

**Complex Performance Analysis through Statistical Experimental Design:
An Evaluation of Parameters Associated with Speed in Parallel Phylogenomics**

David G. Whiting¹, Quinn Snell³, Rebecca R. Nichols¹, Megan L. Porter², Kevin Tew³, Keith A. Crandall^{2,4}, Michael F. Whiting², Mark Clement³

¹Department of Statistics, Brigham Young University, Provo UT 84602, USA

²Department of Integrative Biology, Brigham Young University, Provo UT 84602, USA

³Department of Computer Science, Brigham Young University, Provo UT 84602, USA

⁴Department of Microbiology and Molecular Biology, Brigham Young University, Provo UT 84602, USA

Abstract

For complex designs using multiple factors, traditional methods of performance analysis are inadequate for assessing the improvement of program speed by increasing the number of processors. In this paper, we suggest an approach to performance analysis based on the statistical design of experiments paradigm. This approach allows for the estimation of multiple factors simultaneously as well as provides the machinery to statistically evaluate significance. Using as an illustration the analysis of an experiment to estimate the effects of multiple parameters on program speed in a phylogenetic estimation study, we show that relying on current methods of performance analysis can be misleading and inefficient.

INTRODUCTION

Performance analysis of parallel algorithms is traditionally completed by computing either relative or absolute speedup, where the numerator for relative speedup is the execution time of the parallel algorithm and the denominator for absolute speedup is the execution time of the best known serial algorithm. Relative speedup is very useful in scalability studies, however, it yields no information regarding actual performance. Absolute speedup, on the other hand, is directly compared against the best known serial algorithm. It reports how much faster the answer is expected to be found.

However, currently established methods of performance analysis are strained, if not completely inadequate, when confronted with complex real-world problems such as those encountered in computational biology. For example, although the use of absolute speedup is a good metric for performance analysis, it is based on the fact that the serial algorithm and the parallel algorithm complete the same computation. In phylogenetic inference, for instance, the search space is exponentially large; thus, any search performed on a reasonably sized dataset does not exhaust the search space. Algorithms terminate not because they have completely searched the space, but rather because the algorithm choices are complete or patience has worn out. Indeed, most often serial and parallel algorithms simply search different sections of the search space. Since the computations being performed are not identical, direct comparisons of performance may in fact be meaningless.

Additionally, when there are multiple parameters present in a study which may influence computational speed, it is often not clear which of these should be measured against the number of processors to assess performance. Thus, while some researchers have reported linear speedup in the size of the search space being sampled using parallel algorithms, the amount of time to find the “correct” (or, more accurately, the “best”) answer did not correspondingly decrease with processors in a linear fashion. Indeed, in one case the best answer found by a serial algorithm was not found at all by the parallel algorithm, although the parallel algorithm ran “faster.”

Furthermore, the performance of parallel algorithms with respect to serial codes and the impact of various parameters is especially difficult to determine when multiple parameters are involved. Traditional performance analysis forces us to investigate these parameters on a one-by-one basis. Not only is this inefficient, but the results obtained will be misleading if there are any interactions between the study parameters.

A more accurate and efficient way to do performance analysis in such complex, real-world problems is by implementing statistical experimental designs. In this paper, we demonstrate how a complex computational phylogenomics problem, one involving the creation of a best phylogenetic tree, can be evaluated for performance using a statistical experimental design. The remainder of the paper proceeds as follows. First we introduce the phylogenetic estimation problem. Next we discuss a parallel implementation of phylogenetic software. Section 3 is a brief introduction to statistical experimental design which is followed by its application for the analysis of the parallel parsimony ratchet.

THE PHYLOGENETIC ESTIMATION PROBLEM

Phylogenies are diagrams of branching patterns representing the estimated evolutionary histories among organisms or their parts (Crandall 2001). The phylogeny is indeed the foundation for all comparative biological analyses (Harvey et al. 1996; Harvey and Nee 1994),

and is widely used in studies in systematics, molecular biology, physiology, developmental biology, ecology, virology, and genomics (Pagel 1999). However, our ability to generate nucleotide sequence data for the phylogenetic estimation has significantly outpaced our ability to actually estimate these phylogenies. This is due primarily to the fact that the number of possible unrooted bifurcating trees (B) increases dramatically according to the number of sequences (S) in the analysis

$$B(S) = \prod_{i=3}^S (2i - 5) \quad (1)$$

(Felsenstein 1978). For example, for a modest sized molecular phylogenetic data set of 100 sequences, there are 2×10^{182} possible unrooted bifurcating tree topologies. Thus, the phylogenetic estimation problem is a classic NP-complete problem (Day 1987).

