THE UNIVERSITY *of* EDINBURGH

Simulated Annealing in the Search for Phylogenetic Trees

Daniel Barker

PhD, University of Edinburgh, 1999

Acknowledgements

This PhD was supervised by Andrew Coulson (University of Edinburgh Institute of Cell and Molecular Biology), Quentin Cronk (University of Edinburgh Institute of Cell and Molecular Biology and Royal Botanic Garden Edinburgh) and Mark Watson (Royal Botanic Garden Edinburgh), and funded for three years by the Biotechnology and Biological Sciences Research Council.

Drafts of parts of the thesis were read by the following: Jörn Scharlemann, Andrew Coulson, Jon Bennet, Mark Watson, Quentin Cronk, Toby Pennington, John Pannell, Mukund Unavane, Jill Harrison and Rob Andrews. I am grateful for their many useful suggestions.

Mark Chase supplied his 500-sequence *rbc*L data matrix, and a bracketed-text version of the tree used to prepare Figure 3.1. Susumu Nakayama supplied the details of his work described in Section 6.4. Sub-programs for generating pseudorandom numbers and initialising the generator were supplied by the Edinburgh Parallel Computing Centre (Section G.6), and advice on the subject was provided by Stephen Booth. Tetsuo Amaya suggested the values for the astrophysical numbers used in Chapter 2.

Abstract

I investigate use of the simulated annealing heuristic to seek phylogenetic trees judged optimal according to the principle of parsimony. I begin by looking into the central data structure in phylogenetic research, the tree. I discuss why it is usually necessary to employ a heuristic, rather than an exact method, when seeking parsimonious trees. I summarise different heuristic approaches. I explain how to use the program LVB, written to use simulated annealing in the search for parsimonious trees. I use LVB, with different combinations of values for parameters controlling the annealing search, to re-analyse two DNA sequence data matrices, one of 50 objects and one of 365 objects. Equations to estimate suitable control parameters, on the basis of desired run time and quality of result, are fitted to data obtained by these analyses. Future directions of research are discussed.

Contents

Chapter 1

Introduction

This thesis is about the search for phylogeny. Ironically, many terms in systematics have various, conflicting meanings. Phylogeny is sometimes used to mean the pattern of evolutionary relationships ('a phylogeny is presented in Figure ...') and sometimes used to mean the history of some biological entity ('ontogeny recapitulates phylogeny'). The latter sense is of greatest interest to the evolutionary biologist, but is often so hidden in the past that it cannot be proposed with much certainty. Reconstructing the pattern of relationships is also speculative, since it relies on hypothetical ancestors, but less so. To an extent, reconstructing patterns of relationship must precede proposals of historical processes. I will use phylogeny to mean 'pattern of evolutionary relationships' in this thesis.

To summarise phylogeny in this sense, the following questions are raised:

1. Which data structure best represents this pattern? Data structures range from simple, such as *bit*, to more complicated, such as *string*, *set*, *matrix* and *tree*.

2. How is an instance of that data structure to be evaluated?

3. How are optimal instances of the data structure to be sought?

In our choice of a data structure, we are defining the range of forms our concept of phylogeny may take. That this corresponds to reality is unknowable in an absolute

sense, but if the data structure does not suit its purpose, we may be able to perceive this darkly. For example, if we decided to represent the phylogeny of all organisms as a line of length 10 mm, we would be left with the feeling of not 'getting anywhere'. The data structure would not convey meaning because both the data structure (line) and a specific instance of that data structure (line of length 10 mm) were decided on without reference to any facts. But if we turn the line into a number line, with one end representing 'most primitive' and the other end 'most advanced', we may mark the position of biological entities on this line. The line is now a continuous axis of variation and, providing the nature of that variation may be clearly expressed, it can convey meaning.

The initial problem with such a line is in the definition of the axis, 'primitive' to 'advanced'. If the first organism that ever lived is the most 'primitive', what if, in a more recent organism, the only difference is replacement of a complex aspect of its structure by something simpler? Is the more recent organism more primitive because it is simpler, or more advanced because it has undergone a change from the earlier state? It would be possible to define answers to all such questions. These axioms would allow us to place all organisms on the line and, knowing the axioms, the positions of organisms on the line would convey meaning.

Apart from Sporne's advancement index [1, 2], there have been few recent uses of such a line. The suspicion is strong that the pattern of relationships is really nothing like a number line. Also, it is not necessary to restrict the data structure so much. We are capable of making, reading and distributing two-dimensional images. Since the origin of phylogenetic research as we know it [3], phylogenies have been proposed mainly as trees. *Vertices* represent actual and hypothetical biological entities, or *objects*, and *edges* represent paths between them. The advantage a tree has over a number line is that it illustrates the topology of descent rather than a simple index of primitiveness for each extant object. A typical *rooted tree* (Figure 1.1*) has the following components:

*Figures are grouped at the end of each chapter.

1. One vertex with two descendants, and no ancestor. This represents the hypothetical common ancestor of all other objects in the study, extant or hypothesised, and is known as the *root* of the tree.

2. One vertex per extant object in the study, each with one ancestor and no descendants. These represent the extant objects and are known as the *leaves* of the tree.

3. For a tree with n extant objects, $n-2$ other vertices, each with one ancestor and two descendants. These vertices represent hypothetical ancestors in various positions intermediate between the root (ultimate ancestor) and leaves (ultimate descendants).

For $n-2$ in (3) to have meaning as a count of vertices, it follows that n, the number of extant objects, must be at least two.

Because each hypothetical ancestor, including the root, has precisely two descendants, the tree bifurcates as we move away from the root, and is known as a *binary tree*. In the binary tree, there is a distinction between *left* and *right* descendants, but this difference, vital in many uses of the binary tree (e.g., [4]), is of no significance in phylogeny (Figure 1.2). The topology, or pattern of connections (branching pattern of the tree), is the feature of interest. Following biological use, I will refer to trees of identical topology as if they were identical trees, even if they differ in whether certain descendants occur to the right or to the left.

It is easy to generalise the data structure, to allow trifurcations and other higher-order branching events. These are assumed to be rare in evolution, but may be used on a phylogenetic tree where there is no evidence for either of two bifurcations occurring first (Figure 1.3). As with the binary tree, the left-to-right order of descendants of a single ancestor is meaningless in phylogeny, although it is important in other uses of non-binary trees (e.g., [5]).

The edges of the rooted tree have direction, representing passage of time *from* an ancestor *to* one of its descendants. The amount of time from one vertex to its

neighbour is irrelevant to the topology of the tree. Sometimes, the topology is combined with aspects of the number line, by representing some feature of each edge by the length of the edge relative to others. This may be used to show some sort of distance, such as time or genetic distance.

In an *unrooted tree* [Figure 1.4(i)], there is no root, and there is one less vertex and one less edge in the tree. Edges in the unrooted tree lack direction, and so ancestors and descendants are not distinguishable within the tree, other than to say that the leaves represent the ultimate descendants (since they have no descendants themselves). The unrooted tree has the following components:

1. One vertex per extant object in the study, each adjoining one other vertex. These represent the extant objects and are known as the *leaves* of the tree.

2. For a tree with n extant objects, $n-2$ other vertices, representing hypothetical ancestors. Each such vertex adjoins three other vertices. These vertices represent hypothetical ancestors in various positions intermediate between the leaves and each other.

For $n-2$ in (2) to have meaning as a count of vertices, it follows that n, the number of extant objects, must be at least two.

An unrooted tree may be rooted by breaking any edge in two with a new vertex, and calling this vertex the root [Figure 1.4(ii)–(iv)]. Edges then gain direction, since the root is known to be the ultimate ancestor in the tree.

The phylogenetic statement in a rooted tree is more specific than that made by an unrooted tree. Since there is often little evidence for which position the root should take, a rooted tree is also more likely to be incorrect. (The rooted tree like is an explanation of an unrooted tree.)

Having arrived at a data structure of rooted or unrooted tree, the next question is how to *evaluate* trees. A phylogenetic tree that showed a duck and an amoeba to be more closely related than a duck and a hen would be 'absurd', yet this absurdity must be quantified since it is not always so obvious. One simple way to 'evaluate' trees is

to construct a tree in a stepwise (sequential) manner such that there is only one result. This is assumed correct and all other possible trees are assumed incorrect. However, a sequential method ignores the overall probability of the whole tree, failing to revise earlier steps if they place an unreasonable constraint on later ones. Because of this, sequential methods (e.g., UPGMA, the unweighted paired-groups method using arithmetic averages [6, 7]) have fallen out of favour for phylogenetic work.

It is instead popular to divorce construction of the tree and evaluation of the tree, and evaluate a whole tree at once, for example using parsimony [8]. Trees must still be generated in some way, and one approach is to examine all possible trees connecting the extant objects in the study. As explained in Chapter 2, this is absolutely impractical for large studies. The problem then becomes to generate less than the complete set of possible trees for the objects at hand, and yet still obtain a result that is optimal, or near-optimal, with respect to our method of evaluating trees. Investigation of one such approach, simulated annealing [9, 10], is the subject of this thesis.

1.1 Bibliographic Note

Though their presentation is my own, most of the ideas in this chapter are not original [11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

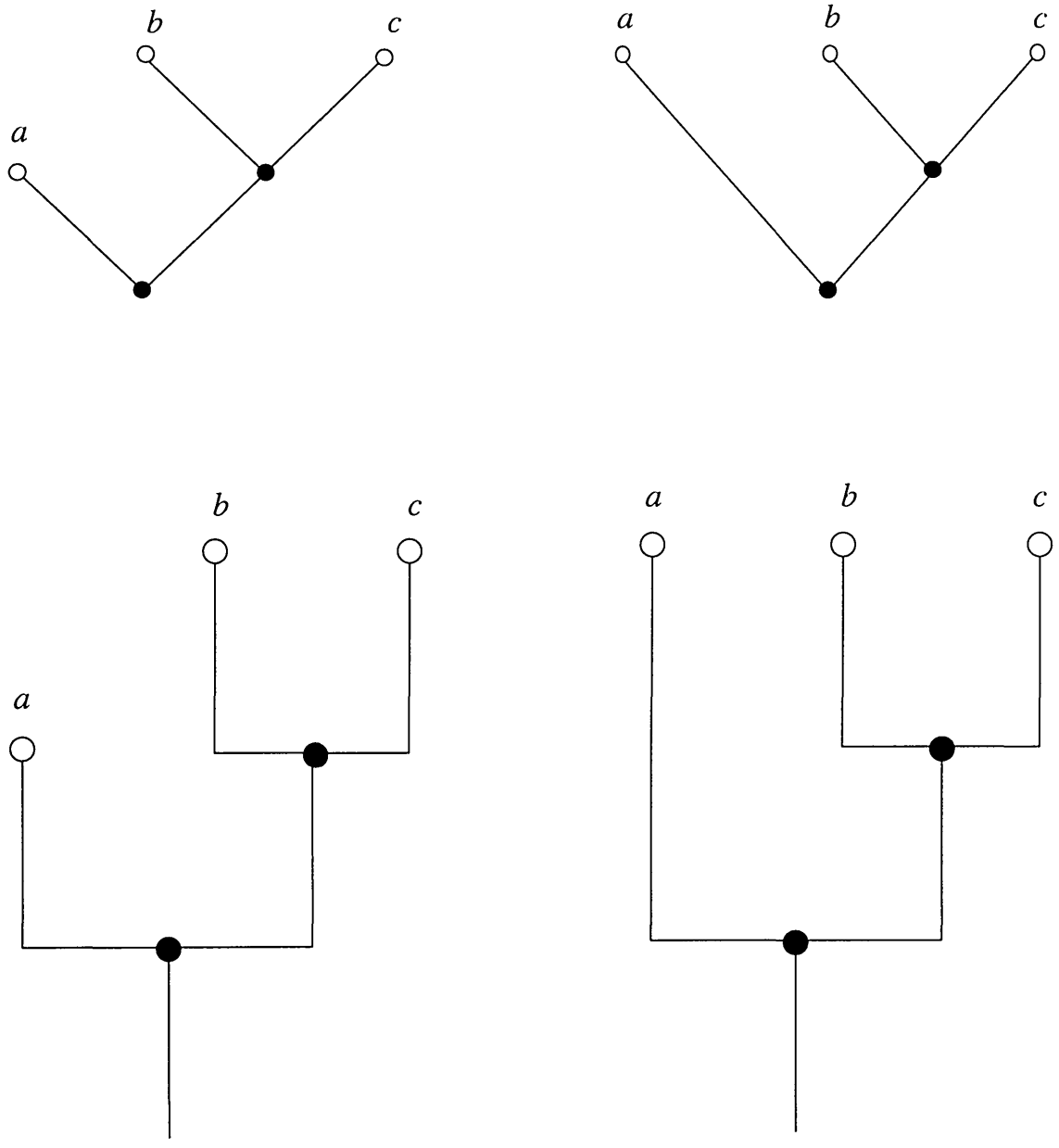


Figure 1.1 Equivalent representations of a single rooted tree illustrating relationships between three extant objects, a, b and c. The four versions of the tree differ only in the details of presentation. Biologically and mathematically, they have the same meaning. Vertices representing extant objects are signified by open circles, and those representing hypothetical ancestors by closed circles. In all cases, the lower of the two hypothetical ancestors is the root. Circles representing vertices would normally be omitted in such diagrams.

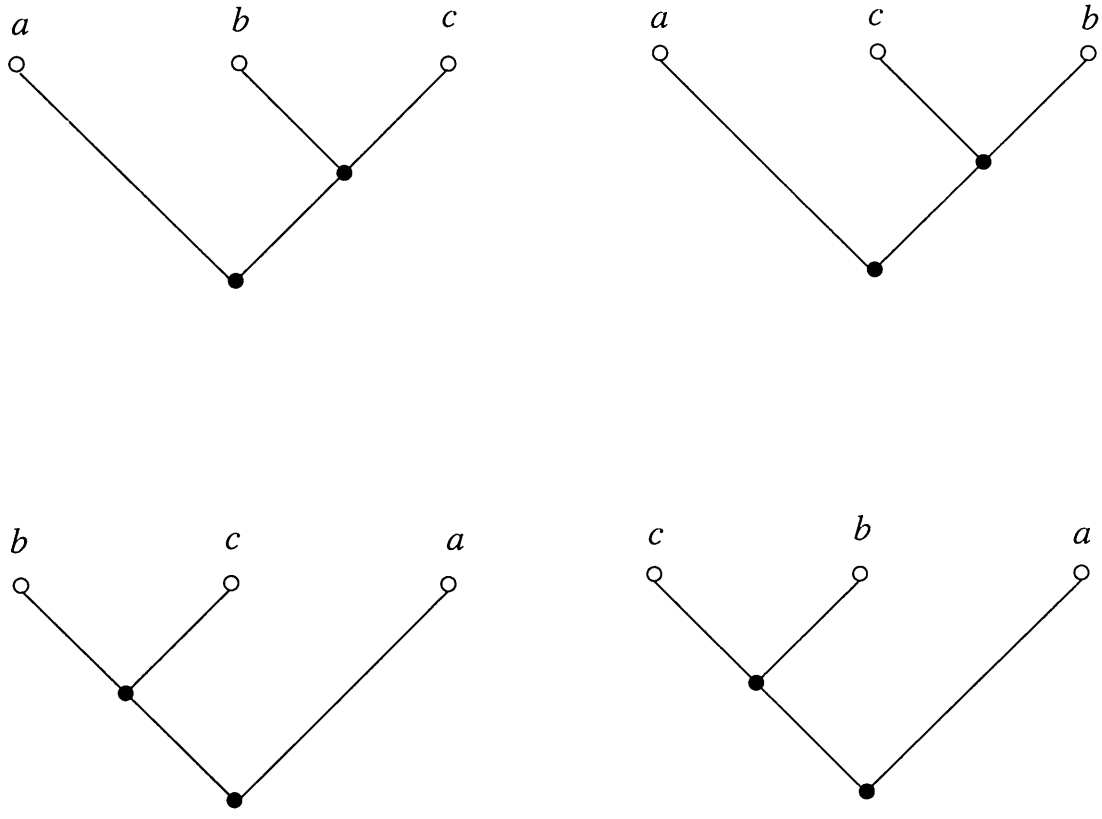


Figure 1.2 Biologically equivalent representations of a single rooted tree illustrating relationships between three extant objects, a, b and c. The topology of each tree is the same, but which descendants occur to the left and which to the right of their ancestor vary. Because of this, mathematically, each of the four rooted trees is a different binary tree. However, this variation has no meaning in phylogeny, where these trees are regarded as the same.

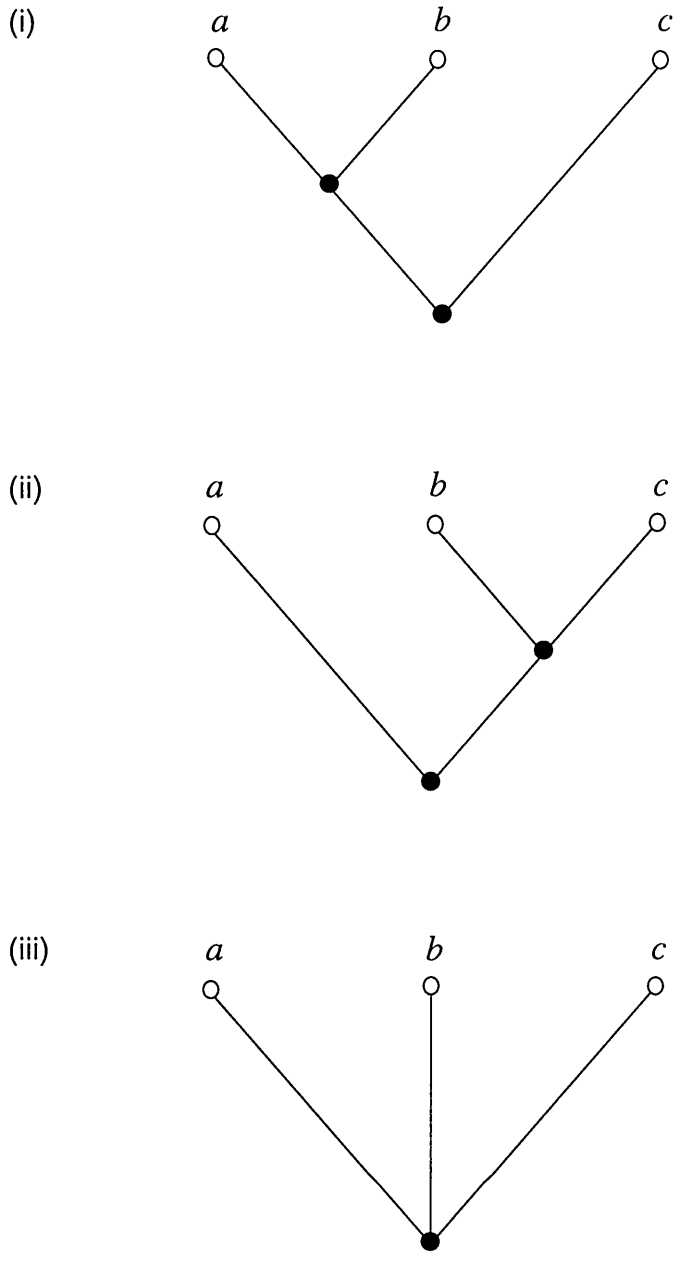
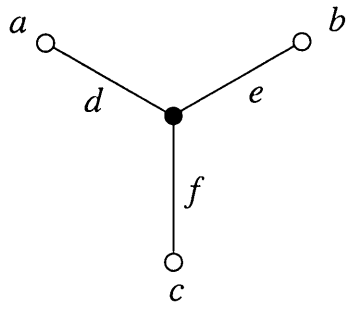
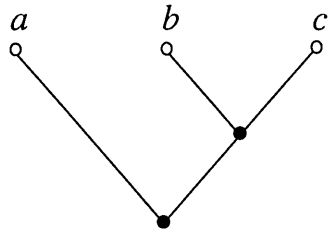


Figure 1.3 Use of a trifurcation to show ambiguity in the pattern of relationships. (i) and (ii) are two of the three possible rooted trees for three extant objects, a, b and c. If it is unclear which of these is correct, (iii) might be presented as the current solution. It implies uncertainty rather than a simultaneous split of one lineage into three. There would be loss of information, since, as well as (i) and (ii), (iii) equally suggests a grouping of a with c more closely than with b, which is not present in (i) or (ii).

