19

Tree Construction using Singular Value Decomposition

Nicholas Eriksson

We present a new, statistically consistent algorithm for phylogenetic tree construction that uses the algebraic theory of statistical models (as developed in Chapters 1 and 3). Our basic tool is *Singular Value Decomposition* (SVD) from numerical linear algebra.

Starting with an alignment of n DNA sequences, we show that SVD allows us to quickly decide whether a split of the taxa occurs in their phylogenetic tree, assuming only that evolution follows a tree Markov model. Using this fact, we have developed an algorithm to construct a phylogenetic tree by computing only $O(n^2)$ SVDs.

We have implemented this algorithm using the `SVDLIBC` library (available at `http://tedlab.mit.edu/~dr/SVDLIBC/`) and have done extensive testing with simulated and real data. The algorithm is fast in practice on trees with 20–30 taxa.

We begin by describing the general Markov model and then show how to flatten the joint probability distribution along a partition of the leaves in the tree. We give rank conditions for the resulting matrix; most notably, we give a set of new rank conditions that are satisfied by non-splits in the tree. Armed with these rank conditions, we present the tree-building algorithm, using SVD to calculate how close a matrix is to a certain rank. Finally, we give experimental results on the behavior of the algorithm with both simulated and real-life (ENCODE) data.

19.1 The general Markov model

We assume that evolution follows a tree Markov model, as introduced in Section 1.4, with evolution acting independently at different sites of the genome. We do not assume that the transition matrices for the model are stochastic. Furthermore, we do not assume the existence of a global rate matrix (as in Section 4.5).

This model is called the general Markov model. It is a more general model than any in the Felsenstein hierarchy (Figure 4.7). The main results in this chapter therefore hold no matter what model in the Felsenstein hierarchy one works with.

Under the general dogma that statistical models are algebraic varieties, the polynomials (called "phylogenetic invariants") defining the varieties are of great interest. Phylogenetic invariants have been studied extensively since [Lake, 1987, Cavender and Felsenstein, 1987]. Linear invariants for the Jukes–Cantor model have been used to infer phylogenies on four and five taxa; see [Sankoff and Blanchette, 2000]. Sturmfels and Sullivant finished the classification of the invariants for group-based models [Sturmfels and Sullivant, 2005]; see Chapter 15 for an application of these invariants for constructing trees on four taxa. Invariants for the general Markov model have been studied in [Allman and Rhodes, 2003, Allman and Rhodes, 2004a].

The main problem with invariants is that there are exponentially many polynomials in exponentially many variables to test on exponentially many trees. Because of this, they are currently considered impractical by many and have only been applied to small problems. However, we solve the problem of this combinatorial explosion by only concentrating on invariants which are given by rank conditions on certain matrices, called "flattenings".

19.2 Flattenings and rank conditions

Recall from Chapters 2 and 17 that a *split* $\{A, B\}$ in a tree is a partition of the leaves obtained by removing an edge of the tree. We will say that $\{A, B\}$ is a *partition* of the set of leaves if it is not necessarily a split but merely a disjoint partition of the set of leaves into two sets.

Throughout, all trees will be assumed to be binary with n leaves. We let m denote the number of states in the alphabet Σ. Usually $m = 4$ and $\Sigma = \{A, C, G, T\}$ or $m = 2$ and $\Sigma = \{0, 1\}$. We will write the joint probabilities of an observation on the leaves as $p_{i_1 \ldots i_n}$. That is, $p_{i_1 \ldots i_n}$ is the probability that leaf j is observed to be in state i_j for all $j \in \{1, \ldots, n\}$. We write P for the entire probability distribution.

Although the descriptions of tree-based models in this book all deal with rooted trees, we will mostly consider unrooted tree models, which are equivalent to them for the general Markov model; see [Allman and Rhodes, 2004a] for details on this technical point. Our tree-building algorithm constructs an unrooted tree, additional methods would be required to find the root.

Definition 19.1 A *flattening* along a partition $\{A, B\}$ is the $m^{|A|}$ by $m^{|B|}$ matrix where the rows are indexed by the possible states for the leaves in A and the columns are indexed by the possible states for the leaves in B. The entries of this matrix are given by the joint probabilities of observing the given pattern at the leaves. We write $\mathrm{Flat}_{A,B}(P)$ for this matrix.