There are two steps involved in the reconstruction of any large phylogeny: 1) defining an optimality criterion by which trees can be compared in determining the optimal or set of optimal trees, and 2) establishing a strategy for searching the set of possible trees. The two most common optimality criteria employed for the reconstruction of phylogenies are maximum parsimony (MP) and maximum likelihood (ML). In MP searches, the best tree is defined as the one that minimizes the total tree length in terms of character state transformations. In contrast, ML searches choose the phylogeny that maximizes the likelihood of observing the data given a specified model of evolution (Felsenstein 1981). Because this criterion incorporates explicit assumptions about the evolutionary process and attempts to maximize the probability of these processes across all characters, the search process is computationally more intensive than MP searches, where only variable characters are considered in evaluating the tree. This disparity in complexity makes direct comparisons of the parameters affecting computational speed between criteria unrealistic.

Because the number of possible phylogenetic tree topologies is so large, researchers have moved to heuristic search strategies and innovative implementations of such strategies. Indeed, a variety of novel methodological and computational approaches have been attempted to overcome the computational burden of the phylogeny estimation (efficiency) as well as increase the accuracy of estimation, often with conflicting results relative to these two criteria (see (Hillis 1995) for definitions of efficiency, accuracy, and related terms in a phylogenetic context). These approaches can be classified into three basic groups: 1) enhancement in the speed of a search algorithm through changes to an existing algorithm (e.g., Felsenstein 1981; Hendy 1982); 2) development of new swapping approaches within existing algorithms for better evaluation of the tree space (e.g., Page 1993); and 3) development of novel search strategies (Edwards 1996; Felsenstein 1981; Saitou and Nei 1987).

Novelty in search strategies itself has come in a variety of forms. Some researchers have developed alternative optimality criteria (e.g., minimum evolution, Bayesian inference) (Huelsenbeck et al. 2001; Rzhetsky and Nei 1992), while others have concentrated on alternative algorithms for optimizing existing criteria (e.g., genetic algorithms) (Lemmon and Milinkovitch 2002; Lewis 1998).

While most heuristic search strategies are hill-climbing methodologies, a relatively new strategy, the ratchet, uses a statistical approach for sampling the tree “landscape” to find the most optimal trees for a dataset (Nixon 1999, #3). The strategy consists of iterations of: 1) obtaining a starting tree, 2) perturbing the dataset via random reweighting of characters, 3) searching for an optimal tree using the perturbed dataset, and 4) returning to the original dataset with the

perturbed tree and searching using branch-swapping methods. By randomly perturbing the dataset, a larger proportion of the set of possible trees can be explored.

Another option for increasing efficiency and accuracy for a variety of optimality criteria is through the parallelization of existing computational algorithms for phylogeny estimation. The advantages of parallelization have recently been explored using genetic algorithms with the ML optimality criterion (Brauer et al. 2002). They demonstrated a nearly linear improvement in computational speed for large phylogenetic problems using both simulated and real data sets. However, the best-known solution was not found using the genetic algorithm approach (Brauer 2002, #2213). Similarly, a parallel version of fastDNAmI presented at Supercomputing 2001 (Stewart 2001) also demonstrated near linear improvement in speed. However, the results were not compared against serial algorithms such as PAUP* (Phylogenetic Analysis Using Parsimony and other methods) (Swofford 2000), which, we will show, is able to find shorter trees in less time on a single machine. Since phylogenetic inference algorithms need no communication among workers, one would expect linear improvement in the number of trees evaluated. However, this says nothing about the quality of the trees considered.

PARALLEL IMPLEMENTATION

The Distributed Object Group Metacomputing Architecture (DOGMA) is a parallel and distributed programming architecture (Judd 1998 #1; Snell 2002). DOGMA is a resource management system that supports traditional parallel programming and also allows the option to create a parallel application by “wrapping” existing, commonly used sequential programs (e.g., PAUP* and POY for phylogenetic inference) and using them as workers in concert. A benefit of this approach is that any improvements made to the sequential code will also be seen in the parallel implementation.

DOGMA manages processors that are currently available for a computational task and becomes a broker that matches application requirements with available resources. Processors involved in parallel processing can be supercomputers, dedicated nodes, or idle-time nodes such as university lab facilities where tasks may be interrupted. For nodes where other tasks or users have higher priority, processors communicate with DOGMA via a screen saver. Job submission is accomplished via an internet web page. A test version of DOGMA allowing submission of datasets for analysis, as well as a downloadable software package, is available from the BYU-CPRG homepage (<http://genome.byu.edu>).