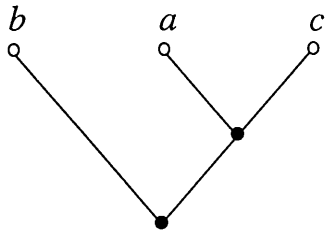
(i)



(ii)



(iii)



(iv)

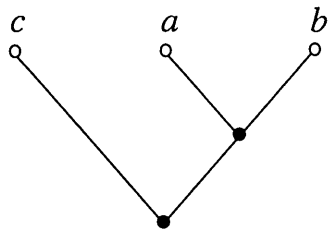


Figure 1.4 (i) An unrooted tree representing relationships between three extant objects, a, b and c. For any three extant objects this is the only possible unrooted tree topology. The tree may be rooted by splitting any one of the edges d, e or f in two with a root vertex, giving the topologies shown in (ii), (iii) and (iv) respectively.

Chapter 2

Simulated Annealing

One way to find the best trees for the data matrix is to generate and evaluate all possible trees. The number of unrooted trees for n extant objects, t_n, is given by:

$$(\forall n \geq 3) t_n = \prod_{i=3}^{n} (2i - 5) \tag{2.1}$$

This equation arises as follows [12, 21, 16].

No tree is possible for less than two extant objects (Chapter 1). For two extant objects, there is only one possible unrooted tree [Figure 2.1(i)], with a single edge. This tree is unusual. It is distinguished by being the only tree in which one edge leads to two *extant* objects. It has no internal edges, and is the only tree in which there is no hypothetical ancestor. Being the only possible tree for the given number of objects, it can reveal nothing about relationships, and is of little interest in phylogeny.

For three extant objects, there is still only one unrooted tree [Figure 2.1(ii)]. Again, being the only possibility, this tree is of little interest. It has three edges, each leading to a different extant object. Like the tree for two extant objects, it lacks internal edges. However, there is now one vertex representing a hypothetical ancestor.

To increase the number of extant objects to four, we split any one of the edges in this tree, and insert an edge leading to the new extant object at the point of the split. Since the tree for three extant objects has three branches, any one of which may be split in this way, there are three possible trees for four extant objects [Figure 2.1(iii)].

Whichever edge is chosen, splitting an existing edge increases the number of edges

by one. The new edge is always internal, except when the original tree has only one edge (i.e., represents relationships between two extant objects), in which case it must be external. Adding the edge leading to the extra extant object also increases the number of edges by one, but here, the new edge is necessarily external. In all, adding one extant object increases the total number of edges by two. If b_n is the number of edges in the unrooted tree representing relationships between n extant objects, then

$$(\forall n \geq 3) b_n = b_{n-1} + 2 \tag{2.2}$$

The number of internal edges in a tree for n extant objects, b_{ni}, starts at zero for trees representing relationships between two or three extant objects, and increases by one per extant object added:

$$(\forall n \geq 3) b_{ni} = n - 3 \tag{2.3}$$

The number of external edges, b_{ne}, is simply the number of extant objects:

$$(\forall n \geq 3) b_{ne} = n \tag{2.4}$$

Combining Equations 2.3 and 2.4,

$$(\forall n \geq 3) b_n = b_{ni} + b_{ne} \tag{2.5}$$

$$= 2n - 3 \tag{2.6}$$

Equation 2.6 is consistent with Equation 2.2, but more convenient, since non-recursive. The relation is illustrated for two to 2 000 extant objects in Figure 2.2.

To obtain a tree with *five* extant objects, any of the five edges of *any* of the three trees representing four objects may be split in two with the addition of a new branch, giving 15 possible trees [Figure 2.1(iv)]. It is clear that the number of possible trees for n extant objects depends on both b_{n-1} and the number of different trees for $n-1$ extant objects, t_{n-1}:

$$(\forall n \geq 3) t_n = t_{n-1} b_{n-1} \tag{2.7}$$

b_{n-1} may be obtained from Equation 2.6, giving:

$$(\forall n \geq 3) t_n = t_{n-1}[2(n-1) - 3] \tag{2.8}$$

$$= t_{n-1}(2n - 5) \tag{2.9}$$

Given that $t_2 = 1$,

$$(\forall n \geq 3) t_n = \prod_{i=3}^{n} (2i - 5)$$

Which is Equation 2.1.

The factorial increase in number of possible trees with number of extant objects presents even more of a problem than exponential increase. This is shown in Figure 2.3. The gradient of this semi-log plot would be constant if number of trees increased exponentially with number of extant objects. But, in fact, the gradient increases as the number of extant objects increases.

Because of this, it is not practical to evaluate each possible tree unless the number of extant objects is small. Assuming a computer takes 10^{-5} seconds per edge to calculate tree length, which would be about the speed if there were 200 varying characters, it would take 37 days to examine all trees for 13 extant objects. Increasing the number of extant objects by just one, to 14, would cause the search time to increase to 2.5 years, and increasing the extant objects to 16 would lead to a search taking almost two millennia (Figure 2.4). If the search could be distributed in parallel over 1 000 processors, without any loss of efficiency, the search time, for 14 extant objects, would reduce to a quite practical one day. The search time for 15 objects would be 25 days, but increasing the number of extant objects to 16 would again leave us with an impossibly long search, this time taking two years. To reduce the search time to two days, for 16 extant objects, the number of processors would have to be increased to 360 000 (Figure 2.5). In practice, more processors would be required, to make up time lost on communication between processors. Since it is impossible to reduce the time taken by a program to less than that of any necessarily serial component, such a speed-up might actually be impossible. (This limit is known as Amdahl's law [22].)

The above examples would all represent quite small studies by modern standards. The speed with which a gene may be sequenced has increased to the point that it is common to include several dozen extant objects in the same data matrix. If there were

100 extant objects, an exhaustive search would require examination of 1.7×10^{182} trees, taking 1.1×10^{172} years on a serial machine or 1.1×10^{169} years on the 1 000-processor machine (assuming run time per edge as above, and no loss of efficiency due to parallelisation). For comparison, the estimated age of the universe is about 10^{10} years. For the search to take just two days, again assuming perfectly efficient parallelisation, 1.9×10^{174} processors would be required. (The estimated number of atoms in the universe is 10^{82}.)

Using the estimated time per edge above, a tree of 13 extant objects may be evaluated in 2.3×10^{-4} seconds. If, due to changes to the program or computer hardware, the tree for 100 extant objects could now be evaluated at this speed, little would be gained, since only the linear component of the problem, time per tree, is affected. The serial exhaustive search would take 1.2×10^{171} years, the parallel search on a 1 000-processor machine would take 1.2×10^{168} years, and the two-day search would require 2.3×10^{173} processors. The problem is reduced by a single order of magnitude in each case, and remains enormous.

The problem may be reduced algorithmically, using *branch-and-bound* [23, 24]. This approach is useful for a wide range of problems in computation. It was brought to phylogeny by Hendy and Penny [25].

The set of possible trees is partitioned into smaller and smaller subsets, and a lower bound on tree length is calculated for the solutions in each subset (cf. [26]). If the bound is greater than the length of the shortest tree found so far, no trees in the subset have to be evaluated. They can only be longer than the best known, and so can be rejected.

Branch-and-bound gives an exact solution. Compared to an exhaustive search, it reduces the amount of time that must be devoted to evaluating trees, in a rather unpredictable manner. In practice, even with branch-and-bound, the maximum number of extant objects cannot be more than about 17 or the search will take so long that it will never finish before the computer suffers down-time. The exact figure depends on the precise problem at hand, including the nature of the data matrix. It is

not straightforward to predict run-time of branch-and-bound algorithms in general [27], and 17 is only a guess based on some experience. With a data matrix of even 25 objects, it may be worth running an initial branch-and-bound analysis, and terminating it in favour of a heuristic if it has not completed after a few days.

Exhaustive searching and branch-and-bound are the only known searches guaranteed to give the best tree or trees. There is little reason to perform an exhaustive search rather than branch-and-bound. On the other hand, branch-and-bound does not increase the size of the data matrix that may be used very much. Seeking parsimonious trees falls into a class of problems known to mathemeticians as *NP-complete*. For these, there is no reliable way to obtain an exact solution (in a realistic amount of time) for large sizes of problem, though a smaller problem may be very simple to solve (cf. Figure 2.1). It would take a great revolution in mathematics or computing for us to be able to solve these problems precisely. Whether such a revolution will occur is not yet clear. Quantum computers [28] might, eventually, have a practical application in this area.

The only option today, if we are to stick with global tree evaluation, is to use a *heuristic* rather than an exact search. A heuristic gives a solution whose difference from the true optimum is unknown and whose probable difference from the optimum is difficult or impossible to quantify. (In this the heuristic differs from an *approximate* method.) It is not possible to say, 'this heuristic will give a solution that differs from the true optimum by no more than 10%'. However, though heuristic solutions cannot be compared against the 'gold standard' of the true optimum, they can be compared against each other. The question becomes, which heuristic achieves the best solution with a certain range of data matrices in a certain amount of time? There are several kinds of heuristics, though the division between most of them is artificial and there are plenty of mixed approaches.

2.1 Greedy Algorithm

This heuristic builds a solution a step at a time, on the assumption that the local solution for the current step will not be greatly at odds with the global solution. The greedy algorithm has probably been used for time immemorial and has many applications in everyday life, such as designing timetables or loading vehicles. For some problems, for example the minimal spanning tree, it is guaranteed to give the globally best solution [29]. However, this is not the case in general. 'Just as in life, a greedy strategy may produce a good result for a while, yet the overall result may be poor' [24].

In phylogeny, this heuristic is also known as *stepwise addition.* It may be used to seek a parsimonious tree as follows. An initial tree is constructed for the extant objects represented by the first three rows in the data matrix. The effect on tree length of adding the fourth object in each of the three possible places is assessed, and it is added in the place where it causes minimum increase in tree length. (If two places are equally good in this respect, some arbitrary decision is made to chose between them.) The same criterion is applied in adding the fifth and subsequent objects, until all objects have been added. The heuristic is in fact a sequential method of tree construction (Chapter 1), and so the result is not expected to equal the globally optimal tree.

Randomising the sequence in which objects are added will often affect the result, but not necessarily improve it. If the rows of the matrix have been arranged according to some prior concept of the phylogeny, with rows representing objects believed to be closely related placed close together in the matrix, and this prior concept coincides closely with the phylogeny according to the optimality criterion in use (e.g., parsimony), randomising the addition sequence will often decrease the quality of the solution. However, running *many* different analyses with different random addition sequences is likely to overcome this: one addition sequence of, say, 100 might happen to be auspicious.

In practice, since it is rather fast and not very good, the greedy algorithm is rarely applied in a 'pure' form. Improvements include:

1. making occasional rearrangements of the existing tree;
2. running the search many times with different addition sequences;
3. using the greedy algorithm to obtain initial trees for a second heuristic, that uses global rearrangement.

Some combination of two or more of these is usually used, e.g., with the program PAUP [30] or `dnapars` in the PHYLIP package (Appendix G.2).

2.2 Hill Climbing

Hill climbing begins with a complete solution to the problem, in our case, tree. This may be constructed at random, by a greedy algorithm, or according to an arbitrary procedure. This is the first *current* tree. A *mutation* of the current tree is proposed, giving a tree that is in the *neighbourhood* of the current one. Neighbours, for this purpose, have been defined in different ways [17]. The important characteristic of a neighbour is that it is generated from the current tree by a simple change, and so is similar (*local*) to the current tree. The program LVB [31] generates neighbours as shown in Figure 2.6.

The mutation function should be designed so that it only ever produces viable mutations, i.e., it does not damage the data structure but always returns an unrooted bifurcating tree for the known objects at hand. If the mutated tree is more optimal than the current tree, given the data matrix, it is accepted as a replacement for the current tree. Otherwise, it is rejected, and the current tree remains unchanged. The mutate, evaluate, accept/reject procedure is repeated, either for a given number of iterations, or until no new mutant has been accepted for a given number of iterations. Here is an outline of a simple hill climbing procedure, in which the number of trees is fixed prior to the start of the heuristic:

```
while (the number of trees examined is less than the number requested)
{
    propose a change to the current tree;
    if (the changed tree is no longer than the current tree)
        accept the change;
}
```

Unlike a greedy algorithm, hill climbing does not construct trees sequentially. The effect of each step on the optimality of the whole tree is evaluated. This could be seen as an improvement, since the optimality criterion, for example, steps (parsimony criterion) or total length in some space (minimum evolution, Section 6.3) is a property of the whole tree. However, it shares the short-sightedness of the greedy algorithm. If the tree could be overall improved by making a change that *increased* length, followed by a change that decreased length, this opportunity is missed.

Like the greedy algorithm and branch-and-bound, hill climbing has been used to seek answers to many classes of problem for many years. The first presentation of hill climbing I know is from 1958 [32]*.

Because each change proposed is a neighbour of the current solution, hill climbing is a type of *local search* [24]. Hill climbing is also known as *iterative improvement* [33]. When each possible mutation of the current tree is considered, and only one that causes the maximum benefit to the solution is actually accepted, hill climbing is also known as *steepest descent*, for example in the program PAUP [30]. (It is easy to adapt a heuristic for minimisation, or descent, to one for maximisation, or 'climbing', and *vice versa*. The two, related, goals have led to opposite-sounding names.)

*This paper is especially interesting in that the mutation function used is *inversion*. This is usually considered a genetic operator, as used in the more recent approach known as genetic algorithms (Section 2.5).

2.3 Stochastic Optimisation

Stochastic optimisation includes a random element in the heuristic. It could be regarded as, for example, hill climbing with random wrong decisions:

```
while (the number of trees examined is less than the number requested)
{
    propose a change to the current tree;
    if (the changed tree is no longer than the current tree)
        accept the change;
    else
        accept the change with a certain probability;
}
```

The advantage of the random element is that it allows the search to retreat from local optima. The disadvantage is that it may cause the search to retreat from the *approach* to optima, even from the approach to the global optimum.

Hill climbing may be regarded as a special case of stochastic optimisation, in which the stochastic element is zero.

2.4 Simulated Annealing

Simulated annealing is like stochastic optimisation, except that the probability of making changes that decrease optimality is allowed to vary. It varies both throughout the run and according to the effect on optimality of the proposed change.

Although simulated annealing was developed for optimising the design of microchips [9], its authors realized it could have much wider application. It seems to have been first used in phylogeny by Lundy [10]. It has been considered for *covering*-based classification by Lefkovitch [34]. (A covering is a data structure that may have uses in phylogeny, but is not widely used in this area at the moment.) As a simplification, I talk of simulated annealing as a method of optimising phylogenetic

trees, although many other data structures could be substituted.

The name 'simulated annealing' derives from the equation usually used to obtain the probability, p, of accepting a change that gives a tree *worse* than the current tree:

$$p = e^{-(k/T)\Delta H} \tag{2.10}$$

where T is an arbitrary, real control parameter known as *temperature*, H is a non-negative real value inversely related to the optimality of the solution and is known as *energy*, and ΔH is the change (increase) in energy caused by the change to the tree. k is a constant, to ensure the power to which e is raised has no units.

The probability of accepting a change that either increases or does not change optimality (decreases or does not change energy) is not given by Equation 2.10, but is fixed at one.

With Equation 2.10, if T is constant and has units of Kelvin, ΔH has units of Joules and k is the Boltzmann constant, simulated annealing at fixed temperature becomes the Metropolis algorithm [35]. This is used to quickly estimate average quantities distributed in a Boltzmann distribution over particles of a system. In fact, simulated annealing is derived from the Metropolis algorithm rather than the reverse. The differences are

1. In simulated annealing, T decreases, whereas in the Metropolis algorithm, it is constant.

2. In the Metropolis algorithm, the aim is to accurately and quickly estimate some average quantity of elements of the system. In simulated annealing, the aim is to seek a data structure of low energy.

Because of (1), the simulated annealing process is like physical annealing, during which temperature decreases and particles change state from liquid to solid. In the liquid state, the energy of each particle tends to be high, but the variation in energy among particles is considerable. The system at this stage is relatively disordered. As temperature falls, both the average energy of particles and the variation in energy

among particles decreases. The system eventually crystallises, i.e., attains a state where particles have low energy and variation in energy between particles is slight. The system is now 'frozen' and highly ordered. However, the energy of particles in the frozen system depends on *how* temperature decreased during annealing. During annealing, the system converges on a particular structure. Lower temperature gives less freedom to explore other possible structures. Therefore, if the energy of the final crystal is to be very low, it is important to decrease temperature slowly, especially during the phase change between liquid and solid. This allows exploration of various coarse structures before 'homing in' on one, and the overall result will tend to have very low energy. This is important in, for example, metal manufacture, when the lower-energy solid will be stronger.

The situation is similar where 'solution to the problem' or, in our case, 'tree' is substituted for 'particle' in the above. Since energy is inversely related to optimality, the final trees will tend to be better (if evaluated by parsimony, with fewer steps or *shorter*) when temperature falls more slowly.

Given that we are dealing with trees rather than physical particles, why use Equation 2.10 rather than, say, $p = k(\Delta H)/T$? The Boltzmann distribution has no special meaning outside physics. However, it has been proved that, using Equation 2.10, the final result will, if temperature decreases in a certain way, give a good result independent of the initial solution used [36]. Thus, the search is relatively unbiased, in contrast to hill climbing where the initial tree constrains the final result considerably. Unfortunately, for this to hold, temperature must decrease more slowly than is practical. For the global solution to be found irrespective of the initial solution with probability one, temperature must decrease infinitely slowly [37]. The hope is that a faster decrease of temperature will still give low bias in the search. There is a trade-off between quality of result and time.

With hill climbing, the search should be performed many times, with different initial trees. With simulated annealing, one or a few searches should suffice, if each search is sufficiently long. If having to re-start the hill climbing search many times

were unproductive compared to a small number of simulated annealing runs, simulated annealing would have the advantage in that, over a given run time, it would tend to find solutions of higher optimality. Run-time is already as long as possible in many phylogenetic analyses, e.g., a weekend or even a month, depending on available systems. Given that run time cannot be increased, and that parallelisation is a losing battle, a more efficient heuristic is perhaps the easiest way to increase the quality of results for a given data matrix. It is in this area that simulated annealing holds promise in the search for phylogenetic trees.

In non-physical applications, it is convenient if both T and H have no units and vary in the interval [0..1]. This allows $k = 1$ to be used rather than the Boltzmann constant (i.e., allows k to be omitted from Equation 2.10). If the optimality criterion is parsimony, this may be achieved by defining H as the *homoplasy index*, or HI [30]. In the absence of missing data or polymorphic character states, HI may be obtained from the consistency index (CI) as follows:

$$\mathrm{HI} = 1 - \mathrm{CI} \tag{2.11}$$

CI is explained below.