Example 19.2 (Flattening a partition on 4 taxa) Let T be a tree with 4 leaves and let $m = 4$, $\Sigma = \{\mathtt{A},\mathtt{C},\mathtt{G},\mathtt{T}\}$. The partition $\{1, 3\}, \{2, 4\}$ flattens to the 16×16 matrix $\mathrm{Flat}_{\{1,3\},\{2,4\}}(P)$ where the rows are indexed by bases of

taxa 1 and 3 and the columns by bases of taxa 2 and 4:

$$\mathrm{Flat}_{\{1,3\},\{2,4\}}(P) = \begin{array}{c} \\ \mathtt{AA} \\ \mathtt{AC} \\ \mathtt{AG} \\ \mathtt{AT} \\ \mathtt{CA} \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathtt{AA} & \mathtt{AC} & \mathtt{AG} & \mathtt{AT} & \mathtt{CA} & \mathtt{CC} & \dots \end{array} \\ \begin{pmatrix} p_{\mathtt{AAAA}} & p_{\mathtt{AAAC}} & p_{\mathtt{AAAG}} & p_{\mathtt{AAAT}} & p_{\mathtt{ACAA}} & p_{\mathtt{ACAC}} & \cdots \\ p_{\mathtt{AACA}} & p_{\mathtt{AACC}} & p_{\mathtt{AACG}} & p_{\mathtt{AACT}} & p_{\mathtt{ACCA}} & p_{\mathtt{ACCC}} & \cdots \\ p_{\mathtt{AAGA}} & p_{\mathtt{AAGC}} & p_{\mathtt{AAGG}} & p_{\mathtt{AAGT}} & p_{\mathtt{ACGA}} & p_{\mathtt{ACGC}} & \cdots \\ p_{\mathtt{AATA}} & p_{\mathtt{AATC}} & p_{\mathtt{AATG}} & p_{\mathtt{AATT}} & p_{\mathtt{ACTA}} & p_{\mathtt{ACTC}} & \cdots \\ p_{\mathtt{CAAA}} & p_{\mathtt{CAAC}} & p_{\mathtt{CAAG}} & p_{\mathtt{CAAT}} & p_{\mathtt{CCAA}} & p_{\mathtt{CCAC}} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \end{array}.$$

□

Next we define a measure of how close a general partition of the leaves is to being a split. If A is a subset of the leaves of T, we let T_A be the subtree induced by the leaves in A (in Chapter 18 this subtree is denoted by $[A]$). That is, T_A is the minimal set of edges needed to connect the leaves in A.

Definition 19.3 Suppose that $\{A, B\}$ is a partition of $[n]$. The *distance* between the partition $\{A, B\}$ and the nearest split, written $e(A, B)$, is the number of edges that occur in $T_A \cap T_B$.

Notice that $e(A, B) = 0$ exactly when $\{A, B\}$ is a split.

Consider $T_A \cap T_B$ as a subtree of T_A. Color the nodes in $T_A \cap T_B$ red, the nodes in $T_A \setminus (T_A \cap T_B)$ blue. Say that a node is *monochromatic* if it and all of its neighbors are of the same color. We let $\mathrm{mono}(A)$ be the number of monochromatic red nodes. That is:

Definition 19.4 Define $\mathrm{mono}(A)$ as the number of nodes in $T_A \cap T_B$ that do not have a node in $T_A \setminus (T_A \cap T_B)$ as a neighbor.

See Figure 19.1 for an example of $e(A, B)$ and $\mathrm{mono}(A)$.

Our main theorem ties together how close a partition is to being a split with the rank of the flattening associated to that partition.

Theorem 19.5 *Let $\{A, B\}$ be a partition of $[n]$, let T be a binary, unrooted tree with leaves labeled by $[n]$, and assume that the joint probability distribution P comes from a Markov model on T with an alphabet with m letters. Then the generic rank of the flattening* $\mathrm{Flat}_{A,B}(P)$ *is given by*

$$\min\left(m^{e(A,B)+1-\mathrm{mono}(A)}, m^{e(A,B)+1-\mathrm{mono}(B)}, m^{|A|}, m^{|B|}\right). \tag{19.1}$$