The DOGMA implementation of the parallel phylogenetic ratchet search is master-worker based. A master process is launched in the DOGMA system that creates a set of tasks and then launches worker processes on available machines. Each “worker” is simply wrapper code that is used to interact with the newest release version of PAUP*. The wrapper application is directly connected to the standard input and output of the sequential application. When a parallel tree search is performed, for instance, each worker gives the search commands to the unmodified sequential application. When the task is complete, the wrapper code parses the output to determine the length of the shortest trees found and then sends this length along with the actual trees back to the master task.

The ratchet methodology allows for even more communication between the parallel processes. When a worker requests a task involving a ratchet iteration, the master process selects one of the best available trees and sends it to worker machines for the next iteration. Because ML searches are more computationally difficult, ratchet was first described as an addition for MP-

based methods. Results of previous MP studies have shown the ability of this strategy to recover better trees in significantly reduced time frames under normal, serially implemented processes (Snell 2000). Now using the DOGMA system, ratcheted ML searches have also been implemented. As far as we are aware, this is the first attempt at a likelihood ratchet.

The implementation of the ML ratchet involved several decisions that were not needed in the parallel parsimony ratchet. In particular, the ratchet relies on the fact that branch swapping will terminate fairly quickly when it is only allowed to hold the single best tree at all times, rather than hold and examine multiple trees. In a maximum likelihood search, the tree cost evaluation takes much longer. If a tree branch can be placed anywhere during branch swapping, the resulting tree is more likely to be slightly better than the current tree. Thus, ML searches rarely terminate. Most often, patience determines the termination of the search. As the ratchet depends on termination of the branch swapping algorithm for a given tree, our algorithm forces ML branch swapping to terminate after a specified amount of time. The algorithm proceeds as follows:

- 1) *Obtain a starting tree.* This is obtained by first generating a Wagner tree, followed by some minimal level of branch swapping; various alternative ways of obtaining starting trees are available, such as randomly generating trees or using trees from previous analyses. Building this kind of starting tree prevents the algorithm from getting stuck during the creation of the initial tree.
- 2) *Randomly perturb the data set.* Originally, the perturbation used was to set all characters weights to 1 (or some original weighting scheme) and randomly select a subset of characters that would be upweighted (usually by 1). However, various modifications of the perturbation method have now been tested, and it turns out that removing (weighting to 0) is probably as effective as adding weight to selected characters.
- 3) Holding a single or a few trees, perform a standard tree search on the perturbed data. Terminate this search after a specified amount of time. This step “warps” the search space so that the algorithm may leave local minima.
- 4) Reweight characters to original weights.
- 5) Using the tree(s) found in step 3 as a starting point, perform SPR or TBR tree search on the UNPERTURBED data, still holding a single (or few) trees. Again terminate this search after a specified amount of time.
- 6) Return to 2) and repeat, using tree(s) from step 5 as a starting point.

Steps 2 through 5 constitute a single iteration; typically, from 100-200 iterations will be performed on a large data set (e.g., more than 200 taxa), while many fewer may be performed on smaller data sets. We have found that the most effective part of the search is the randomness involved with the reweighting of the data set. This reweighting effectively steps the search away from the local optima that ML often gets bogged down in. As an alternative to timed termination of the ML branch swap searches that take place using the perturbed data, the parallel likelihood ratchet allows the use of a parsimony branch swap search. The parsimony branch swap is much quicker and it clearly steps the search away from any local optima.

STATISTICAL EXPERIMENTAL DESIGN

Performance analysis, as traditionally implemented, is a one-at-a-time approach to experimentation in that the levels of one factor (i.e., the number of processors) are varied while holding all other factors that may affect the speed of computation constant. The presence of

multiple factors in such real-world problems further increases the number and complexity of the comparisons which a researcher might wish to make. This one-at-a-time approach is, at best, inefficient; at worst, it can be highly misleading, particularly if interactions exist between factors. Additionally, the potential for inadequately modeling the effect of the number of processors on computational speed increases when experimental error and variability (which, we argue, is always present) add further noise (Lawson, 2000). Moreover, performance analysis as it is currently implemented lacks associated statistical tests to distinguish important factors from inherent variation.

