An open source phylogenetic search and alignment package

Hyrum Carroll, Adam Teichert, Jonathan Krein, Kenneth Sundberg, Quinn Snell and Mark Clement*

Computer Science Department,
Brigham Young University,
Provo, Utah 84602, USA
E-mail: hyrumcarroll@gmail.com
E-mail: adamteichert@gmail.com
E-mail: jonathankrein@gmail.com
E-mail: kasundberg@gmail.com
E-mail: snell@cs.byu.edu
E-mail: clement@cs.byu.edu
*Corresponding author

Abstract: PSODA is a comprehensive phylogenetics package, including alignment, phylogenetic search under both parsimony and maximum likelihood, and visualization and analysis tools. PSODA offers performance comparable to PAUP* in an open source package that aims to provide a foundation for researchers examining new phylogenetic algorithms. A key new feature is PsodaScript, an extension to the nearly ubiquitous NEXUS format, that includes conditional and loop constructs; thereby allowing complex meta-search techniques like the parsimony ratchet to be easily and compactly implemented. PSODA promises to be a valuable tool in the future development of novel phylogenetic techniques. This paper seeks to familiarize researchers with PSODA and its features, in particular the internal scripting language, PsodaScript. PSODA is freely available from the PSODA web site: `http://csl.cs.byu.edu/psoda`.

Keywords: PSODA, PsodaScript, phylogenetic inference, open source

Reference to this paper should be made as follows: Carroll, H., Teichert, A., Krein, J., Sundberg, K., Snell, Q., and Clement, M. (2008) 'An open source phylogenetic search and alignment package', *Int. J. Bioinformatics Research and Applications*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Hyrum Carroll received his Bachelors degree from the Computer Engineering Department and his Masters degree from the Computer Science Department at Brigham Young University. He is currently working on his Ph.D. in Computer Science at Brigham Young University with an emphasis in multiple sequence alignment.

Adam Teichert is completing his Bachelors degree in Computer Science at Brigham Young University. His educational goals include a doctoral

Copyright © 200x Inderscience Enterprises Ltd.

degree and research in Natural Language Processing.

Jonathan Krein is currently working on his Bachelors degree in Computer Science. In 2009 he will begin an advanced degree program in the field of Computer Science.

Kenneth Sundberg received his Bachelors and Masters degrees from the Department of Computer Science at Utah State University. He is currently working on his Ph.D. in Computer Science at Brigham Young University with an emphasis in phylogenetic search.

Quinn Snell is an Associate Professor of Computer Science at Brigham Young University. His research areas are Computational Biology and Parallel and Distributed Processing. Quinn has been at Brigham Young University since 1997.

Mark Clement is an Associate Professor in the Computer Science Department at Brigham Young University. His research has included new methods for phylogenetic analysis, sequence alignment and population genetics. He was responsible for the development of the TCS population genetics application as well as the DOGMA phylogenetic parallel processing system. In addition to his interests in Bioinformatics, Dr. Clement has performed research in parallel processing and Internet protocols and has published more than 65 papers in interdisciplinary venues.

1 Introduction

Phylogenetic trees model the evolutionary relationships among species, enabling researchers to both analyze species differences (Felsenstein, 2004) and to better understand the processes by which those differences arise (Mooers and Heard, 1997). Phylogenetic trees represent an important research tool in many scientific areas. In AIDS research, for example, scientists are using phylogenies to better understand how the Human Immunodeficiency Virus (HIV) mutates in response to the human immune system. Because HIV evolves faster than most known organisms, understanding its evolution will hopefully lead to improved vaccines (Rambaut *et al.*, 2004). Phylogenetic research is also contributing to many other areas of study, including nucleic acids research (Eisen, 1998) and endangered species conservation (Wolfe *et al.*, 1998).

A major task related to phylogenetic research is that of efficiently inferring, from molecular data, the correct phylogenetic tree for a set of organisms. One approach to the problem is to arrange a data set into all possible phylogenies, scoring each tree to find the one that most likely represents the actual evolutionary relationships among the organisms. Unfortunately, because phylogenetic inference is an NP-hard problem (Chor and Tuller, 2005), such an exhaustive approach is typically unreasonable. Therefore, to infer trees from data sets with more than a few taxa, phylogenetic inference software must rely heavily on heuristic algorithms (Sanderson, 1990). This is significant because most realistic biological studies require data sets with at least 20 taxa (Graham and Foulds, 1982), if not hundreds or thousands.

Table 1 Accessibility

	License Fee	Open-source
PHYLIP		✓
POY		✓
PAUP*	✓	
BioPerl		✓
TNT		
PSODA		✓

A high quality phylogeny, or evolutionary tree, is important to accurately determine the relationships between species. Phylogenies have been in use for over a hundred years and software has been employed to produce better phylogenies for a couple of decades. These applications are among the most frequently cited papers in the field of bioinformatics with over 10,000 citations for both PHYLIP (Felsenstein, 2007) and PAUP* (Swofford, 2003) (according to `scholar.google.com`). Although many people use these applications, they have not been thoroughly maintained in the past several years. Furthermore, most existing phylogenetic reconstruction packages either have a licensing fee and are not extendible to experiment with new algorithms and methods or have serious performance limitations. This work presents an open source phylogenetic search and multiple sequence alignment package that is free to use and has phylogenetic search performance comparable to PAUP*.