Proof We claim that $\mathrm{Flat}_{A,B}(P)$ can be thought of as the joint distribution for a simple graphical model. Pick all the nodes that are shared by the induced subtrees for A and B: call this set R. If R is empty, then $\{A, B\}$ is a split; in that case let R be one of the vertices of the edge separating A and B. Notice that $|R| = e(A, B) + 1$. Think of these vertices as a single hidden random variable which we will also call R with $m^{|R|} = m^{e(A,B)+1}$ states. Group the states of the nodes in A together into one $m^{|A|}$-state observed random variable;

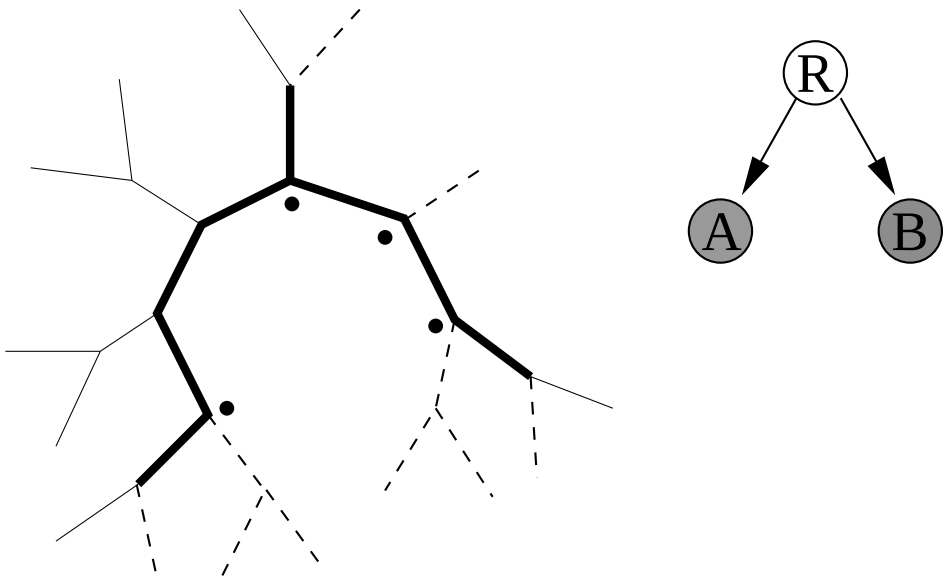

Fig. 19.1. If A is given by the 8 dashed leaves and B by the 7 solid leaves, then $e(A, B) = 8$ (shown in bold) and $\mathrm{Flat}_{A,B}(P)$ is the joint distribution for a 3-state graphical model where the root R has m^9 states and the descendents A and B have m^8 and m^7 states, respectively. Here $\mathrm{mono}(B) = 4$ (indicated by the dots), so the $m^9 \times m^8$ matrix M_A has rank $m^{9-4} = m^5$, which is the rank of $\mathrm{Flat}_{A,B}(P)$.

similarly the nodes in B are grouped into a $m^{|B|}$-state random variable. Then create the graphical model with one hidden $m^{|R|}$-state random variable and two descendent observed variables with $m^{|A|}$ and $m^{|B|}$ states. Notice that $\mathrm{Flat}_{A,B}(P)$ is the joint distribution for this model. See Figure 19.1 for an example.

Furthermore, the distribution for this simplified model factors as

$$\mathrm{Flat}_{A,B}(P) = M_A^{\mathrm{T}} \,\mathrm{diag}(\pi(R)) M_B \tag{19.2}$$

where $\pi(R)$ is the distribution of R and M_A and M_B are the $m^{|R|} \times m^{|A|}$ and $m^{|R|} \times m^{|B|}$ transition matrices. That is, the (i, j)th entry of M_A is the probability of transitioning from state i at the root R to state j at A.

To say the tree distribution factors as (19.2) just means that

$$\mathrm{Prob}(A = i, B = j) = \sum_k \mathrm{Prob}(R = k)\mathrm{Prob}(A = i \mid R = k)\mathrm{Prob}(B = j \mid R = k).$$

Notice that all of the terms in this expression can be written as polynomials in the edge parameters (after choosing a rooting). Therefore the rank of $\mathrm{Flat}_{A,B}(P)$ is at most $m^{\min(|R|,|A|,|B|)}$.