Statistical experimental design procedures allow a researcher to simultaneously assess the effects of all factors involved in a study and any possible interactions that may exist between these factors. It also provides a measure of the experimental error and overall variability in the study, as well as providing statistical tests to evaluate the importance of the experimental factors. We give a short overview of statistical experimental design procedures in this paper. More thorough coverage can be found in any introductory statistical experimental design book (see, e.g., Petersen (1985) or Lawson (2001)).

The most basic experimental design is a factorial design, which examines the effects of two or more factors on a response simultaneously. Full factorial designs observe the response variable at all possible combinations of the factor levels. It can be used to estimate and evaluate the main effects (how the factor alone, averaged across all levels of the other factors, affects the response) and all possible interaction effects (when the effect of one factor depends on the levels of another factor), including the highest-order interaction (the interaction between all of the factors). However, the number of data points required by a full factorial design increases exponentially as the number of factors and/or factor levels are increased. Thus, full factorial designs are used only in the smallest of problems.

Fractional factorial designs require far fewer runs than the full factorial design while still allowing specific main effects and interactions to be estimated. Reducing the number of experimental runs limits which factor effects are estimable by confounding certain “unimportant” effects (generally higher-order interactions) with those we want to estimate (typically main effects or lower-order interactions). Confounded effects cannot be separately estimated. These designs consequently choose specific points from the full factorial design to control the confounding structure so that important effects remain estimable with a smaller and more reasonable number of runs.

Fractional factorials are especially useful if the number of runs in the full factorial design exceed available resources, such as time and money. Often, information is required only on main effects and lower-order interactions, because at some point higher-order interactions typically become negligible. As a rule of thumb, main effects tend to be more important than two-way interactions, two-way interactions tend to be more important than three-way interactions, and so on. Additionally, full as well as fractional factorial designs have a “hidden replication” property. This means that any runs used to estimate an effect (either main or interaction) that is subsequently found to be insignificant are not wasted. Rather, since they also contain information about other factors and interactions due to the design structure, they act as partial replicates. Thus, the most efficient use of information is inherent in such designs.

If there are many potential factors in an experiment, screening designs may be used to identify important factors. Screening designs are actually specific fractional factorial designs intended to estimate factor main effects and possibly one or two two-way interactions. Factors that are determined to be important in a screening design are typically included in a follow-up

experiment designed to more fully estimate interaction effects. The number of possible runs in the follow-up experiment is thus greatly reduced and only the few significant factors found by the screening design need further investigation.

D-optimal designs are variations of fractional factorial designs that choose factor level combinations for experimental runs according to the D-optimality criterion (Dykstra 1971). A common strategy for creating complex experimental designs is to augment a fractional factorial or screening design with points chosen to according to D-optimality. D-optimal designs augment the existing data with new runs by maximizing the determinant of the $X^T X$ matrix, where the X matrix includes the values of the design (or predictor) variables. The determinant of the $X^T X$ matrix is maximized by selecting the run in the design space where the variance of the predicted response is the greatest. Collecting data at the points with the greatest variance increases the confidence in these areas and reduces the overall variance of the design the most. Thus, the resulting design is efficient and can then be used to uniquely estimate each of the main effects and chosen interactions using standard general linear model analysis of variance methods (Dykstra 1971).

Software exists for creating D-optimal designs and for augmenting existing fractional factorial and screening designs (see, e.g., SAS Proc Optex) while ensuring that user-specified main and interaction effects are estimable. Although other optimality criteria exist, we will not utilize them in this paper.

PERFORMANCE ANALYSIS FOR A PHYLOGENETIC ESTIMATION EXPERIMENT

Parallel Ratchet Performance

As an example of the complexity of the performance measurement, we use traditional methods to determine the performance of our new parallel ratchet algorithm. In all cases, the parallel ratchet times are averages over 5 runs.

Table 1. Parallel Ratchet Performance

#Taxa	SOURCE	PAUP* Time (seconds)	PAUP* Length	Parallel Ratchet Time (seconds)	Parallel Ratchet Length
200	Simulated	180	3015	7	3015
1500	Simulated	200 hrs	10294*	500	10290
100	HIV	59	581	2.7	581
200	HIV	200 hrs	766*	21	765
500	HIV	200 hrs	2040*	1274	2028
100	18S	180	2054	18	2054
200	18S	200 hrs	4539*	200	4537
500	Zilla	200 hrs	16222*	300	16218

While it is obvious that the parallel ratchet would evaluate a linear factor times more trees in the same amount of time, Table 1 also shows that under all scenarios the parallel ratchet algorithm found the shortest tree in considerably less time.

