

# Software manual for PAUPRat: A tool to implement Parsimony Ratchet searches using PAUP\*

by

**Derek S. Sikes and Paul O. Lewis**

*Department of Ecology and Evolutionary Biology  
University of Connecticut  
Storrs, CT 06269*

April, 2001

PAUPRat is a program to generate a text file that contains commands which, when read by PAUP\* (with an open and executed data file), will implement the Parsimony Ratchet of Kevin Nixon (1999b). The Parsimony Ratchet is a search strategy that is surprisingly efficient at finding shortest trees for data sets too large for traditional heuristic search methods. This manual provides a basic introduction to the Parsimony Ratchet and information on how to use PAUPRat. PAUPRat comes in three flavors: Mac, PC and Linux (Intel executable)

## Section I: The Parsimony Ratchet

**Brief History:** In May of 1998, Kevin Nixon presented results of a novel method to quickly find optimal parsimony trees for even large datasets (Horovitz, 1999). He named this method the "Island Hopper" or the "Parsimony Ratchet" and demonstrated its effectiveness at finding optimal trees using the 500 taxon seed plant *rbcL* data set of Chase *et al.* (1993). This method was described formally by Nixon (1999b). In contrast to the 11.6 months of CPU time consumed by Rice *et al.* (1997) in searching for optimal topologies with this data set, Nixon's Parsimony Ratchet, run using the program NONA, would typically find within 22 h shorter trees than Rice *et al.* (1997) found (Nixon, 1999b).

Nixon (1999b) reports that the Parsimony Ratchet has been implemented in the following programs: DADA (Nixon, 1998), Winclada (Nixon 1999a), and NONA (Goloboff, 1998). Goloboff, Farris, and Nixon (2001) are working on another program, TNT, designed specifically for analyzing large data sets, which implements additional fast-search algorithms first presented in Goloboff (1999). TNT is available for beta-testing by contacting Goloboff.

Regarding the Parsimony Ratchet Nixon (1999b) states "*The method can be easily implemented with existing phylogenetic software by generating batch command files.*" It was only a matter of time before the idea was implemented in PAUP\*, and we know of at least one additional, independent implementation of the Ratchet with PAUP\*.

The first author conceived of the idea to use the Ratchet with PAUP\* while working as a teaching assistant. He was preparing a laboratory exercise on phylogenetic analysis of large data sets for David Wagner's BioSystematics course at the University of Connecticut in the Fall of 1999 and demonstrated the Ratchet's abilities with the Chase *et al.* (1993) data set using PAUP\*. The second author improved on the idea by writing an executable to generate the batch files.

**How it works:** Prior to the development of the Ratchet, the best search strategy for large datasets involved a trade-off between visiting many tree islands and searching each island thoroughly. We, and many others, have found that the most time-optimal solutions tend to sacrifice thorough island searching of each island in exchange for increases in the number of islands visited. Figure 1 below displays the effect of minimizing search time spent on any given island using the *rbcL* dataset. This approach to heuristic searching involves performing many independent search replicates, each starting from a stepwise addition tree in which the addition sequence has been selected randomly. These searches, being independent of each other, could be arranged in any chronological order. There is the same probability of hitting the shortest tree on the first replicate as the last replicate.

Rather than perform many thorough, independent heuristic searches, the Ratchet performs what amounts to a single long search comprising a series of short heuristic searches. One of the trees on a tree island is in memory at this point, but this

island is often not the island containing the globally most parsimonious tree.<sup>1</sup> The Ratchet (now “stuck” on a suboptimal tree island) gets itself “unstuck” by warping treespace slightly, which often turns a gap between adjacent islands in treespace into a bridge connecting the two. An “adjacent” tree island might be defined as an island whose component trees are only two (or a very few) steps away from the trees in the current island. When treespace is then returned to its original shape, the search can proceed from its new starting point, which will either be on the same island, a better island or a worse island. If the island is the same or a worse island, then no progress is made in that particular segment of the overall search. However, if the warping of tree space allows the Ratchet to slip over to a better island, then substantial progress can be achieved because of the excellent position of the new starting point.

Our version of the Ratchet saves time (sacrificing thoroughness) by always starting with just one (arbitrarily chosen) starting tree from the previous iteration, even though there may have been many most parsimonious trees found during that previous iteration. This is accomplished in PAUP\* by setting `multrees=no`, `nchuck=1` and `chuckscore=1`.