PSODA (Phylogenetic Search using Open Source Data Analysis) Carroll *et al.* (2007) is an open source phylogeny reconstruction application. It provides basic and advanced phylogeny searching capabilities and a progressive multiple sequence alignment method. It can perform phylogenetic analysis with Maximum Parsimony and Maximum Likelihood. Additionally, it comes with a graphical user interface (GUI) that allows for visualizing individual phylogenies as well as the progress of a search. PSODA's performance is comparable to existing phylogenetic programs (see section 6). One of PSODA's distinguishing characteristics, PsodaScript (Krein *et al.*, 2007), is the ability to script meta-searches with advance language constructs. PSODA can be obtained for free from `http://csl.cs.byu.edu/psoda`.

This paper seeks to familiarize researchers with PSODA and its features, in particular PsodaScript.

2 Related Work

While several phylogenetic reconstruction packages exist, they are either too slow for analysis on medium to large data sets and/or are proprietary.

Over the past few decades, developers have produced a number of phylogenetic inference software packages. Even a superficial summary of the many packages is beyond the scope of this paper. We discuss only a few of the more influential packages—PHYLIP (Felsenstein, 2007), POY (Wheeler *et al.*, 2003), PAUP* (Swofford, 2003), and TNT (Goloboff *et al.*, 2007) (see Table 1).

PHYLIP, an open source package, is one of the longest maintained phylogenetic analysis applications, offering a collection of separate programs that can be used as a toolkit in phylogenetic research. It was first released in 1980 by Joe Felsenstein and is one of the first programs to perform Maximum Likelihood (ML) searches. It is an open source package that focuses on ML, but also allows for Maximum Parsimony (MP) searches. PHYLIP's component programs are written in C and are relatively easy to install and use. However, it also runs much more slowly on large data sets than other phylogenetic applications (Sanderson, 1990).

PAUP* (Phylogenetic Analysis Using Parsimony *and other methods) is believed to be the most widely used phylogenetic search program. It is both feature-rich (analysis in both MP and ML) and robust. Unfortunately, it is proprietary software, requires a licensing fee to use and does not perform multiple sequence alignment.

POY, on the other hand, is free and open source software. It differs from the other programs here in that it employs *Optimization Alignment* to build phylogenies (Wheeler, 1996). Optimization Alignment builds a phylogeny from unaligned data, then produces an alignment of the sequences based on that phylogeny. The phylogeny is evaluated using the newly calculated alignment.

Another phylogenetic search package is TNT (Tree analysis using New Technology), written by Pablo Goloboff, Steve Farris, and Kevin Nixon. Although TNT has become a pacesetter in phylogenetic software, offering excellent search speeds (Meier and Ali, 2005), it ultimately limits users because it is closed source. For instance, TNT scores all trees with parsimony; users cannot score trees with likelihood, and they cannot extend TNT to implement experimental algorithms. Another limitation is that TNT does integrated alignment methods. TNT is proprietary, but as of November 2007 when the Willi Hennig Society subsidized the project, it is free to download (see `http://www.zmuc.dk/public/phylogeny/TNT/`).

3 Features

Analyzing phylogenetic trees currently requires the use of several different programs. PSODA bridges the gap and brings the many features necessary to analyze phylogenetic trees together in one package. Some of the features PSODA provides are discussed in this section.

3.1 Open-Source Code

PSODA uses open source code licensed under the GNU General Public License, Version 2 (see `http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`). This license allows others not only to collaborate and make improvements to the package, but also to extend it and perform algorithmic experiments with a stable code foundation. We envision many researchers finally being able to implement, in code, concepts that they've envisioned, but have not implemented due to the hurdles of developing a fast and reliable foundation of code. Example modifications include a different enumeration of topologies during a TBR search and integrating both Maximum Likelihood and Maximum Parsimony into a single search algorithm (see (Sundberg *et al.*, 2007)). To implement the latter would require minor changes and additions to the existing code.

3.2 Performance

The performance of a phylogenetic search application is usually measured by the phylogeny scores that it achieves over time. PSODA has comparable performance to other phylogenetic search packages in terms of the trees scores obtained over time. For a more detailed treatment of PSODA's performance, see section 6, Results.

3.3 Models of analysis

The two main models of phylogenetic analysis are Maximum Parsimony (Camin and Sokal, 1965) and Maximum Likelihood (Felsenstein, 1981). MP has been used for phylogeny analysis longer than ML. It is based on the same principles as Occam's razor—the simplest solution is the best solution. Joe Felsenstein fathered the ML movement when he discovered inconsistencies with MP when long branches are present in a phylogeny. ML uses different models of evolution. PSODA allows users to perform both Maximum Parsimony and Maximum Likelihood searches. For ML searches, PSODA uses the GTR+Γ evolutionary model by incorporating RAxML (Stamatakis, 2006).