The consistency index for *character* i, c_i, is the ratio between the minimum number of changes character i could show on a tree, m_i, and the number of changes that are *actually* indicated by the parsimony algorithm [8], s_i:

$$c_i = \frac{m_i}{s_i} \tag{2.12}$$

A character i has to at least exhibit each of the t_i states found for it among the extant objects. In evolutionary terms, it must begin with one state and change at least once per extra state. So the minimum number of changes for a character may be obtained as follows:

$$(\forall t_i \geq 2) m_i = t_i - 1 \tag{2.13}$$

CI, the consistency index for a *tree*, given a specific data matrix of n characters, is

given by:

$$\mathrm{CI} = \frac{\sum_{i=1}^{n} m_i}{\sum_{i=1}^{n} s_i} \tag{2.14}$$

This is sometimes referred to as *ensemble consistency index.* Often, both the tree and character consistency indexes are referred to as 'consistency index' or 'CI', and it is only clear from context which is meant.

CI, the figure for the whole tree, depends on how well the tree fits all of the data in the matrix. It lies in the interval [0..1], although it is never actually zero. (The precise lower bound depends on the data matrix.) For a given data matrix, CI is inversely proportional to tree length. Since, with simulated annealing, H is a quantity we seek to *minimise*, I have instead equated H with the HI (Equation 2.11).

Since T is arbitrary, it may simply be *defined* to have an absolute lower limit of 0 and absolute upper limit of 1.

Simulated annealing at a constant temperature of *eps*, where *eps* is a positive real value just distinguishable from zero, would be hill climbing. Simulated annealing at a constant temperature of 1 would be a fairly unconstrained 'walk' through possible trees. Hence, the simulated annealing search, by starting at high temperature and progressing to low temperature, begins as something similar to a random walk, and ends up as hill climbing. In the final stage of the search, only solutions of quality equal to the current solution are accepted. If a different, equally optimal tree is encountered, $p = 1$. This ensures good representation of the whole local optimal region in the final set of solutions. If we are fortunate, this local optimum will be similar to, or even *be*, the global optimum.

Temperature could be decreased in many ways during the run. I have chosen the cooling schedule of Kirkpatrick *et al.* [9] in which temperature stays at a given level for a certain amount of time, rather than decreasing in a continuous manner, and in which the relation between successive temperatures is that of exponential decrease. Each temperature level, T_n, is obtained from the initial temperature, T_0, and the

second temperature, T_1, $T_1 \leq T_0$, as follows:

$$T_n = \left(\frac{T_1}{T_0}\right)^n T_0 \tag{2.15}$$

If $T_1 = T_0$, temperature remains fixed and we have the Metropolis algorithm (above).

It would be possible to use another relation between successive temperatures (Section 6.5), such as a linear relation [37].

With the program LVB, the amount of time spent at each temperature and when to terminate the search are controlled by the parameters `maxaccept`, `maxpropose` and `maxfail` (Chapter 3).

2.5 Genetic Algorithms

A simple genetic algorithm [38, 39] (GA) is shown in Figure 2.7.

The solution to the problem is coded as a 'chromosome'. This differs from a biological chromosome in that any data structure is allowed, rather than just a string of nucleic acids. A GA differs from simulated annealing in having a *population* of current solutions rather than just one, and in using *genetic operators* to generate new solutions. One such operator is mutation, also used in hill climbing (Section 2.2) and simulated annealing (Section 2.4). Others are used as well.

For example, chromosomes may recombine by crossover. This differs from biological crossover, since the GA chromosome is usually haploid (a single data structure representing a solution to a problem), yet crosses over in the manner of a sexually reproducing diploid organism. Inversion is also possible, though this operator is not always used.

The genetic operators generate and maintain diversity. A contrasting force is provided by *selection pressure*, whereby 'fitter' (more optimal) chromosomes are more likely to survive and produce offspring. Depending on the exact GA, the analogy with biological evolution may be close. In the simple GA, the population is panmictic, i.e., the probability of crossover between any two solutions is equal. More elaborately, a GA may include a population structure, with chromosomes spread over

more than one 'island'. The population of each island is panmictic within itself, and there is also a certain, typically low, migration rate between islands.

It is possible to introduce the cooling cycle of simulated annealing into a GA [40], in which case the GA gains the property of simulated annealing, that the heuristic can become an exact method when the run is infinitely long.

For biologists, GAs are easier to understand than simulated annealing. For programmers, they are easier to parallelise (Section 6.2), due to the inherently parallel evolution of populations on separate islands. (Parallelisation can be arranged automatically with the RPL2 package [41]). On the other hand, genetic algorithms are awkward to use with a tree data structure.

A GA has been found to give excellent results in seeking maximum likelihood trees [42]. GAs are discussed further in Section 6.6.2.

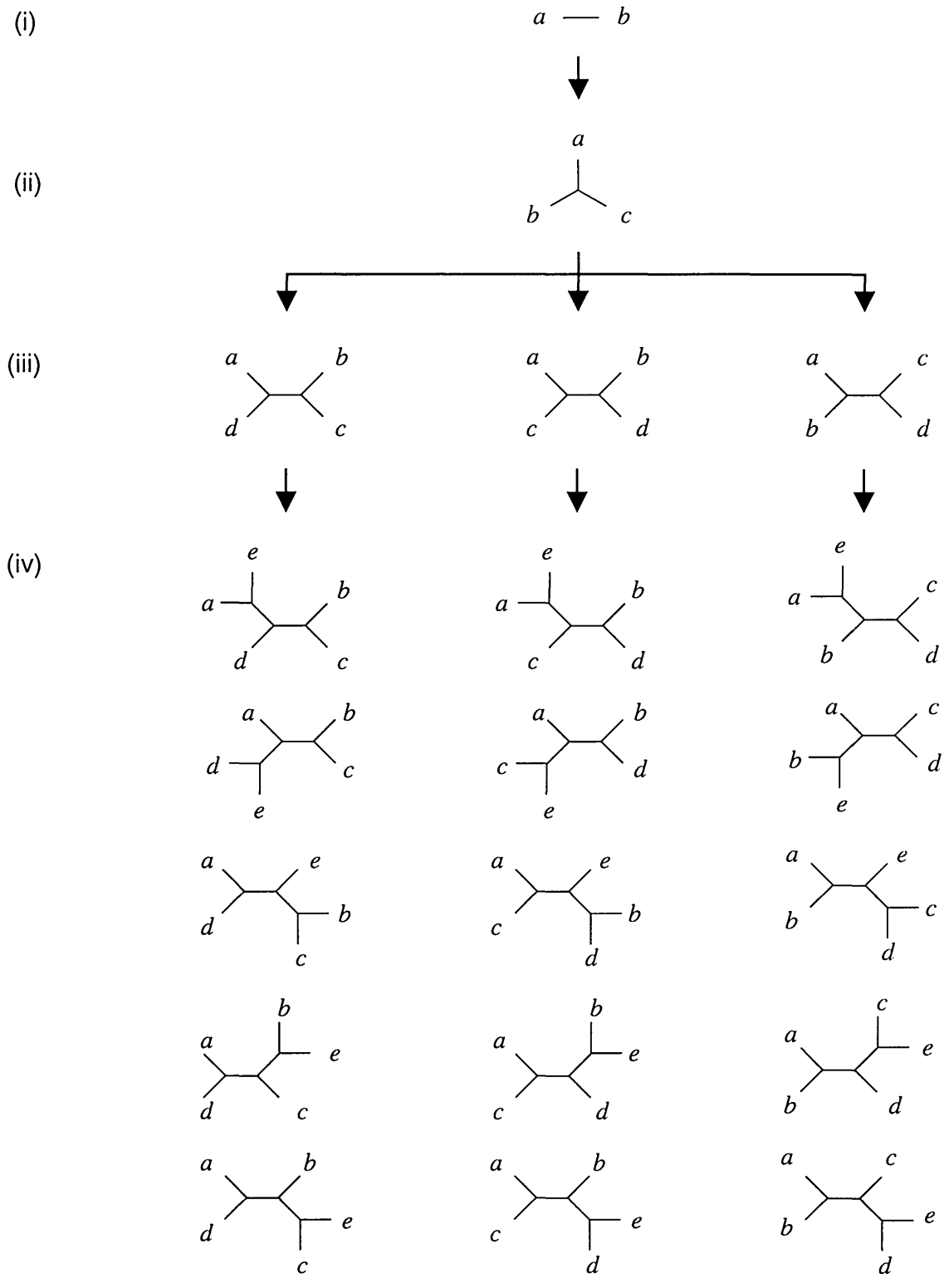


Figure 2.1 Increase in number of possible unrooted trees with number of extant objects. (i) and (ii) If there are two or three extant objects, only one tree is possible. (iii) Possible trees with four extant objects. (iv) Possible trees with five extant objects.

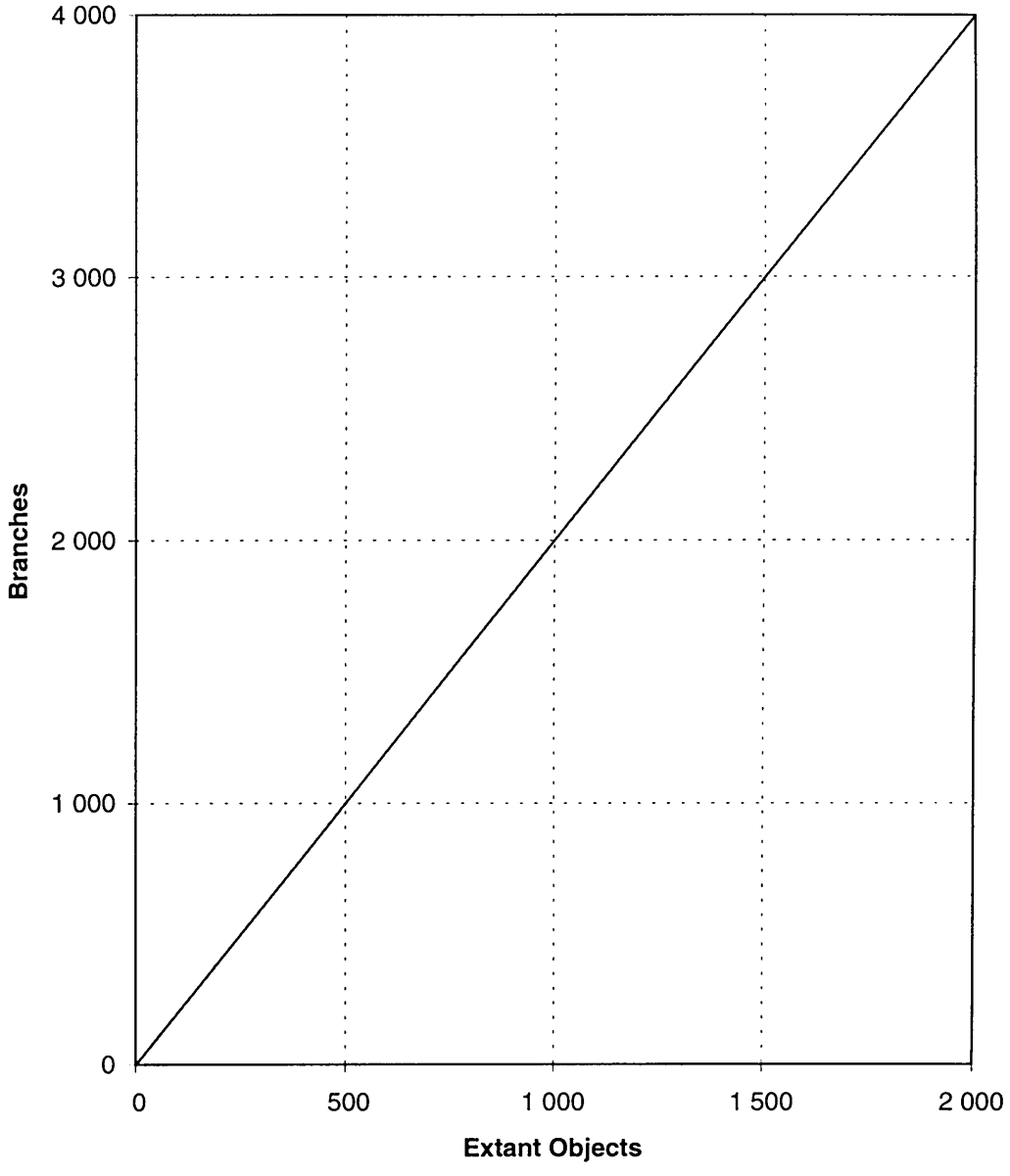


Figure 2.2 The relationship between branches per unrooted tree and number of extant objects.

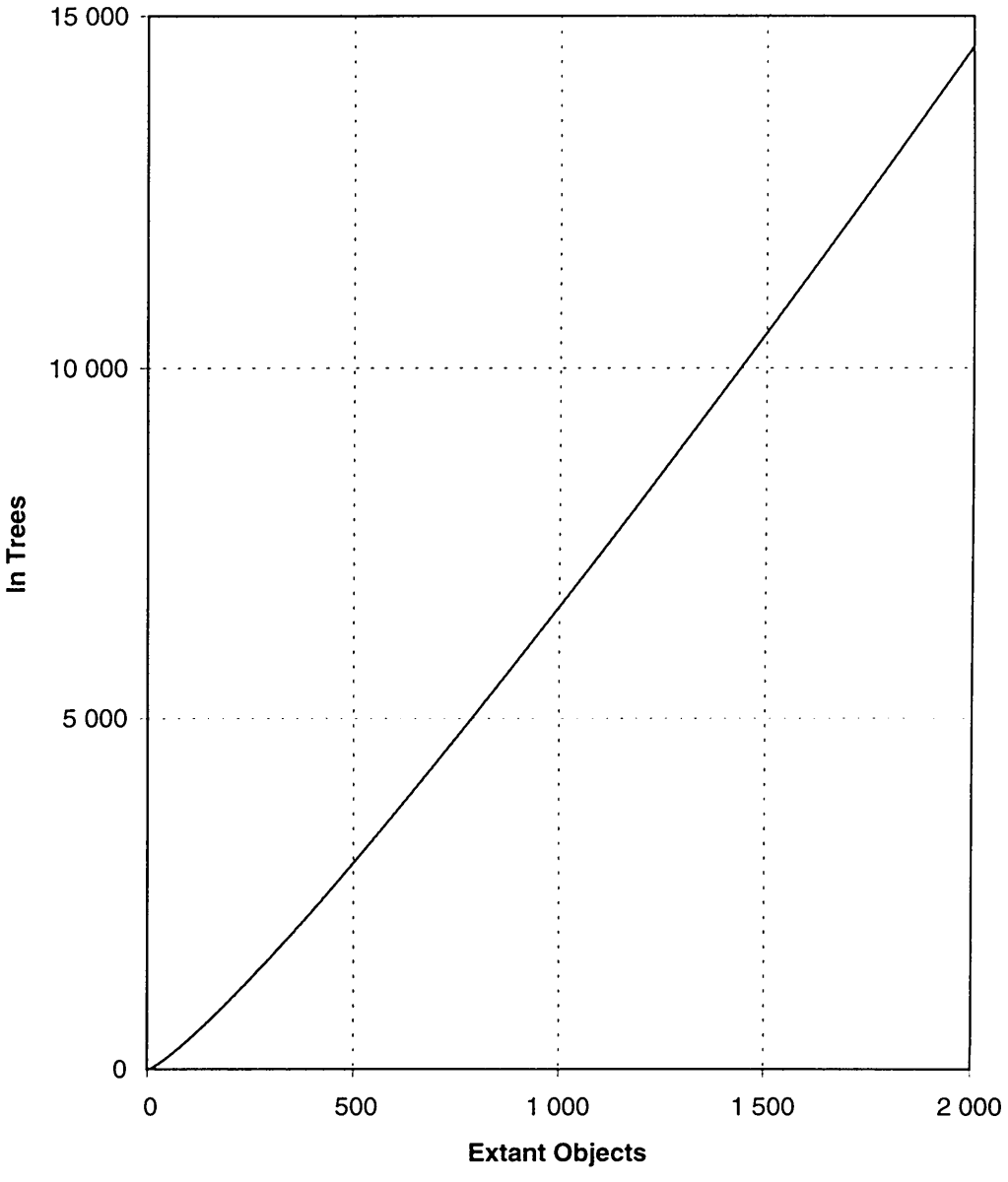


Figure 2.3 The relationship between ln(number of possible trees) and number of extant objects.

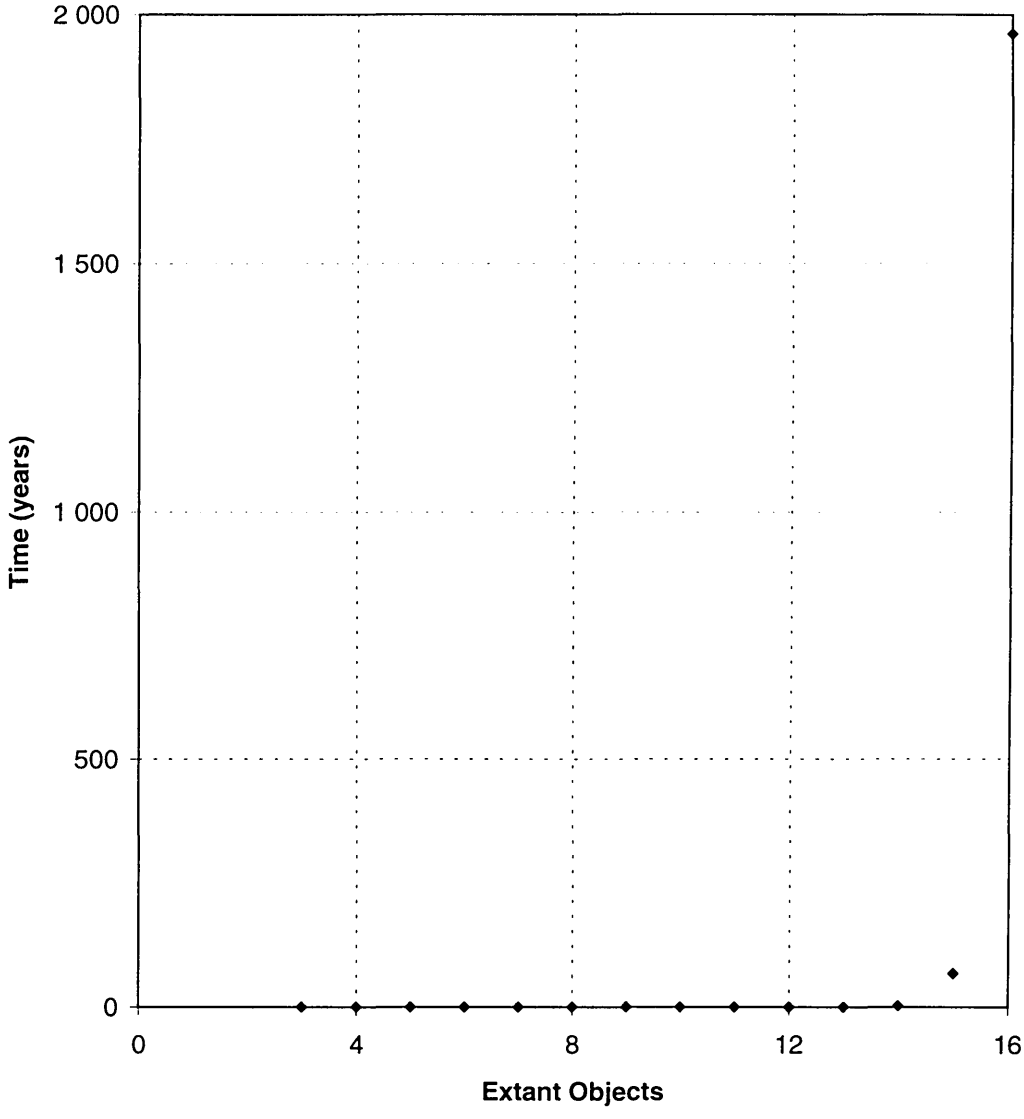


Figure 2.4 The relationship between CPU time for an exhaustive search and number of extant objects, if searching takes 10^{-5} seconds per edge per tree.