The Parsimony Ratchet, in contrast, as can be seen with the *rbcL* data set in Fig.2 below, proceeds through tree space generally without losing any progress achieved in prior iterations, thus the name ‘Ratchet.’ Therefore, in Ratchet searches, each subsequent iteration is not independent of the previous, and the probability of hitting the shortest tree usually increases with each iteration.

The Ratchet is simple to implement; see Nixon's (1999b) paper for a complete list of the steps involved. Briefly, the mechanics involve creating a starting tree (which is the only tree created ‘from scratch’ during a Ratchet run) and then performing two searches for each Ratchet iteration: one using a subset of the characters (randomly chosen but of a fixed count) that are reweighted or ‘perturbed’ and another with the characters returned to their original weights. The best tree found for each search is passed along to the next search. The search with the perturbed weights tends to push the search into a new island of tree space, but not a randomly chosen island, rather a neighboring island that can be reached by altering the shape of the tree landscape. The fact that the Ratchet works suggests that good islands tend to be clustered in treespace.

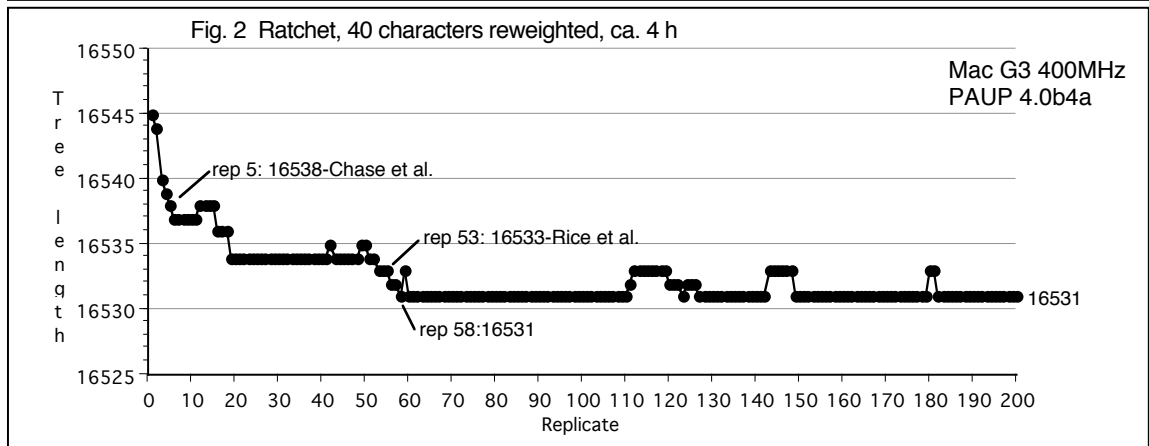
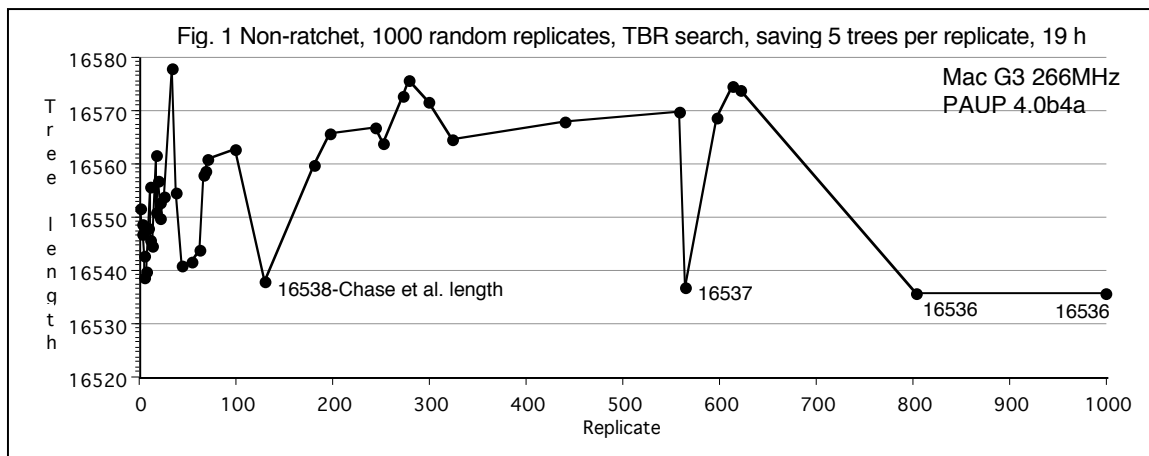
In general, using the 500 taxon *rbcL* data set as a test piece, the Parsimony Ratchet implemented in PAUP\* takes from about 2 to 8 hours to complete 200 iterations with 50 characters perturbed per iteration, depending on the speed of the CPU being used. Using the latest PAUP\* build (4.0b8) and a 733MHz G4 Macintosh it takes just under 1.75 hours to complete a single 200 iteration search (compare to 6 to 9 hours per run on the fastest machines of 1999). The first two runs failed to find the shortest tree but came within 1 step. The third run found the shortest tree (length 16531 [=length 16218 of Nixon (1999b)])<sup>2</sup> by iteration 39, only 20 minutes after starting. For comparison, a 200 iteration run using this dataset, was performed using the Windows and Linux versions of PAUP\* and PAUPRat. The results were: Windows 2000 machine (Pentium 800 MHz): 1 hour, 53 minutes; Red Hat Linux 6.2 machine (Pentium 500 MHz): 3 hours, 1 minute.

