema that facilitates quick topological queries. Specifically, the need for recursive traversal commonly presented by schemata based on adjacency lists is largely obviated. This is accomplished by computing pre- and post-order indexes and node heights on the topology as it is being ingested.

3. The resulting toolkit provides several command line tools to do the transformation and to extract subtrees from the resulting database files. In addition, reusable library code with object-relational mappings for programmatic access is provided. To demonstrate the utility of the general approach I also provide database files for trees published by Open Tree of Life, Greengenes, D-PLACE, PhyloTree, the NCBI taxonomy and a recent estimate of plant phylogeny.

4. The database files that the toolkit produces are highly portable (either as SQLite or tabular text) and can readily be queried, for example, in the R environment. Programming languages with mature frameworks for object-relational mapping and phylogenetic tree analysis, such as Python, can use these facilities to make much larger phylogenies conveniently accessible to researcher-programmers.

Keywords

phylogenetics, scripting, databases, topological queries, object-relational mapping

Introduction

Larger and larger phylogenies are being published. The contexts in which these trees appear vary somewhat. Sometimes, a tree is published as a ‘one off’ estimate needed for testing a hypothesis in a phylogenetic comparative framework (e.g. Zanne et al. 2014). In other cases, the tree demonstrates the capabilities of initiatives to produce megatrees (e.g. Hinchliff et al. 2015; Smith and Brown 2018). In yet other cases, the trees are provided as snapshots of the diversity contained in a database (e.g. Oven and Kayser 2009; Kirby et al. 2016; Federhen 2012; DeSantis et al. 2006).

All these trees coming publicly available is a wonderful development. However, the format in which they are published is not always convenient for reuse. Most commonly, large phylogenetic trees are made available in Newick format (Felsenstein n.d.), as other formats (e.g. Maddison, Swofford, and Maddison 1997; Vos et al. 2012) are too verbose. From this perspective of conciseness, Newick is a sensible choice. However, the researcher-programmer who wants to reuse such large trees in a scripting environment is then faced with the need to parse complex parenthetical tree descriptions and load some kind of graph structure or
object into memory every time the script is run. With large trees, this takes a lot of time and consumes a lot of working memory. For example, loading the latest Open Tree of Life estimate (v10.4, see Hinchliff et al. 2015) into DendroPy (Sukumaran and Holder 2010) takes about 15 minutes and consumes over 8 GB of RAM. This might be fine for some use cases (e.g. for processes that subsequently run for very long) but it can be a limitation in other situations.

An alternative approach is to ingest a large tree into a portable, on-disk database as a one-time operation, and then access the tree through a database handle. No more recurrent, complex text parsing, and the tree does not have to be loaded into memory to query its topology. The NCBI taxonomy (Federhen 2012) is distributed as database tables with this usage in mind, for example. In that case, and indeed in most cases where trees that might have polytomies are represented in databases, the topology is captured using adjacency lists, where each database record for a node (except the root) contains a reference to its parent by way of a foreign key relation. The downside of this is that tree traversal requires recursive queries: to get from a tip to the root, each focal node along the path has to be visited in turn to look up the foreign key relation to its parent. This is relatively slow. A possible solution to this is to use relational database engines that compute transitive closures, but not all commonly-used engines support those, and their computation imposes additional computational cost on the ones that do.

Pre-computing certain metrics and topological indexes as column values can obviate the need for some recursions entirely, speeding up topological queries significantly. The general idea is illustrated in Fig 1. The topology shown is represented in the table, with one record for each node, by way of the following columns:

- **name** - the node label. The values in this column correspond to those in the tree.
- **length** - the branch length.
- **id** - a primary key, generated as an autoincrementing integer.
- **parent** - a foreign key, whose value references the primary key of the parent node.
- **left** - an index generated as an autoincrementing integer in a pre-order traversal: moving from root to tips, parent nodes are assigned the index before their child nodes.
- **right** - an index generated as an autoincrementing integer in a post-order traversal: moving from root to tips, child nodes are assigned the index before their parents. That is, “on the way back” in the recursion.
- **height** - the node height, i.e. the distance from the root.