There is a greater differential in performance (in terms of speed and the ability to find shorter trees) with large data sets. Notice that for the 500 taxa HIV data set, the parallel ratchet algorithm found a tree 12 steps shorter than the tree found by PAUP*. For this example, PAUP* took an unreasonable amount of time to find trees that were less optimal than those found by the parallel ratchet in a fraction of an hour. As is obvious in Table 1, this is not an uncommon occurrence.

The parallel ratchet made it possible to find trees that were unable to be found by the sequential algorithms. Based on our initial tests, the parallel ratchet always finds a tree at least as optimal as those reported by NONA and PAUP*. Most often it finds a tree that is more optimal and finds that more optimal tree in less time.

In like manner, we compared the parallel maximum likelihood ratchet algorithm with sequential PAUP* and fastDNAm1. Table 2 summarizes these results. In all cases, the algorithms were allowed to place branches anywhere on the tree, not just within a specified distance. The parallel algorithms each ran on 16 processors.

Table 2. Parallel Maximum Likelihood Ratchet

Taxa	SOURCE	PAUP* Time	PAUP* Length	fastDNA ml Time	FastDNAm l Length	Parallel ML Ratchet Time	Parallel ML Ratchet Length	Parallel ML Ratchet with MP Time	Parallel ML Ratchet with MP Length
50	fastDNAm1 test data	3:42:08	16800	3:23:13	16826	3:7:42	16792	2:16:41	16792
75x500	Deep-six	39:50:00	4326	***	***	0:60:37	4326	0:21:25	4326
75x800	Deep-six	3:28:00	6057	***	***	0:30:29	6057	0:3:5	6057

*** Unable to complete or yield intermediate results after 1 week on 16 processors

Clearly, the parallel likelihood ratchet outperforms other algorithms by finding better trees in less time. It is interesting to note the performance increase using the parallel likelihood ratchet with parsimony branch swap on the perturbed data. This alternative allowed us to find the same tree in even less time. We are currently examining this further to verify if this behavior exists on all data sets.

Although these comparisons are promising for the parallel likelihood ratchet, performance analysis is difficult because of the number of factors involved in evaluating speed. Most importantly, the individual influence of these factors on computational speed is impossible to determine without a properly designed experiment.

The Experimental Design

We now illustrate the design and analysis of a statistical experiment to evaluate performance as applied to a phylogenetic estimation problem. In our design, we included all factors which were potentially influential in computational speed. The factors in question are the number of processors, the number of characters (i.e., the length of each nucleotide sequence), the number of taxa (i.e., the number of nucleotide sequences), whether the ratchet search algorithm was used, and whether data were real or simulated.

Because of the disparity in complexity between the MP and ML criteria as previously mentioned, and due particularly to the divergent computational requirements of each of these criterion, separate experiments were designed for each. In this paper we present only the construction and analysis of the MP design, noting only that the ML design was created in a similar fashion.

The factors and their respective levels in the MP design are

- A. Processors: 3 levels (1, 8, 32)
- B. Characters: 3 levels (500, 1000, 2158)

- C. Taxa: 3 levels (75, 200, 332)
- D. Ratchet: 2 levels (Yes, No)
- E. Data Type: 2 levels (Real, Simulated)

To estimate the effects of all of these factors and all of their possible interactions using a full factorial design (with no degrees of freedom for estimating error) would require at least $3^3 \times 2^2 = 108$ runs. However, by using the principles of statistical experimental design previously explained, we can greatly reduce the number of runs while ensuring that the effects of interest are all estimable (Kuehl 2000; Peterson 1985).

Our design consisted of an adaptation of the L_{18} orthogonal array, which we augmented with additional runs using the D-optimality criterion (Dykstra 1971). The L_{18} orthogonal array is a commonly used 2×3^7 screening design for up to eight factors and 18 runs (see Figure 1).

Figure 1. The L_{18} Screening Design

run	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	2	2	2	2	2	2
3	1	1	3	3	3	3	3	3
4	1	2	1	1	2	2	3	3
5	1	2	2	2	3	3	1	1
6	1	2	3	3	1	1	2	2
7	1	3	1	2	1	3	2	3
8	1	3	2	3	2	1	3	1
9	1	3	3	1	3	2	1	2
10	2	1	1	3	3	2	2	1
11	2	1	2	1	1	3	3	2
12	2	1	3	2	2	1	1	3
13	2	2	1	2	3	1	3	2
14	2	2	2	3	1	2	1	3
15	2	2	3	1	2	3	2	1
16	2	3	1	3	2	3	1	2
17	2	3	2	1	3	1	2	3
18	2	3	3	2	1	2	3	1

Of the eight possible factors shown in the columns of the L_{18} design, the first factor has two levels and the remaining seven factors each have three levels. All eight main effects as well as the two-way interaction between the factors in column 1 and 2 are uniquely estimable, but the other two-way and higher-order interactions are confounded and not uniquely estimable (Lawson, 2000).