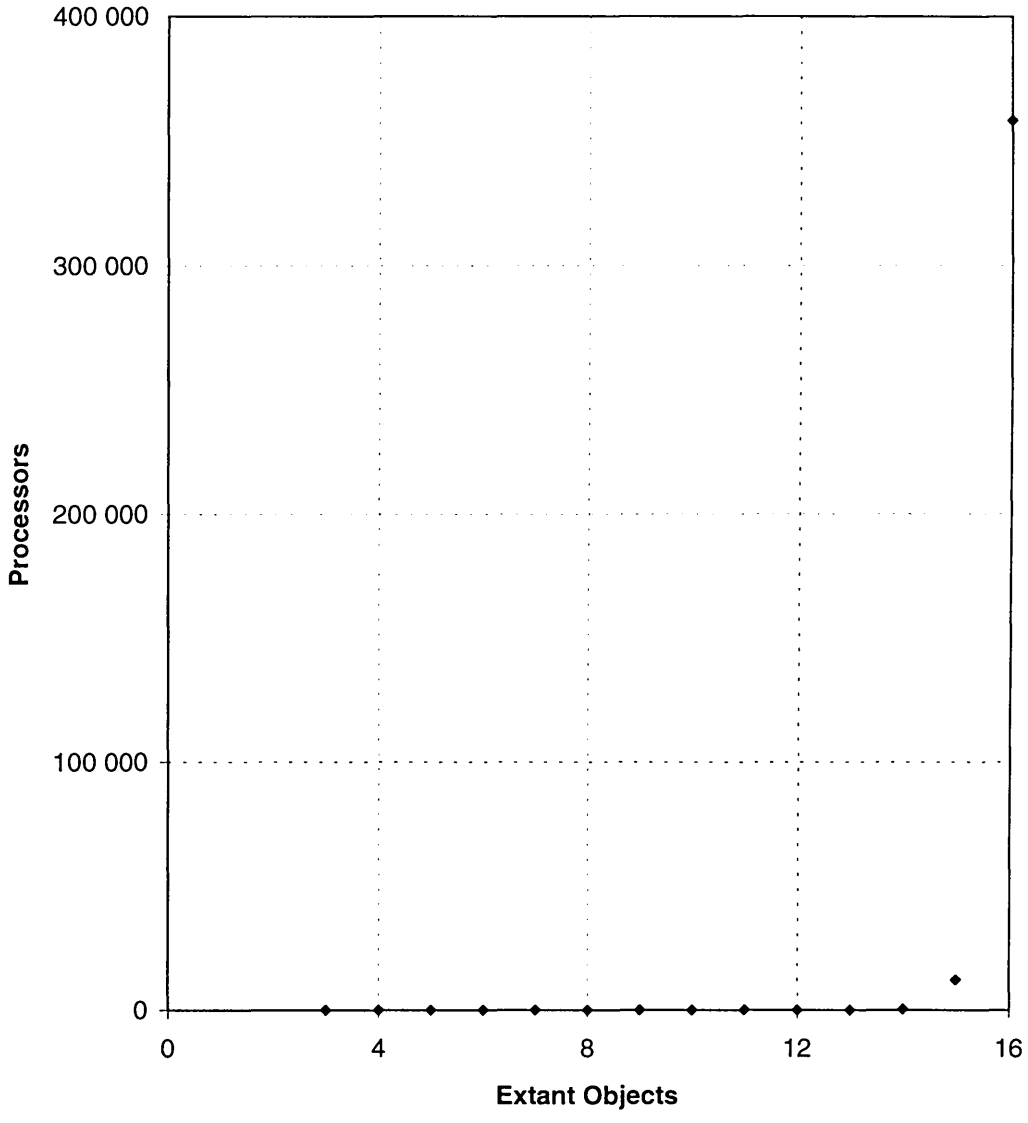


Figure 2.5 The relationship between number of processors for an exhaustive search and number of extant objects, if searching takes 10^{-5} seconds per edge per tree and the search is to be complete within two days. Perfect efficiency of parallelisation and no serial bottleneck are assumed.

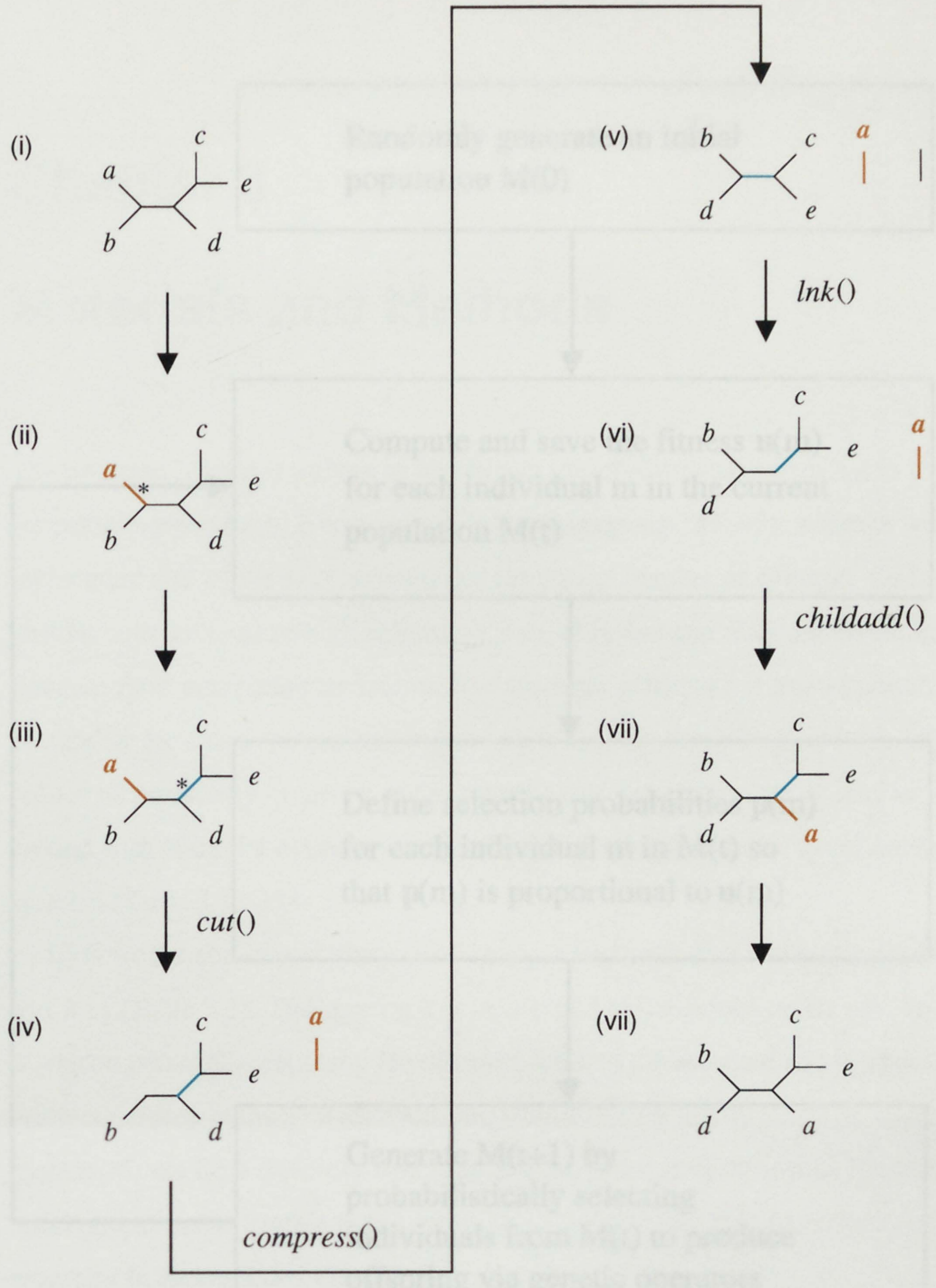


Figure 2.6 The mutation function of LVB. Sub-programs called by `mutate()` (Appendix D) are indicated. (i) Example input. (ii) *Source* leaf (marked with an asterisk) selected at random. (iii) *Destination* (marked with an asterisk) selected at random out of all edges excluding both the source and any of the two edges with which it is in direct contact. (iv) Source excised. (v) Tree repaired. (vi) Space made using the internal edge excised in (v). (vii) Source inserted again. (viii) Output.

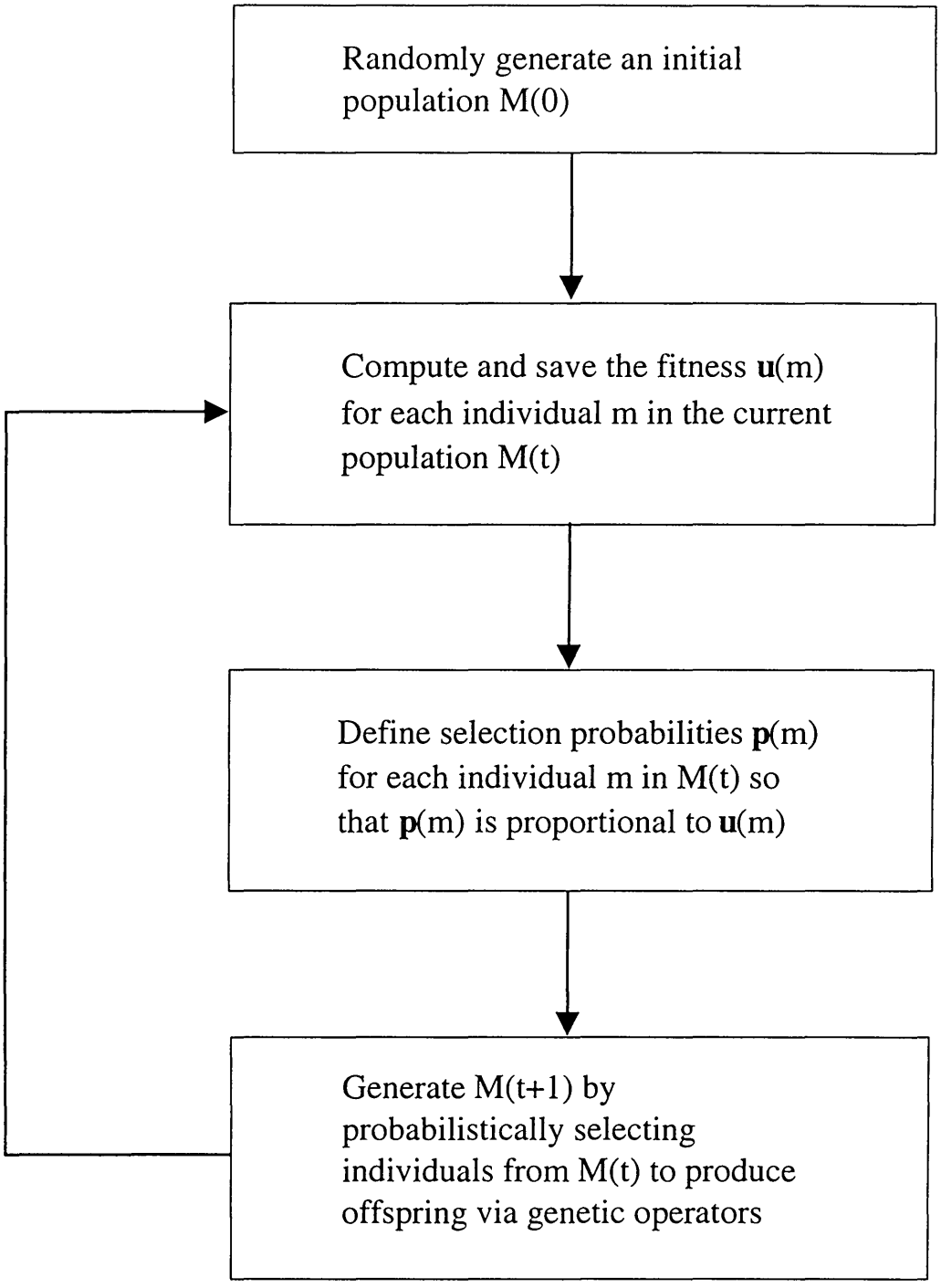


Figure 2.7 A simple genetic algorithm, re-drawn from [43].

Chapter 3

Materials and Methods

The program LVB was written to investigate use of simulated annealing in the search for parsimonious trees. It was written in the C language [44, 45], which is well understood and which may be compiled for a large number of systems. Code was written with the aims of high portability, ease of re-use and reasonable efficiency. Because most computers are idle most of the time, efficiency is less important than portability for a non-interactive program such as LVB. Some bugs in LVB 1.0A reduce its portability in theory, but in practice they are not serious or may be worked around with compiler options. LVB has been run with success on many different systems (Table 3.1, Table 3.2).

LVB works non-interactively, reading input from text files and writing output to text files (Table 3.3). This approach is well suited to batch jobs on servers, but also works on personal computers. No directory structure is assumed and file names are brief, consisting entirely of numbers and letters. This is highly portable, since some systems do not have directory structure, and many symbols other than letters and numbers can cause trouble in file names (e.g., the full stop is used as directory separator in Acorn RISC OS).

In this chapter I first describe how the program LVB may be used. This chapter was written after two years of use of the program and manual [31] (Appendix D) by myself and others. The description here, written after this experience, is intended to be simpler than the manual but no less accurate. For documentation of the source

Processor family	Operating system	Compiler	Remarks
Alpha	Digital Unix 4	EGCS	`-mieee` used; source file `lvb.h` changed to work around name space bug in the compilation system
Alpha	Linux 2	EGCS	requires reasonable IEEE compliance (compiler option `-mieee`)
K6	OS/2 version 3	IBM	binary fails during Test 2 with Watcom compiler
Pentium	Windows 98	Metrowerks	`NaN` may appear instead of `0` in statistics files
Pentium II	Windows NT 4	Metrowerks	none
PowerPC	MacOS 8	Metrowerks	none
R10000	IRIX 6	MIPS	none
R4000	IRIX 5	MIPS	none
R5000	IRIX 6	MIPS	none
SPARC	Linux 2	GNU	none
SPARC	Solaris 2	Sun	none
UltraSPARC	Solaris 2	Sun	none

Table 3.1 Systems on which the author has run, or knows results of running LVB with the test suites 'Test 1' and 'Test 2' (Appendix C). There was variation in statistics files for Test 2, either due to trivial differences in output of real numbers, especially numbers very close to zero, or due to output of `NaN` in place of some such numbers. Output of `NaN` and run-time failure after some kinds of compilation are due to reliance on IEEE-conformant arithmetic in source file `solve.c` (Appendix D). This was accidental. LVB was developed on Solaris and IRIX systems, where this problem did not reveal itself.

Processor family	Operating system	Compiler
80486	DOS	Microsoft
80486	OS/2	GNU
Alpha	Unicos	Cray
J90	Unicos	Cray
Pentium	Linux	GNU
Pentium	Windows 95	Metrowerks
RS-6000	AIX	IBM

Table 3.2 Systems on which the author, or a user known to the author, has compiled LVB, but on which (to the knowledge of the author) the Test 2 test suite has not been run.

code, however, the reader is referred to the manual which explains it in considerable detail. The source code has not changed since version LVB 1.0A 18 August 1997, which was released soon after LVB 1.0 to fix some minor bugs (Appendix E).

I then describe how LVB 1.0A was used to investigate the nature of simulated annealing in the search for parsimonious phylogenetic trees.

3.1 LVB Input Files

3.1.1 `data`

This file contains the data matrix to be analysed. The following text characters are not allowed in the file:

```
, : ; ( ) [ ] ` ' "
```

The null character is also not allowed. (Users who do not know what the null character is are unlikely to violate this rule. It is a special character used internally by C programs.)

Each row of the matrix must occupy one line. There must be at least four and no more than 16 383 rows. Rows must take the form

File	Description	Data flow
`data`	data matrix	in
`ini`x	initial trees; $x = 0$ for the first run and increases by 1 per run (e.g., `ini0`, `ini1`, ..., `ini`n, where $n =$ `runs` $- 1$)	out
`ini`x`r`	the current tree on beginning a re-start, where x is the number of the current run (e.g., `ini0r` if LVB was paused during the first run)	out
`lock`	lock file, preventing other copies of LVB from running in the same directory (folder); also, remains in place after a fatal error which cannot be logged to the `log` file	in/out
`log`	log file, indicating progress and errors	out
`params`	run-time configuration file	in
`pause`	file whose presence causes LVB to exit early, saving its internal state for a later re-start	in
`res`x	results trees; $x = 0$ for the first run and increases by 1 per run (e.g., `res0`, `res1`, ..., `res`n, where $n =$ `runs` $- 1$)	out
`save`	internal state details: LVB will read this to continue execution from the point where it was interrupted by a `pause` file	in/out
standard error	error message if an existing `lock` file is detected on start-up	out
`stat`x	progress for each run; $x = 0$ for the first run and increases by 1 per run (e.g., `stat0`, `stat1`, ..., `stat`n, where $n =$ `runs` $- 1$)	out
`stop`	file whose presence causes LVB to exit early, with output of results so far	in
`sum`	tabulated summary of progress for all runs	out

Table 3.3 Files used by LVB. `runs` is explained in Table 3.4. Files with data flow *in* are read, with data flow *out* are written, and with data flow *in/out* are either written or read. All named files are in the current directory (folder) where LVB is run. The standard error stream will usually be associated with the computer screen, but may be directed or mirrored to a named file on some systems.

name states

where *name* and *states* are separated by some white space. *name* is a string identifying the extant object, and must not itself contain white space. Nor must it equal none, since this has special meaning when configuring LVB, meaning, literally, 'none' in some circumstances (Table 3.4). *name* must be at least one and no more than 120 characters long. Each row's *name* must be different.

states contains symbols representing character states for the object. Each symbol must be a single machine character. Symbols may occur as one continuous string, or be separated from each other by white space. (Since spaces and tabs are ignored, they cannot themselves be used as character states.) There must be at least 1 and no more than 32 766 states per row. The number of states must be the same for each row. If this is not the case, it must be made so before using LVB, for example by sequence alignment. LVB is not able to align sequences itself.

White space at the start or end of the line is ignored. Both *name* and *states* are treated in a case-insensitive manner, so, for example, states A and a will be treated as equal.

