TCS: Estimating Gene Genealogies

Mark Clement, Quinn Snell, Peter Walker
Department of Computer Science, Brigham Young University
David Posada, Keith Crandall
Department of Zoology, Brigham Young University

Abstract

Phylogentic analysis is becoming an increasingly important tool for customized drug treatments, epidemiological studies, and evolutionary analysis. The TCS method provides an important tool for dealing with genes at a population level. Existing software for TCS analysis takes an unreasonable amount of time for the analysis of significant numbers of Taxa. This paper presents the TCS algorithms and describes initial attempts at parallelization. Performance results are also presented for the algorithm on several data sets.

1 Introduction

Phylogenies are extremely useful tools, not only for establishing genealogical relationships among a group of organisms or their parts (e.g. genes), but also for a variety of research once the phylogenies are estimated. In a recent review, Pagel (1999) eloquently outlined a number of uses for phylogenetic information. These uses include the analysis of drug resistance and the study of evolutionary relationships between species. Phylogenies have also been used to predict future trends in infectious disease (Bush *et al*. 1999). Yet phylogenies are only as useful as they are accurate.

Estimating genealogical relationships among genes at the population level presents a number of difficulties when compared to traditional methods of phylogeny reconstruction. These traditional methods such as parsimony, neighbor-joining, and maximum likelihood make assumptions that are invalid at the population level.

For example, these methods assume ancestral haplotypes are no longer in the population, yet coalescent theory predicts that ancestral haplotypes will be the most frequent sequences sampled in a population level study (Watterson & Guess 1977; Donnelly & Tavaré 1986; Crandall & Templeton1993). Figure 1 shows a traditional parsimony or maximum likelihood tree. Note that all of the haplotypes occur at the leaves of the tree. With population studies, many of the individuals sampled will be internal nodes or ancestors of other individuals sampled as shown in Figure 2. Cycles can also occur along with recombination in trees derived from population studies.

Traditional methods assume that recombination will not occur. The failure to incorporate the possibility of recombination in phylogeny reconstruction can lead to grave errors in the resulting estimated phylogeny.

The combination of these effects can lead parsimony methods to infer a cumbersome amount of most parsimonious trees at the population level with no resolution among the set (e.g. over one billion trees for a set of human mitochondrial DNA (mtDNA), Excoffier & Smouse 1994). These effects can also lead neighbor-joining and traditional maximum-likelihood methods to be over confident in the resulting relationships (Bandelt *et al.* 1995). Therefore, an alternative approach is needed to provide accurate estimates of gene genealogies at the population level that take into account these population level phenomena not addressed by traditional methods.

Multiple groups have looked to network representations for population level genealogical information (Bandelt & Dress 1992; Templeton *et al.* 1992; Excoffier & Smouse 1994; Fitch 1997). Networks

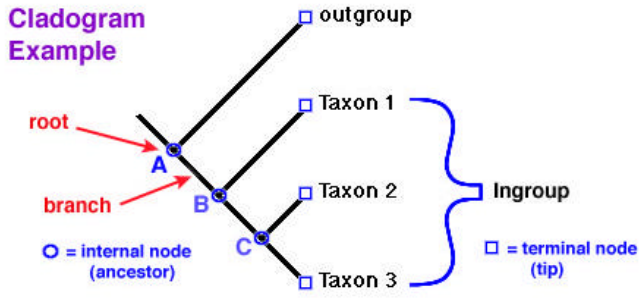

Figure 1: Traditional Parsimony or Maximum Likelihood Tree

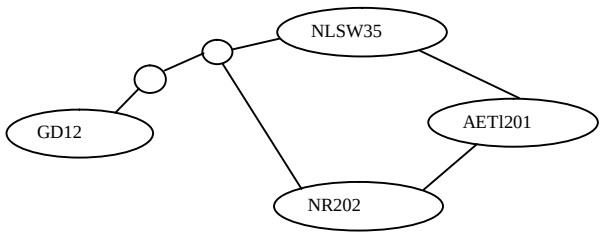

Figure 2: Taxa can exist as ancestors and cycles can occur in a population study tree.