The L_{18} screening design is quite flexible and can be modified using the pseudo factor method and dummy levels as described in Lawson (2000). For the phylogenetics example in this paper, we modified this screening design for two 2-level factors and three 3-level factors. We augmented our modified L_{18} design to ensure that all possible two-way interactions between the four factors in our study were estimable. Our final D-optimal design consisted of a 36-run experiment with 10 degrees of freedom for error.

The Response Variable: Measuring Performance

The goal of all phylogenetic search algorithms is to find the most optimal tree. Since the programs to be considered never finish searching the search space, and since they often search different parts of that space, a metric which accounts for computational speed without including a measure of tree quality is, in the phylogenetic performance setting, of no value. We suggest a consistent methodology that should be followed when making such algorithm comparisons.

- 1) Always compare results against the best available serial algorithm. Parallel algorithms legitimately search through more trees than do sequential algorithms, but they don't

necessarily search more efficiently. We must always check to see that the parallel algorithms find better trees as well as covering more of the search space.

- 2) Run the sequential algorithm either to completion or up to a specified time limit, then record the time and the resulting tree length.
- 3) Run the parallel algorithm until a tree of equal or better length is found.
- 4) Because many parallel and sequential search algorithms use random jumps to locate islands of local optima, sometimes a high quality tree will be found quickly, while in other cases a tree as good as or better than that found by the sequential algorithm may never be attained. Thus, to account for potentially anomalous results, a measure of variation over a number of program executions should be reported. This can either be accomplished by averaging a number of runs and reporting the average, or through the use of a valid statistical experimental design which, by design, incorporates a measure of variability directly.

The Data

We constructed datasets of the desired taxa and character number from one master dataset named Deepsix consisting of 18s sequences from 21 orders of insects (Whiting, unpublished data). To construct the smaller datasets required for the experimental design, a program called “Resize” (available from <http://genome.byu.edu/resize.html>) was used which takes an entered dataset and pares it down to a specified size and divergence through a given number of iterations. In this manner, the level of divergence, a possible confounding factor, was kept constant.

Each trial was run on a dedicated cluster of computers using a standard set of parameters. For trials where ratchet was employed, we used a standard set of ratchet parameters (Table 3). For the likelihood trials, we employed a standardized model (JC69).

Table 3. Standardized ratchet-only parameters

initial tree number of repetitions	1
initial tree maximum trees per repetition	10
initial tree epsilon	3
initial tree maximum time for repetition (seconds)	120-140
minimum percentage of characters to reweight	5
maximum percentage of characters to reweight	25
minimum reweight value	2
maximum reweight value	10

The Analysis

Because our experimental data is unbalanced by design, general linear model analysis of variance (GLM ANOVA) is necessary to estimate and determine the statistical significance of the main effects and two-way interactions of the factors on computational time (Lawson 2000). Because the response variable (time to best tree) is heavily right skewed, the natural log of the time to best tree was used. This better meets the normality assumption required by analysis of variance methods.

The SAS GLM procedure was used to estimate the main and two-way interaction effects and to determine their significance statistically. Figure 2 shows the SAS GLM output when fitting the full model with all the main effects and two-way interactions. When the full model is fit, only the main effect of taxa appears statistically significant, with a p-value of 0.0004.

Figure 2. SAS GLM Output for Parsimony Full Model Analysis

The SAS System
The GLM Procedure

Dependent Variable: logmin

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	25	849.1782784	33.9671311	3.06	0.0233
Error	12	133.1524754	11.0960396		
Corrected Total	37	982.3307538			

	R-Square	Coeff Var	Root MSE	logmin Mean
	0.864453	2990.035	3.331072	0.111406

Source	DF	Type III SS	Mean Square	F Value	Pr > F
processor	2	7.3985672	3.6992836	0.33	0.7229
taxa	2	349.8232428	174.9116214	15.76	0.0004
processor*taxa	4	31.0657702	7.7664425	0.70	0.6067
character	2	26.0621057	13.0310529	1.17	0.3421
processor*character	4	6.3268466	1.5817117	0.14	0.9629
taxa*character	4	41.1804525	10.2951131	0.93	0.4799
ratchet	1	12.9745097	12.9745097	1.17	0.3008
processor*ratchet	2	6.1822653	3.0911327	0.28	0.7616
taxa*ratchet	2	1.1295016	0.5647508	0.05	0.9506
character*ratchet	2	5.5442974	2.7721487	0.25	0.7829

This model therefore shows no evidence of significant two-way interactions between the factors studied. Furthermore, none of the factors other than the number of taxa appear to significantly affect the speed of computation.