However, the matrices in this factorization do not necessarily have full rank. For example, if one of the nodes in R has only neighbors that are also in R, then the $m^{|R|} \times m^{|A|}$ transition matrices from R to A have many rows that are the same, since the transition from a state of R to a state of A does not

depend on the value of this one node. More generally, if a node of R has no neighbors in $T_A \setminus (T_A \cap T_B)$, then the entries of the transition matrix M_A do not depend on the value of this node. But the entries do depend on the values of all other nodes of R (that is, those with neighbors in $T_A \setminus (T_A \cap T_B)$). So R really behaves like a model with $m^{|R|-\text{mono}(A)}$ states on the transition to A and $m^{|R|-\text{mono}(B)}$ states for the transition to B. There are enough parameters so that after canceling out these equal rows, all other rows are linearly independent. Therefore, the rank of M_A is $\min\left(m^{|R|-\text{mono}(A)}, m^{|A|}\right)$ (and similarly for M_B), so the theorem follows. □

This theorem gives rise to a well-known corollary upon noticing that if $\{A, B\}$ is a split, then $e(A, B) = 0$ (see [Allman and Rhodes, 2004a], for example).

Corollary 19.6 *If $\{A, B\}$ is a split in the tree, the generic rank of $\text{Flat}_{A,B}(P)$ is m.*

A partial converse of Corollary 19.6 will be used later.

Corollary 19.7 *If $\{A, B\}$ is not a split in the tree, and we have $|A|, |B| \geq 2$ then the generic rank of $\text{Flat}_{A,B}(P)$ is at least m^2.*

Proof Since we have $|A|, |B| \geq 2$, we must show that the two other exponents in (19.1) are at least 2. That is, we have to show that $e(A, B)+1-\text{mono}(A) \geq 2$ (the case for B is symmetric). This term counts the number of nodes in $T_A \cap T_B$ that are directly connected to a part of T_A outside of $T_A \cap T_B$. Since $\{A, B\}$ is not a split, we know that $|T_A \cap T_B| = e(A, B) + 1 \geq 2$. Consider $T_A \cap T_B$ as a subtree with at least 2 nodes of T_A. The only way for all but one of these nodes to be isolated from the rest of the tree is to have the two consist of a leaf and its parent. However, this is impossible since $\{A, B\}$ is a disjoint partition of the set of leaves, so $T_A \cap T_B$ contains no leaves. □

Example 19.8 In Example 19.2, the 16×16 matrix $\text{Flat}_{\{1,3\},\{2,4\}}(P)$ has rank 4 if the split $\{\{1, 3\}, \{2, 4\}\}$ occurs in the tree, otherwise, it has rank 16. □

In fact, if $m = 2$, it has recently been shown [Allman and Rhodes, 2004b] that the rank conditions in Corollary 19.6 generate the ideal of invariants for the general Markov model. However, they do not suffice if $m = 4$, since in that case a polynomial of degree 9 lies in the ideal of invariants (see [Strassen, 1983, Garcia *et al.*, 2004]) but this polynomial is not generated by the degree 5 rank conditions (see [Landsberg and Manivel, 2004]).

19.3 Singular Value Decomposition

Singular Value Decomposition provides a method to compute the distance between a matrix and the nearest rank k matrix. In this section, we briefly introduce the basic properties of SVD for real matrices. See [Demmel, 1997] for a thorough treatment.

Definition 19.9 A *singular value decomposition* of a $m \times n$ matrix A (with $m \geq n$) is a factorization $A = U\Sigma V^{\mathrm{T}}$ where U is $m \times n$ and satisfies $U^{\mathrm{T}}U = I$, V is $n \times n$ and satisfies $V^{\mathrm{T}}V = I$ and $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are called the *singular values* of A.

Definition 19.10 Let a_{ij} be the (i, j)th entry of A. The *Frobenius norm*, written $\|A\|_{\mathrm{F}}$, is the root-sum-of-squares norm on $\mathbb{R}^{m \cdot n}$. That is,

$$\|A\|_{\mathrm{F}} = \sqrt{\sum a_{ij}^2}.$$

The L_2 *norm* (or operator norm), written $\|A\|_2$, is given by

$$\|A\|_2 = \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \left\{ \frac{\|Ax\|}{\|x\|} \right\},$$

where $\|x\|$ is the usual root-sum-of-squares vector norm.