Not every 200 iteration run of the Ratchet will find the shortest tree for this dataset. Although it is unusual to not find the shortest tree in two successive runs of 200 iterations each, Nixon (1999b) reported that with this dataset, his implementation of the Ratchet using NONA found shortest trees in three of every four 200 iteration runs. This highlights the importance of doing multiple Ratchet runs, preferably 10 or more, before feeling confident in the results.

---

<sup>1</sup> A tree island is defined as a group of trees having the same parsimony score, and this score is better than that of any other tree reachable by a single branch swapping move. (The definition of a tree island thus changes depending on which branch swapping strategy you elect to use; NNI branch swapping generally yields far more islands than the TBR strategy because TBR can “reach further” with a single move.)

<sup>2</sup> **Note regarding differences of tree lengths reported between Nixon (1999b) and Rice *et al.* (1997):** Surprisingly there is no consensus on how to count steps for a parsimony tree; users of NONA apparently do not include uninformative characters in their tree lengths, whereas users of PAUP\* do. For the *rbcL* matrix PAUP\* reports 1428 characters, 30 of which are excluded because they represent the primer sequences, thus leaving a matrix of 1398 characters for tree searches. Of this total, 376 characters are constant, and 263 variable characters are parsimony uninformative leaving 759 parsimony informative characters. In Nixon (1999b) and Goloboff (1999) these authors state that this data matrix has 1428 characters (without indication of whether the primer sequences were included in their searches) and Goloboff (1999) states that the shortest trees found by Rice *et al.* (1997) were “16,220 steps (they counted uninformative characters thus adding 313 steps).” Rice *et al.* (1997) reported tree lengths of 16,533 – a difference of 313 steps. Throughout Nixon (1999b) and Goloboff (1999) this difference of 313 steps is maintained, thus our shortest trees for this matrix, with lengths of 16531, equal Nixon’s lengths of 16218, a difference of 313 steps.



Note that in Fig. 1 only trees of lengths not previously found are plotted, thus as the search replicates proceed, the number of tree lengths not previously found declines and the plotted tree lengths become less dense.

## Section II: Using PAUPRat

Below is the NEXUS file (named *verbose\_setup.nex*) that is output by PAUPRat upon typing "help" into the program and hitting "OK". This file contains commands understood by PAUPRat. Note that a non-annotated setup file (named *setup.nex*) can be generated by following step 3 below. Either *verbose\_setup.nex* or *setup.nex* should be edited before being submitted to PAUPRat. At a minimum, the number of characters in the data set to be analyzed must be specified in the dimensions (*nchar=*) command. Comments in brackets [] are ignored by PAUPRat and PAUP\*; commands are in **bold**. See Notes at the bottom for additional information.