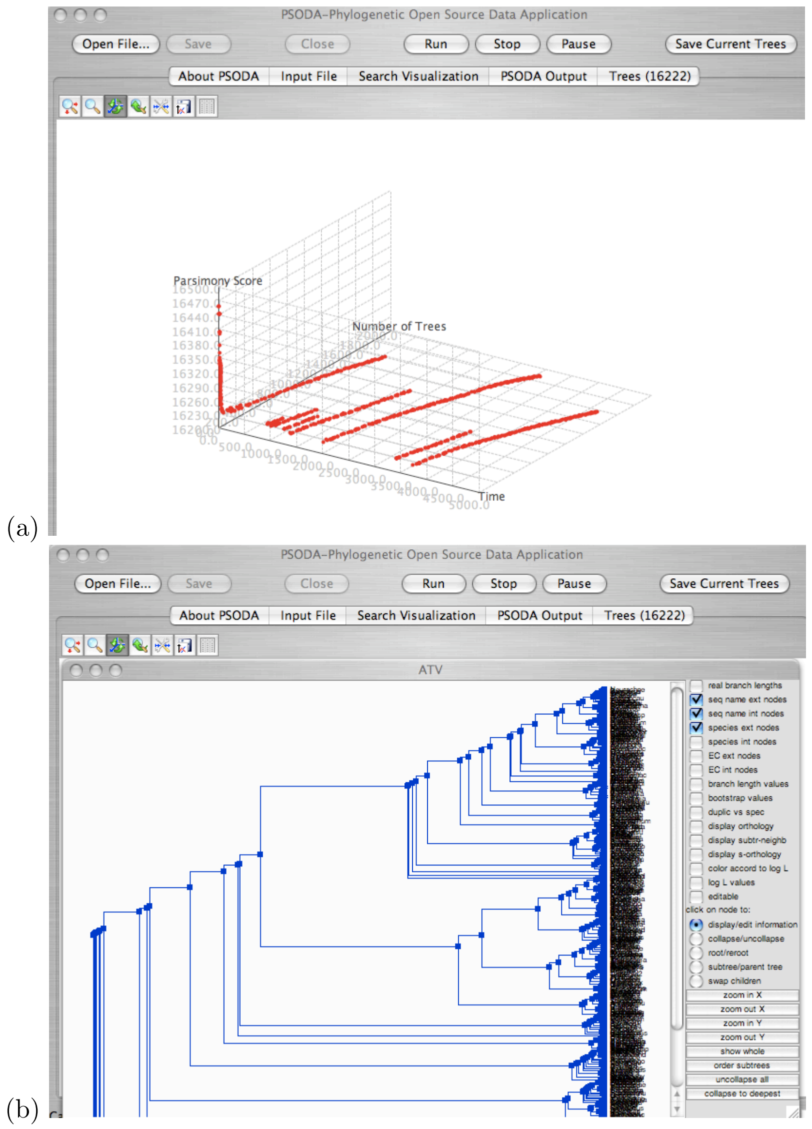
3.4 Graphical User Interface

PSODA's graphical user interface (GUI) (see Figure 1) is an important part of making PSODA user-friendly and portable. All of the other programs used in searching tree space have only a command-line interface, or the GUI that is available only works on older operating systems.

There are several features offered in PSODA's GUI in order to facilitate many of the tasks required to run and analyze datasets. Three of the most valuable features in the GUI are: data format conversion, a 3D visualization of the progress of the search and an individual phylogeny viewer. Each of these features are discussed in this section.

Data Format Conversion

Several formats currently exist for genetic and phylogeny data such as PHYLIP (Felsenstein, 2007), Clustal (Thompson *et al.*, 1994), MEGA (Kumar *et al.*, 1994), NEXUS (Maddison *et al.*, 1997), and FASTA (Lipman and Pearson, 1985); however, none of the standard programs accept all of these formats, nor do they perform all types of analyses common for phylogenetic tree. To do so requires using multiple programs and multiple file formats. The process of converting from one format to another can be difficult for many users, so DataConvert (David McClellan, `http://biology.byu.edu/faculty/dam83/cdm/`), which is capable of converting each file format to any other, is included in PSODA. When converting the formats, DataConvert offers the option of interleaving the sequences or leaving them discrete. Also, if there is an entire directory of files that needs to be converted, DataConvert can convert all of those files instead of requiring the user to convert each file individually.



- [ ] real branch lengths
- [✓] seq name ext nodes
- [✓] seq name int nodes
- [✓] species ext nodes
- [ ] species int nodes
- [ ] EC ext nodes
- [ ] EC int nodes
- [ ] branch length values
- [ ] bootstrap values
- [ ] duplic vs spec
- [ ] display orthology
- [ ] display subtr-neighb
- [ ] display s-orthology
- [ ] color accord to log L
- [ ] log L values
- [ ] editable

click on node to:

- [✓ display/edit information]
- collapse/uncollapse
- root/reroot
- subtree/parent tree
- swap children

Figure 1 PSODA GUI illustrating the search results (tree scores and number of trees over time) (in (a)) and using ATV (Zmasek and Eddy, 2001) to display a phylogeny (in (b)).

3D Search Visualization

While tree space visualization is a current topic of research (Hillis *et al.*, 2005), PSODA provides a 3D graph of the progress of the search (see Figure 1). The default graph's axes are elapsed time, tree score and the number of trees found of the score. The graph is updated in real-time. Such a visualization can provide insights into the progress of the search and help a researcher gauge when to terminate a search.

Since PSODA is open source, it is possible for others to contribute by defining new dimensions to better map the phylogenetic tree space.

Phylogeny Viewer

To analyze a specific tree for biological accuracy it is necessary to view it. While there are several tree viewers available, other phylogeny search applications do not integrate a viewer into their program. To view the saved trees from a search often requires converting to a new format specific to the tree viewer of choice. Integrated into PSODA is ATV (A Tree Viewer) (Zmasek and Eddy, 2001). ATV is a powerful tree viewer written in Java, and therefore provides the same level of portability enjoyed by PSODA. Among the most useful features of ATV are its ability to view trees with a large number of taxa, view branch lengths and zoom in and out.

