BIOINFORMATICS ORIGINAL PAPER Vol. 27 no. 23 2011, pages 3250–3258
doi:10.1093/bioinformatics/btr553

Sequence analysis Advance Access publication October 7, 2011

FastSP: linear time calculation of alignment accuracy

Siavash Mirarab and Tandy Warnow*

Department of Computer Science, The University of Texas at Austin, 1616 Guadalupe Street, Austin 78701, USA

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Multiple sequence alignment is a basic part of much biological research, including phylogeny estimation and protein structure and function prediction. Different alignments on the same set of unaligned sequences are often compared, sometimes in order to assess the accuracy of alignment methods or to infer a consensus alignment from a set of estimated alignments.

Three of the standard techniques for comparing alignments, Developer, Modeler and Total Column (TC) scores can be derived through calculations of the set of homologies that the alignments share. However, the brute-force technique for calculating this set is quadratic in the input size. The remaining standard technique, Cline Shift Score, inherently requires quadratic time.

Results: In this article, we prove that each of these scores can be computed in linear time, and we present FastSP, a linear-time algorithm for calculating these scores. Even on the largest alignments we explored (one with 50 000 sequences), FastSP completed <2 min and used at most 2 GB of the main memory. The best alternative is qscore, a method whose empirical running time is approximately the same as FastSP when given sufficient memory (at least 8 GB), but whose asymptotic running time has never been theoretically established. In addition, for comparisons of large alignments under lower memory conditions (at most 4 GB of main memory), qscore uses substantial memory (up to 10 GB for the datasets we studied), took more time and failed to analyze the largest datasets.

Availability: The open-source software and executables are available online at http://www.cs.utexas.edu/~phylo/software/fastsp/.

Contact: tandy@cs.utexas.edu

Received on June 16, 2011; revised on September 21, 2011; accepted on September 29, 2011

1 INTRODUCTION

Estimation of multiple sequence alignments for molecular datasets is fundamental to many problems in biology, including the prediction of protein function and structure and phylogeny estimation. Estimated alignments are often compared with other alignments in order to assess accuracy or to determine the features shared by two or more alignments. In addition, since different alignment methods can produce alignments that differ enough to introduce phylogenetic uncertainty (Wong *et al.*, 2008) and alignment error increases with the size of the dataset (Liu *et al.*, 2009, 2010), the use of several alignments, and comparisons of these alignments, is advisable for large-scale phylogenetic studies.

Of the various methods for comparing an estimated alignment to a reference alignment, four are generally in use: the Developer score (also called the SP-score, for sum-of-pairs), Modeler score, Total Column score and Cline Shift Score. The SP-score and Modeler scores are quite similar: the SP-score is the percentage of the homologies in the reference alignment that appears in the estimated alignment, and the Modeler score is the percentage of the homologies in the estimated alignment that appears in the reference alignment. Thus, each can be obtained by computing the number of shared homologies and then normalizing by either the number of homologies in the reference or true alignment. The Total Column score is the number of alignment columns shared by both alignments, and can also be normalized by the number of columns in one of the alignments. Finally, the Cline Shift Score (Cline *et al.*, 2002) is computed by averaging the Cline Shift scores for each of the induced pairwise alignments. Each of these normalized scores ranges from 0% to 100%, with 0 indicating that the two alignments are maximally dissimilar for the criterion and 100% indicating that the two alignments are considered identical with respect to the criterion. Thus, these scores represent accuracy measures, and complementing these scores (subtracting them from 100%) produces the corresponding error metrics.

While several methods have been developed for comparing alignments, only qscore (Edgar, 2004) and Lobster (available online at http://www.drive5.com/qscore and http://www.drive5.com/lobster, respectively) correctly compute the SP-score. However, the computational complexity of these methods—i.e. their asymptotic running time—has never been established. Clearly, all four scores can be computed in quadratic time using brute-force techniques, and the Cline Shift Score (by definition) requires quadratic time. However, the SP-score, Modeler score and Total Column scores might not require quadratic time.

In this article, we show that the SP-score, Modeler score and Total Column scores can each be computed in linear time [i.e. in $O(nk)$ time where n is the number of sequences and k is the length of the longer alignment]. We present a method, FastSP, to compute the number of shared homologies between two alignments and prove that it runs in linear time. The result then follows, since each of the three scores can be obtained directly from this number. Since error scores are complements of these accuracy scores, FastSP can be used to compute error rates such as SP-FN (the percentage of true homologies missing in the estimated alignment) and SP-FP (the percentage of predicted homologies that are not present in the true alignment). Furthermore, FastSP can be used to evaluate the reliability of a given alignment with respect to a set of alignments, since it enables fast all-pairs comparisons. Thus, FastSP is a basic tool for analyzing sets of multiple sequence alignments.

*To whom correspondence should be addressed.

Several aspects of the algorithm design lead to the linear-time complexity of FastSP. First, we represent the set of homologies defined by a single multiple sequence alignment in an auxiliary data structure, and by toggling back and forth between the two representations of the alignment, we are able to implicitly represent the common homologies between a pair of alignments in linear time. We then show that computing the number of shared homologies from that implicit representation can also be done in linear time through the use of pre-computed values.

We have also taken care with the memory usage in our implementation of FastSP; as a result, our code is not only fast, but also never used more than 2 GB on any of our datasets. We found that FastSP used much less memory than qscore, the fastest competitor. Thus, while FastSP and qscore had essentially the same running time when memory is unlimited (both completing in just a few minutes on all datasets), FastSP ran much faster when analyzing large alignments when the available memory was limited to 4 GB. Thus, FastSP improved upon all current methods for computing the accuracy scores between large alignments: it reduces running time on low memory conditions, and matches running time and reduces memory usage on high memory conditions.

2 APPROACH

Let S be a set of n sequences over a fixed alphabet Σ (where Σ is arbitrary but is typically nucleotides or amino acids). Let $\mathcal{A}$ and $\mathcal{A}'$ be two input alignments on S. We can define an alignment by its set of homologies, where a homology is a pair of letters from the input sequences that occur in the same site. Although the alphabet can be arbitrary, we will illustrate the definitions using nucleotides.