The following is Theorem 3.3 of [Demmel, 1997]:

Theorem 19.11 *The distance from A to the nearest rank k matrix is*

$$\min_{\mathrm{Rank}(B)=k} \|A - B\|_{\mathrm{F}} = \sqrt{\sum_{i=k+1}^{m} \sigma_i^2}$$

in the Frobenius norm and

$$\min_{\mathrm{Rank}(B)=k} \|A - B\|_2 = \sigma_{k+1}$$

in the L_2 norm.

One way of computing the singular values is to compute the eigenvalues of $A^{\mathrm{T}}A$; the singular values are the square roots of these eigenvalues. Therefore, general techniques for solving the real symmetric eigenvalue problem can be used to compute the SVD. These various methods, both iterative and direct, are implemented by many software packages for either sparse or general matrices. We will discuss the computational issues with SVD after we describe how to use it to construct phylogenetic trees.

19.4 Tree-construction algorithm

Now that we know how to tell how close a matrix is to being of a certain rank, we can test whether a given split comes from the underlying tree or not by using the SVD to tell how close a flattening matrix is to being rank m. However, since there are exponentially many possible splits, we must carefully search through this space. Following a suggestion by S. Snir, we do this by building the tree bottom up, at each step joining cherries together, in a method reminiscent of neighbor-joining (Algorithm 2.41).

It is an interesting open question whether the additional information in Theorem 19.5 about non-splits that are almost splits can be harnessed to produce an improved algorithm.

Algorithm 19.12 (Tree construction with SVD)
Input: A multiple alignment of genomic data from n species, from the alphabet Σ with m states.
Ouptut: An unrooted binary tree with n leaves labeled by the species.

Initialization: Compute empirical probabilities $p_{i_1 \ldots i_n}$. That is, count occurrences of each possible column of the alignment, ignoring columns with characters not in Σ. Store the results in a sparse format.

Loop: For k from n down to 4, perform the following steps.

For each of the $\binom{k}{2}$ pairs of species compute the SVD for the split $\{\{\text{pair}\}, \{\text{other } k-2 \text{ species}\}\}$. Pick the pair whose flattening is closest to rank m according to the Frobenius norm and join this pair together in the tree. That is, consider this pair as a single element when picking pairs at the next step.

Proposition 19.13 *Algorithm 19.12 needs the computation of at most* $(n-1)^2 - 3$ *SVDs.*

Proof At the first step, we compute an SVD $\binom{n}{2}$ times. At each subsequent step, we only need to compute those splits involving the pair that we just joined together. Thus we compute $(n-2)+(n-3)+\cdots+3 = \binom{n-1}{2} - 3$ total SVDs after the first step for $\binom{n}{2} + \binom{n-1}{2} = (n-1)^2 - 3$ SVD computations in total. In fact, not all of these are even necessary; some steps will involve computing both partitions $\{A, B\}$ and $\{B, A\}$, in which case one can be ignored. □

The flattenings are very large (size $m^{|A|} \times m^{|B|}$), yet they are typically very sparse. If an alignment is of length L, at most L entries of the flattening, typically many fewer, are non-zero. Generally, computing all singular values of an $a \times b$ matrix takes $O(a^2 b + a b^2)$ time. However, Lanczos iterative methods (cf. Chapter 7 of [Demmel, 1997]) allow singular values to be computed quickly individually, starting with the largest. Furthermore, sparse matrix techniques allow us to take advantage of this structure without having to deal with matrices of exponential size.

Since we will be comparing the SVD from different sized splits, we need to compute distances in the Frobenius norm, which does not change as the dimensions of the matrices change (as long as the number of entries is constant). This means that we should compute all singular values; however that is difficult computationally. But in practice, the singular values typically decrease very quickly, so it suffices to compute only the largest singular values to estimate the Frobenius norm.

By exploiting the sparsity and only computing singular values until they become sufficiently small, we find that we are able to very quickly compute the SVD for flattenings coming from trees with at most 31 leaves with binary data ($m = 2$) and up to 15 leaves with DNA data ($m = 4$). This limitation is due to limits on the size of array indices in `SVDLIBC` and can probably be exceeded. Furthermore, there are good approximation algorithms for SVD that could make very large problems practical [Frieze *et al.*, 1998].