**NOTE for Macintosh users:** Unlike some other command line Mac programs PAUPRat responds to text typed into the rectangular box that appears when the program is first launched. Do not click on the button marked 'quit.' To create a terse setup file to edit rather than the annotated one below: launch PAUPRat, click the button labeled 'OK' and when PAUPRat is done displaying the results you can quit by typing command-Q. In the same folder as PAUPRat will appear a text file named 'setup.nex' that can be clicked to edit in the PAUP or any text editor. Once this setup.nex file has been edited (at least the *nchar* is specified) launch the PAUPRat program again and type setup.nex into the field and click OK. Next (when it is done spewing text onto the screen), quit PAUPRat and choose 'Don't save'. PAUPRat will have created the batch file that you will feed into PAUP and it will be titled *Ratchet.nex*

-----  
#nexus

[  
PAUPRat example setup file. Sikes, D. S. and P. O. Lewis. 2001. beta software, version 1. PAUPRat: PAUP\* implementation of the parsimony ratchet. Distributed by the authors. Department of Ecology and Evolutionary Biology, University of Connecticut, Storrs, USA. April 2001. Based on Kevin Nixon's Parsimony Ratchet as described in: Nixon, K. C. 1999. The Parsimony Ratchet, a new method for rapid parsimony analysis. Cladistics 15: 407-414.  
]

[  
Notes:

- \* This file can be regenerated automatically by running PAUPRat with the single command 'help' (first make sure there is no file named verbose\_setup.nex in PAUPRat's folder)
- \* PAUPRat ignores comments (like this one) delimited by square brackets

Suggested Protocol:

1. Read through this NEXUS file, if the PAUP\* commands are alien to you it might be best to familiarize yourself with PAUP\* command line terminology by reading through the pertinent sections of the PAUP\* manual (obtainable from : <http://www.lms.si.edu/PAUP/> ). Also read Nixon's (1999b) paper describing the Parsimony Ratchet, which describes his method in greater detail.
  2. Create a folder/directory in which your PAUPRat files will reside. Place the application/executable PAUPRat in this folder.
  3. Create a setup file for PAUPRat to read with instructions on how the Ratchet batch file for PAUP\* should be made. A non-annotated version of this file can be generated by launching PAUPRat, leaving the argument line blank, and hitting OK (if using the Windows version, run PAUPRat from an MS-DOS command line). The file created will be named setup.nex and should need only minor modification. NOTE: the names of all files made should have no spaces in them.
  4. Execute/launch PAUPRat, which is command-line-driven, and type the name of the setup file you created in step 3 (which must be in the same folder/directory).
  5. Hit return and PAUPRat should generate a batch file named ratchet.nex in the same folder/directory.
  6. Your data file and PAUP\* do not have to be in the same directory. Launch PAUP\* and execute your datafile. Then tell PAUP\* to execute your batch file (named ratchet.nex in step 5 above) using the 'File/Open' menu commands. PAUP\* should implement a Parsimony Ratchet search of your description.
- Note: on a Macintosh keep PAUP\* as the frontmost application to give it top CPU priority (=maximum speed); and, use the 'windowshade' feature to collapse the display buffer when you are not reading it while the ratchet is running - this tends to save some time in redrawing the screen - more noticable with smaller datasets.
7. Upon completion of the search, PAUP\* will indicate that all trees

are saved and it is safe to quit the program. Tree lengths for each iteration are displayed (see below) and all shortest trees are saved in the file mydata.tre.

Note: without a search monitor box, a PAUP\* search can be aborted on a Macintosh by pressing the command and period (.) keys simultaneously.

]

**begin pauprat;**

[specify below the total number of characters, e.g. nchar=1428;]

**dimensions nchar=;**

[  
Notes about nmod and pct:

- 
- \* use nmod to specify absolute number of characters to perturb each iteration
  - \* use pct to specify a percentage of characters to perturb each iteration
  - \* do not use both nmod and pct simultaneously (if you do, the last one used wins)
  - \* Nixon (1999b) states that perturbing between 5% and 25% of the total number of informative characters usually works well.