allow one to naturally incorporate the often-times non-bifurcating genealogical information associated with population level divergences. The method of Templeton, Crandall and Sing.(1992) (TCS) has been used extensively with restriction site and nucleotide sequence data to infer population level genealogies when divergences are low (Georgiadis *et al.* 1994; Routman *et al*. 1994; Gerber & Templeton 1996; Hedin 1997; Schaal *et al.* 1998; Vilá *et al*. 1999, Gómez-Zurita *et al*. 2000).

TCS has been used with traditional methods to estimate relationships among organisms that span a wide range of divergence (Crandall & Fitzpatrick 1996; Benabib *et al.* 1997). The approach has also been used extensively with a nested analysis procedure to partition population structure from population history (Templeton *et al*. 1995; Templeton 1998) and explore the phylogeographic history of a diversity of organisms (e.g. Johnson & Jordon 2000; Turner *et al.* 2000).

The TCS software opens nucleotide sequence files in either nexus (Maddison *et al.* 1997) or phylip (Felsenstein 1991) sequential format. The program collapses identical sequences into haplotypes and calculates the frequencies of the haplotypes in the sample. These frequencies are used to estimate haplotype outgroup probabilities, which correlate with haplotype age (Donnelly & Tavaré 1986; Castelloe & Templeton 1994).

An absolute distance matrix is then calculated for all pairwise comparisons of haplotypes. The probability of parsimony [as defined in Templeton *et al.* (1992), equations 6, 7, and 8] is calculated for pairwise differences until the probability exceeds 0.95. The number of mutational differences associated with the probability just before this 95% cut-off is then the maximum number of mutational connections between pairs of sequences justified by the 'parsimony' criterion.

These justified connections are then made resulting in a 95% set of plausible solutions. The program outputs the sequences, the pairwise absolute distance matrix, probabilities of parsimony for mutational steps just beyond the 95% cut-off, a test listing of connections made and missing intermediates generated, and a graph output file containing the resulting network. This graph output file can be opened in the freeware VGJ 1.0.3 (McCreary 1998)(distributed under the terms of the GNU General Public License, Version 2), which is packaged with the TCS algorithm.

This paper describes the implementation of the TCS algorithm and a parallel version recently completed. Performance results are presented for several data sets along with algorithm analysis results.

2 Algorithm Description

This section describes the overall architecture of the TCS program. It focuses on sections of code that are most computationally intensive and attempts to highlight opportunities for parallelization.

After the sequence file is read in, the algorithm calculates the distance between each taxa and every other taxa. This calculation is performed by comparing the characters for each sequence and recording the raw number of changes between the sequences. Table 1 shows sequence data and Table 2 shows the distance matrix for the taxa shown in Figure 2.

Taxa Name	Sequence Data
NLSW35	ACGCA
AETl201	ACGCC
NR202	ACGAC
GD12	TTGAA

Table 1: Sample Sequence data for the tree in Figure 2.

Distance	NLSW35	AETl201	NR202	GD12
NLSW35	0	1	2	3
AETl201	1	0	1	4
NR202	2	1	0	3
GD12	3	4	3	0

Table 2 Distance Matrix for the tree in Figure 2.

Once the distance matrix has been computed, the TCS algorithm proceeds to connect the taxa into a cladogram using the following algorithm:

1) A cluster is created for each of the *N* taxa in the sequence file.
2) The distance matrix is examined to determine which two clusters have the minimum distance *M*. This distance is computed by taking each taxa in one cluster and finding the taxa in each of the other clusters that has the smallest number of changes. In the worst case, there are *N-1* clusters that each have 1 taxa for a complexity = $O(N^2)$ for this step.
3) All of the taxa in the two minimum distance clusters that have distance *M* are then joined. For this discussion we assume that taxa *S* in the source cluster and taxa *D* in the destination cluster are two of these taxa that have distance M. Connections that have a distance greater than 1, will be made by adding intermediates in the following way: (In the worst case *N*/2**N*/2 connections will be made)
   a. The minimum number of intermediates should be added to make the distance between the two taxa correct while preserving other distances in the matrix. Intermediates from another connection can be used in joining a pair of taxa as long as the connection does not form a connection that is shorter than the minimum distance between any two taxa in the source and destination clusters.