3.5 Cross-platform architecture

PSODA has been carefully designed to run on the most popular operating systems. Executable binaries of PSODA for Mac OS X, Linux and Windows operating systems are available from the PSODA website, `http://csl.cs.byu.edu/psoda`. Additionally, the source code is also available for contribution and modification.

3.6 Input format

PSODA uses the NEXUS format for inputting sequences, trees and commands. The format is familiar to many researchers, and there is a wealth of supporting tools that exist to convert existing data and create new NEXUS files. Additionally, PSODA allows auxiliary input beyond the NEXUS format to handle features not present in other phylogenetic search applications. An example of this is unaligned data. Currently, the NEXUS format does not support unaligned data (unless the sequences are all the same length). PSODA uses unaligned data to perform a multiple sequence alignment.

3.7 Multiple Sequence Alignment

While most phylogenetic search applications only handle previously aligned data sets, PSODA also can perform a progressive multiple sequence alignment (Feng and Doolittle, 1987) of unaligned data. Users can either provide a guide tree, or PSODA produces one. A guide tree is calculated by clustering sequences using the Neighbor-Joining algorithm (Saitou and Nei, 1987) according to their pairwise alignment scores. PSODA traverses the tree, aligning the most closely related sequences first (the leaves of the tree). After those sequences are aligned, it aligns sub-alignments of sequences. This continues and results in an alignment of all the sequences. PSODA uses the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970) to perform the alignments. Including an alignment algorithm in a phylogenetic search application allows for interesting combinations of alignment and phylogeny search to be efficiently combined.

Table 2 Meta-search Capabilities

	Wrappable	Internal Commands	Internal Scripting
PHYLIP	✓		
POY	✓	✓	
PAUP*	✓	✓	
BioPerl	✓	✓	✓
TNT	✓	✓	✓
PSODA	✓	✓	✓

3.8 PsodaScript

PSODA has a built-in language that allows users to create phylogenetic searches of their own design (Krein *et al.*, 2007). A full treatment of PsodaScript is given in the following sections.

4 PsodaScript

One of the driving purposes of PSODA is to provide a more complete framework for developing meta-search algorithms. To accomplish this, PSODA has been equipped with its own internal scripting language, PsodaScript.

4.1 Motivation

Researchers have developed a number of heuristic algorithms that perform reasonably well on large data sets; nevertheless, local optima in a search space often prevent these algorithms from finding the globally optimal solution. This problem is exacerbated by the inability of researchers—due to the vastness of the search space—to confidently distinguish between local and global optima.

One answer to the local optimum problem is meta-searching, which combines various heuristic searches together into a single search algorithm (Miolanen, 2000). In phylogenetic inference software packages, meta-searches may be constructed from existing algorithms either via an internal scripting language or an external wrapper. Unfortunately, wrapper applications tend to be cumbersome, often requiring considerable effort to orchestrate the interaction among sub-searches. Furthermore, having to code meta-search algorithms in a wrapper-script increases complexity and reduces clarity. In an effort to build upon predecessor applications and provide a solid, usable framework for developing meta-search algorithms, PSODA implements a scripting language—PsodaScript—with syntax and constructs for clearly and concisely defining phylogenetic searches.

4.2 Existing Approaches to Phylogenetic Meta-searching

Existing packages offer varying degrees of meta-searching capabilities (see Table 2). However, to perform advanced meta-searches most of these packages (including PHYLIP, POY, and PAUP*) must be wrapped by an external program or language such as DCM (Roshan *et al.*, 2004), Perl, or shell scripting. Although wrapping phylogenetic software with other applications or code enables some meta-searches, the process is unnecessarily complex.

One problem with wrapper applications and languages is that they introduce the complexity of coordinating and converting inputs and outputs between the various pieces of the meta-search. BioPerl, in particular, addresses this difficulty, and many researchers use BioPerl to make needed conversions. However, because BioPerl is based on the Perl programming language, researchers must have a Perl programming environment in order to use BioPerl. Unfortunately, installing Perl and the necessary modules can be frustrating, particularly for users with no experience in computer systems administration. Moreover, Perl syntax can be an obstacle for researchers who wish to write complex algorithms, but who do not have the time to learn the finer points of a powerful, yet complex language structure.

Additionally, scripting languages generally require researchers to use syntax and terms that are foreign to the phylogenetic research domain. POY and PAUP* have done well to tailor their syntax to the target audience, and these packages also facilitate some basic meta-searches. Nevertheless, both applications still require external programming to run more advanced meta-search algorithms.

TNT implements its own internal scripting language which is capable of performing advanced meta-searches. However, TNT is closed-source, and its syntax—like Perl's—provides great power, but at the expense of simplicity. PsodaScript takes the middle ground between the command languages of POY and PAUP* and the heavy scripting languages of Perl and TNT. It is an internal scripting language with a simplified syntax and command names from the phylogenetic domain. Therefore, PsodaScript not only makes writing advanced meta-search algorithms possible, but also more intuitive. Further, it frees researchers from the burdens of coordinating and converting inputs and outputs between sub-applications and the need to setup and maintain auxiliary applications and programming environments.

4.3 PsodaScript: Methods & Procedures

To date, researchers have developed several successful meta-searching techniques (Nixon, 1999; Goloboff, 1999). However, without internal scripting capabilities and more advanced language constructs, using and experimenting with these techniques and other meta-search algorithms is awkward and limited. Our challenge is to design and implement an internal scripting language for PSODA in a way that would enable (even encourage) meta-search exploration, without compromising simplicity or usability.