In relational database implementations of trees that use adjacency list of this form, the children of \texttt{n1} can be selected like so (returning \texttt{C} and \texttt{D}):

```sql
select CHILD.* from node as PARENT, node as CHILD
where PARENT.name='n1'
and PARENT.id==CHILD.parent;
```

The inverse, getting the parent for an input node, should be readily apparent. Beyond direct adjacency, traversals that would otherwise require recursion can be executed as a single query with the aid of the additional indexes. For example, to identify the most recent common ancestor MRCA of input nodes \texttt{C} and \texttt{F}, we can formulate:

```sql
select MRCA.* from node as MRCA, node as C, node as F
where C.name='C' and F.name='F'
and MRCA.left < min(C.left,F.left)
and MRCA.right > max(C.right,F.right)
order by MRCA.left desc limit 1;
```

The query selects all nodes whose \texttt{left} index is lower, and whose \texttt{right} index is higher than that of either of the input nodes. This limits the result set to those nodes that are ancestral to both. By then ordering these on the \texttt{left} index in descending order they are ranked from most recent to oldest. Limiting the result set to only the first record in this ordered list returns MRCA. Variations on this query to obtain, for example, all ancestors or descendants of input nodes follow similar logic. The precomputed node heights can be exploited, for example, to compute patristic distances between nodes, such as:
```sql
select (C.height-MRCA.height)+(F.height-MRCA.height)
from node as MRCA, node as C, node as F
where C.name='C' and F.name='F'
and MRCA.left < min(C.left,F.left)
and MRCA.right > max(C.right,F.right)
order by MRCA.left desc limit 1;
```

In this query, the final result is 3.3, i.e. the sum of the heights of C and F, as the root has no height. Other calculations that take advantage of the extra indexes are also possible as single queries. For example, several metrics capturing the tendency of nodes towards the tips (such that the tree is “stemmy”) or towards the root (“branchy”) are used to summarize the mode of diversification in a clade (e.g., apparently accelerating or slowing down, respectively).

One of these metrics (Fiala and Sokal 1985) iterates over all internal nodes and for each calculates the ratio of the focal node’s branch length over the sum of descendent branch lengths plus the focal length, and then averages over these ratios. This can be expressed in a single query:

```sql
select avg(ratio) from (select INTERNAL.length/(sum(CHILDREN.length)+INTERNAL.length) as ratio
from node as INTERNAL, node as CHILDREN
where INTERNAL.left!=INTERNAL.right
and CHILDREN.left>INTERNAL.left
and CHILDREN.right<INTERNAL.right
and INTERNAL.parent!=1
group by INTERNAL.id)
```

These examples illustrate that access to large tree topologies indexed in this way is quite powerful, especially when integrated in scripting environments that provide additional functionality. The toolkit presented here provides such access.

### Materials and Methods

#### Database schema and object-relational mapping

A database schema that provides the functionality described in the Introduction is shown in Table 1. In addition to the column names and their data types, shown are the indexes for the database engine to compute. To avoid confusion with the usage of ‘index’ elsewhere in this manuscript, what is referred to here are B-Trees that the database engine computes for internally organizing and searching the data held by a column (or combination of columns) to allow it to find matches more quickly, sort result sets, and enforce certain constraints (such as uniqueness). In other words, this is something else than the topological indexing described at greater length in this paper. Nevertheless, these B-Tree indexes also influence performance greatly so I note them here in the interest of any re-implementations by readers.

As the database consists of a single table, mapping its structure onto an object-oriented class is straightforward. Many programming languages have tools for this. Commonly-used examples are Hibernate for Java, SQLAlchemy for Python, and DBIx::Class for Perl, which I used. I then modified the generated code so that it inherits from a tree node class of Bio::Phylo (Vos et al. 2011), providing it with the additional functionality of this package (e.g. export to various flat file formats; tree visualization). Infeasibly large phylogenies can thus be programmed like any other tree object that Bio::Phylo operates on, provided a database is populated with them.