This step is implemented using the following algorithm

i. All of the possible connections between *S*, an intermediate in the source cluster, an intermediate in the destination cluster and *D* are evaluated. There are at most $(N/2)^2$ of these connections possible ($N/2$ intermediates in the source cluster and $N/2$ intermediates in the destination cluster).
ii. These connections are evaluated to determine which connection has the maximum metric. In the worst case, N^2 distances will be compared for each possible connection. The metric is computed in the following way:
    1. The distance between every pair of taxa in the source and destination cluster is examined to determine the global quality of a possible connection. A distance metric is created by comparing each of these possible distances with the real distance computed from the sequence file.
    2. If a possible connection creates a distance that is correct, the metric is incremented by 20 points.
    3. If the distance is shorter than the correct distance, but longer then the minimum distance for the taxa, then the metric is decremented by 10 points.
    4. If the distance is less than the minimum for the taxa, then the metric is set to negative infinity (to indicate that this connection is undesirable).
    5. If the distance is longer than the correct value, then the metric is decremented by 5 points.
iii. The connection with the best metric is made in the tree data structure.

b. Combine the two clusters into one, reducing the number of clusters by one.

4) If there is more than one cluster, go to step 2.

2.1 Example

Figure 1 provides an example data set for the TCS algorithm. Initially, there are N clusters, one for each taxa. The minimum distance is between NLSW35 and AET1201 (the distance between NR202 and AET1201 is also minimal, and will be dealt with next. Taxa NLSW35 and AET1201 are connected with this minim distance (1) and are joined into a cluster. Figure 4 shows the first connection made in the algorithm.

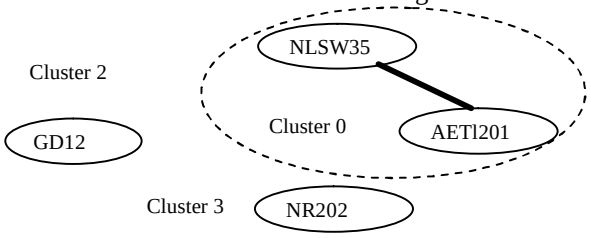

Figure 4: Configuration after the first two clusters have been collapsed.

At this point the minimum distance between clusters must be computed. The minimum distance between GD12 and any of the taxa in Cluster 0 is 3. Since the distance between NR202 and AET1201 is one, this is the minimum distance between any two clusters and a connection will be made in the next step between Cluster 0 and Cluster 3. During the next iteration of step 2 in the algorithm Taxa GD12 with be joined to the cluster consisting of NLSW35, AET1201 and NR202. The minimum distance is 3 between the two clusters and the algorithm makes all of the connections at that minimum distance. The first connection is made between GD12 and NLSW35 by adding two intermediates as shown in Figure 5.

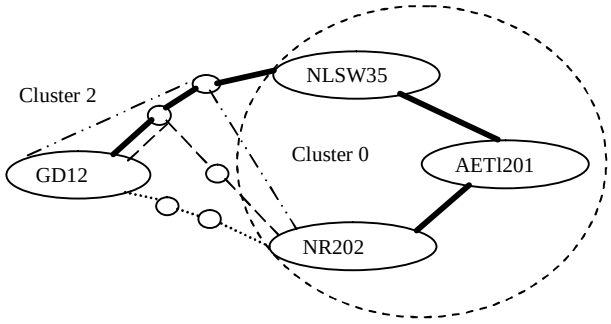

Figure 3: Configuration after the first connection has been made between Taxa GD12 and Cluster 0 (Taxa NLSW35). Possible connections are shown with different line patterns for the next connection to Taxa NR202.