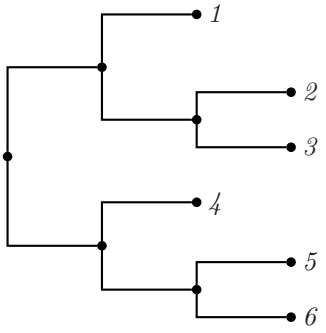

Fig. 19.2. The 6-taxa tree constructed in Example 19.15.

Theorem 19.14 *Algorithm 19.12 is statistically consistent. That is, as the probability distribution converges to a distribution that comes from the general Markov model on a binary tree T, the probability that Algorithm 19.12 outputs T goes to 1.*

Proof We must show that the algorithm picks a correct split at each step; that is, as the empirical distribution approaches the true distribution, the probability of choosing a bad split goes to zero. By Corollary 19.6, we see that a true split will lead to a flattening that approaches a rank m matrix, while Corollary 19.7 shows that other partitions will approach a matrix of rank at least m^2 (except for partitions where one set contains only one element; however, these are never considered in the algorithm). Therefore, as the empirical distribution approaches the true one, the distance of a split from rank m will go to zero while the distance from rank m of a non-split will not. □

Example 19.15 We begin with an alignment of DNA data of length 1000 for 6 species, labeled $1, \ldots, 6$, simulated from the tree in Figure 19.2 with all branch lengths equal to 0.1. For the first step, we look at all pairs of the 6 species. The score column is the distance in the Frobenius norm from the flattening to the nearest rank 4 matrix:

```
Partition        Score
2 3 | 1 4 5 6    5.8374
5 6 | 1 2 3 4    6.5292
1 2 | 3 4 5 6    20.4385
1 3 | 2 4 5 6    20.5153
4 6 | 1 2 3 5    23.1477
4 5 | 1 2 3 6    23.3001
1 4 | 2 3 5 6    44.9313
3 4 | 1 2 5 6    52.1283
2 4 | 1 3 5 6    52.6763
1 6 | 2 3 4 5    52.9438
1 5 | 2 3 4 6    53.1727
```

```
3 6 | 1 2 4 5   59.5006
3 5 | 1 2 4 6   59.7909
2 6 | 1 3 4 5   59.9546
2 5 | 1 3 4 6   60.3253
picked split 1 4 5 6 | 2 3
tree is 1 4 5 6 (2,3)
```

After the first step, we see that the split $\{\{2,3\},\{1,4,5,6\}\}$ is the best, so we join nodes 2 and 3 together in the tree and continue. Notice that the scores of the partitions roughly correspond to how close they are to being splits:

```
Partition       Score
1 2 3 | 4 5 6   5.8534
5 6 | 1 2 3 4   6.5292
4 6 | 1 2 3 5   23.1477
4 5 | 1 2 3 6   23.3001
1 4 | 2 3 5 6   44.9313
2 3 4 | 1 5 6   45.1427
1 6 | 2 3 4 5   52.9438
2 3 6 | 1 4 5   53.0300
1 5 | 2 3 4 6   53.1727
2 3 5 | 1 4 6   53.3838
picked split 1 2 3 | 4 5 6
tree is 4 5 6 (1,(2,3))
```

After the second step, we join node 1 to the $\{2,3\}$ cherry and continue:

```
Partition       Score
5 6 | 1 2 3 4   6.5292
4 6 | 1 2 3 5   23.1477
4 5 | 1 2 3 6   23.3001
picked split 1 2 3 4 | 5 6
tree is 4 (1,(2,3)) (5,6)

Final tree is (4,(1,(2,3)),(5,6))
```

We have found the last cherry, leaving us with 3 remaining groups which we join together to form an unrooted tree. □

19.5 Performance analysis

19.5.1 Building trees with simulated data

The idea of simulation is that we first pick a tree and simulate a model on that tree to obtain aligned sequence data. Then we build a tree using Algorithm 19.12 and other methods from that data and compare the answers to the original tree.