Notes about random number seeds:

- 
- \* set seed=0 tells PAUPRat to generate a random number seed using the system clock
  - \* set seed=1 or other number to specify your personal favorite random number seed
  - \* the random number seed determines the sequence of 'random' numbers used in choosing characters to perturb for each iteration
  - \* running PAUPRat twice with the same random number seed (other than 0) will result in exactly the same characters being perturbed each iteration
  - \* the safest thing to do is let PAUPRat choose a new seed each time you run it (by setting seed=0)

Notes about nreps:

- 
- \* nreps specifies the number of Ratchet iterations to have PAUP\* perform
  - \* each iteration consists of a search under perturbed character weights followed by a second search under the original weights

Notes about wtmode:

- 
- \* use 'wtmode=uniform' to ensure that original weights are always 1 for all characters
  - \* use 'wtmode=additive' to handle the case of unequal original weights in an additive way. 'additive' means a character originally weighted 10 will act just like 10 separate characters each weighted 1: it will be 10 times more likely to be chosen for perturbation than a character whose original weight was 1, but each time it is chosen, its weight will be incremented by just 1
  - \* use 'wtmode=multiplicative' to handle the case of unequal original weights in a multiplicative way. If 'multiplicative' mode is

chosen, a character originally weighted 10 will increase its weight by 10 each time it is selected for perturbation, but it will not be any more likely to be chosen than a character whose original weight was 1

Notes about verbose and terse:

- \* specify 'verbose' if you want comments in the output indicating which characters were perturbed and how much their weights changed for each iteration
- \* normally, you want to specify 'terse' because these comments add a lot of bytes to the file and don't change the analysis at all
- \* do not specify both 'terse' and 'verbose' - if you do, the last one specified wins

]

```
set terse seed=0 nreps=200 pct=15 wtmode=uniform;
```

[

You can specify unequal original weights using a wtset command, for example:

```
WTSET * UNTITLED =
0: 1 55 78 81 85 90 94 101 104 106 111-112 120-121 126 132 138 140,
10: 2-15 17-19 21-45 48-54 56-59 66 69 71 79 86 102-103 108-109
117-118 130 134-137 141-142, 1: 16 20 46 60-65 67-68 70 72-77 80
82-84 87-89 91-93 95-100 105 107 110 113-116 119 122-125 127-129 131
133 139, 5: 47;
```

This works just like the equivalent command in PAUP\* (allowing the wtset command to simply be copied directly from the original nexus data file), except that the asterisk is ignored in PAUPRat (if you put in a wtset command, it will be used by PAUPRat whether or not there is an asterisk).

If you leave out the wtset command or specify wtmode=uniform in the set command, PAUPRat will behave just like you used this wtset command:

```
wtset * allequalone = 1: 1-1428;
```

Several commands may be used at this point:

PAUPRat command	When PAUP* will execute command
startcmd	before ratchet has begun
normcmd	only when characters have their original weights
rewtdcmd	only when characters have their perturbed weights
paupcmd	both when characters have their perturbed weights AND when characters have their original weights
stopcmd	only after the ratchet has finished

Note: we have not found any use for rewtdcmd, but it has nevertheless been implemented just in case someone else finds a use for it.

]

[

Lines below create the starting tree. If status=yes PAUP\* will show the search status for the starting tree but will also require a human to click 'close' after the starting tree is found, if status=no, the

ratchet begins automatically after finding the starting tree and does not display the search status for that tree.

]

```
startcmd "[!*****]";
startcmd "[!* ----- PAUP* Ratchet ----- *]";
startcmd "[!*   Derek S. Sikes & Paul O. Lewis   *]";
startcmd "[!*     University of Connecticut     *]";
startcmd "[!*           March, 2000           *]";
startcmd "[!* Based on Kevin Nixon's Parsimony *]";
startcmd "[!* Ratchet as described in: Nixon, *]";
startcmd "[!* K. C. 1999. The Parsimony Ratchet *]";
startcmd "[!* a new method for rapid parsimony *]";
startcmd "[!* analysis. Cladistics 15: 407-414. *]";
startcmd "[!*****]";
startcmd "[exe rbcl.nex]";
startcmd "log file=paupratchet.log";
startcmd "set increase=auto;
startcmd "hs status=no nrep=1 swap=tbr start=stepwise addseq=random
      nchuck=1 chuckscore=1";
```

[  
Line below creates master tree file in same folder as batch file -  
will have, upon completion, the same number of trees as iterations  
plus the starting tree  
]

```
startcmd "savetrees file=mydata.tre replace";
```

[  
Line below creates temp tree file in same folder as batch file  
- will only hold the best original weight tree found of the most  
recent unpermuted search  
]

```
startcmd "savetrees file=mydata.tmp replace";
```

[  
Line below sets multistate taxa to be treated as uncertainty  
rather than polymorphism - allows tree lengths to be comparable  
with those found by NONA  
]

```
paupcmd "pset mstaxa=uncertain";
```