Consider $S=\{s_1,s_2\}$ where $s_1=ACAT$ and $s_2=GA$ and alignment $\mathcal{A}$ be given by

$$s_1=-A\ \ C\ A\ T$$
$$s_2=GA\ \ -\ -\ -$$

Let alignment $\mathcal{A}'$ be given by

$$s_1=A\ C\ A\ T$$
$$s_2=-\ G\ A\ -$$

Then these two pairwise alignments share *no* common homologies, even though both have a site that contains two *A*'s. That is, we need to distinguish between the different occurrences of each nucleotide within each sequence. Thus, instead of writing $s_1=ACAT$, we will write $s_1=N_{1,1},N_{1,2},N_{1,3},N_{1,4}$, so that $N_{i,j}$ indicates that nucleotide N appears as the j-th letter of sequence s_i (if we are interested in the actual nucleotides, we would save the information that $N_{1,1}=A,N_{1,2}=C$, etc.). Then the homologies defined by an alignment consist of pairs of these subscripted nucleotides. Thus, $\mathcal{A}=\{(N_{1,1},N_{2,2})\}$, and $\mathcal{A}'=\{(N_{1,2},N_{2,1}),(N_{1,3},N_{2,2})\}$. With this representation, it is clear that although both alignments contain sites that are uniformly occupied by the nucleotide A, they do not share any common homologies. This representation of an alignment as a set of homologous pairs extends to multiple alignments by taking the union of homologous pairs of each of the pairwise alignments induced by the multiple alignment. We will denote by $H(\mathcal{A})$ the set of homologous pairs defined by the alignment $\mathcal{A}$, and by $h(\mathcal{A})=|H(\mathcal{A})|$. Note that if $\mathcal{A}$ has R sites and r_i non-gap letters in the i-th site, then $h(\mathcal{A})=\sum_{i=1}^{R}\frac{r_i(r_i-1)}{2}$. Thus, every alignment of n sequences with length R consists of $O(Rn^2)$ homologous pairs.

Under the assumption that the true alignment is known, the error in an estimated alignment can be defined in several ways. One of the various ways of defining this is the SP-FN error, where 'SP' stands for *sum-of-pairs*, and 'FN' stands for 'false negative'. Thus, the SP-FN error of an estimated alignment is the number of homologous pairs in the true alignment that are missing in the estimated alignment (i.e. false negatives). Similarly, the SP-FP error is the number of pairs that are predicted to be homologous in the estimated alignment that are not present in the true alignment (i.e. false positives). Both quantities can be then normalized by the number of homologous pairs in the relevant alignment to produce a value between 0 and 1 (so that the SP-FN error is normalized by the number of homologies in the true alignment, and SP-FP is normalized by the number of homologies in the estimated alignment).

As an example, suppose we have $S=\{s_1,s_2,s_3,s_4,s_5\}$ where $s_1=AC,s_2=C,s_3=A,s_4=T$ and $s_5=G$. Let alignment $\mathcal{A}$ on S be given by

$$s_1=AC$$
$$s_2=-C$$
$$s_3=A-$$
$$s_4=T-$$
$$s_5=G-$$

We rewrite this as

$$s_1=N_{1,1}N_{1,2}$$
$$s_2=-N_{2,1}$$
$$s_3=N_{3,1}-$$
$$s_4=N_{4,1}-$$
$$s_5=N_{5,1}-$$

The set $H(\mathcal{A})$ of homologous pairs defined by $\mathcal{A}$ is therefore $H(\mathcal{A})=\{(N_{1,1},N_{3,1}),(N_{1,1},N_{4,1}),\ (N_{1,1},N_{5,1}),\ (N_{3,1},N_{4,1}),\ (N_{3,1},N_{5,1}),\ (N_{4,1},N_{5,1}),\ (N_{1,2},N_{2,1})\}$.

Note that the first site in $\mathcal{A}$ has four nucleotides and hence contributes six homologous pairs to this set, whereas the second site has two nucleotides and so contributes only one homologous pair to the set.

Now suppose $\mathcal{A}'$ is another alignment on the same set S given by

$$s_1=AC$$
$$s_2=-C$$
$$s_3=-A$$
$$s_4=-T$$
$$s_5=G-$$

Then we rewrite $\mathcal{A}'$ in the same way we rewrote $\mathcal{A}$ (using the $N_{i,j}$ notation), and obtain its set of homologous pairs as $H(\mathcal{A}')=\{\ (N_{1,1},N_{5,1}),\ (N_{1,2},N_{2,1}),\ (N_{1,2},N_{3,1}),\ (N_{1,2},N_{4,1}),\ (N_{2,1},N_{3,1}),\ (N_{2,1},N_{4,1}),\ (N_{3,1},N_{4,1})\}$.

Thus, the set of homologies shared in common between these two alignments is $H(\mathcal{A})\cap H(\mathcal{A}')=\{(N_{1,1},N_{5,1}),(N_{3,1},N_{4,1}),(N_{1,2},N_{2,1})\}$. If we were to treat one alignment (say, $\mathcal{A}$) as the true alignment, then the alignment SP-FN error for $\mathcal{A}'$ would be the

number of homologies in $\mathcal{A}$ missing from $\mathcal{A}'$, or $|H(\mathcal{A})-H(\mathcal{A}')|$. In this particular case, that would mean that there were four missing homologies, since the two alignments shared three common homologies and $\mathcal{A}$ had seven homologies.

3 ALGORITHM

Throughout this section, we will assume that we have two alignments $\mathcal{A}$ and $\mathcal{A}'$, each on n sequences, and with k the length of the longer alignment.

Several brute-force approaches can be used to calculate the number of shared homologies between two alignments, and hence the true positive rate, as well as the false positive rate. For example, if each sequence has length at most L (in the unaligned form), then each alignment can be represented as a presence/absence matrix with nL rows and columns; consequently, the true positive rate thus can be calculated in $O(n^2L^2)$ time. However, we can calculate this quantity more efficiently, as we now show.

We begin by defining an equivalence relation on the letters $N_{i,j}$ that appear in a site x of alignment $\mathcal{A}$. We say that two elements of the x-th site are *equivalent* if they are in the same site of $\mathcal{A}'$. For a given site x in $\mathcal{A}$ and a given alignment $\mathcal{A}'$, there will be t equivalence classes (where an equivalence class is a maximal set of letters that are pairwise equivalent), and we let $m(i)$ denote the cardinality of the i-th equivalence class.

For example, the nucleotides that appear in site 1 in alignment $\mathcal{A}$ are $N_{1,1},N_{3,1},N_{4,1}$ and $N_{5,1}$. In this definition, $N_{1,1}$ and $N_{5,1}$ are equivalent since they appear in site 1 in $\mathcal{A}'$, and $N_{3,1}$ and $N_{4,1}$ are equivalent since they appear in site 2 in $\mathcal{A}'$. Thus, $\mathcal{A}'$ defines an equivalence relation on these elements of two equivalence classes (one for each site in $\mathcal{A}'$), each having two elements.

Note then that the number of homologous pairs that are in common between $\mathcal{A}$ and $\mathcal{A}'$ is the sum, over all the sites x in $\mathcal{A}$, of the number of pairs of letters that appear in site x that are equivalent to each other. Furthermore, the number of such pairs contributed by an equivalence class of size z is exactly $\frac{z(z-2)}{2}$. This means we can calculate the total number of common homologies by computing the equivalence relation defined by each site in $\mathcal{A}$, and the number of elements in each of the equivalence classes for that site. We now show how to do this efficiently.