We used the program `seq-gen` [Rambaut and Grassly, 1997] to simulate data of various lengths for the tree in Figure 19.3 with the two sets of branch

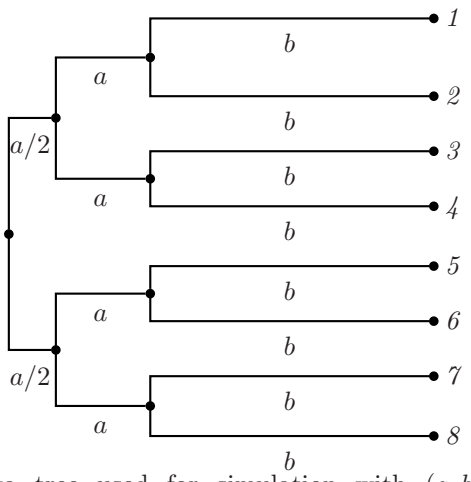

Fig. 19.3. The 8-taxa tree used for simulation with $(a, b) = (0.01, 0.07)$ and $(0.02, 0.19)$.

lengths given in Figure 19.3. This tree was chosen as a particularly difficult tree [Strimmer and von Haeseler, 1996, Ota and Li, 2000].

We simulated DNA data under the general reversible model (the most general model supported by `seq-gen`). Random numbers uniformly distributed between 1 and 2 were chosen on each run for the six rate matrix parameters (see Figure 4.7). The root frequencies were all set to 1/4.

Next, the data was collapsed to binary data (that is, `A` and `G` were identified, similarly `C` and `T`). We used binary data instead of DNA data because of numerical instability with SVD using the much larger matrices from the DNA data. It should be noted that Algorithm 19.12 performed better on binary data than on DNA data. This may be due to the instability, but it may also be because the rank conditions define the entire ideal for binary data.

We ran all tests using our Algorithm 19.12 as well as two algorithms from the `PHYLIP` package (see Section 2.5): neighbor-joining (i.e., Algorithm 2.41), and a maximum likelihood algorithm (`dnaml`). We used Jukes–Cantor distance estimation for neighbor-joining and the default settings for `dnaml`. All three algorithms took approximately the same amount of time, except for `dnaml`, which slowed down considerably for long sequences.

Figures 19.4 and 19.5 show the results of the simulations. Each algorithm was run 1000 times for each tree and sequence length. While SVD performed slightly worse than the others, it showed very comparable behavior. It should be noted that SVD constructs trees according to a much more general model than the other two methods, so it should be expected to have a higher variance.

19.5.2 Building trees with real data

For data, we use the October 2004 freeze of the ENCODE alignments. For detailed information on these, see Section 4.3, Chapters 21 and 22.

As in Chapter 21, we restricted our attention to 8 species: human, chimp,

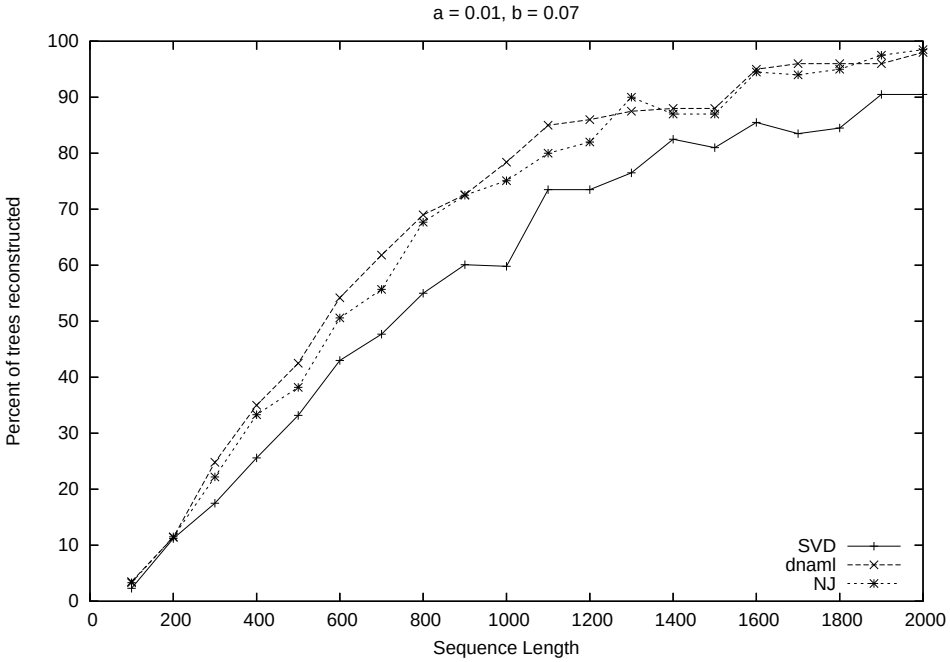

Fig. 19.4. Percentage of trees reconstructed correctly (for the 8-taxa tree with branch lengths $(a, b) = (0.01, 0.07)$) using our SVD algorithm and two `PHYLIP` packages.