The next minimum distance connection between Taxa GD12 and Cluster 0 is to Taxa NR202. Several possible connections exist. Two new intermediates could be added between GD12 and NR202. The distances for this tree would all still be correct. By reusing intermediates I1 and I2, the metric will be higher and this connection will be used. If the minimum distance for Taxa NLSW35 or NR202 was greater than 2, then reusing these intermediates would have caused the distance between NLSW35 and NR202 to be less than the minimum distance and this connection would not be used.

3 Complexity and Performance Analysis

In analyzing the complexity of the TCS algorithm, the loop starting with step 2 will occur *N* times. There can be as many as N^2 possible connections to perform metric evaluation for at each of these steps for a total of $O(N^3)$ computations. For each of these connections, a worst case of N^2 distance comparisons must be made to calculate the metric. The total complexity of the algorithm is $O(N^5)$.

Although the complexity of the TCS algorithm grows rapidly as the number of taxa increases, the problem is not nearly as difficult as total exploration of the tree space. Table 3 shows the number of computations for each problem. Execution time for TCS varies significantly depending on the data set. If the sequence data is organized so that a single additional taxa is added to a growing large single cluster, then the computation will proceed more quickly. If there is a step where two large clusters are joined, the computation will take much longer. Many data sets exist with more than 10,000 taxa and even the most efficient traditional algorithms do not examine a significant percentage of the total trees. Table 4 shows run times for several data sets with different number of taxa.

Table 3: Number of trees for the Parsimony computation, compared to the number of computations for TCS.

Data Set	Number of Taxa	Execution Time
LTRA Sequences	30	4 seconds
LTRA Sequences	40	17 seconds
LTRA Sequences	72	40 seconds
HIV	100	720 seconds
Simulated Sequences	200	23 seconds
HIV	200	27,120 seconds

Table 4: Execution time for TCS with different sequence data.

Number of Taxa (*N*)	**Number of Trees**	**Worst Case TCS Computations**
10	2×10^{6}	10^{5}
22	3×10^{23}	5×10^{6}
50	3×10^{74}	3×10^{8}
100	2×10^{182}	10^{10}
1,000	$2 \times 10^{2,860}$	10^{15}
10,000	$8 \times 10^{38,658}$	10^{20}
100,000	$1 \times 10^{486,663}$	10^{25}

3.1 Parallelization

For thousands of taxa, the run time of TCS is still excessive. A parallel implementation of the algorithm has been completed in order to increase the problem sizes that can be effectively dealt with. There are several opportunities for parallelization in the TCS algorithm.

One possibility for parallelization would be to have each processor independently combine clusters (iterations of step 2). There are data dependencies between iterations as each cluster is collapsed that makes this level of parallelization impossible.

Parallelizing the loop that begins at step 3 is also difficult since intermediates inserted by one connection can be used by other connections between clusters. If this problem could be solved, then there would be approximately $O(N^4)$ computations possible without communications and the algorithm would be much more efficient.

The initial parallel implementation chose to assign the evaluation of the metric for each connection (step 3.a.ii) to a different processor ($O(N^2)$ computations for each processor). This approach results in more inter-processor communication and results in load imbalance since the evaluation of some metrics will require much more time than others.

3.2 Parallel Implementation

The parallelization has been completed and the initial results appear promising. Since TCS is written in Java, multiple threads were spawned and these threads were mapped to different processors on a shared memory machine.

Before optimization was done, profiling information showed that the most time consuming operation was java Vector lookups. Java Vectors were used significantly in nested for loops that recalculated distances. Java Vectors are synchronized, causing an object lock to be requested for any method call. In a multithreaded environment, this could mean unacceptably long waiting times for object locks to do read-only operations. Further, java Vectors do not allow direct access, as do arrays, meaning that each element retrieved from a Vector incurs the overhead of a method call.