The first thing we do is to replace the nucleotide entries in the alignment by entries of the form $N_{i,j}$ (i.e. we show the indices). In fact, the indices are the only things we need to know—we do not need to know the actual nucleotides in each position of the alignment. Given this representation of the alignment, we then define an $n\times k$ matrix $S[i,j]$ of ordered pairs, so that $S[i,j]=(a,b)$ implies that $N_{i,j}$ appears in site a for alignment $\mathcal{A}$ and in site b for alignment $\mathcal{A}'$. If j is larger than the length of the i-th sequence (after gaps are removed), then we set $S[i,j]$ to $(0,0)$.

As an example, given the alignments $\mathcal{A}$ and $\mathcal{A}'$ from the previous section, we have $n=5$ and $k=2$. For this pair of alignments, we obtain $S[1,1]=(1,1)$, $S[1,2]=(2,2)$, $S[2,1]=(2,2)$, $S[2,2]=(0,0)$, $S[3,1]=(1,2)$, $S[3,2]=(0,0)$, $S[4,1]=(1,2)$, $S[4,2]=(0,0)$, $S[5,1]=(1,1)$ and $S[5,2]=(0,0)$.

- Step 0: we set $A(j)=\frac{j(j-1)}{2}$ for each $j=1,2,\ldots,n$. For each site x in alignment $\mathcal{A}$, let b_x be the number of non-gapped entries in the site. Then the number of homologous pairs in $\mathcal{A}$ is $h(\mathcal{A})=\sum_x A(b_x)$.
- Step 1: for each site x in $\mathcal{A}$, let $N(x)$ be the number of homologies in $\mathcal{A}'$ that are shared with the x-th site in $\mathcal{A}$. Thus, $N(x)=\sum_{i=1}^{t}A(m(i))$, where $m(i)$ is the number of elements in the i-th equivalence class, and there are t equivalence classes.
- Let $\mathcal{N}=\sum_{x=1}^{k}N(x)$, where k is the number of sites in $\mathcal{A}$.

Since $N(x)$ is the number of shared homologies for site x in $\mathcal{A}$, it follows that $\mathcal{N}$ is the total number of shared homologies. Hence the SP-FN score is $H(\mathcal{A})$-$\mathcal{N}$. As an illustration, using the alignments $\mathcal{A}$ and $\mathcal{A}'$ given previously, $N(1)=2$, since the first site in $\mathcal{A}$ contains six homologous pairs, two of which are also present in $\mathcal{A}'$. Similarly we see that $N(2)=1$. Therefore, $m(1)=m(2)=2$. Note that $A(m(1))=A(m(2))=A(1)=A(2)=1$, and so $N(1)=A(1)+A(2)=1+1=2$. A similar calculation yields $N(2)=1$. Therefore, $\mathcal{N}=N(1)+N(2)=3$, as expected.

We now show how to compute $\mathcal{N}$ and $h(\mathcal{A})$ efficiently. Consider a fixed site x in $\mathcal{A}$. We examine alignment $\mathcal{A}'$ and write down the (at most k) sites in $\mathcal{A}'$ for the elements of site x.

Then we examine each site x in $\mathcal{A}$, and write down its nucleotides. For each i,j such that $N_{i,j}$ appears in site 1, we examine $S[i,j]$. Since $N_{i,j}$ appears in site x for alignment A, $S[i,j]=(x,b)$, and hence $N_{i,j}$ appears in site b in alignment $\mathcal{A}'$. We then write down b in the list of sites in $\mathcal{A}'$ associated with site x. That is, we note the set $Sites(x)$ of $\mathcal{A}'s$ sites y that appear, and the multiplicity $m(y)$ with which each site y appears. As an example, consider the same pair of alignments $\mathcal{A}$ and $\mathcal{A}'$, as before. We examine the first site in $\mathcal{A}$, and we see $N_{1,1},N_{3,1},N_{4,1}$ and $N_{5,1}$. We then examine $S[1,1],S[3,1],S[4,1]$ and $S[5,1]$. We see $S[1,1]=(1,1),S[3,1]=(1,2),S[4,1]=(1,2)$ and $S[5,1]=(1,1)$. We write down the sequence $1,2,2,1$, so that $Sites(x)=\{1,2\}$, and $m(1)=m(2)=2$. This sequence gives us two equivalence classes, each of size two. Thus, $m(1)=m(2)=2$.

The algorithm will compute the following values:

- $List(x)$: the list of sites in $\mathcal{A}'$ that appears in site x of $\mathcal{A}$. This is initialized to the empty list.
- $m(y)$: a non-negative integer for the number of occurrences of y in $List(x)$, for each y that appears in $List(x)$. This is initialized to 0 for all $y=1,2,\ldots,k$.
- $Processed(y)$: a Boolean variable that indicates that we have added $A(y)$ to $N(x)$. This is initialized to False for all $y=1,2,\ldots,k$.
- $N(x)$: a non-negative integer that will be $\sum_{y\in List(x)}A(m(y))$. This is initialized to 0.

To set all these values, we examine the site x in $\mathcal{A}$. As we visit each element of site x in $\mathcal{A}$, we append the site in $\mathcal{A}'$ to $List(x)$. Thus, for $N_{i,j}$ in site x of alignment $\mathcal{A}$, with $(a,b)=S[i,j]$, we note that $a=x$ and b is the site in $\mathcal{A}'$ for $N_{i,j}$. We therefore append b to $List(x)$.

We initialize $m(y)\leftarrow 0$ for all $y=1,2,\ldots,k$, where k is the number of sites in $\mathcal{A}'$. We then scan $List(x)$, and for each element y in $List(x)$, we add 1 to $m(y)$. At the end of the scan of $List(x)$, all the counts $m(y)$ are set properly, but $Processed(y)$ is still $False$ for every y.

We process $List(x)$ again. For each element y of $List(x)$, if $Processed(y)$ is $False$, we set $N(x)\leftarrow N(x)+A(m(y))$, and set $Processed(y)$ to $True$. Thus, $A(m(y))$ is added to $N(x)$ exactly once during the pass through $List(x)$.