We next remove the insignificant two-way interactions from the model (in this case, all of them) and analyze the remaining main effects.

Figure 3. SAS GLM Output for the Parsimony Main Effects Model

The SAS System
The GLM Procedure

Dependent Variable: logmin

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	745.3476785	106.4782398	13.48	<.0001
Error	30	236.9830753	7.8994358		
Corrected Total	37	982.3307538			

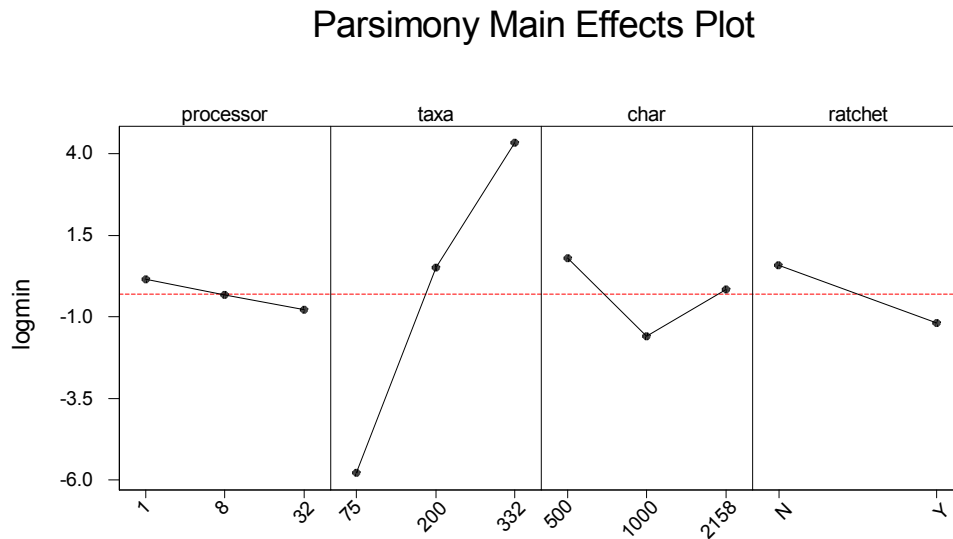
	R-Square	Coeff Var	Root MSE	logmin Mean
	0.758754	2522.843	2.810594	0.111406

Source	DF	Type III SS	Mean Square	F Value	Pr > F
processor	2	5.6791071	2.8395536	0.36	0.7010
taxa	2	626.8251394	313.4125697	39.68	<.0001
character	2	38.5676739	19.2838370	2.44	0.1042
ratchet	1	28.2986094	28.2986094	3.58	0.0681

The main effects model shows that the number of taxa is still significant (p-value < 0.0001) and the ratchet is significant at the $\alpha = 0.10$ level (and barely insignificant at the $\alpha = 0.05$ level).

Surprisingly, the processor effect is not significant, a fact we would not have picked up doing a conventional performance analysis. This insignificance is probably due to the fact that there is so much variability in the time to find the best tree using the MP criterion. (Because the ML criterion is computationally more intensive, the processor effect was found to be significant.) A wider range of processor values, i.e., with a maximum number of processors greater than 32, would possibly have resulted in a significant processor effect.

Figure 3. Parsimony Experiment Main Effects Plots.



The main effects plots show the direction of increase or decrease of the effects. Note, for example, that additional processors do decrease the time to best tree in log minutes, but this decrease is nonsignificant. Ratchet likewise shows a decrease when it is included. The taxa main effect, as discussed above, shows a significant increase in time. Although the character main effect has an apparent quadratic effect, since character is nonsignificant, this can be accounted for by random variation. Additional runs might reveal whether a real quadratic effect exists; however, this appears rather to be simply an anomaly of these particular data.