[  
The commands below implement the ratchet proper. After searches  
using the normal weights, the single tree held in memory is saved  
to the mydata.tmp file, replacing the existing file. Then  
all the trees from k previous iterations (in the mydata.tre file)  
are added (mode=7 means add rather than replace) to the tree in  
memory, and then these k+1 trees are saved to mydata.tre, replacing  
the existing file. Finally, the best tree from the most recent  
iteration (which was saved in mydata.tmp) is brought back into  
memory for use as the starting tree for the next iteration.  
]

```
paupcmd "hsearch status=no start=1 swap=tbr multrees=no";
normcmd "savetrees file=mydata.tmp replace";
normcmd "gettrees file=mydata.tre mode=7";
```

```
normcmd "savetrees file=mydata.tre replace";
normcmd "gettrees file=mydata.tmp mode=3 warntree=no";
normcmd "time";
```

[  
 Lines below pull all the trees found (1 per iteration plus the starting tree) from the master tree file, mydata.tre, into memory and tell PAUP\* to display their scores - note that the last iteration is tree #1 and the starting tree is the last tree listed  
 ]

```
stopcmd "gettrees file=mydata.tre mode=3";
stopcmd "pscores all";
stopcmd "time";
stopcmd "log stop";
stopcmd "[!*****]";
stopcmd "[!* -- THIS SEARCH IS COMPLETE -- *]";
stopcmd "[!* A LOG FILE HAS BEEN WRITTEN *]";
stopcmd "[!* AND ALL TREES HAVE BEEN SAVED. *]";
stopcmd "[!* IT IS OK TO QUIT PAUP* *]";
stopcmd "[!*****]";
stopcmd "[quit]";
```

[  
 The line below tells PAUPRat to take the information above and go ahead and generate the PAUP\* batch file, naming it ratchet.nex. The replace keyword tells PAUPRat to replace the file ratchet.nex if it already exists. If you leave out the replace keyword, and the file exists, PAUPRat will terminate with an error message.

Don't forget this line, without it PAUPRat will not create a batch file!  
 ]

```
write file=ratchet.nex replace;

end;
```

-----

### Notes

**Output of results:** PAUPRat will have PAUP\* list all the shortest tree lengths of each unperturbed iteration, but in reverse order (such that tree #1 is the last iteration and tree #200 is the first of a 200 iteration run, tree #201 is the starting tree) like this:

Tree #	1	2	3	4	5	6	7	8	9	10	11	12	13
Length	16531	16531	16531	16531	16531	16531	16531	16531	16532	16531	16531	16531	16531
Tree #	14	15	16	17	18	19	20	21	22	23	24	25	26
Length	16531	16531	16532	16532	16532	16532	16532	16532	16532	16532	16532	16532	16532
Tree #	27	28	29	30	31	32	33	34	35	36	37	38	39
Length	16532	16532	16532	16532	16532	16532	16532	16534	16534	16532	16532	16532	16532
Tree #	40	41	42	43	44	45	46	47	48	49	50	51	52
Length	16532	16532	16532	16532	16532	16532	16532	16531	16531	16531	16531	16531	16531
Tree #	53	54	55	56	57	58	59	60	61	62	63	64	65
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	66	67	68	69	70	71	72	73	74	75	76	77	78

Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	79	80	81	82	83	84	85	86	87	88	89	90	91
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	92	93	94	95	96	97	98	99	100	101	102	103	104
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	105	106	107	108	109	110	111	112	113	114	115	116	117
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	118	119	120	121	122	123	124	125	126	127	128	129	130
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	131	132	133	134	135	136	137	138	139	140	141	142	143
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	144	145	146	147	148	149	150	151	152	153	154	155	156
Length	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531	16531
Tree #	157	158	159	160	161	162	163	164	165	166	167	168	169
Length	16531	16531	16531	16531	16531	16531	16532	16532	16532	16533	16532	16532	16532
Tree #	170	171	172	173	174	175	176	177	178	179	180	181	182
Length	16532	16532	16532	16532	16532	16532	16532	16532	16532	16532	16532	16532	16532
Tree #	183	184	185	186	187	188	189	190	191	192	193	194	195
Length	16532	16532	16532	16532	16532	16532	16532	16532	16534	16535	16535	16535	16536
Tree #	196	197	198	199	200	201							
Length	16536	16536	16536	16536	16551	16554							

Once a Ratchet search has completed, all the shortest, unperturbed trees of each iteration are stored in the file named *mydata.tre*. The trees in this file can be brought into PAUP\* (using the TREES menu and the GET TREES FROM FILE option) and filtered to keep only those of the shortest length which can then be made into a consensus tree.