As we process $List(x)$, we can also find out whether the entire column is aligned correctly (required for TC-score). Note that

column x is aligned correctly in $\mathcal{A}'$ if and only if two conditions hold. First, there should exists only one value of y for which $m(y)$ is non-zero. If there are multiple values of y with $m(y)>0$, then the column x in $\mathcal{A}$ has been divided into multiple columns in $\mathcal{A}'$. Second, the number of characters in column y of $\mathcal{A}'$ [call this quantity $c(y)$] should be equal to $m(y)$. If the first condition holds but the second condition does not hold, the column in alignment $\mathcal{A}'$ that corresponds to column x in $\mathcal{A}'$ has some extra characters and should not be considered correctly aligned. Therefore, to compute the column score we need to check (for each column x in $\mathcal{A}$) whether $m(y)$ has non-zero values for only one y, *and* that $c(y)=m(y)$. This can be easily checked while scanning elements of $List(x)$ by keeping a count of non-zero $m(y)$ values and by checking $m(y)$ against $c(y)$. Note that $c(y)$, the number of characters in columns of $\mathcal{A}'$, can be easily computed and kept in memory as matrix S is being created. TC can be computed at the end by dividing the number of correctly aligned sites by the total number of sites in $\mathcal{A}$. Columns that have only one non-gap character do not represent homology statements, and are ignored in the calculation of the TC-score. After processing site x, we reset all the variables to their initial values.

Running time analysis: recall that we have n sequences and the longer of the two alignments has k sites. Calculating the matrix $S[i,j]$ takes $O(nk)$ time, as follows. We initialize $S[i,j]=0$ for all i,j. We rewrite each alignment so that we write each nucleotide in the form $N_{i,j}$. We then visit each site in $\mathcal{A}$, and for each nucleotide $N_{i,j}$ that appears in site x we set the first element of $S[i,j]$ to x. We repeat this for $\mathcal{A}'$, and for each nucleotide $N_{i,j}$ that appears in site y in $\mathcal{A}'$ we set the second element of $S[i,j]$ to y.

The preprocessing step of the algorithm involves the calculation of $A(j)$ for each $j=1,2,...k$, and so requires $O(k)$ arithmetic steps involving numbers up to k; thus, the preprocessing costs $O(k)$ (since each arithmetic step has unit cost). Calculating each $N(x)$ involves $O(n)$ operations, and hence calculating all $N(x)$ involves $O(nk)$ operations. Finally, calculating $\mathcal{N}=\sum_{x=1}^{k}N(x)$ costs $O(k)$ work. Thus, this algorithm costs $O(nk)$ time, which is linear in the input size.

4 THE FastSP CODE

The algorithm described above is implemented in a publicly available Java program, FastSP. The implementation has been designed to reduce the memory usage. Therefore, although FastSP keeps the reference alignment in memory as a list of strings, it does not keep the entire estimated alignment in memory; instead, it only keeps the matrix S. Also the first elements of the tuples defined for S are not necessary and therefore our implementation omits them. In addition, the number of columns in S can be reduced to the number of non-gap characters in the longest sequence in the reference alignment. Therefore, if the reference alignment has R sites and the longest sequence has L non-gap characters, then FastSP needs Ln integers (each 4 bytes) to store S, and Rn characters (each 2 bytes) to store the reference alignment. Since $L \leq R$, this means that FastSP needs only $O(Rn)$ memory, and so scales linearly with the reference alignment size, and not also with the estimated alignment size.

The implementation first reads the reference alignment and saves it in memory. It then reads the estimated alignment and creates the S matrix as an $n \times k$ array of integers. Next, it iterates over the columns of the reference alignment. In each iteration, first, referring to matrix S, a data structure is created (using a Java HashMap) that for each column of the estimated alignment holds the number of letters in that column that correspond to a letter in the reference alignment column [similar to $m(y)$ introduced before]. Then, $A(x)=\frac{x(x-1)}{2}$ is evaluated for the elements in this data structure, and the sum of these values is added to the total number (N) of shared homologies. Each iteration also computes the number of homologies in the current column of the reference alignment, and adds this value to the total number of homologies in the reference alignment $h(\mathcal{A})$. In addition, we keep track of the number of correctly aligned columns (C), using the technique described earlier. Once the iterations finish, N and $h(\mathcal{A})$ provide the required information to compute the SP-score and Modeler score, and C provides the information required to compute TC.

We tested our implementation on many different datasets, and compared SP, Modeler and TC scores computed to the scores computed using existing tools. For every case where there was a disagreement between our scores and those produced by other software, we were able to find a bug in existing tools (as described later in the article).

5 PERFORMANCE STUDY

5.1 Datasets

We studied the performance of the proposed algorithm on a variety of datasets, both simulated and biological (see Table 1 for the dimensions of a representative sample of the alignments on each of these datasets). Our datasets were selected to span a wide range of alignment sizes, from 100 to 78 000 sequences.

We used four simulated nucleotide datasets, three with 100, 500 and 1000 taken from Liu *et al.* (2009) and one much larger dataset with 78 000 sequences from Price *et al.* (2010); these are all available online at http://www.cs.utexas.edu/users/phylo/datasets/ (Liu *et al.*, 2010). For the biological datasets, we included the datasets used in Liu *et al.* (2009), taken from the Gutell Comparative Ribosomal Website (CRW) (http://www.rna.ccbb.utexas.edu/). For each of these datasets, we have a curated alignment based upon confirmed secondary (and higher order) structures, which is highly reliable. We also included another nucleotide alignment with 50 000 sequences, 16S.GG-50K, which is a random sample from a dataset of 237 882 sequences obtained from the Green Genes database (DeSantis *et al.*, 2006), and studied previously (Price *et al.*, 2010).

On every dataset, we obtained at least one (typically more than three) estimated alignments and a reference alignment. On the biological datasets, in all but one case we used an available curated alignment as the reference. In the case of 16S.GG-50k, no curated alignment was available; however, the original publication of this dataset includes an automatically created alignment (Price *et al.*, 2010), which we used as our reference alignment. On the simulated datasets, we used the true alignment as the reference alignment. In total, our study involves the calculation of alignment scores for 75 pairs of alignments, one estimated and one reference alignment.