Overall, the conclusions one can draw from these preliminary data are that 1) the number of taxa is the most important factor in determining the computational speed of a phylogenetic analysis, 2) employing the ratchet reduces computation time nearly significantly, and 3) the number of processors used, after correcting for all other factors, did not have a significant impact in computation speedup, although it did reduce computation time slightly. Due to the speed of the computer nodes used in the parallel computation, and perhaps due to the inherent speed of the parsimony sequential code, speedup may not be seen unless either much larger datasets are analyzed which strain the computational abilities of the algorithm, or more processors than 32 are used in the analysis.

Because an effect is nonsignificant, it does not mean that a real underlying effect does not exist. For example, character and processor might both be insignificant because the levels of

these effects were not widely spaced enough. These results differ quite a bit from our preliminary data on the ML experimental approach. There characters, taxa, processors, and ratchet were found to be significant.

CONCLUSIONS

In this paper, we have presented a performance analysis methodology and shown its use through the example of the parallel parsimony ratchet. Through this analysis it was shown that larger datasets (more taxa, hence more branches and a larger potential search space) require significantly more time to find an optimal tree than smaller datasets, and that the ratchet algorithm is significant for performance. Preliminary results show that not only the ratchet but also the number of processors is also significant for performance in parallel maximum likelihood analysis.

The methodology employed in our study demonstrates the usefulness of the statistical experimental design approach. Performance analysis done in the traditional manner both lacks the ability to statistically test for significance and is unable to simultaneously account for multiple factors in a study. Computer scientists will need to adopt this approach as they increasingly become involved in complex, multi-parameter studies and if they wish to more realistically assess the overall performance of an algorithm based on number of processors.

REFERENCES

- Brauer, M. J., M. T. Holder, L. A. Dries, D. J. Zwickl, P. O. Lewis, and D. M. Hillis. 2002. Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference. *Molecular Biology and Evolution* 19:1717-1726.
- Crandall, K. A. 2001. Phylogeny. Pp. 1465-1466 *in* S. Brenner and J. H. Miller, eds. *Encyclopedia of Genetics*. Academic Press, London.
- Day, W. H. E. 1987. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology* 49:461-467.
- Dykstra, O. 1971. The augmentation of experimental data to maximize $|X'X|$. *Technometrics* 13:682-688.
- Edwards, A. W. F. 1996. The origin and early development of the method of minimum evolution for the reconstruction of phylogenetic trees. *Systematic Biology* 45:79-91.
- Felsenstein, J. 1978. The number of evolutionary trees. *Systematic Zoology* 27:27-33.
- Felsenstein, J. 1981. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution* 17:368-376.
- Harvey, P. H., A. J. Leigh Brown, J. Maynard Smith, and S. Nee. 1996. *New Uses for New Phylogenies*. Oxford University Press, Oxford, England.
- Harvey, P. H., and S. Nee. 1994. Phylogenetic epidemiology lives. *Trends in Ecology and Evolution* 9:361-363.
- Hillis, D. M. 1995. Approaches for assessing phylogenetic accuracy. *Systematic Biology* 44:3-16.

- Huelsenbeck, J. P., F. Ronquist, R. Nielsen, and J. P. Bollback. 2001. Bayesian inference of phylogeny and its impact on evolutionary biology. *Science* 294:2310-2314.
- Lemmon, A. R., and M. C. Milinkovitch. 2002. The metapopulation genetic algorithm: an efficient solution for the problem of large phylogeny estimation. *Proceedings of the National Academy of Sciences, U.S.A.* 99:10516-10521.
- Lewis, P. O. 1998. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution* 15:277-283.
- Pagel, M. 1999. Inferring the historical patterns of biological evolution. *Nature* 401:877-884.
- Rzhetsky, A., and M. Nei. 1992. A simple method for estimating and testing minimum-evolution trees. *Molecular Biology and Evolution* 9:945-967.
- Saitou, N., and M. Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4:406-425.
- Snell, Q., K. Tew, J. Ekstrom, and M. Clement. 2002. An Enterprise Based Grid Resource Management System. *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, 83-92.
- Snell, Q., M. Whiting, M. Clement, and D. McLaughlin. 2000. Parallel Phylogenetic Inference. *Proceedings of Supercomputing 2000*, Dallas, TX.
- Stewart, C., D. Hart, D. Berry, G. Olsen, E. Wernert, and W. Fischer. 2001. Parallel implementation and performance of fastDNAml - a program for maximum likelihood phylogenetic inference. *Proceedings of Supercomputing 2001*, Denver, CO.
- Swofford, D. L. 2000. *PAUP*: phylogenetic analysis using parsimony (*and other methods)*. Sinauer Associates, Sunderland, Mass.