Nixon (1999b) and Goloboff (1999) argue that because the Ratchet does such a good job of covering tree space that the consensus tree of a sufficient number of Ratchet searches will, in theory, be identical to the consensus of all shortest trees for any given dataset. Thus a recommended strategy would be to first determine what percent of characters to perturb using some preliminary Ratchet searches, then run 10 or 20 independent Ratchet searches and create a consensus tree from the combined results of all these searches. Once the consensus tree has stabilized, *i.e.* new searches don't change the topology, it should be accurate (Goloboff 1999).

**Performing multiple Ratchet searches automatically:** We have found, as did Nixon (1999b), that performing several, shorter Parsimony Ratchet searches is preferable to performing a single, longer search. To this end one can create multiple folders each with a separate batch file and into which PAUP\* will create output files. This allows one to start a search with one command that proceeds uninterrupted through numerous (e.g. 10-20) independent Parsimony Ratchet searches of, say 200 iterations each. Most modern computers should be able to complete 15 unattended runs of the *rbcL* dataset over a weekend using this strategy. To accomplish this on a Macintosh you would first create your 10 to 20 folders and generate a different batch file for each using PAUPRat in each folder. Then write a single controlling batch file that directs PAUP\* to execute all the batch files you've created. The master batch file for 5 searches might look like this:

```
#nexus

begin paup;
execute 'Mac harddrive:Folderinroot:batchfolder1:ratchet.nex';
execute 'Mac harddrive:Folderinroot:batchfolder2:ratchet.nex';
execute 'Mac harddrive:Folderinroot:batchfolder3:ratchet.nex';
execute 'Mac harddrive:Folderinroot:batchfolder4:ratchet.nex';
execute 'Mac harddrive:Folderinroot:batchfolder5:ratchet.nex';
end;
```

Where, in this example, *Mac harddrive* is the name of your harddrive, *Folderinroot*, is the name of the folder in the root of your harddrive that contains the separate batch file folders, *batchfolder1... batchfolder2* etc. are the names of all the folders which contain the batch files, and *ratchet.nex* is the name of the batch file to be executed within each folder.

The same strategy works for the Windows version of PAUP\*:

```
#nexus

begin paup;
execute 'C:\FolderInRoot\BatchFolder1\ratchet.nex';
execute 'C:\FolderInRoot\BatchFolder2\ratchet.nex';
execute 'C:\FolderInRoot\BatchFolder3\ratchet.nex';
execute 'C:\FolderInRoot\BatchFolder4\ratchet.nex';
execute 'C:\FolderInRoot\BatchFolder5\ratchet.nex';
end;
```

A similar strategy could be implemented using a shell script in Unix. In this case, there is no need for multiple folders because: 1) the Unix versions of PAUP\* will automatically execute a file whose name is specified on the command line when PAUP\* is invoked; and 2) Unix commands can be inserted in the script between invocations of PAUP\* to rename the log file from the previous run. Below is an example shell script. Two important modifications are needed to the *setup.nex* file to make this shell script work as expected. First, insert **startcmd="exe rbcl.nex"** as the first **startcmd** in your *setup.nex* file, replacing *rbcl.nex* of course with the name of your own data file. Second, insert **stopcmd="quit"** as the last **stopcmd** in your *setup.nex* file. These two modifications are present in the automatically-generated *setup.nex* file, but are commented out. Finally, it is assumed that the PAUP\* executable is in a directory listed in the current PATH environmental variable, and the shell script should be run from the directory containing the *ratchet.nex* file.

```
#!/bin/sh

paup ratchet.nex
mv -f paupratchet.log paupratchet1.log
paup ratchet.nex
mv -f paupratchet.log paupratchet2.log
paup ratchet.nex
mv -f paupratchet.log paupratchet3.log
paup ratchet.nex
mv -f paupratchet.log paupratchet4.log
paup ratchet.nex
mv -f paupratchet.log paupratchet5.log
rm -f mydata.tre
rm -f mydata.tmp
```