Due to these factors, the code was changed so that Vectors were copied on demand into Arrays local to each thread. The relatively small amount of overhead caused by the array copy was more than compensated for by the speed gained in using arrays. While not eliminating the possibility of thread blocking, the probability was reduced dramatically. This change doubled the performance of both serial and parallel code.

After finishing the optimization of the code, attention was turned to improving the load balancing of the parallel code. Initially, load balancing was static. Domain partitioning gave each thread an exact number of taxa in the source cluster. A thread would then iterate through these taxa for each taxa in the destination cluster.

Timing of the initial parallel code showed that some threads ran as much as ten times longer than other threads under this partitioning method. This caused an enormous bottleneck and limited parallel performance, since the longest running thread determined the actual run-time. The exact cause of such wide variances in run-times is unknown, however, thread synchronization seems to be a likely cause

To empirically argue that this was the case, a simple experiment was performed. The program was run

on the same set of data several times, noting each time which thread was slowest. Results showed that the longest running thread always took from eight to ten times the amount of time for the fastest thread. However, the slowest thread changed each time the program was run. This eliminated the possibility that the data caused the discrepancy in run-times (since data partitioning was static), and argued for the cause being a synchronization issue (such as waiting for object locks).

Due to the dynamic impact of the above issues, the second attempt at parallelization used finer grained, dynamic partitioning, assigning each thread a pair of source and destination taxa, whenever the thread was available. This finer grained approach resulted in near identical run-times in each thread. Moreover, it decreased the impact of a thread getting "stuck" while waiting for an object lock.

3.3 Results and Analysis

Performance of the final version of the parallel code was evaluated on a quad-processor Windows 2000 machine. Four data sets, one small, two medium, and one large, were run using one to four threads. The small data set, called Snails.nex, had thirty taxa with 359 characters each. The medium data set LTRA.nex had 72 taxa, each with 725 characters. The other medium data set LTRB00.nex had 143 taxa with 518 characters each. The big file, LTRB01-4.nex, had 217 taxa with 521 characters each. The characters provide a way to calculate distances between taxa, but have no direct impact on the run-time of the parallel code. The number of Taxa and the distance between taxa are the determining factors in run time.

Although the parallelized version of the code offered performance improvements with larger data sets, the results from smaller data sets also proved interesting. Figure 6 shows that the application runs slower with four processors than two or three threads. Because this is a small data set, there is a high probability that several processors will request locks on the data structure. When this occurs, performance is diminished in several ways. First, an expensive context switch must occur since one of threads will not obtain the object lock and must go into a sleep state until the lock is available. There is also more barrier overhead when there are more threads. Finally, more threads create contention for memory, bus, and other computer resources.

Figure 7 shows that four threads run twice as fast as one thread. Though far from a linear speedup, the results show promise for larger data sets. Current research is being performed to allow threads on other shared memory nodes to communicate through network connections. This will allow several shared memory machines to participate in the computation. Increased problem sizes will be used on these larger systems.

The experimental results show that four threads are only 5% - 15% faster than three threads. Part of the problem lies in the sequential component of the computation. Although metric evaluation can be performed in parallel, cluster joining is a sequential computation. The next step in parallelization will be to eliminate sequential components when possible.

4 Acknowledgements

This work was supported in part by the Alfred P. Sloan Foundation, a Shannon Award from the National Institutes of Health, and NIH R01-HD34350.

5 Conclusions

Prior to this research, TCS calculations were all performed by hand and the corresponding tree was also drawn by hand. The TCS software has allowed systematic researchers to deal with much larger problems in population genetics. The run times for TCS analysis are much smaller than exhaustive search of the tree space and several options are possible for parallelization. The TCS software package has proved to be a valuable tool in DNA analysis.