An example data file is this, containing DNA sequences:

```
Species_1    ACGNN-AAT
Species_1a   CCGNN-AAG
Species_2    ACGNN-AAT
Species_3    ACGNN-AAG
```

Here, an underscore has been used for spacing within *name*, which is allowed. (The PHYLIP tree-drawing programs drawgram and drawtree expect this.) The start of each *states* string does not have to be at the same position in each line so, for example, the following would be an equally good version of the above matrix:

Description	Type	Name	Acceptable values	Default value
Heuristic search	string	`method`	`anneal`, `hillclimb` or `randomwalk`	`anneal`
Number of independent searches	integer	`runs`	$0 <$ `runs` $<$ `MAX_LINT`	`1`
Trees to examine per search	integer	`trees`	$0 <$ `trees` $<$ `MAX_LINT`	`20000`
Character states to be regarded as equivalent	string	`equivalent`	up to 32 comma-separated strings each of up to 32 characters, or `none`	`none`
Symbol in data matrix indicating same state as in the first row	string	`matchchar`	a single machine character, or `none`	`none`
Symbols whose presence in a column of the data matrix will cause the column to be ignored	string	`ignorechars`	a string of one or more machine characters, or `none`	`none`
Outgroup	string	`outgroup`	the name of a row in the data matrix, or `none`	`none`
Seed for pseudorandom number generator	integer	`seed`	$0 \leq$ `seed` $\leq$ `MAX_SEED`	(varies)
Text to precede column titles in statistics files	string	`titleprec`	string of up to 32 machine characters containing no white space, or `none`	`none`
Amount of information to be logged	integer	`verbose`	`0` or `1` (`0` for few details, `1` for many)	`1`

Table 3.4 Variables used to configure LVB, excluding those that are specific to simulated annealing. `MAX_LINT` is a system-dependent constant such that `MAX_LINT` $\geq 2^{32}/2 - 1$. `MAX_SEED` is a system-dependent constant such that $2^{16}/2 - 1 \leq$ `MAX_SEED` $\leq 9 \times 10^8$. The default value of `seed` is taken from the system clock. If this is unavailable or unsuitable, an attempt to rely on the default will cause LVB to log an error message and exit.

```
Species_1 ACGNN-AAT
Species_1a CCGNN-AAG
Species_2 ACGNN-AAT
Species_3 ACGNN-AAG
```

It would also be possible to leave spaces or tabs between each state, for example:

```
Species_1 A C G N N - A A T
Species_1a C C G N N - A A G
Species_2 A C G N N - A A T
Species_3 A C G N N - A A G
```

The intention of the format for LVB's `data` file was to allow easy interaction with spreadsheets. A spreadsheet would be able to export tab- or space-delimited text in a form immediately suitable for use by LVB. This has not proved very useful, because non-specialist spreadsheets are rarely used to manipulate data matrices. Fortunately, PHYLIP (Section G.2) and NEXUS [46] files may be converted to LVB's format without too much effort using a text editor (e.g., `vi` or `emacs`) or text-oriented scripting languages (e.g., `awk`) [47, 48]. A program converting various formats, including PHYLIP, to LVB format is available as part of the Web-based LVB service at the Institut Pasteur (Section G.4).

3.1.2 `params`

This file contains instructions for run-time configuration of LVB. Instructions take the form

`var val`

This assigns value *`val`* to the variable *`var`*. *`var`* is the name of one of 15 variables known to LVB (Table 3.4, Table 3.5). *`var`* and *`val`* occur on the same line and must be separated by some white space. White space at the start or end of the line is

Parameter	Type	Name	Range acceptable to LVB	Default value
Initial temperature, T_0	real	`t0`	$0 <$ `t0` ≤ 1	`1.0`
Second temperature, T_1	real	`t1`	$0 <$ `t1` $\leq$ `t0`	`0.9`
Maximum changes proposed at a temperature	integer	`maxpropose`	$0 <$ `maxpropose` $<$ `MAX_LINT`	`5000`
Maximum new trees at least as good as the then-current tree accepted at a temperature	integer	`maxaccept`	$0 <$ `maxaccept` $<$ `MAX_LINT`	`50`
Maximum consecutive temperatures allowed at each of which less than `maxaccept` new better trees are accepted, per search	integer	`maxfail`	$0 <$ `maxfail` $<$ `MAX_LINT`	`2`

Table 3.5 Variables used to control the simulated annealing search in LVB. They are ignored in the absence of `method anneal`. `MAX_LINT` is explained in the caption to Table 3.4.

ignored.

val is not allowed to contain white space. *val* may be of type *integer*, *string* or *real*. Integers are not allowed to contain punctuation or letters. For example, `10 000`, `10,000`, `10000.0` and `ten thousand` are not acceptable values for an integer, but `10000` is. With real values, standard form is possible, but $x \times 10^n$ must be expressed as *x*E*n* or *x*e*n*. For example, 2.6×10^{-4} should be written as one of the following: `0.00026`, `2.6e-4` or `2.6E-4`. It is not necessary for real values to contain a decimal point: both `1` and `1.0` are acceptable.

Comments are allowed in the file. They are introduced by `%` at the start of a line:

```
%comment_line
```

comment_line, the remainder of the line, will be ignored. Blank lines, or lines containing only white space, are also ignored.

The purpose of each of the variables is explained in Table 3.4 and below for general variables, and in Table 3.5 for variables controlling simulated annealing only. Some variables are only meaningful in certain contexts (Table 3.6).

3.1.2.1 method

`method` gives the heuristic search to use. It may have value `anneal`, for simulated annealing (Chapter 2), `hillclimb`, for hill climbing, or `randomwalk`, for a random walk. LVB implements hill climbing using the extremely simple algorithm in Section 2.2, so use of `method hillclimb` is not recommended.

The random walk *always* accepts the proposed update to a tree, and, as such, is a very poor heuristic. It is only provided for comparison.

3.1.2.2 runs

`runs` gives the number of independent searches to perform. For example,

Variable	Value used
`equivalent`	always
`ignorechars`	always
`matchchar`	always
`maxaccept`	with `method anneal`
`maxfail`	with `method anneal`
`maxpropose`	with `method anneal`
`method`	always
`outgroup`	always
`runs`	always
`seed`	always
`t0`	with `method anneal`
`t1`	with `method anneal`
`titleprec`	with `verbose 1`
`trees`	with `method hillclimb` or `method randomwalk`
`verbose`	always

Table 3.6 Dependencies of variables used to control LVB. Some variables may or may not be used, depending on the value of another variable. Unused variables are ignored.

```
method anneal
runs 1
```

will cause LVB to perform one simulated annealing search, but

```
method anneal
runs 10
```

will cause it to perform ten simulated annealing searches, each starting from scratch.

3.1.2.3 trees

`trees` is the number of trees to examine per search. For example, the following will cause LVB to consider 10 000 trees in a single hill climbing search.

```
method hillclimb
trees 10000
```

After 10 000 trees have been considered, the search will cease, irrespective of its progress so far.

3.1.2.4 equivalent

Sometimes, it may be desired to treat certain character states as identical. This can be done without modifying the data matrix using the variable `equivalent`. This is best described through example.

With the following DNA sequence matrix,

```
Sequence4 aagrr
Sequence3 yayac
Sequence2 aggty
Sequence1 cagac
```

adenine is represented by `a`, cytosine by `c`, thymine by `t` and guanine by `g`. `r` is used to mean purine (`a` or `g`) and `y` is used to mean pyrimidine (`c` or `t`), without narrowing down the state to a particular residue. Letting LVB interpret this matrix 'as is' would be a mistake, since LVB does not 'understand' this usage, and each of `a`, `c`, `t`, `g`, `r` and `y` would be treated as a unique state. One way to remedy this would be to reduce each position in the sequence to two states, 'purine' and 'pyrimidine'. This may be done as follows:

```
equivalent agr,cty
```

Exactly the same effect could be achieved by editing the data matrix, for example to

```
Sequence4 rrrrr
Sequence3 yryry
Sequence2 rrryy
Sequence1 yrrry
```

or

```
Sequence4 11111
Sequence3 21212
Sequence2 11122
Sequence1 21112
```

in which case the default of `equivalent none` could be used. (It would be more usual to use `0` and `1` as states of a two-state character. Using these symbols would have exactly the same effect as using `r` and `y`, or `1` and `2`. However, there has been confusion in the past since some methods other than Fitch parsimony [8] expect `0` to represent 'absence' and `1` to represent 'presence', with the two states treated in different ways [49].)

Where there is more than one equivalent list, as in my example, they must all be given in the same line, each separated from its neighbour by a single comma. There must be no space before or after the comma.

3.1.2.5 `matchchar`

Sometimes, only the character states of the first row are given in full, and in subsequent rows, a symbol is used to indicate that the state is the same as in the first row. A full stop is a common choice for this purpose. For example, the matrix in Section 3.1.1 might be written in this form:

```
Species_1    ACGNN-AAT
Species_1a   C.......G
Species_2    .........
Species_3    ........G
```

By default, LVB will interpret a matrix in this form incorrectly, by treating . as a state in its own right. The position of the first row in the results tree(s) would be random since, according to LVB, it would share no state with any other row. The purpose of `matchchar` is to fix this problem. For this matrix, the following should appear in the `params` file:

```
matchchar .
```

This will cause LVB to translate each instance of . into the state present at that position in the first row. (The translation is applied to LVB's copy of the matrix in computer memory. The `data` file is not changed.) The same effect could be achieved either by replacing each state in the first row with . or by replacing each instance of . in the matrix with an absolute state prior to the analysis. Such a change would allow use of the default `matchchar none`.

3.1.2.6 ignorechars

To ignore columns in the matrix containing any one or more specific states, `ignorechars` may be used. For example, if - is used for missing data, all columns containing missing data could be omitted from the analysis using

```
ignorechars -
```

If more than one symbol is given, any of the symbols will cause a column to be ignored. For example, using the standard codes for DNA nucleotides (Section G.5), the following causes all columns containing missing, ambiguous or unknown nucleotides to be ignored:

```
ignorechars RYWSMKHBVDN-?+
```

There must be no white space between the states. Nor should there be any commas, unless the comma itself is a state to be ignored.

3.1.2.7 outgroup

LVB will usually output results trees in unrooted form. To make it output rooted trees instead, `outgroup` is set to the name of an object. This will cause trees to be output with the root as ancestor of that object. The root's other descendent will consist of the ancestor of all objects *excluding* itself, the root and the outgroup. LVB does not allow an outgroup to consist of more than one object.

Naming an outgroup may have another effect, in addition to rooting the tree. In rooted trees, LVB will collapse edges that can never have length under the accelerated transformation (ACCTRAN) model of character state change [50], giving trichotomies or higher-order branching events (Figure 1.3).

3.1.2.8 seed

The seed for the pseudorandom number generator may be given as the value of `seed`. The `uni` pseudorandom number generator is used [51]. This must be seeded with four integers, but for simplicity, the true seeding function has been 'wrapped' with a function that splits up a single integer for this purpose, so that the LVB user only has to supply one seed per analysis. The wrapper is based on the `rmarin` subroutine [52].

3.1.2.9 titleprec

To 'comment out' column titles in statistics files (Section 3.2.5), a value may be assigned to `titleprec`. For example, to make it easier to use statistics files with the GNUPLOT graph-plotting program (Section G.7), use

```
titleprec #
```

This will cause #, followed by a space, to precede column titles on the first line of each statistics file.

The same effect could be achieved by editing the statistics files when LVB has finished. However, for very large files, this could be inefficient or lead to corruption of data.

3.1.2.10 `verbose`

This controls how many details are logged by LVB as it runs. With

```
verbose 1
```

`statx` files (Section 3.2.5) are written for each run. To prevent LVB writing such files,

```
verbose 0
```

is used.

3.1.3 `pause`

Presence of a readable text file called `pause` will make LVB exit early, saving its state for a re-start later. `pause` may take some time to have an effect, since LVB only checks for its presence once per 5 000 trees examined.

The `pause` file is deleted by LVB.

Use of `pause` is not recommended where the analysis is intended to be repeatable. Rather than saving the internal state of the pseudorandom number generator, it saves a random integer, for use as the seed (Section 3.1.2.8) on re-start. Repeating the same analysis would require both that this same value were used, and that LVB were paused at exactly the same stage, which may prove difficult in practice.

3.1.4 `stop`

Presence of a readable text file called `stop` will make LVB exit early, saving the best trees found so far. `stop` may take some time to have an effect, since LVB only checks

for its presence once per 5 000 trees examined.

The `stop` file is deleted by LVB.

3.2 LVB Output Files

3.2.1 `ini`*x*

The initial tree for each search is output, in unrooted form, to a file whose name takes the form `ini`*x*, where *x* is the number of the run.

3.2.2 `ini`*x*`r`

On a re-start after pausing LVB (see Section 3.1.3), the current tree at that point is written to a file whose name takes the form `ini`*x*`r`, where *x* is the number of the run.

3.2.3 `log`

LVB always writes a brief progress report to a file called `log`. Near the start of this file, values of pertinent variables are repeated. This may be used to check that the `params` file has been understood.

On an error that is not fatal but which indicates some quite serious problem, LVB will log a line beginning with the following:

```
ERROR:
```

after which there will be some explanation of the problem. On an error which is even more serious and prevents LVB from continuing, LVB will usually print a line beginning with the following:

```
FATAL ERROR:
```

after which, again, there will be some explanation of the problem. Such explanations may occupy several lines.

If there is an error that is so very serious it cannot be logged, this may be discovered by *absence* of a line either beginning

```
End time:
```

or consisting of

```
End of execution
```

at the end of the `log` file after LVB has finished. The `lock` file will also remain in place under such conditions (Section 3.3.1).

3.2.4 `res`*x*

Results trees for each run are written to a file whose name takes the form `res`*x*. Trees may be unrooted or rooted, according to the value of `outgroup` (Section 3.1.2.7). All best trees found per run are given.

3.2.5 `stat`*x*

Each `stat`*x* file contains detailed information on the progress of run *x*. Columns are tab-delimited, and there is one row per tree examined. The columns are explained in Table 3.7. `stat`*x* files may or may not be output, depending on the value of `verbose` (Section 3.1.2.10).

3.2.6 Standard Error Stream

If LVB is apparently running in the current directory (folder) already, the new copy will not generate any file output. Instead it will write an error message to the standard error stream (usually the screen) and exit. This is the only situation in which LVB may write to the screen. It cannot write to the `log` file, because this may be in use by the copy of LVB already running. The mechanism is explained in Section 3.3.1.

If LVB *has* run in the current directory, but is no longer running, LVB will launch without trouble. Its output files will over-write those output by the previous run,

Heading	Contents	Present with `method`
`Tree`	Number of the current tree in the search (first tree is numbered 0)	`anneal`, `hillclimb` or `randomwalk`
`BestLen`	Length of the best tree encountered in the search so far	`anneal`, `hillclimb` or `randomwalk`
`TempNo`	Number of the current temperature level (first temperature, T_0, is numbered 0)	`anneal`
`Temp`	Current temperature level	`anneal`
`CurrLen`	Length of the current tree	`anneal`, `hillclimb` or `randomwalk`
`PropLen`	Length of the proposed replacement tree	`anneal`, `hillclimb` or `randomwalk`
`ProbAcc`	Probability of accepting the replacement tree	`anneal`

Table 3.7 Columns in LVB statistics files. Columns occur in the order they are given in this table. Some columns are only present with `method anneal`.

unless this is a re-start (Section 3.1.3).

3.3 LVB Files Used for Input or Output

3.3.1 `lock`

LVB writes a small file called `lock` to the current directory (folder) when it runs. Attempting to run LVB in a directory that already contains a file called `lock` will fail, with an error message to the standard error stream. This will usually prevent two copies of LVB from running in the same directory and writing to the same output files. (There is a small window of opportunity, between start of execution of LVB and the file `lock` becoming readable by other processes, where such mayhem could still occur. This is only likely to happen if the first copy of LVB is delayed very soon after starting, for example by a batch scheduling system.)

The `lock` file is deleted on exit, except when there has been an error so serious that it cannot be logged to the `log` file, in which case the `lock` file is deliberately left in

place to indicate this. Once the user has made sure that the copy of LVB running in a directory has really terminated, it is safe to delete any `lock` file and perhaps run LVB again.

3.3.2 save

A `save` file is generated when LVB exits early due to presence of a `pause` file (Section 3.1.3). If present when LVB starts, data in `save` will be used to continue execution at the point it was paused. The `params` and `data` files must not have changed during the interval. If they have, the effect is undefined.

The `save` file should not normally be edited, since this may have undefined effect.

The `save` file is deleted by LVB on successful restart.

3.4 The Experiments

The data used comes from the 500-sequence analysis of seed plant relationships performed by Chase *et al.* [53]*. Sequences were for the chloroplast gene *rbc*L, which codes for the large subunit of ribulose-1,5-bisphosphate carboxylase/oxygenase (RuBisCO). The sequences represent seed plants, from gymnosperms to monocotyledons, including most families of angiosperm. The original analysis was revolutionary in scope and its impact will forever be felt in plant systematics.

The total number of trees for 500 sequences is 10^{1278}. The phylogenetic analysis of this data performed by Chase *et al.* ran for four weeks on an Apple Quadra computer, during which 9.7×10^6 trees were examined. The consensus tree resulting from that analysis is presented in Figure 3.1.

The whole data matrix had a considerable number of missing entries. Because LVB cannot fill missing entries, I removed the 135 sequences that had most missing data. The 365 sequences remaining form what I refer to as the *general* data matrix. To

*The 500-sequence matrix is referred to as a 499-taxon matrix in the paper. The 500 sequences included two different sequences from *Canella* (Canellaceae). In the published tree, these were reduced to a single taxon.

Variable	Levels
t0	1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001
t1	$0.99 \times T_0$, $0.099 \times T_0$, $0.0099 \times T_0$
maxpropose	50, 500, 5000
maxaccept	5, 50
maxfail	1, 10, 100
runs	1, 5, 10

Table 3.8 Levels of LVB variables investigated for each data matrix.

examine the effect of matrix size on results, a smaller matrix was extracted from the general data matrix. This consisted of all monocotyledon sequences from the general data matrix, plus *Brasenia* (Cabombaceae) to act as outgroup. *Brasenia* was chosen because, in addition to being in the 365-sequence matrix, it falls in the sister group to the monocotyledon clade in the 500-sequence tree of Chase *et al.*, and also because it is in Nymphaeales. This order is regarded, on the basis of morphological and fossil evidence, as similar to the ancestors of the monocotyledons [54]. In all, the *monocot* data matrix contains 50 sequences. The names of the sequences used for each data matrix are given in Appendix A.

After some runs performed in an informal manner, it was decided to investigate all combinations of the levels of the parameters given in Table 3.8 for each data matrix. This gives levels of T_1 (LVB variable `t1`) in relation to T_0 (LVB variable `t0`), because the second temperature is of little interest in itself, but the relation between T_0 and T_1 controls the *rate* of decrease of temperature levels (Section 2.4). An absolute value of `t0` was used in the `params` file for each analysis, as required by LVB (Table 3.5).

Many levels of `t0` were used so it could confidently be treated as a continuous variable or *covariate* when analysing the results. This would have been useful for all variables, but was not possible due to the large amount of computer time required even for the current experiment.

The three levels of all other simulated annealing parameters except `maxaccept`

allowed them to be treated as covariates as well, but, due to the smaller number of levels, the requirement for and suitability of transformations (Section 4.2.1.5) is less easy to judge.

Only two levels of `maxaccept` were used, with the highest equal to the lowest level of `maxpropose`. Introducing a level of `maxaccept` in excess of the minimum `maxpropose` would serve no purpose, but would increase the interaction between the two variables. Because, with two data points, it is impossible to detect whether a relationship is linear or of some other kind, `maxaccept` could only be treated as a categorical variable of *factor* when analysing the results. The data matrix used (monocot or general) was also treated as a factor. Had other data matrices been used as well, it would still be difficult to regard the nature of the data matrix as a continuous variable. The monocot data matrix consists of sequences that are, on average, more closely related than those in the general data matrix. This could have an effect additional to the simple effect of data matrix size.

A different value of `seed` was used for every analysis, except in 11 cases where one of nine seeds was inadvertently re-used. For the general data matrix, many values were obtained by default from the computer system clock (Section 3.1.2.8). During the summer of 1998, the clock of Unix systems began to return a value too large to be used for this purpose. Use of the default then caused LVB to exit with an error message. Remaining analyses of the general data matrix were performed using seeds chosen by the author. With the monocot data matrix, for which all analyses were performed during 1999, most seeds were generated using the `randbetween` subroutine of Excel (Microsoft Corporation) to give suitable pseudorandom integers from a uniform distribution.

For the general data matrix, `outgroup none` was used, because an outgroup consisting of more than one sequence would be appropriate and this is not allowed by LVB (Section 3.1.2.7). For the monocot matrix, `outgroup Brasenia` was used. The value of `outgroup` does not affect the search for the most parsimonious tree.

Other than `seed` and the parameters under investigation (Table 3.8), the values of

Parameter	Value
method	anneal
equivalent	none
matchchar	.
ignorechars	KYMSWRN-
titleprec	#
verbose	1

Table 3.9 Variables of fixed value for all analyses performed.