Compatibility with the NEXUS File Format and PAUP's Command Set*

As is the case with the popular package PAUP*, input files for PSODA generally follow the NEXUS file format. This format was proposed in 1997 to be an extensible format for describing systematic information (Maddison *et al.*, 1997). PSODA is also designed to execute PAUP* blocks, although it's language does not yet implement every PAUP* command. Before execution, the PSODA interpreter warns the user about any unimplemented commands, references the commands by line

Sample 1 PsodaScript is based on PAUP*'s command syntax and the NEXUS file format

```
  hsearch start = current swap = TBR;
```

number, and then skips over them during execution. As PSODA continues to be extended, more features and algorithms will be incorporated into the application.

There are two important reasons why PsodaScript was developed on the foundation of the NEXUS file format and PAUP*'s command set (see Sample 1). First, many people are already familiar with this format and command style, making it easier for users to begin using PsodaScript. The other motivation lies in the simplicity of PAUP*'s syntax and commands, which seem to be well suited to phylogenetic researchers. Building on PAUP*'s syntax, PsodaScript is intended to be understandable by someone with little programming experience. Where appropriate, PSODA uses words rather than obtuse symbols, and the more advanced language constructs are intended to read like English statements. A person who is familiar with PAUP* commands should be able to understand the general idea of what a PSODA program does, even if they have had no prior exposure to it.

A Quick Introduction to PsodaScript Syntax

In keeping with PAUP*'s style, PSODA instructions are each listed as distinct statements terminated by a semicolon. The instruction in Sample 1 tells PSODA to run a heuristic search beginning with the current tree(s) in the tree repository and using the TBR method to explore the search space. Even with very little exposure, the syntax and terminology make sense.

In addition to running PAUP* commands, PsodaScript also allows users to store values in user variables. Assignment to a variable is performed via the `=` operator. For instance, the statement `height = 5;` assigns the value of 5 to the variable `height`. If the variable does not yet exist, it is created and initialized to the value `5`. To avoid unintended side effect of assigning variables, PsodaScript uses the notion of variable scoping, which means that a variable exists within a well defined region in the program. If a variable is created within a loop, for instance, then it will not be visible, or accessible, outside of that loop; it is considered, therefore, local to that loop. If a variable is created outside of all constructs and user defined commands, then it will be visible throughout the program.

Another point to note about variables in PsodaScript is that they are loosely typed, as in Perl or JavaScript. This means that users do not have to explicitly declare the type of data that a variable will hold when they create it. One variable in PsodaScript can be assigned data of any type, and when necessary, the PSODA interpreter will attempt to perform conversions. PsodaScript variables can also be combined with operators to form arithmetic and logical expressions.

Conditional constructs allow users to determine how a program (or meta-search) should proceed based on the current state of the search. Sample 2(a) illustrates the syntax for a conditional construct. In a conditional construct, the instructions following the first true condition are executed once, after which the interpreter skips to the end of the conditional (`endif`) to continue on with the program. If none of the conditions evaluate to true, then the `else` component is executed. A

Sample 2 (a) Syntax for a Conditional Construct (b) Syntax for a Loop Construct

```
if (condition-1)
 ...
elsif (condition-2) ...
elsif (condition-3) ...
else ...
endif;
```

(a)

```
while (condition) ...
endwhile;
```

(b)

conditional statement may or may not have an `else` component, and it may have zero or more `elsif` components.

The loop construct is written in a similar format (see Sample 2(b)). In a while loop construct, the body is repeatedly executed as long as the condition expression evaluates to true.

Error Checking and Handling

Given its critical impact on usability, an important effort in PsodaScript is the development of effective error checking and handling. Because phylogenetic searches frequently run for several days or weeks at a time, error handling should include some degree of data recovery following a fatal scripting error. Good error checking and handling is even more important when running meta-search algorithms, which are more likely (due to code branching) to run longer before encountering faulty logic. Before execution, the PSODA interpreter informs the user of fatally incorrect syntax and gives warnings for several non-fatal errors. For fatal logic errors, which may surface during execution, the interpreter informs the user of the error and the need to quit. It then gives the user an opportunity to save the trees in the repository before exiting.

User-defined Commands: Facilitating Algorithm Readability and Design

PsodaScript also supports user-defined commands for grouping program statements together under a single command name. The ability to group statements is a simple but powerful concept. Consider a large meta-search algorithm that has the ability to perform various heuristic searches, each of which is derived from a base search and modified via a specific configuration of parameters, weights, and so forth. Duplicating the code for each heuristic search every time it is used leads to a messy program block, which translates into more mistakes and decreased algorithm readability and re-usability. In short, grouping and labeling program segments is a simple technique that makes an algorithm much easier to understand as a whole; it helps current and future researchers answer the question, "What is this algorithm really doing?"

An illustrative user-defined command in Sample 3(a), `randomReweight`, randomly skews a given percentage of column weights—part of a technique used by the ratchet (Nixon, 1999) to escape a local optimum. It is called as if it were a built-in PsodaScript command. Notice that the first statement in the `randomReweight` command creates two variables, `range` and `percent`, with the default values of `3` and `25` respectively. Just like any other PsodaScript commands, this command could be called with a parameter: `randomReweight percent = 20;`. Listing `percent = 20` as a parameter, overrides the default value of 25.