Future work will include several projects to increase the utility and performance of TCS. The first project will include a benchmarking experiment with sequences from known trees to compare the trees generated with TCS to the original tree. The parallel code will be profiled and tested on larger data sets. The next step in population analysis is to determine the nesting values that help to separate the ingroup and the outgroup. Most of this code has been written and should be tested and benchmarked. Efforts are also underway to assign sequence data to the intermediate taxa that are inserted in the process.

The TCS software has been included in several NSF grant proposals and will continue to be enhanced to provide better solutions for systematic problems.

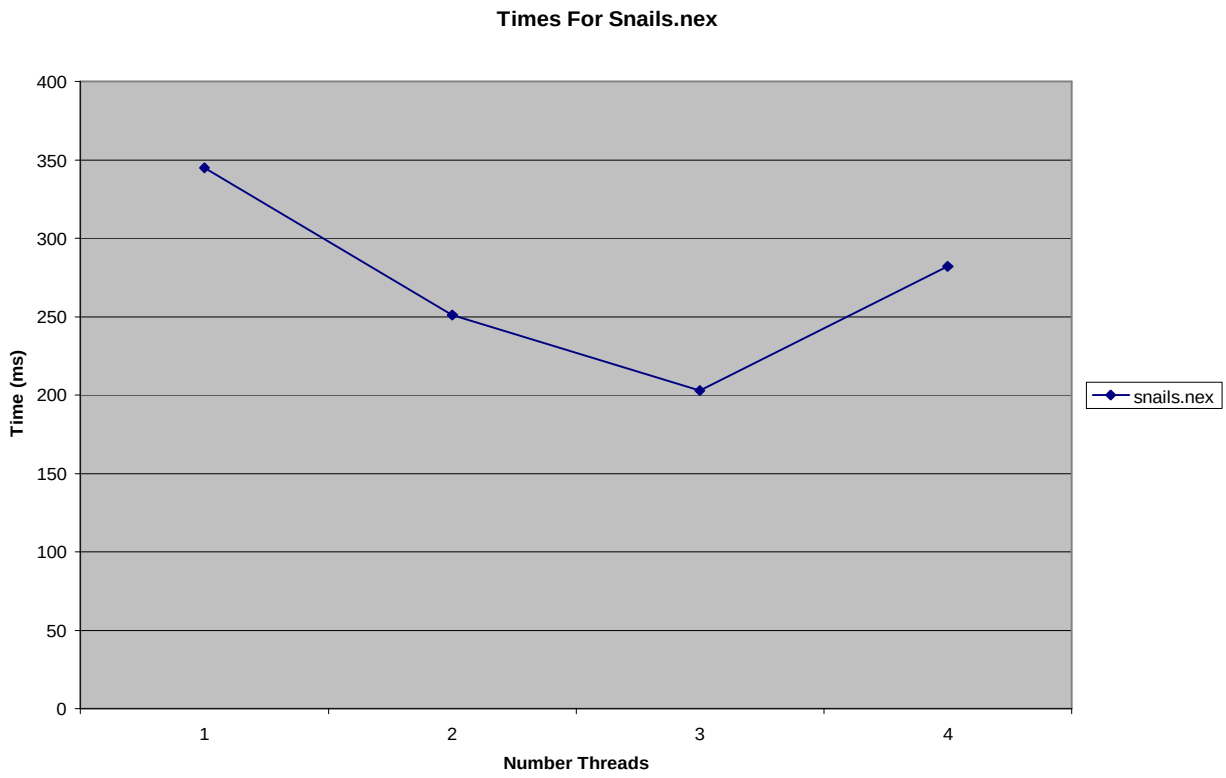

Figure 6: Small data set run times