### Populating databases

My approach for processing input parenthetical statements and emitting these as database records of the form discussed in the Introduction is described in the following prose algo-
1. Apply an auto-incrementing label to each node, i.e., reading the tree statement from left to right, append a unique identifier to each closing parenthesis. Careful tokenization, taking into account the Newick rules (loosely observed as they are) for single and double quoting, spaces, underscores, and square bracketed comments, must be applied diligently here and throughout the algorithm.

2. Remove the closing semicolon of the parenthetical statement. From here on, every nested taxon - including the entire tree - is syntactically self-same: it has a name, either tagged using the labeling scheme from step 1, or a previously provided one, and it may have a branch length (the last colon symbol followed by a number).

3. Emit the focal taxon to the database handle. In the root case, no parent of the focal taxon is in the tree, and so the default value for parent is used, i.e. 1. The length and name are parsed out of the string. An id is generated as an auto-incrementing integer and is stored as the value for name in a lookup table (hash table, dictionary). In cases other than the root case, the parent has been processed and so the generated identifier for parent can be fetched from the lookup table. What is passed to the database handle is thus a new record with values for the fields id, parent, name, and length.

4. Strip the outermost set of decorated parentheses (if any) from the tree string, storing the parent label attached to the closing parenthesis. Split the remaining string in the two or more (in case of polytomies) direct children, by scanning for comma symbols that are not nested inside parentheses. This involves keeping track of the nesting levels of opening and closing parentheses while scanning through the string. Pass each of these direct children to step 3. The recursion between 3 and 4 continues until all taxa have been emitted.

5. The Newick string has now been consumed. In a second pass, the left and right indexes and the node height are computed by traversing through the now populated database. Starting with the root (i.e. parent==1), a depth-first traversal is performed by recursively fetching the immediate child nodes from the database (as per the first query example from the Introduction). The pre-order processing of the children is to store the value of an auto-incrementing integer as left, and the value of height as carried over from the parent increased with the value of length. After treating any children, the post-order processing then applies the value of the auto-incrementing integer (unchanged in the case of terminal nodes) to right.

I implemented this basic algorithm in a script and applied it to the following, published trees:

- A tree of human societies from the D-PLACE database (Kirby et al. 2016). 1,647 nodes using the release that was current as of 04.02.2017.
- A tree of 16S rRNA gene sequences from the Greengenes database (DeSantis et al. 2006), release gg_13_5, current as of 11.10.2017. Contains 406,903 nodes.
- A synthesis of plant phylogeny from (Smith and Brown 2018), identified as ALLMB.tre, version v1.0, current as of 29.08.2019. Contains 440,712 nodes.
- A release of the Open Tree of Life project (Hinchliff et al. 2015). Identified as v10.4, current as of 24.09.2018. Contains 2,902,755 nodes.

In addition, I implemented two scripts that process tree descriptions in proprietary, tabular formats:

- The tabular dump of the NCBI taxonomy. The database I generated with this is from GenBank release current as of 03.02.2017, and contains 1,554,272 nodes.

- A custom format that captures a tree of Y-chromosome haplotype diversity backing the PhyloTree database (Oven and Kayser 2009), build 17, current as of 11.10.2017. Contains 5,438 nodes.
Results

The substantial results of this study comprise library code and scripts. The library code introduces two namespaces compatible with the standardized class hierarchy for Perl5:

- **Bio::Phylo::Forest::DBTree** - a class containing factory methods for instantiating databases and utility methods for persisting and extracting trees. This subclasses the core tree class in Bio::Phylo and inherits its decorations.
- **Bio::Phylo::Forest::DBTree::Result::Node** - the generated object-relational mapping class, modified to inherit from the core tree node class of Bio::Phylo. In addition, this class contains several query methods of the sort described in the Introduction.