**Unequal original weights:** One feature that Nixon's implementation of the Ratchet did not cover was how to handle unequal starting weights. PAUPRat allows users to implement the Ratchet with data sets that have unequal starting weights (see commands in example setup file above). However, our testing of this feature indicates that the Ratchet searches do not behave as they do with equal starting weights. As can be seen in Fig. 3 below, the searches do not plateau on the shortest tree lengths. This result is similar to what occurs with equal starting weights if one perturbs too many characters for each iteration. Nixon (1999b) states that he has found good results with between 5 and 25 % of the characters perturbed for each iteration. If too many characters are perturbed, say 50-70%, the results are much like in Fig. 3. So, if users wish to explore the behavior of the Ratchet with unequal starting weights, we would be very interested in hearing your results.

Fig. 3. Nicrophorinae matrix, 77 taxa, ratchet, 20% perturbed, unequal original weights, multiplicative perturbation



**Support and future plans:** We encourage ideas to improve on this program. However, we expect it will sooner or later be incorporated into PAUP\* itself. If significant bugs are found and reported, we will do our best to squash them.

**Citation:** If you find this program useful and wish to cite it, please use the following citation:

Sikes, D. S. and P. O. Lewis. 2001. beta software, version 1. PAUPRat: PAUP\* implementation of the parsimony ratchet. Distributed by the authors. Department of Ecology and Evolutionary Biology, University of Connecticut, Storrs, USA.

### Acknowledgments

We thank David Wagner for running such a cutting-edge BioSystematics course (University of Connecticut EEB 458), and all his students who were the first beta-testers of PAUPRat. In addition we thank the few other beta testers who used PAUPRat in their research: Charles Davis, Brian Farrell, Elizabeth Jockusch, Piotr Naskrecki, Brian O'Meara, and Yin-Long Qiu. We also thank Kevin Nixon for his discovery of the Parsimony Ratchet.

### Literature Cited

- Chase, M. W., D. E. Soltis, R. G. Olmstead, D. Morgan, D. H. Les, B. D. Mishler, M. R. Duvall, R. A. Price, H. G. Hillis, Y.-L. Qui, K. A. Kron, J. H. Rettig, E. Conti, J. D. Palmer, J. R. Manhart, K. J. Sytsma, H. J. Michaels, W. J. Kress, K. G. Karol, W. D. Clark, M. Hedrén, B. S. Gaut, R. K. Jansen, K.-J. Kim, C. F. Wimpee, J. F. Smith, G. R. Furnier, S. H. Strauss, Q.-Y. Xiang, G. M. Plunkett, P. S. Soltis, S. M. Swensen, S. E. Williams, P. A. Gadek, C. J. Quinn, L. E. Eguiarte, E. Golenberg, G. H. Learn, Jr., S. W. Graham, S. C. Barrett, S. Dayanandan, & V. A. Albert. **1993**. Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene *rbcL*. *Ann. Mo. Bot. Gard.* 80: 528-580.
- Goloboff, P. 1998. Nona. Computer program and software. Published by the author. Tucuman, Argentina.
- Goloboff, P. 1999. Analyzing large data sets in reasonable times: Solutions for composite optima. *Cladistics* 15: 415-428.
- Horovitz, I. 1999. A report on "One Day Symposium on Numerical Cladistics." *Cladistics* 15: 177-182.
- Nixon, K. C. 1998. Dada ver 1.9. Software and manual. Published by the author, Trumansburg, NY.
- Nixon, K. C. 1999a. Winclada (beta) ver. 0.9 Published by the author, Ithaca, NY [Available at <http://www.cladistics.com>]
- Nixon, K. C. 1999b. The Parsimony Ratchet, a new method for rapid parsimony analysis. *Cladistics* 15: 407-414.
- Rice, K. A., M. J. Donoghue, and R. G. Olmstead. 1997. Analyzing large data sets: *rbcL* 500 revisited. *Syst. Biol.* 46: 554-563.

Swofford, D. L. 1998. PAUP\* Phylogenetic analysis using parsimony (\*and other methods), 4.0b4a. Sinauer Associates, Sunderland, Massachusetts.

Goloboff, P., S. Farris, K. Nixon. 2001. TNT: Tree Analysis Using New Technology <http://www.cladistics.com/webtnt.html>