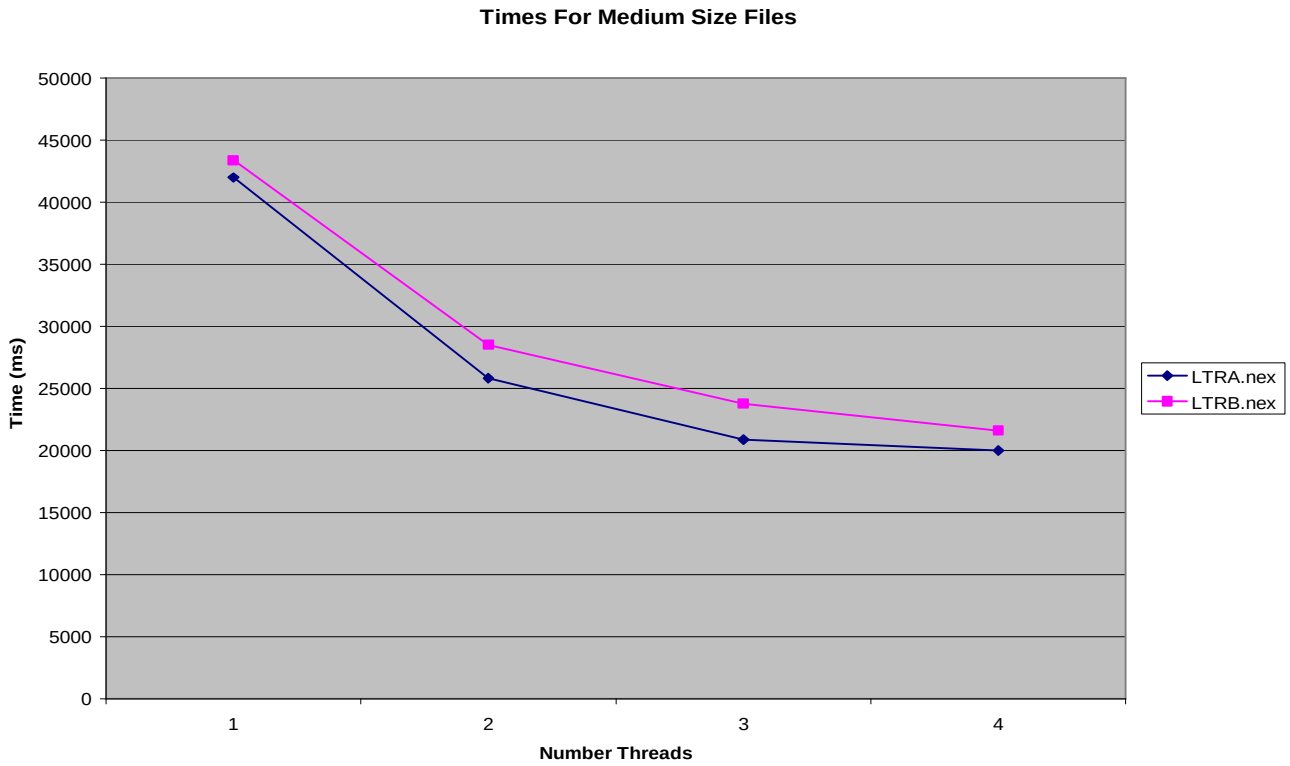

Figure 7: Medium data set run times.

6 References

Bandelt H-J, Dress AWM (1992) Split decomposition: a new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*,**1**, 242–252.

Bandelt H-J, Forster P, Sykes BC, Richards MB (1995) Mitochondrial portraits of human populations using median networks. *Genetics*,**141**, 743–753.

Benabib M, Kjer KM, Sites JW Jr (1997) Mitochondrial DNA sequence-based phylogeny and the evolution of viviparity in the *Sceloporus scalaris* group (Reptilia, Squamata). *Evolution*,**51**,1262–1275.

Bush RM, Bender CA, Subbarao K, Cox NJ, Fitch WM (1999) Predicting the evolution of human influenza A. *Science*,**286**,1921–1925.

Castelloe J, Templeton AR (1994) Root probabilities for intraspecific gene trees under neutral coalescent theory. *Molecular Phylogenetics and Evolution*,**3**, 102–113.