The scripts are:

- **megatree-loader** - Newick tree parser/loader
- **megatree-ncbi-loader** - parser/loader of NCBI taxonomy dump
- **megatree-phylotree-loader** - PhyloTree parser/loader
- **megatree-pruner** - extracts subtrees from a database

Applying the loader scripts to the trees listed in the Methods resulted in databases that can be queried in SQL (e.g. in the SQLite shell, a 3rd party database browser, or from a scripting environment via a database handle) or using the object library code presented here. I describe in Data Availability how to obtain these generated databases and the tools to make more. As an example of the time it takes to do the latter: indexing the largest tree in the set (and the largest published phylogeny I am aware of), the Open Tree of Life release, took approximately one hour on a current MacBook Pro. This is thus a somewhat costly operation that, mercifully, needs to be run only once.

Discussion

The concepts, tools and data files presented here are intended to make life easier for researchers in computational biology. I would therefore like to reassure the reader that there is no need to dust off any lingering knowledge of SQL or Perl to be able to take advantage of the outcomes of this study. The databases produced in this study can be navigated conveniently in R by accessing them as data frames and processing them with `dbplyr` and related tools. I provide an R Markdown document on the git repository (see Data Availability) that provides a simple run through of how to operate on the databases, showing how to extract clades, MRCAs, and pairwise distances.

For programming languages where object-relational mapping is a more common, mature technique, the schema and databases presented here may form the basis for extending the functionality of some popular toolkits. For example, generating a mapping for Python results in a tiny SQLAlchemy class that, thanks to Python’s multiple inheritance model, might subclass DendroPy’s tree node model, thus making persistently databased trees accessible through the same programming interface as memory resident trees. I invite authors of libraries that could take advantage of this to consider this possibility.

Acknowledgements

I would like to thank Bill Piel for the numerous conversations we’ve had over the years on the topic of representing trees in relational databases, from which I learned some of the concepts and ideas presented here. I would also like to thank Mannis van Oven, who kindly provided me with the data dump of the PhyloTree project.
Data availability

The source code of this project is available under the same terms as the Perl5 core itself, i.e. a combination of the GNU General Public License (v.3) and the Artistic License, and is being developed further in a git repository at: https://github.com/rvosa/bio-phylo-forest-dbtree

The version of the software presented in this manuscript has been stored permanently under a DOI at: https://doi.org/10.5281/zenodo.1035856 and is released through the Comprehensive Perl Archive Network at: https://metacpan.org/release/Bio-Phylo-Forest-DBTree

The database files discussed in this manuscript are available at the following locations:

| Name          | Citation                        | Database DOI                  |
|---------------|---------------------------------|-------------------------------|
| PhyloTree     | (Oven and Kayser 2009)          | 10.6084/m9.figshare.4620757   |
| D-PLACE       | (Kirby et al. 2016)             | 10.6084/m9.figshare.4620217   |
| NCBI Taxonomy | (Federhen 2012)                 | 10.6084/m9.figshare.4620733   |
| Green Genes   | (DeSantis et al. 2006)          | 10.6084/m9.figshare.4620214   |
| ALLMB         | (Smith and Brown 2018)          | 10.6084/m9.figshare.9747638   |
| Open Tree of Life | (Hinchliff et al. 2015) | 10.6084/m9.figshare.9750509   |
Figures and tables

Figure 1: representation of a tree shape in a relational database, with additional, precomputed indexes and values. See text for details.

| id | parent | left | right | name | length | height |
|----|--------|------|-------|------|--------|--------|
| 2  | 1      | 1    | 10    | n3   | 0.0    | 0.0    |
| 3  | 2      | 2    | 5     | n1   | 0.9    | 0.9    |
| 4  | 3      | 3    | 3     | C    | 0.6    | 1.5    |
| 5  | 3      | 4    | 4     | D    | 0.8    | 1.7    |
| 6  | 2      | 6    | 9     | n2   | 0.6    | 0.6    |
| 7  | 6      | 7    | 7     | F    | 1.2    | 1.8    |
| 8  | 6      | 8    | 8     | G    | 1.0    | 1.6    |

| Name  | Type      | Index        |
|-------|-----------|--------------|
| id    | int       | primary key, not null |
| parent| int       | index        |
| left  | int       | index        |
| right | int       | index        |
| name  | varchar(20)| index       |
| length| float     |              |
| height| float     |              |

Table 1: schema for DBTree databases.
References

DeSantis, T Z, P Hugenholtz, N Larsen, M Rojas, E L Brodie, K Keller, T Huber, D Dalevi, P Hu, and G L Andersen. 2006. “Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB.” Applied and Environmental Microbiology 72 (7): 5069–72. https://doi.org/10.1128/AEM.03006-05.