The real benefit of user-defined commands, therefore, is their ability to abstract away substantial segments of code, which can then be executed as often as needed via a single, parameterized command call.

5 Applications

To demonstrate the use of PsodaScript for meta-search construction, we present here the implementation of two specific meta-search algorithms: parsimony ratchet (section 5.1) and iterative alignment and phylogeny search (section 5.2).

5.1 Parsimony Ratchet

As discussed above, heuristic searches for optimal phylogenetic trees are often caught in local optima and thus fail to find the global optimum. The parsimony ratchet attempts to avoid this problem by executing a series of heuristic searches alternating between skewed and normal weightings for tree scoring (Nixon, 1999). For the sake of comparison, we present a ratchet that could be run by PAUP* together with a possible implementation of a ratchet search in PsodaScript (see Sample 3). Developers have created tools that facilitate using the ratchet in PAUP* (or PSODA) by generating long sequences of commands that simulate looping and random number generation (Sikes and Lewis, 2001). The PAUP* ratchet dramatically improves search performance and can be run in PSODA; however, using the programming constructs of PsodaScript to implement the ratchet offers a number of advantages in addition to the efficiency gained from using the ratchet.

For instance, during the ratchet, better scoring trees are replaced by other (perhaps worse) trees on subsequent iterations. With variables and conditional constructs, however, PsodaScript can also track the best score found so far, and save only those trees that achieve a new best score. Additionally, PAUP* merely *simulates* a loop by running a long sequence of heuristic searches, the length of which is ultimately finite and must be predetermined. On the other hand, with a simple loop, PsodaScript can run indefinitely, generating new random numbers on each iteration.

Another disadvantage to the PAUP* ratchet is the added dependence on external software, which further obfuscates the algorithm. On the other hand, consider the while loop in the main body of the PSODA program (see Sample 3(a)). It makes the algorithm of the ratchet search clear. First, twenty percent of the weights are randomly re-weighted with skewed values, and the search continues with the skewed weights. Afterward, the weights are reset to their initial values, and the search con-

Sample 3 (a) Implementation of the ratchet in PsodaScript. (b) Example implementation of the ratchet in PsodaScript. Note, due to space constraints, text was omitted and only a few iterations are shown. Normally, a ratchet search includes several iterations.

```
#NEXUS

BEGIN PSODA;

begin randomReweight (range=3, percent=25);
  numColumns = getWeightsLength();
  numToChange = numColumns * percent / 100;
  j = 0;
  while (j < numToChange)
    column = random (max=numColumns) + 1;
    newWeight = random (max=range);
    weights newWeight:column;
    j++;
  endWhile;
end;

set (maxtrees=1, criterion=parsimony);
hsearch (start=stepwise, swap=tbr);
while (true)
  randomReweight (percent=20);
  hsearch (start=current, swap=tbr);
  weights reset;
  hsearch (start=current, swap=tbr);
  print (text = "Score: ");
  print (text = getBestScore() . endline);
endwhile;

END;
```

(a)

```
#NEXUS

BEGIN PAUP;
set criterion=parsimony;
set maxtrees=1 increase = no;
hsearch start=stepwise swap=TBR;

weights 2 : 7 14 17 26 27 31 34 45 50 52 54 57 60 63 64
77 86 91 92 102 103 107 115 117 121 122 124 127 131 133
134 140 142 155 156 163 173 176 183 185 187 195 197 198
202 204 209 219 221 222 225 226 230 237 238 240 241 244
248 252 254 255 258 269 276 279 283 284 291 294 295 297
309 310 315 319 321 323 325 326 342 346 349 350 351 353
354 355 359 360 363 366 370 377 378 390 393 394 398 408
412 413 418 419 423 425 426 428 429 432 450 456 467 475
479 482 484 489 492 493 497 498 500;
hsearch start=current swap=TBR;
weights 1:all;
hsearch start=current swap=TBR;

weights 2 : 1 5 17 22 26 31 34 39 44 46 58 ... 474 483;
hsearch start=current swap=TBR;
weights 1:all;
hsearch start=current swap=TBR;

weights 2 : 2 9 11 14 16 18 20 24 28 36 40 ... 488 491;
hsearch start=current swap=TBR ;
weights 1:all;
hsearch start=current swap=TBR;

weights 2 : 1 5 17 22 26 31 34 39 44 46 58 ... 485 490;
hsearch start=current swap=TBR;
weights 1:all;
hsearch start=current swap=TBR;

weights 2 : 2 3 9 20 22 24 25 26 28 30 32  ... 494 498;
hsearch start=current swap=TBR;
weights 1:all;
hsearch start=current swap=TBR;

weights 2 : 2 3 6 12 25 29 30 37 50 56 60  ... 485 495;
hsearch start=current swap=TBR;

...
```

(b)

tinues under normal weights. This process of alternating the search between skewed and normal weights is repeated until the user tells PSODA to stop.

A further benefit of the PsodaScript framework is the way it facilitates experimentation with parameters, such as the range of random numbers used in the weighting or the percentage of skewed weights. Additionally, researchers can use print statements to periodically output information about the state of the search. Performing these simple tasks without a programming language is impractical, and as noted above, external languages introduce unnecessary complexity.