5.2 Alignment methods

We included alignments estimated on these datasets using SATé (Liu *et al.*, 2009, 2011), MAFFT (Katoh and Toh, 2008), MAFFT-PartTree (Katoh and Toh, 2007), Muscle (Edgar, 2004), Prank+GT

Table 1. Datasets and their sizes

Dataset	n[a]	L	Ref.	MAFFT	OPAL	PART	PRANK	QUICK	SATé	SATé-II
100L1-R0	100	1089	2287	1563						1737
500L1-R0	500	1110	4992	3307						3421
1000L1-R0	1000	1079	3517	907						2856
Price-78K	78132	1286	1287			1504				
23S.E	117	5317	9079	8929	9860	11018	13941	6796	10352	
23S.E.aa_ag	144	1079	8619	8123	9576	10956	14343	7017	9029	
23S.M.aa_ag	263	4483	10305	7353	13625	12320	13471	5522	7815	
23S.M	278	4216	10738	7478	10447	13384	13639	5311	8746	
16S.M	901	2023	4722	4418	12812	9496	12826	3216	4776	
16S.M.aa_ag	1028	2672	4907	4493	13785	11225	20856	3317	48881	
16S.3	6323	4066	8716			19775		5310	10186	20414
16S.T	7350	4066	11856	10891	43797	25951		6109	12301	29156
16S.B.ALL	27643	1851	6857			14217		3413		8463
16S.GG-50K	50000	1701	7682			14877				

[a]n is the number of sequences. L is the maximum number of nucleotides in any of the sequences. *Ref.* is the length of the reference alignment (i.e. including gaps). The rest of the columns show the length of the alignment for each estimated alignment. An empty cell indicates that the respective alignment method was not run on that particular dataset. The first four datasets are simulated datasets, while the rest are all real biological datasets.
We have included results from two different runs of SATé-II on 16S.B.ALL dataset. The second version had a length of 8209.

(Liu *et al.*, 2009; Loytynoja and Goldman, 2005), Opal (Wheeler and Kececioglu, 2007) and ClustalW in its default and Quicktree versions (Thompson *et al.*, 1994). MAFFT-PartTree and ClustalW-Quicktree methods are specifically designed to be used on very large datasets.

Most of the alignments used in this article are available from the original studies. In addition, we obtained the following alignments. On 16S.GG-50k, we produced a PartTree alignment, and we produced SATé-II [Liu *et al.*, (2011)] alignments on the 16S.B.ALL, 16S.T and 16S.3 datasets. The alignment on the simulated dataset Price-78K is also one of the datasets obtained from Liu *et al.* (2010). For the other simulated datasets, we ran MAFFT, ClustalW (in its default setting) and SATé-II (in its default setting) to obtain estimated alignments. These additional alignments were produced using ClustalW version 2.0.4 (downloaded from http://www.ebi.ac.uk/Tools/msa/clustalw2/), MAFFT version 6.240 (downloaded from http://mafft.cbrc.jp/alignment/software/) and SATé-II version 1.22 (downloaded from http://phylo.bio.ku.edu/software/sate/sate.html). In the commands for each method, given below, `<input>` is a FASTA-formatted input file containing unaligned sequences and `<output>` is the resulting FASTA-formatted output file.

- ClustalW: `clustalw2 -align -infile=<input> -outfile=<output> -output=fasta`
- MAFFT: `mafft –localpair –maxiterate 1000 –quiet <input> > <output>`
- MAFFT-PartTree: `mafft –parttree –retree 2 –partsize 1000 <input> > <output>`
- SATé: `python run_sate.py -i <input>`

5.3 Methods for calculating the SP-score

There are a few existing programs that compute the SP-score, or can be modified to compute this. However, not all software claiming to compute the SP-score actually calculates it correctly. For example, although BALI_SCORE is widely used, it is now known (Blażewicz *et al.*, 2009) that this calculation is incorrect in some cases: BALI_SCORE ignores sites that have >20% gaps and also ignores sites that have a gap in the first sequence (Blażewicz *et al.*, 2009).

Our study used three programs in addition to FASTSP to compute SP-scores: QSCORE (Edgar, 2004) (also known as the PREFAB Q-score, and downloaded from http://www.drive5.com/qscore), LOBSTER (downloaded from http://www.drive5.com/lobster) and BIGMATRIX (Liu *et al.*, 2010) (downloaded from http://www.cs.utexas.edu/users/kliu/public/BigDataMatrix.java). LOBSTER is a precursor to QSCORE, and BIGMATRIX implements a brute-force quadratic time algorithm for the SP-score; these are included in our experiments for completeness, but the main focus of the study is comparing QSCORE and FASTSP.

Our initial attempts to run QSCORE on large datasets failed (i.e. Q-scores >1 were produced). After corresponding with the author, we were able to diagnose the problem as an integer overflow, which we then solved by changing the code to use 64 bit integers (more precisely, we added the following line to fastq.cpp: `#define unsigned int64_t`).

By default, QSCORE computes SP- and TC-scores, both of which by FASTSP is computed as well. However, there is a problem with the way QSCORE calculates the TC-score. Consider the two alignments given by

$\mathcal{A}$	$\mathcal{A}'$
A–CGCC	A–CGCC
ACCT–C	ACCT–C
TG–TCT	TG–TCT
T–GTCG	TG–TCT.

Here, we have four correctly aligned columns (1,3,4 and 5) and therefore correct TC-score is 0.67. QSCORE, however, considers the number of correctly aligned columns to be 5, giving 0.83 for TC-score. QSCORE considers column 2 as correctly aligned, persumably

because all homologies in $\mathcal{A}$ are present in the same column in alignment $\mathcal{A}'$. It seemingly ignores the fact that column 2 of alignment $\mathcal{A}'$ has an extra character. FastSP avoids this problem by making sure the associated column in the estimated alignment does not contain extra characters. qscore can also compute the Modeler score if run with extra options, but for reasons that are not clear to us, qscore runs much slower when computing Modeler score (see the line labeled 'qscore-modeler' in Fig. 3). Also, qscore can compute Cline Shift Score (Cline *et al.*, 2002) if run with extra options, but that uses a different part of the qscore code that runs in quadratic time.

We present results comparing qscore, qscore-modeler, Lobster, BigMatrix and FastSP for calculations of the SP-score. We used the following commands:

- qscore: `qscore -test <estimated alignment> -ref <reference alignment>`
- qscore-modeler: `qscore -modeler -test <estimated alignment> -ref <reference alignment>`
- Lobster: `lobster -score <estimated alignment> -ref <reference alignment>`
- BigMatrix: `java -Xmx3500m BigDataMatrix -v <reference alignment> -f <estimated alignment> -sp`
- FastSP: `java -Xmx2000m FastSP -e <estimated alignment> -r <reference alignment>`

5.4 Evaluation procedure

We compared FastSP and qscore under three conditions—64-bit machines with 2 GB, with 4 GB or with 8–32 GB of the main memory. We also ran Lobster and BigMatrix on the 8–32 GB machines as well. In our first experiment, we performed the analyses on a dedicated 64-bit Linux machine, each with 4 GB of main memory. The subsequent experiments were performed on a heterogeneous Condor cluster (Litzkow *et al.*, 1988).

We compared the SP-scores calculated by different programs to make sure they were identical. After the modification to qscore to ensure that it runs on all datasets, qscore, BigMatrix, Lobster and FastSP all produced the same SP-scores on all the datasets on which they were able to run. We also gathered data regarding the peak memory usage and the running time of each execution, as reported in the next section. We note that small differences in running times should not be considered significant, due to the conditions in which the experiments were run: heterogeneous machines with varying amounts of main memory (unless limited by JVM), and on machines that were being used by other processes.