LVB's parameters for all analyses were as in Table 3.9. The value of `ignorechars` was chosen to cause all columns containing missing or uncertain data to be ignored. (The value of `ignorechars` is not adequate to ensure this with *all* nucleotide data matrices, but it was suitable for the two matrices used.)

Due to the large size of the experiment, runs were performed on various available computer systems. Because of this, comparison of either elapsed or CPU time between analyses is not meaningful. Instead, number of trees examined in an analysis may be used as an indicator of an underlying, 'ideal' time. On a given system with a given load, actual run time will be closely correlated with number of trees examined.

Where available, the name of each machine used is given below, omitting the terminal `.ed.ac.uk` part of the IP address. However, details change fast: the X-lab no longer exists and `waverley` has become part of a larger 'Waverley cluster'. Compilation of LVB for IRIX 5 and IRIX 6 R4000 and R4000-like systems was performed using MIPS C (Silicon Graphics, Inc.), invoked as `cc -mips2 -O -ansi -w2 -fullwarn`; for `nessie.bch` (IRIX 6 R10000), using MIPS C, invoked as `cc -64 -r10000 -O -ansi -ansiE -IPA:echo=on,aggr_cprop=ON,space=500 -INLINE:list=on,must=ssupdate -OPT:IEEE_arithmetic=1,Olimit=0,roundoff=0,swp=ON -TARG:platform=ip25 -fullwarn`; for most SPARC Solaris 2 machines (`waverley`, `holyrood`, EUCS X-lab machines), using Sun C (Sun Microsystems),

invoked as `cc -Xc -native -xO4 -dalign`, once on each of `waverley` (UltraSPARC), `holyrood` (SuperSPARC), and an X-lab workstation (microSPARC); for the Waverley cluster of various SPARC machines running Solaris 2.6, using Sun C, invoked as `cc -Xc -xO4 -dalign` (on `waverley-p1`); for `swann.bru` and `darwin.bru` (SPARC Linux 2), using GNU C (Free Software Foundation), invoked as `gcc -ansi -O3 -fno-short-enums`; for `nova.bru` (Alpha Linux 2), using EGCS 2.91.60 (Free Software Foundation), invoked as `gcc -ansi -mieee -O3 -fno-short-enums`; for `orion.bru` (K2 Windows 3.1), as in the LVB 1.0A Windows 3.1 release notes (Section F.3), and 'XMS Memory Required' was set to 1025 KB and 'XMS Memory Limit' to -1 (meaning, as much as necessary) using `PIFEDIT.EXE`; for `niceone`, of no fixed IP address (Pentium Windows 98), as in the LVB 1.0A Win32 release notes (Section F.2).

3.4.1 Analysis of Results

Firstly, an attempt was made to summarise the relationship between length of the shortest tree found and number of trees examined. This will allow future estimation of run-time required to obtain a given quality of result. This would help researchers efficiently allocate computer time to phylogenetic analyses.

Secondly, the effects of the combinations of data matrix and factors in Table 3.8 on both length of the shortest tree found and number of trees examined were analysed using the general linear model (GLM) implemented by Minitab (Minitab, Inc.) for Windows (Microsoft Corporation). Minitab version 11.21 was used under Windows 3.1 on a Pentium MMX-based computer and under Windows 3.11 on a 6x86-based computer, and Minitab 12.23 was used under Windows 98 (with Service Pack 1) on a Pentium-based computer.

GLM includes anova, or analysis of variance using categorical predictors, and regression, which uses continuous predictors. There may in principle be any number of continuous and/or categorical predictors, although for Minitab imposes an upper limit of nine. When using GLM, a *model formula* is proposed, giving a relationship

between the response variables and the predictors under investigation. GLM then assesses the statistical significance of each predictor, and fits equations relating the responses and predictors. Interaction between predictors, and higher-order interactions (interactions involving interactions) may be included in the model formula. By default, Minitab does not always output values sufficient to discover the full fitted equation. However, these are given after increasing the level of output by the command `brief 3`.

Since, *a priori*, we may assume all simulated annealing parameters and some interactions between them to have a significant effect on length of the shortest tree found and run time (even if there are two few data points to reveal this), GLM is of most interest here for fitting equations. Fitted equations will allow prior estimation of suitable simulated annealing parameters for an analysis, according to the nature of the data matrix and the desired run time and quality of result.

The assumptions of GLM are discussed, with reference to this experiment, in Section 4.2.1.

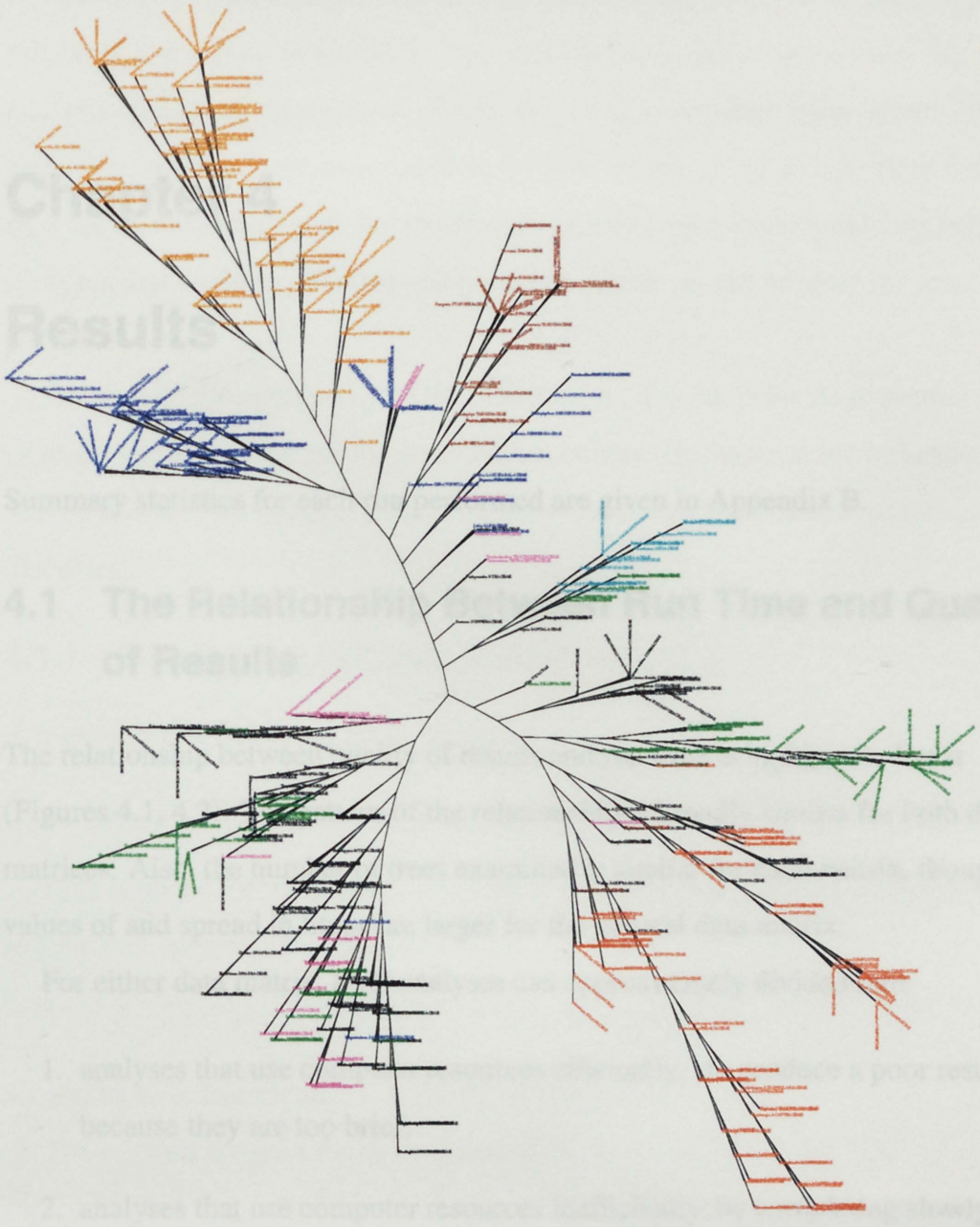


Figure 3.1 Unrooted version of the tree found by Chase *et al.* [53] in their 500-sequence analysis of seed plants. I have coloured sequence names according to their recent, but pre-Chase, higher-level classification. Sub-classes of dicotyledons, delimited according to Cronquist's *Integrated System* [54, 55] are coloured as follows: Magnoliidae are blue, Hamamelidae are pink, Caryophyllidae are turquoise, Dilleniidae are green, Rosidae are black and Asteridae are red. Monocotyledons are orange and Gymnosperms are brown. As originally presented, the tree was rooted with gymnosperms as outgroup, which left *Ceratophyllum* (Ceratophyllaceae) as sister group to all other angiosperms. *Ceratophyllum* may just be distinguished in the figure, as a lone blue label below the cluster of brown.

Chapter 4

Results

Summary statistics for each run performed are given in Appendix B.

4.1 The Relationship Between Run Time and Quality of Results

The relationship between quality of results and run time is highly non-linear (Figures 4.1, 4.2). The pattern of the relationship is broadly similar for both data matrices. Also, the number of trees examined is similar for each matrix, though the values of and spread in Steps are larger for the general data matrix.

For either data matrix, LVB analyses can approximately divided into:

1. analyses that use computer resources efficiently, yet produce a poor result because they are too brief;
2. analyses that use computer resources inefficiently, by completing slowly and producing a poor result;
3. analyses that use computer resources efficiently, by running slowly but producing a good result.

(1) consists of those data points on Figures 4.1 and 4.2 which may be connected by a near-vertical line towards the minimum of the Trees axis. (2) consists of those data points which lie to the right of (1), yet remain at a high position on the Steps axis.

(3) consists of those data points which may be connected by a near-horizontal line at relatively low values on the Steps axis. Among these, Steps varies little. The ideal analysis would fall among these points, but not at a very high value on the Trees axis, since this indicates excessive run-time for little return. If (1) and (3) were connected by a curve, the region near, but to the right of the turning-point would represent analyses that are both effective and quite fast. These would be ideal for most purposes.

Because of the complexity of the relationship, it is not possible to predict quality of result from run time or *vice versa* with accuracy. However, some indication would still be useful. Two kinds of relationship are considered and their fit to the data discussed.

4.1.1 Fitting a Logarithmic Relationship

A logarithmic relationship takes the form

$$y = m \ln x + C \tag{4.1}$$

This is tested for the general data matrix in Figure 4.3. The linear regression line suggests an approximate relation

$$\text{Steps} = -776(\ln \text{Trees}) + 21600 \tag{4.2}$$

As can be seen from Figure 4.3, the fit is rather poor.

The relation is tested for the monocot data matrix in Figure 4.4. The linear regression line suggests an approximate relation

$$\text{Steps} = -60.0(\ln \text{Trees}) + 2280 \tag{4.3}$$

The fit is worse than for the general data matrix, as reflected in the lower R^2.

4.1.2 Fitting an Allometric Relationship

An allometric relationship takes the form

$$y = \alpha x^{\beta} \tag{4.4}$$

Taking a logarithm of this equation gives a linear relationship, e.g.,

$$\ln y = \ln\alpha + \ln x^{\beta} \tag{4.5}$$

$$= \ln\alpha + \beta \ln x \tag{4.6}$$

This is tested for the general data matrix in Figure 4.5. The linear regression line suggests an approximate relation

$$\ln \text{Steps} = -0.0611(\ln \text{Trees}) + 10.121 \tag{4.7}$$

i.e., $\beta = -0.0611$ and $\ln\alpha = 10.121$, giving the approximate allometric relationship

$$\text{Steps} = \frac{24900}{\text{Trees}^{0.0611}} \tag{4.8}$$

The fit is poor, but slightly better than the fit for the logarithmic relationship, as shown by the higher R^2.

The relation is tested for the monocot data matrix in Figure 4.6. The linear regression line suggests an approximate relation

$$\ln \text{Steps} = -0.0346(\ln \text{Trees}) + 7.7579 \tag{4.9}$$

i.e., $\beta = -0.346$ and $\ln\alpha = 7.7579$, giving the approximate allometric relationship

$$\text{Steps} = \frac{2340}{\text{Trees}^{0.0346}} \tag{4.10}$$

The fit is worse than for the general data matrix. On the other hand, as with the general matrix, the fit is slightly better than for the logarithmic relationship.

4.1.3 Summary

The relationship between steps on the shortest tree found and number of trees examined in the search, for the current experiment, is very approximately allometric. The fitted relationship is given for the general data matrix by Equation 4.8 and for the monocot data matrix by Equation 4.10.

4.2 The Effects of LVB Parameters on Run Time and Quality of Results

4.2.1 Assumptions of GLM

The assumptions of GLM are discussed here with respect to the current experiment. They are a mix of the assumptions of Model I anova and Model I regression (for which, see [56]).

4.2.1.1 Random Samples

GLM assumes each data point is an observation drawn at random from a hypothetical population of possible data points for the treatment it received. (A treatment is a given combination of levels of the predictors.)

Whether the assumption of random samples is met here depends on the quality of the pseudorandom number generator used (Section 3.1.2.8). Use of a poor generator, or repeated use of equal seeds, could lead to bias.

It may be assumed that the samples were random in the current experiment. See also Sections 4.2.1.2 and 4.2.2.1.

4.2.1.2 Independent Samples

GLM assumes no *autocorrelation* between data points. Non-random effects due to predictors absent from the model formula are not allowed.

Some analyses launched at similar times used rather similar seeds for the pseudorandom number generator, because these values were derived from the computer system's clock (Section 3.4). The apparent autocorrelation for the general data matrix (Figures 4.7, 4.8) is because LVB analyses with similar treatments were often launched at almost the same time. Seeds for analyses of the monocot data matrix were not derived from a clock, and there is less evidence of a correlation between seed and the response variables here (Figures 4.9, 4.10). The effects of seed are analysed statistically in Section 4.2.2.1.

The clumped distribution of lnTrees in Figures 4.8 and 4.10 can be explained by constraints on the number of trees examined imposed by the predictors. For example, the lower limit on number of trees that could be examined in any run is `maxpropose` + 1.

Since several different computer systems were used (Section 3.4), another possibility of autocorrelation would be system-dependence in the program LVB, causing results from one computer system to be inherently more similar to those from another. Although the `uni` pseudorandom number generator (Section 3.1.2.8) lacks system dependence, other parts of LVB might be dependent on the system. For example, rounding of real values may differ, and this could have an effect on acceptance of worse trees during simulated annealing. Such an effect could interact with the main predictors in the current experiment, since similar analyses were often performed on the same system. The possibility of autocorrelation due to system could have been avoided by only using one system for all LVB analyses, or assigning analyses to systems at random. However, this would not have allowed the experiment to be completed so quickly.

Overall, we may assume the data points to be independent, although this assumption is hard to support without extensive analyses of the source code of LVB and further run-time tests. (Some such tests are planned: see Section 6.7.)

4.2.1.3 Fixed Effects

GLM assumes *fixed* effects*. In other words, levels of predictors must be controlled precisely by the investigator. *Random* effects, due either to predictors observed by the investigator but whose levels are not under his control, or to predictors measured with error, are not allowed.

This assumption is undoubtedly met in the current experiment for all the predictors in Table 3.8 as well as data matrix (monocot *vs* general). The levels were set without error by the author prior to each analysis. The assumption is not met for the predictor

*This assumption may be changed through use of the `test` sub-command of Minitab.

seed, because values of seed were mainly obtained in an unpredictable way (Section 4.2.1.2). However, seed was not included in the main analysis.

4.2.1.4 Normally Distributed Error Term

As already stated (Section 4.2.1.1), each observation must be randomly drawn from a population of potential observations for the same treatment. The difference between the data point and the mean for the population is known as its *error*. GLM assumes error to be distributed normally.

Since the current experiment has no replication (each treatment is represented by only one data point), this assumption cannot be tested prior to use of GLM.

However, the assumption of normality is more likely to be met if the response variables are normally distributed. This was not true of the responses as recorded (Figures 4.11, 4.12, 4.13, 4.14). A logarithmic transformation was found to give an approximately normal distribution for run time (Trees). A logarithmic transformation was also found to increase the normality of the distribution of length of the shortest tree found (Steps), although the effect was much more limited due to the multimodal distribution of this response.

We may assume this assumption is met for the response run time, after transformation, but not for tree length.

4.2.1.5 Linearity

Apart from effects of interactions, the effect of a covariate on a response is assumed to be linear.

After trial and error with visual examination, all covariates other than runs (and, in Section 4.2.2.1, seed) were transformed to their natural logarithm prior to using GLM. As can be seen from graphs of each response against each (perhaps) transformed covariate (Figures 4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, 4.31, 4.32, 4.33, 4.34), this led to an approximately linear relationship with the response in each case. The transformations

also tended to increase homogeneity of variance (Section 4.2.1.6). The relationships between responses and `runs` were approximately linear and required no transformation (Figures 4.31, 4.32, 4.33 and 4.34).

The assumption of linearity is approximately met.

4.2.1.6 Homogeneity of Variance

The variance of each population from which observations are drawn (Section 4.2.1.1) is assumed to be equal. This is known as homogeneity of variance or *homoscedasticity.*

Since the experiment lacks replication, this assumption may not be tested directly. However, the plots of transformed responses against (perhaps) transformed covariates for each data matrix (Figures 4.15, 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, 4.31, 4.32, 4.33, 4.34), may be used. Each vertical row of points should be normally distributed.

For a given data matrix, scatter of points for each value of a given transformed factor seems approximately equal, although this is hard to judge in some cases, especially for the response ln(`Steps`), due to the multi-modal, skewed distribution of this response (e.g., Figures 4.15, 4.27). Also, the different symbols used for two levels of `maxaccept` suggest this has no serious effect on variance. However, variance of the responses is *not* equal for the two data matrices. The scatter of points for given axes with the general data matrix is consistently larger than that for the same axes with the monocot data matrix.

We conclude that this assumption of GLM is not met.

4.2.1.7 Summary

Many of the assumptions of GLM are met closely by the current experiment. Distributions and linearity have only been checked visually, rather than numerically. However, the main violation, of the assumption of homogeneity of variance (Section 4.2.1.6), is obvious without any numerical tests.

Fortunately, the aim of GLM *in this experiment* is to fit equations relating the factors to the responses rather than to test statistical significance. p is used more as an indication of relevance rather than a probability. Because of this, the importance of all assumptions is reduced.

4.2.2 Preliminary Analyses

4.2.2.1 Random Number Seed

Two GLMs were performed as a crude test of the random number generator. The p-values from these tests are not reliable because

1. `seed` is not a fixed effect (Section 4.2.1.3);
2. there are many departures from randomness that will not be revealed by GLM because they are non-linear;
3. patterns in the series of pseudorandom numbers generated will only indirectly effect the responses.

However, the tests still provide a small level of assurance.

In the first GLM, the model formula included all first-order interactions (interactions between main predictors), and all higher-order interactions (interactions between interactions). In Minitab's notation,

```
GLM lnSteps lnTrees = seed|matrix|lnT0|lnT1toT0 &
|lnMaxpropose|maxaccept|lnMaxfail|runs;
covariates seed lnT0 lnT1toT0 lnMaxpropose lnMaxfail runs.
```

In Minitab's command language (entered in the 'Session Window' of Minitab for Windows), the command for GLM is `GLM`. The model formula is given as its argument. In the model formula, responses are placed to the left of the equals sign and predictors to the right. | is used to separate predictors which, in addition to their main, additive effect, interact with each other in every possible way. (They do not, however,

interact with themselves, unless this is explicitly included in the model.) & allows a line to be continued, a semicolon indicates that a sub-command starts on the next line, and a full stop ends the whole command. By default, effects are interpreted as factors. A sub-command, `covariates`, is used to indicate which predictors are covariates.