5.2 Iterative Alignment and Phylogeny Search

Phylogenetic inference frequently begins by producing a multiple sequence alignment (MSA) using software such as ClustalW (Thompson *et al.*, 1994) and an initial phylogenetic guide tree. In preparation for a phylogenetic search, MSA inserts gaps into the sequence data to make all sequences the same length. Research has shown that the quality of the alignment significantly impacts the success of the phylogenetic search (Morrison and Ellis, 1997). The software package POY performs

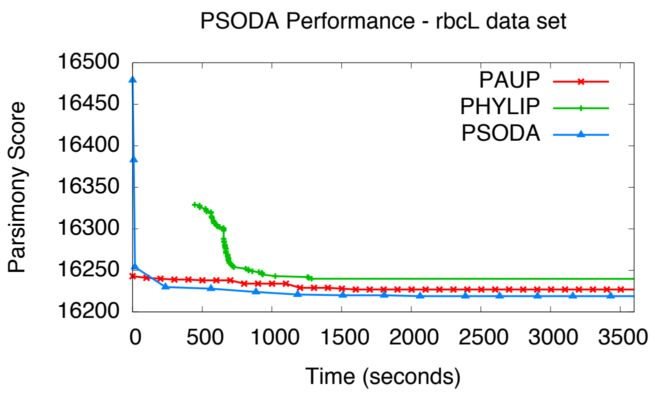


Figure 2 Performance results, in terms of best parsimony score found over time, for a parsimony search using PSODA and PAUP* on the 500 taxa seed plant rbcL data set (Chase *et al.*, 1993). Note: lower parsimony score is better.

a MSA for every tree searched—thereby making expensive MSA calculations on suboptimal trees (Wheeler *et al.*, 2006). A possibly more efficient alternative to POY's approach is to iterate between alignment and phylogeny searches (Gotoh, 1996; Ridge *et al.*, 2006). In PsodaScript, this is easily implemented as a while loop that invokes a parsimony search and alignment on the best tree found.

6 Results

PSODA performs tree searches comparable to other phylogenetic search packages. The results presented here were run at the Ira and Mary Lou Fulton Supercomputing Laboratory at Brigham Young University. Each node of the computers used has two Dual-core Intel Xeon EM64T processors (2.6GHz) and 8 GB of memory. The data set, *rcbL* (Chase *et al.*, 1993), is comprised of 500 plant seed taxa, each with a length of 759 sites. It is one of the most studied data set in systematics. Figure 2 illustrates the results of running a TBR search with PSODA and PAUP on the rcbL data set. While PAUP initially has better performance (the first 20 seconds), PSODA quickly catches up and surpasses PAUP. The best parsimony score found by PAUP is 16,227 (after 1,507 seconds). PSODA finds a phylogeny with a better (of 16,226) after only 653 seconds. Furthermore, it is interesting to note that the best parsimony tree score published for this data set is 16,218 (Nixon, 1999), and PSODA achieved 16,219 (after 2,033 seconds) with simpler methods than those used elsewhere. Using a ratchet-style search implemented in PsodaScript, PSODA also found trees with a 16,218 parsimony score.

7 Conclusion

PSODA is an open source phylogeny reconstruction package made freely available to the public. It implements traditional search algorithms for Maximum Parsimony and Maximum Likelihood as well as more advanced search techniques. It also provides a user-friendly GUI, and is available for multiple operating systems. The input format is compatible with PAUP*. Furthermore, PSODA's performance is comparable with PAUP*. Finally, PSODA has several features unique to itself, such as integrated graphing visualizations, a multiple sequence alignment algorithm and a scripting language, PsodaScript.

PsodaScript allows users to quickly and easily design meta-searches. By designing meta-searches which appropriately combine various heuristics and parameter settings, phylogenists greatly improve the practicability of using inferred phylogenetic trees to solve problems. The internal scripting abilities of PSODA give researchers the flexibility to better explore and exploit the realm of phylogenetic meta-searching while addressing several limitations of previous phylogenetic applications.

Acknowledgments

We would like to thank the beta testers of PSODA for their insightful comments. This material is based upon work supported by the National Science Foundation under Grant No. 0120718.

References and Notes

1 Camin, J. and Sokal, R. (1965). A Method for Deducing Branching Sequences in Phylogeny. *Evolution*, **19**(3), 311–326.

2 Carroll, H., Ebbert, M., Clement, M., and Snell, Q. (2007). PSODA: Better Tasting and Less Filling Than PAUP. In *Proceedings of the 4th Biotechnology and Bioinformatics Symposium (BIOT-07)*, pages 74–78.

3 Chase, M., Soltis, D., Olmstead, R., Morgan, D., Les, D., Mishler, B., Duvall, M., Price, R., Hills, H., Qiu, Y., *et al.* (1993). Phylogenetics of Seed Plants: An Analysis of Nucleotide Sequences from the Plastid Gene rbcL. *Annals of the Missouri Botanical Garden*, **80**(3), 528–580.

4 Chor, B. and Tuller, T. (2005). Maximum likelihood of evolutionary trees is hard. *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2005*, **3500**, 296–310.

5 Eisen, J. A. (1998). A Phylogenomic study of the MutS family of proteins. *Nucleic Acids Research*, **26**(18), 4291–4300.

6 Felsenstein, J. (1981). Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. *Journal of Molecular Evolution*, **17**(6), 368–376.

7 Felsenstein, J. (2004). *Inferring Phylogenies*. Sinauer Associates, Sunderland, MA.

8 Felsenstein, J. (2007). PHYLIP (Phylogeny Inference Package) version 3.67. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.