galago, mouse, rat, cow, dog, and chicken. We processed each of the 44 ENCODE regions to obtain 3 data sets. First, for each region, all of the ungapped columns were chosen. Second, within each region, all ungapped columns that corresponded to RefSeq annotated human genes were chosen. Third, we restricted even further to only the human exons within the genes. Bins without all 8 species and bins with less than 100 ungapped positions in the desired class were removed from consideration. This left us with 33 regions for the entire alignment, and 28 for both the gene and exon regions, of lengths between 302 and over 100000 base pairs. See Chapter 21 for a more thorough discussion of these data sets.

As is discussed in Section 21.4, tree construction methods that use genomic data usually misplace the rodents on the tree. The reasons for this are not entirely known, but it could be because tree construction methods generally assume the existence of a global rate matrix (cf. Section 4.5) for all the species. However, rat and mouse have mutated faster than the other species. Our method does not assume anything about the rate matrix and thus is promising for situations where additional assumptions beyond the Markov process of evolution at independent sites are not feasible.

In fact, Table 19.1 shows that our algorithm performs better than `dnaml` on the ENCODE data sets. Note that the measure used is the *symmetric distance* on trees, which counts the number of splits present in one tree that aren't present in the other.

While neither algorithm constructed the correct tree a majority of the time,

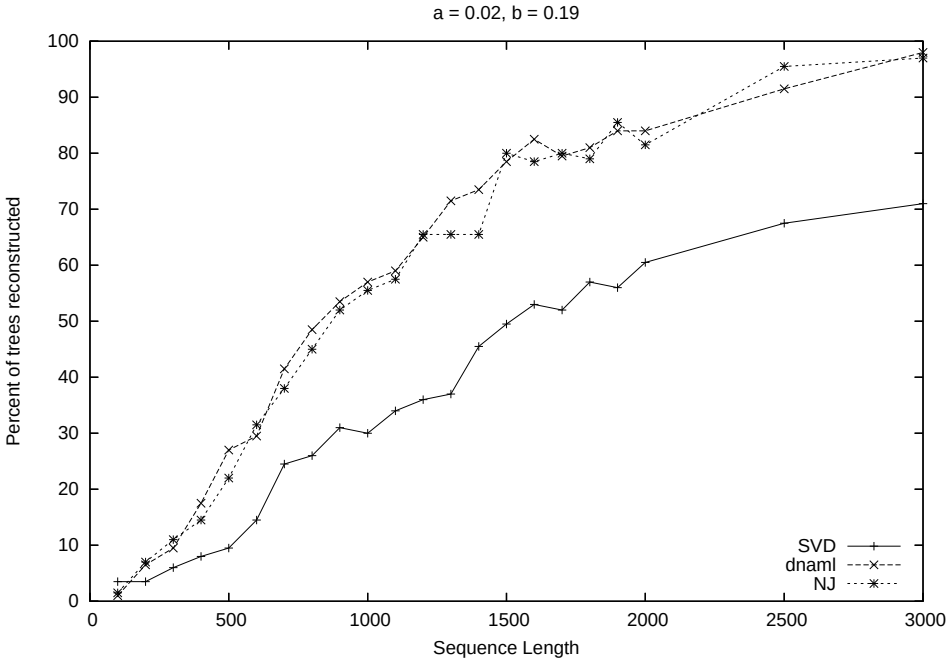

Fig. 19.5. Percentage of trees reconstructed correctly (for the 8-taxa tree with branch lengths $(a, b) = (0.02, 0.019)$) using our SVD algorithm and two PHYLIP packages.

	SVD		dnaml	
	Ave. distance	% correct	Ave. distance	% correct
All	2.06	5.8	3.29	2.9
Gene	1.93	10.3	3.21	0.0
Exon	2.43	21.4	3.0	3.5

Table 19.1. *Comparison of the SVD algorithm and* `dnaml` *on data from the ENCODE project. Distance between trees is given by the symmetric distance, % correct gives the percentage of the regions which had the correct tree reconstructed.*

the SVD algorithm came much closer on average and constructed the correct tree much more often than `dnaml`, which almost never did (see Figure 21.4 for the correct tree and a common mistake).