Crandall KA, Fitzpatrick JF Jr (1996) Crayfish molecular systematics: Using a combination of procedures to estimate phylogeny. *Systematic Biology*, **45**, 1–26.

Crandall KA, Templeton AR (1993) Empirical tests of some predictions from coalescent theory with applications to intraspecific phylogeny reconstruction. *Genetics*,**134**, 959–969.

Donnelly P, Tavaré S (1986) The ages of alleles and a coalescent. *Advances in Applied Probability*,**18**, 1–19.

Excoffier L, Smouse PE (1994) Using allele frequencies and geographic subdivision to reconstruct gene trees within a species: Molecular variance parsimony. *Genetics*,**136**, 343–359.

Felsenstein J (1991) *PHYLIP: Phylogenetic Inference Package*. University of Washington, Seattle, WA.

Fitch WM (1997) Networks and viral evolution. *Journal of Molecular Evolution*, **44**, S65–S75.

Georgiadis N, Bischof L, Templeton A *et al*. (1994) Structure and history of African elephant populations: I. Eastern and Southern Africa. *Journal of Heredity*,**85**, 100–104.

Gerber AS, Templeton AR (1996) Population sizes and within-deme movement of *Trimerotropis saxatilis* (Acrididae), a grasshopper with a fragmented distribution. *Oecologia*, **105**, 343–350.

Gómez-Zurita J, Petitpierre E, Juan C (2000) Nested cladistic analysis, phylogeography and speciation in the *Timarcha goettingensis* complex (Coleoptera, Chrysomelidae). *Molecular Ecology*, **9**, 557–570.

Hedin MC (1997) Speciational history in a diverse clade of habitat-specialized spiders (Araneae: Nesticidae: *Nesticus*): Inferences from geographic-based sampling. *Evolution*, **51**, 1929–1945.

Johnson JB, Jordon S (2000) Phylogenetic divergence in leatherside chub (*Gila copei*) inferred from mitochondrial cytochrome *b* sequences. *Molecular Ecology*, **9**, 1029–1035.

Maddison DR, Swofford DL, Maddison WP (1997) nexus: an extensible file format for systematic information. *Systematic Biology*, **46**, 590–621.

Pagel M (1999) Inferring the historical patterns of biological evolution. *Nature (London)*, **401**, 877–884.

Routman E, Wu R, Templeton AR (1994) Parsimony, molecular evolution, and biogeography: The case of the North American Giant Salamander. *Evolution*, **48**, 1799–1809.

Schaal BA, Hayworth DA, Olsen KM, Rouscher JT, Smith WA (1998) Phylogeographic studies in plants: problems and prospects. *Molecular Ecology*, **7**, 465–474.

Templeton AR (1998) Nested clade analyses of phylogeographic data: testing hypotheses about gene flow and population history. *Molecular Ecology*, **7**, 381–397.

Templeton AR, Crandall KA, Sing CF (1992) A cladistic analysis of phenotypic associations with haplotypes inferred from restriction endonuclease mapping and DNA sequence data. III. Cladogram estimation. *Genetics*, **132**, 619–633.

Templeton AR, Routman E, Phillips CA (1995) Separating population structure from population history: a cladistic analysis of geographical distribution of mitochondrial DNA haplotypes in the tiger salamander, *Ambystoma tigrinum*. *Genetics*, **140**, 767–782.

Turner TF, Trexler JC, Harris JL, Haynes JL (2000) Nested cladisitic analysis indicates population fragmentation shapes genetic diversity in a freshwater mussel. *Genetics*, **154**, 777–785.

Vilá C, Amorim IR, Leonard JA *et al*. (1999) Mitochondrial DNA phylogeography and population history of the Gray Wolf *Canis lupus*. *Molecular Ecology*, **8**, 2089–2103.

Watterson GA, Guess HA (1977) Is the most frequent allele the oldest? Theoretical Population Biology , 11, 141–160.

McCreary C, Barowski L (1998), The VGJ Graph Drawing Tool, http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html, February 18, 1998.