6 RESULTS

6.1 First experiment: 4 GB machines

We begin with a discussion of performance on machines with 4 GB of main memory. Figure 1 shows the running time as a function of the number of cells in the two alignments together, where 'cells' are the nucleotides and gaps; therefore, the number of cells is $n(k_1+k_2)$,

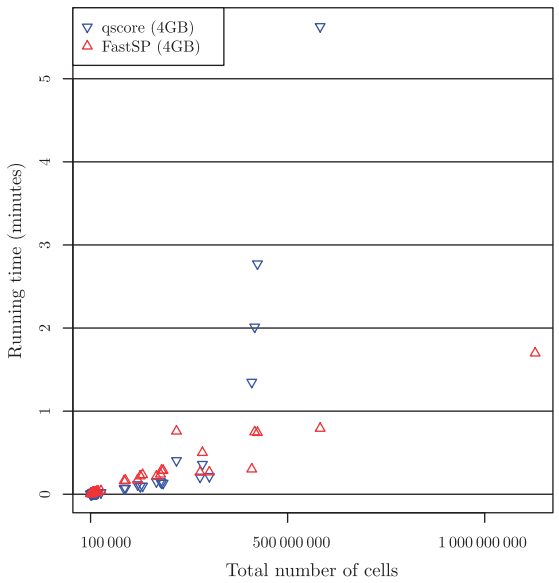

Fig. 1. Log-scaled running time on machines with 4 GB of main memory. qscore is run only in default setting, and so computes only SP- and TC-scores. Note that qscore fails to analyze the largest dataset.

where each alignment is on n sequences and the first alignment has k_1 sites and the second alignment has k_2 sites. Note that qscore failed to score the largest dataset, 16S.GG.50k, with 50 000 sequences, while FastSP successfully completed all analyses. For the remaining datasets, qscore and FastSP were equally fast on the smallest of these datasets, but FastSP was computationally more efficient than qscore on the larger datasets. In particular, on the largest of the datasets on which both methods ran, qscore took >5 min and FastSP finished in <1 min. Averaging over all 27 datasets with more than 10 million cells, and excluding the cases where qscore fails, the running times were 0.54 and 0.25 min and memory usages were 1.38 GB and 502 MB, respectively, for qscore and FastSP.

6.2 Second experiment: 2 GB machines

We then examined the running time on machines with 2 GB of main memory (Fig. 2).

As expected, qscore failed to complete the analysis of the 16S.GG.50k dataset, but it also failed to complete analyses of three pairs of alignments on the 16S.B.ALL dataset, which has 27 643 sequences. In contrast, FastSP completed its analyses of all the datasets under these conditions. Thus, reducing the memory from 4 GB to 2 GB resulted in more failures for qscore. Averaging over all 27 datasets with more than 10 million cells, and excluding the cases where qscore fails, qscore and FastSP took 1.18 and 0.21 min and used 931 MB and 366 MB of memory, respectively. Also, FastSP never used >3 min on any dataset, while qscore used 6 min on one of the larger datasets.

All these results were obtained on 64 bit machines. FastSP, however, can analyze very large datasets on 32 bit desktop machines with even less memory available. To demonstrate this observation, we picked our three largest datasets (16S.GG.50k, 16S.B.ALL/PARTTREE and 16S.B.ALL/SATé) and ran FastSP on those datasets on a desktop of 32 bit 3.16 GHz machine

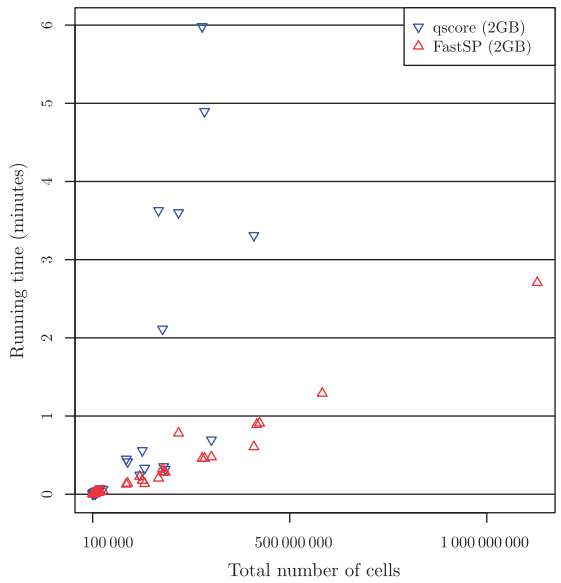

Fig. 2. Running time of QSCORE and FASTSP when limited to 2 GB. QSCORE is run in default mode, and so computes only the TC- and SP-scores. Under these conditions, QSCORE fails to run on some inputs.

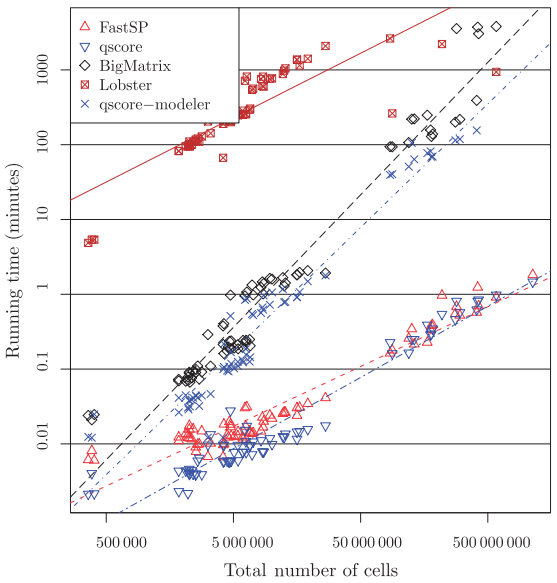

Fig. 3. Log-scaled running time on machines with 'at least' 8 GB of main memory. We show QSCORE run in both the default setting (where it only computes SP and TC) and also to compute the Modeler score. Not all methods succeed in analyzing all datasets.

with only 2 GB of main memory. We also reduced the memory available to JVM by changing JVM's -Xmx option. FASTSP was able to successfully run on the 16S.GG.50k dataset using 1.47 GB of memory in 2.7 min. It also finished in 1.3 min using 865 MB of memory for 16S.B.ALL/PARTTREE, and 0.5 min and 773 MB of memory for 16S.B.ALL/SATé-II. The running times under these low JVM memory conditions are close to the running times reported on the 64 bit cluster machines, and indicate the robustness of FASTSP to conditions with limited memory.