Federhen, Scott. 2012. “The NCBI Taxonomy database.” Nucleic Acids Research 40 (D1): D136–D143. https://doi.org/10.1093/nar/gkr1178.

Felsenstein, Joe. n.d. “The Newick Tree Format.” Accessed August 30, 2019. http://evolution.genetics.washington.edu/phylip/newicktree.html.

Fišer, K L, and R R Sokal. 1985. “Factors Determining the Accuracy of Cladogram Estimation: Evaluation Using Computer Simulation.” Journal article. Evolution 39 (3): 609–22. http://links.jstor.org/sici?sici=0014-3820(198505)39%7B/%%7D253A3%7B/%%7D253C609%7B/%%7D253AFDTAOC%7B/%%7D253E2.0.CO%7B/%%7D253B2-E.

Hinchliff, Cody E, Stephen A Smith, James F Allman, J Gordon Burleigh, Ruchi Chaudhary, Lyndon M Coghill, Keith A Crandall, et al. 2015. “Synthesis of phylogeny and taxonomy into a comprehensive tree of life.” Proceedings of the National Academy of Sciences of the United States of America 112 (41): 12764–9. https://doi.org/10.1073/pnas.1423041112.

Kirby, Kathryn R., Russell D. Gray, Simon J. Greenhill, Fiona M. Jordan, Stephanie Gomez-Ng, Hans Jörg Bibiko, Damián E. Blasi, et al. 2016. “D-PLACE: A global database of cultural, linguistic and environmental diversity.” Edited by Alex Mesoudi. PLoS ONE 11 (7): e0158391. https://doi.org/10.1371/journal.pone.0158391.

Maddison, David R., David L. Swofford, and Wayne P. Maddison. 1997. “NEXUS: An Extensible File Format for Systematic Information.” Journal article. Systematic Biology 46 (4): 590–621. https://doi.org/10.2307/2413497.

Oven, Mannis van, and Manfred Kayser. 2009. “Updated comprehensive phylogenetic tree of global human mitochondrial DNA variation.” Human Mutation 30 (2): E386–E394. https://doi.org/10.1002/humu.20921.

Smith, Stephen A., and Joseph W. Brown. 2018. “Constructing a broadly inclusive seed plant phylogeny.” American Journal of Botany 105 (3): 302–14. https://doi.org/10.1002/ajb2.1019.

Sukumaran, Jeet, and Mark T. Holder. 2010. “DendroPy: a Python library for phylogenetic computing.” Journal article. Bioinformatics 26 (12): 1569–71. https://doi.org/10.1093/bioinformatics/btp228.

Vos, Rutger A., James P. Balhoff, Jason A. Caravas, Mark T. Holder, Hilmar Lapp, Wayne P. Maddison, Peter E. Midford, et al. 2012. “NeXML: Rich, Extensible, and Verifiable Representation of Comparative Data and Metadata.” Journal article. Systematic Biology, February. https://doi.org/10.1093/sysbio/sys025.

Vos, Rutger A., Jason Caravas, Klaas Hartmann, Mark Jensen, and Chase Miller. 2011. “Bio::Phylo-phyloinformatic analysis using perl.” Journal article. BMC Bioinformatics 12 (1). https://doi.org/10.1186/1471-2105-12-63.

Zanne, Amy E., David C. Tank, William K. Cornwell, Jonathan M. Eastman, Stephen A. Smith, Richard G. Fitzjohn, Daniel J. McGlinn, et al. 2014. “Three keys to the radiation of angiosperms into freezing environments.” Nature 506 (7486): 89–92. https://doi.org/10.1038/nature12872.