9 Feng, D.-F. and Doolittle, R. F. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, **25**(4), 351–360.

10 Goloboff, P., Farris, S., and Nixon, K. (2007). TNT: Tree analysis using new technology. `http://www.zmuc.dk/public/phylogeny/TNT/`.

11 Goloboff, P. A. (1999). Analyzing large data sets in reasonable times: Solutions for composite optima. *Cladistics*, **15**(4), 415–428.

12 Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, **264**(4), 823–838.

13 Graham, R. L. and Foulds, L. R. (1982). Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, **60**(2), 133–142.

14 Hillis, D., Heath, T., and St. John, K. (2005). Analysis and Visualization of Tree Space. *Systematic Biology*, **54**(3), 471–482.

15 Krein, J. L., Teichert, A. R., Carroll, H. D., Clement, M. J., and Snell, Q. O. (2007). PsodaScript: Applying Advanced Language Constructs to Open-source Phylogenetic Search. In *Proceedings of the 4th Biotechnology and Bioinformatics Symposium (BIOT-07)*, pages 89–94.

16 Kumar, S., Tamura, K., and Nei, M. (1994). MEGA: Molecular Evolutionary Genetics Analysis software for microcomputers. *Bioinformatics*, **10**, 189–91.

17 Lipman, D. J. and Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. *Science*, **227**(4693), 1435.

18 Maddison, D. R., Swofford, D. L., and Maddison, W. P. (1997). NEXUS: An Extensible File Format for Systematic Information. *Systematic Biology*, **46**(4), 590–621.

19 Meier, R. and Ali, F. B. (2005). The newest kid on the parsimony block: TNT (Tree analysis using new technology). *Systematic Entomology*, **30**(1), 179–182.

20 Miolanen, A. (2000). Simulated evolutionary optimization and local search: Introduction and application to tree search. *Cladistics*, **17**, S12–S25.

21 Mooers, A. O. and Heard, S. B. (1997). Inferring Evolutionary Process from Phylogenetic Tree Shape. *The Quarterly Review of Biology*, **72**(1), 31–54.

22 Morrison, D. A. and Ellis, J. T. (1997). Effects of nucleotide sequence alignment on phylogeny estimation: A case study. *Molecular Biology and Evolution*, **14**, 428–441.

23 Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48**, 443–453.

24 Nixon, K. C. (1999). The Parsimony Ratchet, a New Method for Rapid Parsimony Analysis. *Cladistics*, **15**(4), 407–414.

25 Rambaut, A., Posada, D., Crandall, K. A., and Holmes, E. C. (2004). The causes and consequences of HIV evolution. *Nature Reviews Genetics*, **5**, 52–61.

26 Ridge, P. G., Carroll, H. D., Sneddon, D., Clement, M. J., and Snell, Q. O. (2006). Large Grain Size Stochastic Optimization Alignment. In *Proceedings of the Sixth IEEE Symposium on BionInformatics and BioEngineering (BIBE'06)*, pages 127–134.

27 Roshan, U. W., Moret, B. M. E., Warnow, T., and Williams, T. L. (2004). Rec-I-DCM3: A Fast Algorithmic Technique for Reconstructing Large Phylogenetic Trees. In *Computational Systems Bioinformatics Conference, 2004. CSB 2004. Proceedings.*, pages 98–109. IEEE, IEEE.

28 Saitou, N. and Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, **4**, 406–425.

29 Sanderson, M. J. (1990). Flexible Phylogeny Reconstruction: A Review of Phylogenetic Inference Packages. *Systematic Zoology*, **39**(4), 414–420.

30 Sikes, D. S. and Lewis, P. O. (2001). *PAUPRat: a tool to implement parsimony ratchet searches using PAUP**. Download site: http://www.ucalgary.ca/ dsikes/software2.htm.

31 Stamatakis, A. (2006). RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, **22**(21), 2688.

32 Sundberg, K., O'Connor, T., Carroll, H., Clement, M., and Snell, Q. (2007). Using parsimony to guide maximum likelihood searches. In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering*, volume II, pages 774–779.

33 Swofford, D. L. (2003). *PAUP*. Phylogenetic Analysis Using Parsimony (* and Other Methods). Version 4*. Sinauer Associates, Sunderland, Massachusetts.

34 Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*, **22**, 4673–4680.

35 Wheeler, W., Aagesen, L., Arango, C. P., Faivovich, J., Grant, T., D'Haese, C., Janies, D., Smith, W. L., Varón, A., and Giribet, G. (2006). *Dynamic Homology and Phylogenetic Systematics: A Unified Approach Using POY*, chapter Overcoming Limitations in Phylogenetics, page 34. American Museum of Natural History. Published in cooperation with NASA Fundamental Space Biology.

36 Wheeler, W. C. (1996). Optimization alignment: the end of multiple sequence alignment in phylogenetics? *Cladistics*, **12**, 1–9.

37 Wheeler, W. C., Gladstein, D. S., and Laet, J. D. (1996–2003). POY: Version 3.0.

38 Wolfe, N. D., Escalante, A. A., Karesh, W. B., Kilbourn, A., Spielman, A., and Lal, A. A. (1998). Wild Primate Populations in Emerging Infectious Disease Research: The Missing Link? *Emerging Enfectious Diseases*, **4**(2).

39 Zmasek, C. M. and Eddy, S. R. (2001). ATV: display and manipulation of annotated phylogenetic trees. *Bioinformatics*, **17**(4), 383–384.