6.3 Third experiment: large memory machines

Our third experiment compared FASTSP to all three other methods (LOBSTER, BIGMATRIX and QSCORE) on machines with 8–32 GB of the main memory (Fig. 3). Although FASTSP and QSCORE succeeded in completing all their analyses, LOBSTER and BIGMATRIX failed on several datasets. In particular, LOBSTER failed to analyze 14 of the largest datasets, including all alignments on 16S.T, Price-78K, 16S.GG.50k, and most alignments on 16S.B.ALL and on 16S.3, while BIGMATRIX failed to analyze the two largest datasets, Price-78K and 16S.GG.50K.

Figure 3 shows the running time, given in log scale, as a function of the total number of cells in both alignments together, as well as the regression line in log space for each method. In these analyses, we show results for QSCORE computing the default scores (SP and TC) and then also computing the Modeler score (indicated by QSCORE-modeler).

LOBSTER was the slowest of the three methods, and BIGMATRIX was clearly much slower than either QSCORE or FASTSP. FASTSP and QSCORE took about the same time, and their running times increased at about the same rate. When QSCORE is run so that it computes the modeler score, it runs much slower, and its running time is not linear.

Figures 4 and 5 provide a direct comparison of QSCORE and FASTSP with respect to running time and peak memory usage,

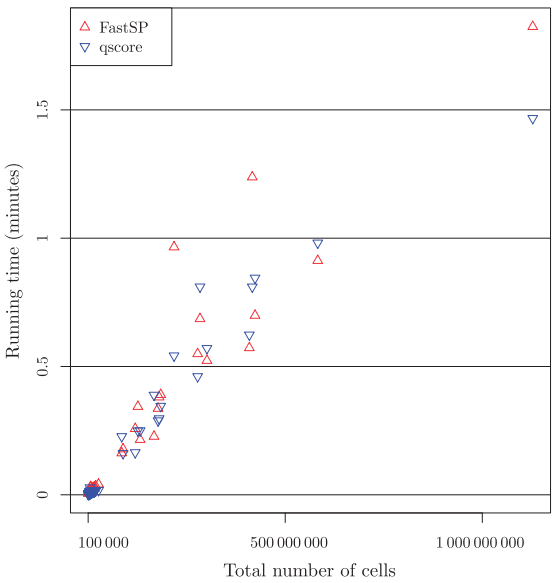

Fig. 4. Comparison of FASTSP and QSCORE-default with respect to running time on machines with 'at least' 8 GB of main memory.

respectively, with QSCORE computing only the default scores (SP and TC). While the running times of FASTSP and QSCORE were close, their peak memory usage differed substantially. These data show that the memory requirement of QSCORE scaled linearly with the total number of cells in both alignments, reaching 10 GB on the largest dataset. In contrast, the memory usage of FASTSP grew more slowly. In the extreme case of the 16S.GG.50k dataset, QSCORE used 10 GB of peak memory, but FASTSP never used more than 2 GB of peak memory.

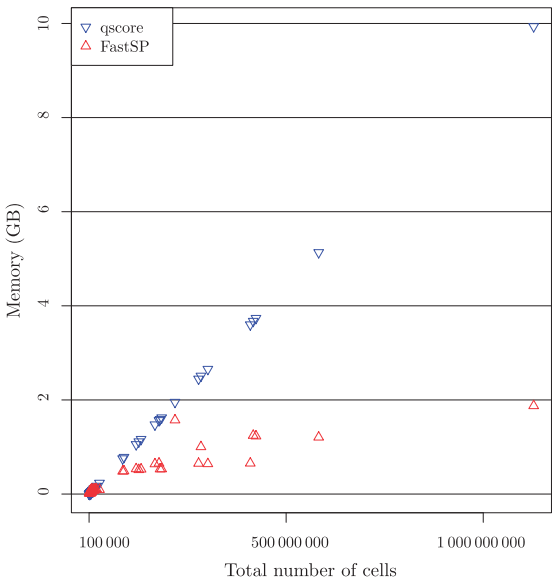

Fig. 5. Comparison of FastSP and QSCORE-default with respect to peak memory usage on machines with 'at least' 8 GB of main memory.

7 DISCUSSION

The main observations we can make are these. When run on machines with sufficient memory (at least 8 GB for these datasets we studied), FastSP and QSCORE have very close running times, but QSCORE has higher peak memory usage. Thus, even though the two methods are not distinguishable by running time in this case, they differ substantially in terms of memory usage. On the other hand, when memory is limited to 4 GB, the two methods have substantially different running times for large alignments. Thus, in general the two methods cannot be distinguished on alignments with small numbers of taxa, but are distinguished on large alignments—either with respect to running time (if memory is limited to 4 GB) or with respect to memory usage (when memory is not limited).

Although the methods have different running times, it is evident that the differences in running time are a result of differences in their memory usage. But, why do we see these differences? Recall that FastSP has memory usage that grows only linearly with the reference alignment size, and does not need to keep the estimated alignment in memory; this can reduce the memory usage substantially. The most likely explanation is a simple one: QSCORE uses more memory simply because it was not implemented with memory usage optimization as one of its objectives.

8 CONCLUSIONS

This article has two main contributions. First, we provide a proof that the number of shared homologies between two alignments can be computed in $O(nk)$ time, where n is the number of sequences and k is the length of the longer alignment. Therefore, the SP-score, Modeler Score and Total Column scores can each be computed in linear time as well. Second, we present FastSP, a new linear-time method to compute these three scores, and we explore its performance on large alignments in comparison to the current best method, QSCORE. The implementation we provide runs efficiently and requires very little memory: even on the very largest alignments with 50 000 sequences and thousands of sites, FastSP finished in <2 min and used less than 2 GB of main memory. The best of the other methods for computing the SP-score is QSCORE. The comparison between FastSP and QSCORE shows that the two methods have roughly the same running time when run on large memory machines with 8–32 GB of main memory [thus suggesting that QSCORE also runs in $O(nk)$ time], but QSCORE uses much more memory than FastSP on the largest dataset. In addition, when run on lower memory machines (e.g. on machines with only 4 GB of memory), QSCORE will fail to analyze some datasets, and will require more time than FastSP to analyze the largest datasets.

Our results show that the memory usage of QSCORE when comparing pairs of large alignments can be quite large, using at least a few gigabytes (and in one case, using 10 GB) of peak memory, whereas FastSP never used more than 2 GB of peak memory. While this level of memory usage may not have a substantial impact for a single comparison, when several pairwise comparisons are desired, either very large memory machines or sequential pairwise analyses will be needed. In contrast, many pairwise comparisons can be run using FastSP on a single machine.