`lnSteps` is the natural logarithm of the length of the shortest tree found, `lnTrees` is the natural logarithm of the number of trees examined during the analysis, `matrix` is the data matrix (coded arbitrarily as 1 to represent *general* and 2 to represent *monocot*), `lnT0` $= \ln(T_0)$, `lnT1toT0` $= \ln(T_1/T_0)$, `lnMaxpropose` $= \ln($`maxpropose`$)$ and `lnMaxfail` $= \ln($`maxfail`$)$.

In this analysis, for both `lnSteps` and `lnTrees`, all interactions involving `seed` were found to be non-significant ($p > 0.1$).

In the next analysis, all interactions were omitted. This allows the purely additive effect of `seed` to be tested for significance. The model formula was:

```
GLM lnSteps lnTrees = seed matrix lnT0 lnT1toT0 &
lnMaxpropose maxaccept lnMaxfail runs;
covariates seed lnT0 lnt1toT0 lnMaxpropose lnMaxfail runs.
```

(Minitab uses no symbol for addition in model formulae. Additive effects may simply be separated by space.)

`seed` was found to be non-significant for `lnSteps` ($p = 0.509$) but highly significant for `lnTrees` ($p < 0.0005$). To determine whether this was a causative relation (Section 4.2.1.2), a further GLM was performed, in which the interaction between `seed` and `matrix` was included but all other interactions were excluded. The model formula was:

```
GLM lnSteps lnTrees = seed|matrix lnT0 lnT1toT0 &
lnMaxpropose maxaccept lnMaxfail runs;
covariates seed lnT0 lnt1toT0 lnMaxpropose lnMaxfail runs.
```

seed × matrix was found to be non-significant for lnSteps ($p = 0.750$), but highly significant for lnTrees ($p = 0.001$). This suggests that the correlation between lnTrees and seed is incidental, resulting from the use of similar seeds for similar analyses *for the general data matrix.* There was no such effect with the monocot data matrix due to the different way seeds were generated (Section 3.4).

The correlation between seed and lnTrees has no generally predictive value. seed is omitted from all further model formulae.

4.2.2.2 Assessment of Interactions

A GLM was performed to investigate first-order interactions between predictors. *A priori*, these are the interactions most likely to have a significant effect. The model formula was:

```
GLM lnSteps lnTrees = matrix lnT0 lnT1toT0 lnMaxpropose &
maxaccept lnMaxfail runs &
matrix*lnT0 matrix*lnT1toT0 matrix*lnMaxpropose &
matrix*maxaccept matrix*lnMaxfail matrix*runs &
lnT0*lnT1toT0 lnT0*lnMaxpropose lnT0*maxaccept &
lnT0*lnMaxfail lnT0*runs &
lnT1toT0*lnMaxpropose lnT1toT0*maxaccept &
lnT1toT0*lnMaxfail lnT1toT0*runs &
lnMaxpropose*maxaccept lnMaxpropose*lnMaxfail &
lnMaxpropose*runs &
maxaccept*lnMaxfail maxaccept*runs &
lnMaxfail*runs;
covariates lnT0 lnT1toT0 lnMaxpropose lnMaxfail runs.
```

First order interactions found to be significant are given in Table 4.1.

The significance of first-order interactions involving every main effect other than runs suggested higher-order interactions might be significant as well. A GLM was

Effect	p for response	
	lnSteps	lnTrees
matrix × lnT0	*NS*	< 0.0005
matrix × lnT1toT0	0.029	< 0.0005
matrix × lnMaxpropose	< 0.0005	< 0.0005
matrix × lnMaxfail	< 0.0005	< 0.0005
lnT0 × lnT1toT0	< 0.0005	< 0.0005
lnT0 × lnMaxpropose	< 0.0005	*NS*
lnT0 × lnMaxfail	0.046	< 0.0005
lnT1toT0 × lnMaxpropose	< 0.0005	< 0.0005
maxaccept × lnT0	*NS*	< 0.0005
maxaccept × lnT1toT0	0.010	< 0.0005
lnT1toT0 × lnMaxfail	< 0.0005	< 0.0005
maxaccept × lnMaxpropose	< 0.0005	< 0.0005
lnMaxpropose × lnMaxfail	< 0.0005	< 0.0005
maxaccept × lnMaxfail	< 0.0005	< 0.0005

Table 4.1 First order interactions found to be significant at $p \leq 0.05$ for one or both responses. *NS:* not significant.

performed to investigate all factors under investigation and all possible interactions, with the exception of any interactions involving runs. Since it has shown no evidence of a multiplicative effect, runs was included as an additive effect only. The model formula was:

```
GLM lnSteps lnTrees = matrix|lnT0|lnT1toT0|lnMaxpropose &
|maxaccept|lnMaxfail runs;
covariates lnT0 lnT1toT0 lnMaxpropose lnMaxfail runs.
```

Interactions of order two or more which were found to be significant are given in Table 4.2.

Effect	p for response	
	lnSteps	lnTrees
lnT0 × lnT1toT0 × lnMaxpropose	0.002	*NS*
lnT0 × lnT1toT0 × lnMaxfail	0.041	*NS*
lnT0 × lnMaxpropose × lnMaxfail	0.012	*NS*
lnT1toT0 × lnMaxpropose × lnMaxfail	*NS*	0.005
maxaccept × lnMaxpropose × lnMaxfail	*NS*	0.028
maxaccept × lnT0 × lnMaxpropose	*NS*	0.003
maxaccept × lnT0 × lnMaxfail	*NS*	0.004
maxaccept × lnT1toT0 × lnMaxpropose	0.027	0.032
matrix × lnT0 × lnT1toT0	*NS*	0.035
matrix × lnT0 × lnMaxpropose	*NS*	0.007
matrix × lnT1toT0 × lnMaxfail	*NS*	0.013
matrix × maxaccept × lnT0 × lnT1toT0	0.033	0.014
matrix × maxaccept × lnT1toT0 × lnMaxfail	0.032	0.016
matrix × lnT0 × lnT1toT0 × lnMaxpropose	*NS*	0.028
matrix × lnT0 × lnMaxpropose × lnMaxfail	0.048	*NS*
matrix × lnT1toT0 × lnMaxpropose × lnMaxfail	*NS*	0.001
maxaccept × lnT0 × lnT1toT0 × lnMaxpropose	< 0.0005	*NS*
maxaccept × lnT0 × lnT1toT0 × lnMaxfail	0.040	*NS*
maxaccept × lnT1toT0 × lnMaxpropose × lnMaxfail	< 0.0005	0.004
matrix × maxaccept × lnT0 × lnT1toT0 ×lnMaxpropose	0.018	0.004
matrix × maxaccept × lnT0 × lnT1toT0 × lnMaxfail	0.040	0.023
matrix × maxaccept × lnT1toT0 × lnMaxpropose ×lnMaxfail	0.014	0.001
maxaccept × lnT0 × lnT1toT0 × lnMaxpropose ×lnMaxfail	0.002	*NS*
matrix × maxaccept × lnT0 × lnT1toT0 ×lnMaxpropose × lnMaxfail	*NS*	0.039

Table 4.2 Interactions of order two or more found to be significant at $p \leq 0.05$ for one or both responses. *NS:* not significant.

4.2.3 Final Analysis

The set of interactions with a significant effect on `lnTrees` is not identical to the set of interactions with a significant effect on `lnSteps` (Table 4.2). Because of these differences, a different analysis was performed for each response, using a different model formula. Each model formula included

1. all main effects;
2. all interactions already found significant at $p \leq 0.05$;
3. all interactions required to make a model formula containing (1) and (2) hierarchical.

(3) is required by GLM. A model is hierarchical if the main effects and any lesser-order interactions that form part of each interaction are included. For example, a model formula including $A \times B \times C$ must also include A, B, C, $A \times B$, $A \times C$ and $B \times C$. Hierarchy is conveniently achieved in Minitab by using | to separate interacting effects (Section 4.2.2.1). Where this leads to a repeated term, for example because `lnT0|lnT1toT0|lnMaxfail` and `lnT0|lnT1toT0` both include `lnT0`, Minitab will ignore the second and subsequent copies of the term. Unless the `tolerance` sub-command is used, Minitab will also remove from the model formula a term which is highly correlated with another term or which is nearly constant. Because this default simplifies fitted equations, `tolerance` was not used.

For simplicity, terms in each fitted equation produced by GLM may be grouped into components a, $-a$, b, $-b$, c, $-c$, d, $-d$, f and $-f$ as follows:

1. a, present in all cases;
2. b, present in all cases when the general data matrix is used;
3. $-b$, present in all cases when the monocot data matrix is used;
4. c, present in all cases when `maxaccept` $= 5$;

Matrix	maxaccept	Effect
general	5	$+d$
general	50	$-d$
monocot	5	$-d$
monocot	50	$+d$

Table 4.3 Values of f, the sum of all non-additive effects involving data matrix and maxaccept, expressed in terms of d (Section 4.2.3).

5. $-c$, present in all cases when maxaccept $= 50$;

6. d, present when the general data matrix is used and maxaccept $= 5$;

7. $-d$, present when the general data matrix is used and maxaccept $= 50$;

8. f, present when the monocot data matrix is used and maxaccept $= 5$;

9. $-f$, present when the monocot data matrix is used and maxaccept $= 50$.

a includes only covariates and interactions involving covariates. b, c, d, e and f include both factors and covariates, as well as interactions involving factors.

Because there are two levels of each of the two predictors which are factors, it happens that

$$f = -d \tag{4.11}$$

Thus we may substitute $-d$ for f (Table 4.3).

Calculation of fitted values for lnSteps and lnTrees from a, b, c and d is as given in Table 4.4. The components of a, b, c and d, from Minitab's output, are given in Sections 4.2.3.1 and 4.2.4.

4.2.3.1 Fitted Equation for Tree Length

The model formula was:

Matrix	maxaccept	$\widehat{resp}$
general	5	$a_{resp} + b_{resp} + c_{resp} + d_{resp}$
general	50	$a_{resp} + b_{resp} - c_{resp} - d_{resp}$
monocot	5	$a_{resp} - b_{resp} + c_{resp} - d_{resp}$
monocot	50	$a_{resp} - b_{resp} - c_{resp} + d_{resp}$

Table 4.4 Calculation of a fitted value $\widehat{resp}$ from a, b, c and d for either level of maxaccept, for either data matrix. If $\widehat{resp}$ is the fitted value for lnSteps, the subscript *resp* is lnSteps. If $\widehat{resp}$ is the fitted value for lnTrees, the subscript *resp* is lnTrees.

```
GLM lnSteps = runs matrix|lnT1toT0 matrix|lnMaxpropose matrix|lnMaxfail &
lnT0|lnT1toT0 lnT0|lnMaxpropose lnT0|lnMaxfail lnT1toT0|lnMaxpropose &
lnT1toT0|lnMaxfail lnMaxpropose|lnMaxfail maxaccept|lnT1toT0 &
maxaccept|lnMaxpropose maxaccept|lnMaxfail lnT0|lnT1toT0|lnMaxpropose &
lnT0|lnT1toT0|lnMaxfail lnT0|lnMaxpropose|lnMaxfail &
maxaccept|lnT1toT0|lnMaxpropose matrix|maxaccept|lnT0|lnT1toT0 &
matrix|maxaccept|lnT1toT0|lnMaxfail matrix|lnT0|lnMaxpropose|lnMaxfail &
maxaccept|lnT0|lnT1toT0|lnMaxpropose maxaccept|lnT0|lnT1toT0|lnMaxfail &
maxaccept|lnT1toT0|lnMaxpropose|lnMaxfail &
matrix|maxaccept|lnT0|lnT1toT0|lnMaxpropose &
matrix|maxaccept|lnT0|lnT1toT0|lnMaxfail &
matrix|maxaccept|lnT1toT0|lnMaxpropose|lnMaxfail &
maxaccept|lnT0|lnT1toT0|lnMaxpropose|lnMaxfail;
covariates lnT0 lnT1toT0 lnMaxpropose lnMaxfail runs.
```

Details of the analysis of variance are given in Table 4.5. For each term, the adjusted sum of squares is equal to the adjusted mean square, as a result of the orthogonal design of the experiment.

Table 4.5 (*overleaf*) Analysis of Variance for lnSteps.

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
runs	1	0.377	0.370	0.370	31.51	0.000
matrix	1	2115.610	9.677	9.677	822.94	0.000
lnT1toT0	1	6.488	0.025	0.025	2.09	0.149
matrix × lnT1toT0	1	0.074	0.182	0.182	15.46	0.000
lnMaxpropose	1	11.804	0.004	0.004	0.33	0.565
matrix × lnMaxpropose	1	1.451	0.103	0.103	8.75	0.003
lnMaxfail	1	11.879	0.009	0.009	0.76	0.384
matrix × lnMaxfail	1	0.233	0.116	0.116	9.84	0.002
lnT0	1	6.746	0.117	0.117	9.96	0.002
lnT0 × lnT1toT0	1	4.443	0.020	0.020	1.67	0.196
lnT0 × lnMaxpropose	1	0.798	0.732	0.732	62.21	0.000
lnT0 × lnMaxfail	1	0.064	0.178	0.178	15.10	0.000
lnT1toT0 × lnMaxpropose	1	0.236	0.270	0.270	22.96	0.000
lnT1toT0 × lnMaxfail	1	0.298	0.132	0.132	11.25	0.001
lnMaxpropose × lnMaxfail	1	0.446	0.004	0.004	0.32	0.571
Maxaccept	1	1.905	0.022	0.022	1.85	0.174
Maxaccept × lnT1toT0	1	0.101	0.001	0.001	0.11	0.743
Maxaccept × lnMaxpropose	1	0.229	0.032	0.032	2.74	0.098
Maxaccept × lnMaxfail	1	1.458	0.015	0.015	1.25	0.263
lnT0 × lnT1toT0 × lnMaxpropose	1	0.343	0.112	0.112	9.51	0.002
lnT0 × lnT1toT0 × lnMaxfail	1	0.833	0.049	0.049	4.18	0.041
lnT0 × lnMaxpropose × lnMaxfail	1	0.239	0.075	0.075	6.39	0.012
Maxaccept × lnT1toT0 × lnMaxpropose	1	0.343	0.057	0.057	4.88	0.027
matrix × Maxaccept	1	0.004	0.018	0.018	1.56	0.212
matrix × lnT0	1	0.017	0.189	0.189	16.10	0.000
Maxaccept × lnT0	1	0.001	0.032	0.032	2.70	0.101
matrix × Maxaccept × lnT0	1	0.070	0.005	0.005	0.41	0.524
matrix × Maxaccept × lnT1toT0	1	0.478	0.001	0.001	0.08	0.777

continued on next page

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
matrix × lnT0 × lnT1toT0	1	0.012	0.201	0.201	17.13	0.000
Maxaccept × lnT0 × lnT1toT0	1	0.449	0.030	0.030	2.54	0.111
matrix × Maxaccept × lnT0 × lnT1toT0	1	0.071	0.019	0.019	1.61	0.204
matrix × Maxaccept × lnMaxfail	1	0.018	0.021	0.021	1.78	0.182
matrix × lnT1toT0 × lnMaxfail	1	0.012	0.024	0.024	2.06	0.151
Maxaccept × lnT1toT0 × lnMaxfail	1	1.066	0.028	0.028	2.37	0.124
matrix × Maxaccept × lnT1toT0 × lnMaxfail	1	0.122	0.019	0.019	1.64	0.201
matrix × lnT0 × lnMaxpropose	1	0.154	0.154	0.154	13.13	0.000
matrix × lnT0 × lnMaxfail	1	0.036	0.008	0.008	0.70	0.402
matrix × lnMaxpropose × lnMaxfail	1	0.569	0.084	0.084	7.13	0.008
matrix × lnT0 × lnMaxpropose × lnMaxfail	1	0.007	0.007	0.007	0.58	0.448
Maxaccept × lnT0 × lnMaxpropose	1	0.150	0.006	0.006	0.49	0.482
Maxaccept × lnT0 × lnT1toT0 × lnMaxpropose	1	0.040	0.154	0.154	13.08	0.000
Maxaccept × lnT0 × lnMaxfail	1	0.113	0.002	0.002	0.17	0.676
Maxaccept × lnT0 × lnT1toT0 × lnMaxfail	1	0.130	0.050	0.050	4.21	0.040
Maxaccept × lnMaxpropose × lnMaxfail	1	0.331	0.011	0.011	0.91	0.341

continued on next page

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.028	0.003	0.003	0.27	0.601
Maxaccept × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.078	0.195	0.195	16.55	0.000
matrix × Maxaccept × lnMaxpropose	1	0.061	0.024	0.024	2.04	0.154
matrix × lnT1toT0 × lnMaxpropose	1	0.095	0.170	0.170	14.42	0.000
matrix × Maxaccept × lnT0 × lnMaxpropose	1	0.000	0.018	0.018	1.56	0.212
matrix × Maxaccept × lnT1toT0 × lnMaxpropose	1	0.381	0.003	0.003	0.25	0.616
matrix × lnT0 × lnT1toT0 × lnMaxpropose	1	0.163	0.162	0.162	13.81	0.000
matrix × Maxaccept × lnT0 × lnT1toT0 × lnMaxpropose	1	0.031	0.031	0.031	2.62	0.106
matrix × Maxaccept × lnT0 × lnMaxfail	1	0.002	0.020	0.020	1.74	0.188
matrix × lnT0 × lnT1toT0 × lnMaxfail	1	0.114	0.114	0.114	9.73	0.002
matrix × Maxaccept × lnT0 × lnT1toT0 × lnMaxfail	1	0.023	0.023	0.023	1.92	0.166
matrix × Maxaccept × lnMaxpropose × lnMaxfail	1	0.263	0.029	0.029	2.50	0.114
matrix × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.000	0.000	0.000	0.00	0.947
matrix × Maxaccept × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.039	0.039	0.039	3.30	0.069

continued on next page

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
Maxaccept × lnT0 × lnMaxpropose × lnMaxfail	1	0.214	0.001	0.001	0.06	0.806
lnT0 × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.002	0.002	0.002	0.17	0.680
Maxaccept × lnT0 × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.117	0.117	0.117	9.95	0.002
Error	1882	22.129	22.129	0.012		
Total	1943	2193.987				

$a_{\texttt{lnSteps}}$ is obtained as follows.

$$
\begin{aligned}
a_{\texttt{lnSteps}} = {} & 8.77390 \\
& - 0.003750 \times \texttt{runs} \\
& - 0.01829 \times \ln(\texttt{t1/t0}) \\
& - 0.003342 \times \ln(\texttt{maxpropose}) \\
& - 0.01106 \times \ln(\texttt{maxfail}) \\
& - 0.017079 \times \ln(\texttt{t0}) \\
& - 0.002349 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\
& + 0.006575 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
& + 0.007075 \times \ln(\texttt{t0}) \times \ln(\texttt{maxfail}) \\
& + 0.009346 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
& + 0.014286 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxfail}) \\
& - 0.001108 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\
& + 0.000863 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
& + 0.001250 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxfail}) \\
& - 0.000709 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\
& - 0.000343 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail})
\end{aligned}
$$

$$+0.000039 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \quad (4.12)$$

$b_{\texttt{lnSteps}}$ is obtained as follows.

$$\begin{aligned} b_{\texttt{lnSteps}} = {} & 0.95173 - 0.03933 \times \ln(\texttt{t1/t0}) \\ & + 0.014908 \times \ln(\texttt{maxpropose}) \\ & + 0.03144 \times \ln(\texttt{maxfail}) \\ & - 0.017762 \times \ln(\texttt{t0}) \\ & - 0.005040 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\ & + 0.003745 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxfail}) \\ & + 0.002415 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\ & + 0.001022 \times \ln(\texttt{t0}) \times \ln(\texttt{maxfail}) \\ & - 0.004006 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\ & + 0.000134 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\ & + 0.005693 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\ & + 0.000658 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\ & + 0.000550 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxfail}) \\ & + 0.000025 \times \ln(\texttt{t1/t0}) \\ & \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \end{aligned} \quad (4.13)$$