There are several applications where many pairwise comparisons between alignments would be made. One obvious application is evaluating sequence alignment methods [a problem that is very important for many applications (Aniba *et al.*, 2010)]. In addition, the phylogenetics research community is increasingly aware of the importance of considering many different alignments (Kemena and Notredame, 2009) and the impact of 'alignment uncertainty' on phylogenetic estimation (Wong *et al.*, 2008). In addition, rather than using one alignment technique, current molecular phylogenetics studies often explore a number of different alignments for each dataset. These alignments can be obtained directly using methods such as SATé or BAliPhy (Suchard and Redelings, 2006) that explore alignment space, or using many different alignment methods. Once the set of alignments is obtained, they can be used to estimate an alignment [as in TCoffee (Poirot *et al.*, 2003) and MCoffee (Moretti *et al.*, 2007)], to produce a consensus alignment (Prasad *et al.*, 2003, 2004) or to evaluate the support for each homology within an alignment (Kim and Ma, 2011; Landan and Graur, 2008). These meta-analyses can then be used to improve biological inferences, such as predicting function (Satija *et al.*, 2009). Alignments can also be compared with each other in order to train alignment estimation methods so that they produce more accurate alignments (Lee *et al.*, 2007).

Thus, real-world applications exist in which many pairwise comparisons between alignments are made. Furthermore, large phylogenetic analyses are becoming the norm, and datasets with tens of thousands of taxa (such as we studied in this article) are being analyzed [e.g. Goloboff *et al.* (2009); Smith *et al.* (2009)]. Therefore, methods, such as FastSP that can compare alignments in a time- and memory-efficient manner, are bioinformatics tools that are likely to have increasing importance for future phylogenetic analyses.

Future work will seek to integrate FastSP into other software, such as alignment visualization tools or methods that annotate alignments using a set of alignments.

ACKNOWLEDGEMENTS

The SATé software used in this study depends upon Dendropy (Sukumaran and Holder, 2010). We thank Robert Edgar for

assistance in using LOBSTER and QSCORE, and Valerie King and Bernard Moret for discussions regarding the algorithmic approach. We also thank the anonymous reviewers for their helpful suggestions.

Funding: US National Science Foundation (Grant No. DEB0733029 to S.M. and T.W.); John P. Simon Guggenheim Foundation; Faculty Research Assignment award from the University of Texas; David Bruton Jr Centennial Professorship in Computer Science (to T.W.). US National Science Foundation; Graduate Fellowship from NSERC (to S.M.).

Conflict of Interest: none declared.

REFERENCES

Aniba,M. *et al.* (2010) Issues in bioinformatics benchmarking: the case study of multiple sequence alignment. *Nucleic Acids Res.*, **38**, 7353–7363.

Blażewicz,J. *et al.* (2009) Some remarks on evaluating the quality of the multiple sequence alignment based on the BAliBASE benchmark. *Int. J. Appl. Math. Comput. Sci.*, **19**, 675–678.

Cline,M. *et al.* (2002) Predicting reliable regions in protein sequence alignments. *Bioinformatics*, **18**, 306–314.

DeSantis,T.Z. *et al.* (2006) Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl. Environ. Microbiol.*, **72**, 5069–5072.

Edgar,R.C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, **32**, 1792–1797.

Goloboff,P. *et al.* (2009) Phylogenetic analysis of 73,060 taxa corroborates major eukaryotic groups. *Cladistics*, **25**, 211–230.

Katoh,K. and Toh,H. (2007) PartTree: an algorithm to build an approximate tree from a large number of unaligned sequences. *Bioinformatics*, **23**, 372–374.

Katoh,K. and Toh,H. (2008) Recent developments in the MAFFT multiple sequence alignment program. *Brief. Bioinformatics*, **9**, 286–298.

Kemena,C. and Notredame,C. (2009) Upcoming challenges for multiple sequence alignment methods in the high-throughput era. *Bioinformatics*, **25**, 2455–2465.

Kim,J. and Ma,J. (2011) PSAR: measuring multiple sequence alignment reliability by probabilistic sampling. *Nucleic Acids Res.*,**39**, 6359–6368.

Landan,G. and Graur,D. (2008) Local reliability measures from sets of co-optimal multiple sequence aligments. *Proc. Pac. Symp. Biocomput.*, **13**, 15–24.

Lee,M. *et al.* (2007) Predicting and improving the protein sequence alignment quality by support vector regression. *BMC Bioinformatics*, **8**, 471.

Litzkow,M. *et al.* (1988) Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*. IEEE Computer Society.

Liu,K. *et al.* (2009) Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science*, **324**, 1561–1564.

Liu,K. *et al.* (2010) Multiple sequence alignment: a major challenge to large-scale phylogenetics. *PLoS Curr. Tree Life*, **2**, RRN1198.

Liu,K. *et al.* (2011) SATé-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Syst. Biol*., (in press).

Loytynoja,A. and Goldman,N. (2005) An algorithm for progressive multiple alignment of sequences with insertions. *Proc. Natl Acad. Sci. USA*, **102**, 10557–10562.

Moretti,S. *et al.* (2007) The M-Coffee web server: a meta-method for computing multiple sequence alignments by combining alternative alignment methods. *Nucleic Acids Res.*, **35**, 645–648.

Poirot,O. *et al.* (2003) Tcoffee@igs: a web server for computing, evaluating and combining multiple sequence alignments. *Nucleic Acids Res.*, **31**, 3503–3506.

Prasad,J. *et al.* (2003) Consensus alignment for reliable framework prediction in homology modeling. *Bioinformatics*, **19**, 1682–1691.

Prasad,J. *et al.* (2004) Consensus alignment server for reliable comparative modeling with distant templates. *Nucleic Acids Res.*, **32**, W50–W54.

Price,M.N. *et al.* (2010) FastTree 2: approximately maximum-likelihood trees for large alignments. *PLoS One*, **5**, e9490.

Satija,R. *et al.* (2009) BigFoot: Bayesian alignment and phylogenetic footprinting with MCMC. *BMC Evol. Biol.*, **9**, 217.

Smith,S.A. *et al.* (2009) Mega-phylogeny approach for comparative biology: an alternative to supertree and supermatrix approaches. *BMC Evol. Biol.*, **9**.

Suchard,M.A. and Redelings,B.D. (2006) BAli-Phy: simultaneous Bayesian inference of alignment and phylogeny. *Bioinformatics*, **22**, 2047–2048.

Sukumaran,J. and Holder,M. (2010) DendroPy: a Python library for phylogenetic computing. *Bioinformatics*, **26**, 1569–1571.

Thompson,J.D. *et al.* (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Wheeler,T.J. and Kececioglu,J.D. (2007) Multiple alignment by aligning alignments. *Bioinformatics*, **23**, i559–i568.

Wong,K.M. *et al.* (2008) Alignment uncertainty and genomic analysis. *Science*, **319**, 473–476.