$c_{\texttt{lnSteps}}$ is obtained as follows.

$$\begin{aligned} c_{\texttt{lnSteps}} = {} & 0.05133 + 0.00415 \times \ln(\texttt{t1/t0}) \\ & - 0.009604 \times \ln(\texttt{maxpropose}) \\ & - 0.01421 \times \ln(\texttt{maxfail}) \\ & + 0.004311 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\ & + 0.008893 \times \ln(\texttt{t0}) \\ & - 0.002896 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \end{aligned}$$

$$
\begin{aligned}
&+ 0.006561 \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxfail}) \\
&- 0.000586 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
&+ 0.001012 \times \ln(\texttt{t0}) \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
&- 0.000760 \times \ln(\texttt{t0}) \times \ln(\texttt{maxfail}) \\
&+ 0.001254 \times \ln(\texttt{t0}) \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxfail}) \\
&+ 0.001863 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\
&- 0.002669 \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\
&+ 0.000069 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\
&- 0.000297 \times \ln(\texttt{t0}) \times \ln(\texttt{t1}/\texttt{t0}) \\
&\times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \qquad (4.14)
\end{aligned}
$$

$d_{\texttt{lnSteps}}$ is obtained as follows.

$$
\begin{aligned}
d_{\texttt{lnSteps}} \;=\; & 0.03721 - 0.002316 \times \ln(\texttt{t0}) \\
&+ 0.00283 \times \ln(\texttt{t1}/\texttt{t0}) \\
&- 0.001547 \times \ln(\texttt{t0}) \times \ln(\texttt{t1}/\texttt{t0}) \\
&- 0.010373 \times \ln(\texttt{maxfail}) \\
&+ 0.003334 \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxfail}) \\
&- 0.006370 \times \ln(\texttt{maxpropose}) \\
&+ 0.000658 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
&- 0.000753 \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
&+ 0.000286 \times \ln(\texttt{t0}) \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
&+ 0.000694 \times \ln(\texttt{t0}) \times \ln(\texttt{maxfail}) \\
&+ 0.000244 \times \ln(\texttt{t0}) \times \ln(\texttt{t1}/\texttt{t0}) \times \ln(\texttt{maxfail}) \\
&+ 0.001743 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \\
&- 0.000673 \times \ln(\texttt{t1}/\texttt{t0}) \\
&\times \ln(\texttt{maxpropose}) \times \ln(\texttt{maxfail}) \qquad (4.15)
\end{aligned}
$$

4.2.4 Fitted Equation for Run Time

The model formula was:

```
GLM lnTrees = runs matrix|lnT0 matrix|lnT1toT0 &
matrix|lnMaxpropose matrix|lnMaxfail &
lnT0|lnT1toT0 lnT0|lnMaxfail &
lnT1toT0|lnMaxpropose maxaccept|lnT0 &
maxaccept|lnT1toT0 lnT1toT0|lnMaxfail &
maxaccept|lnMaxpropose lnMaxpropose|lnMaxfail &
maxaccept|lnMaxfail lnT1toT0|lnMaxpropose|lnMaxfail &
maxaccept|lnMaxpropose|lnMaxfail &
maxaccept|lnT0|lnMaxpropose maxaccept|lnT0|lnMaxfail &
maxaccept|lnT1toT0|lnMaxpropose matrix|lnT0|lnT1toT0 &
matrix|lnT0|lnMaxpropose &
matrix|lnT1toT0|lnMaxfail matrix|maxaccept|lnT0|lnT1toT0 &
matrix|maxaccept|lnT1toT0|lnMaxfail &
matrix|lnT0|lnT1toT0|lnMaxpropose &
matrix|lnT1toT0|lnMaxpropose|lnMaxfail &
maxaccept|lnT1toT0|lnMaxpropose|lnMaxfail &
matrix|maxaccept|lnT0|lnT1toT0|lnMaxpropose &
matrix|maxaccept|lnT0|lnT1toT0|lnMaxfail &
matrix|maxaccept|lnT1toT0|lnMaxpropose|lnMaxfail &
matrix|maxaccept|lnT0|lnT1toT0|lnMaxpropose|lnMaxfail;
covariates lnT0 lnT1toT0 lnMaxpropose lnMaxfail runs.
```

Details of the analysis of variance are given in Table 4.6. For each term, the adjusted sum of squares is again equal to the adjusted mean square, as a result of the orthogonal design of the experiment.

Table 4.6 (*overleaf*) Analysis of Variance for `lnTrees`.

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
runs	1	1730.38	1728.65	1728.65	5820.71	0.000
matrix	1	7.75	0.05	0.05	0.18	0.669
lnT0	1	46.52	2.22	2.22	7.48	0.006
matrix × lnT0	1	13.53	0.43	0.43	1.46	0.226
lnT1toT0	1	4.27	4.29	4.29	14.45	0.000
matrix × lnT1toT0	1	15.57	0.00	0.00	0.02	0.900
lnMaxpropose	1	6138.23	377.00	377.00	1269.42	0.000
matrix × lnMaxpropose	1	10.28	0.01	0.01	0.03	0.870
lnMaxfail	1	4440.33	87.69	87.69	295.28	0.000
matrix × lnMaxfail	1	27.53	0.01	0.01	0.03	0.860
lnT0 × lnT1toT0	1	14.38	0.00	0.00	0.01	0.906
lnT0 × lnMaxfail	1	38.38	0.52	0.52	1.76	0.185
lnT1toT0 × lnMaxpropose	1	38.03	0.43	0.43	1.45	0.229
maxaccept	1	145.51	1.53	1.53	5.14	0.024
maxaccept × lnT0	1	7.37	7.84	7.84	26.41	0.000
maxaccept × lnT1toT0	1	13.98	5.76	5.76	19.40	0.000
lnT1toT0 × lnMaxfail	1	61.43	0.00	0.00	0.01	0.933
maxaccept × lnMaxpropose	1	79.62	2.65	2.65	8.94	0.003
lnMaxpropose × lnMaxfail	1	27.04	0.22	0.22	0.75	0.387
maxaccept × lnMaxfail	1	144.64	1.61	1.61	5.41	0.020
lnT1toT0 × lnMaxpropose × lnMaxfail	1	4.25	2.36	2.36	7.95	0.005
maxaccept × lnMaxpropose × lnMaxfail	1	6.74	1.44	1.44	4.86	0.028
lnT0 × lnMaxpropose	1	0.02	0.72	0.72	2.42	0.120
maxaccept × lnT0 × lnMaxpropose	1	6.68	2.65	2.65	8.91	0.003
maxaccept × lnT0 × lnMaxfail	1	1.64	2.47	2.47	8.32	0.004
maxaccept × lnT1toT0 × lnMaxpropose	1	54.44	1.36	1.36	4.60	0.032

continued on next page

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
matrix × lnT0 × lnT1toT0	1	2.62	1.32	1.32	4.44	0.035
matrix × lnT0 × lnMaxpropose	1	0.35	2.16	2.16	7.26	0.007
matrix × lnT1toT0 × lnMaxfail	1	21.33	1.83	1.83	6.16	0.013
matrix × maxaccept	1	0.48	0.39	0.39	1.32	0.252
matrix × maxaccept × lnT0	1	1.65	0.00	0.00	0.01	0.918
matrix × maxaccept × lnT1toT0	1	27.83	0.09	0.09	0.29	0.590
maxaccept × lnT0 × lnT1toT0	1	12.26	0.00	0.00	0.01	0.907
matrix × maxaccept × lnT0 × lnT1toT0	1	2.75	1.79	1.79	6.03	0.014
matrix × maxaccept × lnMaxfail	1	12.88	0.15	0.15	0.49	0.482
maxaccept × lnT1toT0 × lnMaxfail	1	50.03	0.00	0.00	0.00	0.958
matrix × maxaccept × lnT1toT0 × lnMaxfail	1	15.01	1.73	1.73	5.84	0.016
matrix × lnT1toT0 × lnMaxpropose	1	29.59	0.01	0.01	0.04	0.840
lnT0 × lnT1toT0 × lnMaxpropose	1	0.47	0.62	0.62	2.10	0.148
matrix × lnT0 × lnT1toT0 × lnMaxpropose	1	1.28	1.45	1.45	4.87	0.028
matrix × lnMaxpropose × lnMaxfail	1	23.29	0.00	0.00	0.01	0.918
matrix × lnT1toT0 × lnMaxpropose × lnMaxfail	1	5.85	3.53	3.53	11.89	0.001
maxaccept × lnT1toT0 × lnMaxpropose × lnMaxfail	1	1.84	2.50	2.50	8.42	0.004

continued on next page

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
matrix × maxaccept × lnMaxpropose	1	32.68	0.55	0.55	1.85	0.174
matrix × maxaccept × lnT0 × lnMaxpropose	1	1.40	0.04	0.04	0.13	0.715
matrix × maxaccept × lnT1toT0 × lnMaxpropose	1	40.70	0.05	0.05	0.17	0.683
maxaccept × lnT0 × lnT1toT0 × lnMaxpropose	1	0.21	1.11	1.11	3.74	0.053
matrix × maxaccept × lnT0 × lnT1toT0 × lnMaxpropose	1	1.23	2.48	2.48	8.35	0.004
matrix × lnT0 × lnMaxfail	1	5.18	0.50	0.50	1.67	0.196
lnT0 × lnT1toT0 × lnMaxfail	1	0.43	0.06	0.06	0.20	0.651
matrix × maxaccept × lnT0 × lnMaxfail	1	0.06	0.11	0.11	0.35	0.552
matrix × lnT0 × lnT1toT0 × lnMaxfail	1	1.62	0.94	0.94	3.16	0.076
maxaccept × lnT0 × lnT1toT0 × lnMaxfail	1	1.94	0.29	0.29	0.98	0.323
matrix × maxaccept × lnT0 × lnT1toT0 × lnMaxfail	1	0.32	1.54	1.54	5.17	0.023
matrix × maxaccept × lnMaxpropose × lnMaxfail	1	13.40	0.18	0.18	0.61	0.436
matrix × maxaccept × lnT1toT0 × lnMaxpropose × lnMaxfail	1	2.92	3.59	3.59	12.08	0.001
lnT0 × lnMaxpropose × lnMaxfail	1	0.14	0.34	0.34	1.16	0.282
matrix × lnT0 × lnMaxpropose × lnMaxfail	1	0.37	0.76	0.76	2.56	0.110

continued on next page

Source	*df*	Seq. *SS*	Adj. *SS*	Adj. *MS*	*F*	*p*
maxaccept × lnT0 × lnMaxpropose × lnMaxfail	1	8.37	1.13	1.13	3.79	0.052
lnT0 × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.21	0.21	0.21	0.70	0.403
matrix × maxaccept × lnT0 × lnMaxpropose × lnMaxfail	1	1.28	0.03	0.03	0.09	0.770
matrix × lnT0 × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.39	0.39	0.39	1.32	0.250
maxaccept × lnT0 × lnT1toT0 × lnMaxpropose × lnMaxfail	1	0.97	0.97	0.97	3.26	0.071
matrix × maxaccept × lnT0 × lnT1toT0 × lnMaxpropose × lnMaxfail	1	1.26	1.26	1.26	4.25	0.039
Error	1879	558.03	558.03	0.30		
Total	1943	13961.14				

$a_{\texttt{lnTrees}}$ is obtained as follows.

$$
\begin{aligned}
a_{\texttt{lnTrees}} = \; & -0.3007 \\
& + 0.256130 \times \texttt{runs} \\
& - 0.07438 \times \ln(\texttt{t0}) \\
& - 0.24180 \times \ln(\texttt{t1/t0}) \\
& + 1.03934 \times \ln(\texttt{maxpropose}) \\
& + 1.09576 \times \ln(\texttt{Maxfail}) \\
& - 0.001081 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\
& + 0.012133 \times \ln(\texttt{t0}) \times \ln(\texttt{Maxfail}) \\
& + 0.011791 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose})
\end{aligned}
$$

$$
\begin{aligned}
& - 0.00180 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\
& - 0.008501 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
& + 0.009295 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
& - 0.006515 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
& - 0.002036 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
& - 0.001389 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\
& + 0.001517 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
& + 0.000396 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\
& \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \qquad (4.16)
\end{aligned}
$$

$b_{\texttt{lnTrees}}$ is obtained as follows.

$$
\begin{aligned}
b_{\texttt{lnTrees}} = & -0.0809 \\
& + 0.03292 \times \ln(\texttt{t0}) \\
& - 0.00803 \times \ln(\texttt{t1/t0}) \\
& + 0.00477 \times \ln(\texttt{maxpropose}) \\
& + 0.01127 \times \ln(\texttt{Maxfail}) \\
& + 0.019227 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\
& - 0.011291 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
& - 0.05312 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\
& - 0.001976 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
& - 0.003102 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
& + 0.001011 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
& + 0.011372 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
& - 0.011838 \times \ln(\texttt{t0}) \times \ln(\texttt{Maxfail}) \\
& - 0.005459 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\
& + 0.002257 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail})
\end{aligned}
$$

$$+0.000544 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \quad (4.17)$$

$c_{\texttt{lnTrees}}$ is obtained as follows.

$$\begin{aligned} c_{\texttt{lnTrees}} = & -0.4297 \\ & -0.13979 \times \ln(\texttt{t0}) \\ & -0.28009 \times \ln(\texttt{t1/t0}) \\ & +0.08720 \times \ln(\texttt{maxpropose}) \\ & +0.14833 \times \ln(\texttt{Maxfail}) \\ & -0.021647 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\ & +0.012504 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\ & +0.026390 \times \ln(\texttt{t0}) \times \ln(\texttt{Maxfail}) \\ & +0.021015 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\ & +0.001070 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\ & -0.00114 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\ & +0.009566 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\ & -0.002718 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\ & -0.003037 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\ & -0.002744 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\ & +0.000855 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\ & \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \end{aligned} \quad (4.18)$$

$d_{\texttt{lnTrees}}$ is obtained as follows.

$$\begin{aligned} d_{\texttt{lnTrees}} = & -0.2173 \\ & -0.00280 \times \ln(\texttt{t0}) \\ & -0.03429 \times \ln(\texttt{t1/t0}) \\ & +0.022414 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \end{aligned}$$

$$
\begin{aligned}
&+ 0.04485 \times \ln(\texttt{Maxfail}) \\
&- 0.05170 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\
&+ 0.03971 \times \ln(\texttt{maxpropose}) \\
&- 0.001530 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \\
&+ 0.004010 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
&- 0.004063 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \\
&- 0.005449 \times \ln(\texttt{t0}) \times \ln(\texttt{Maxfail}) \\
&- 0.006983 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \times \ln(\texttt{Maxfail}) \\
&- 0.007646 \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
&+ 0.011459 \times \ln(\texttt{t1/t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
&+ 0.000412 \times \ln(\texttt{t0}) \times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail}) \\
&+ 0.000976 \times \ln(\texttt{t0}) \times \ln(\texttt{t1/t0}) \\
&\times \ln(\texttt{maxpropose}) \times \ln(\texttt{Maxfail})
\end{aligned} \tag{4.19}
$$

4.2.5 Summary

It is possible to predict values for ln(Steps) and ln(Trees), and hence Steps and Trees, on the basis of results obtained. The fitted equations fit the current data well, as shown by the low adjusted mean square for the error terms (Table 4.5, Table 4.6). The fitted equations in this section may be used to predict, and hence relate, both quality of result and run time, with more accuracy than the simple equations obtained in Section 4.1. Uses of the fitted equations are discussed further in Chapter 5.

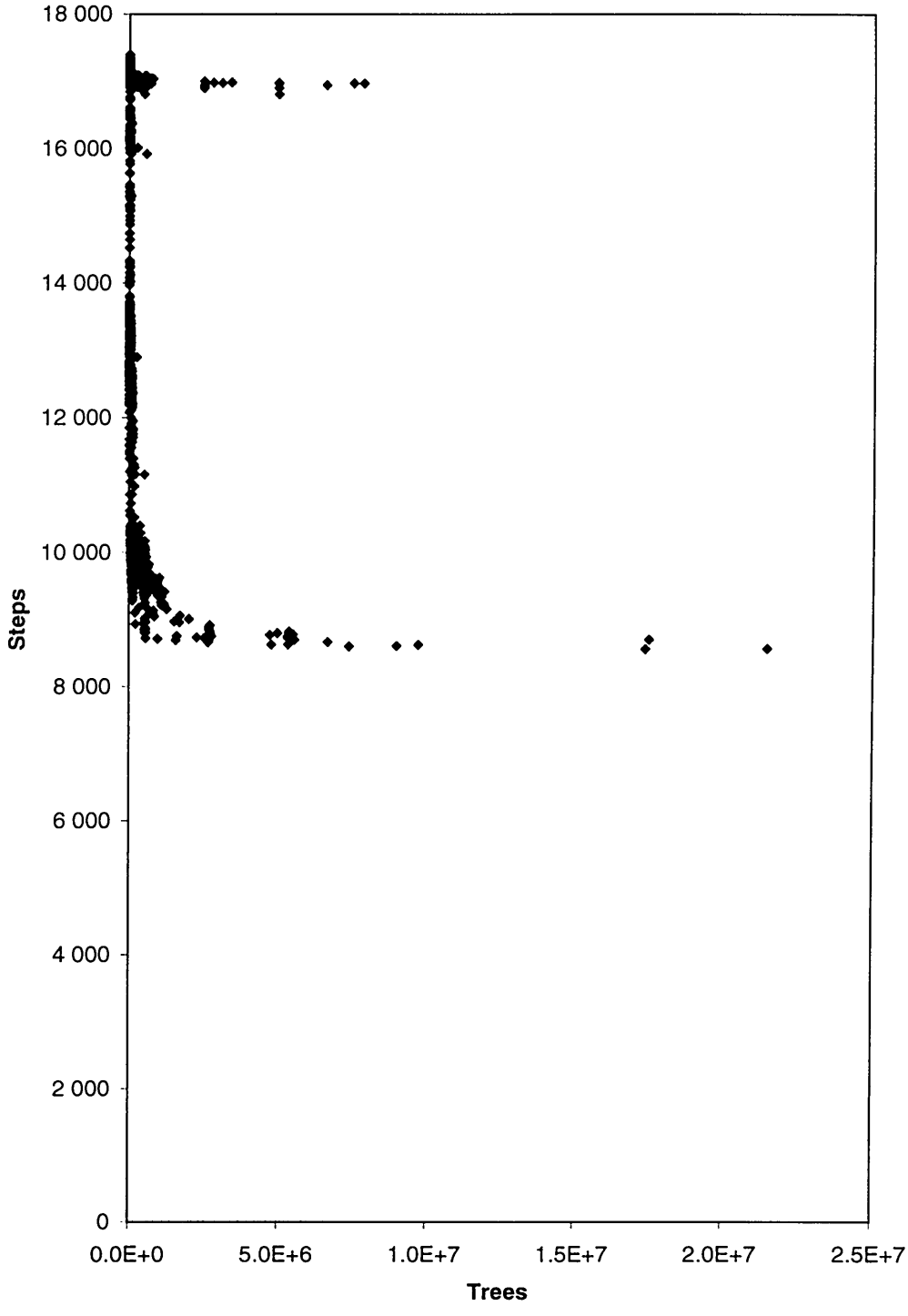


Figure 4.1 The relationship observed between steps on the shortest tree found and number of trees considered, for the general data matrix.

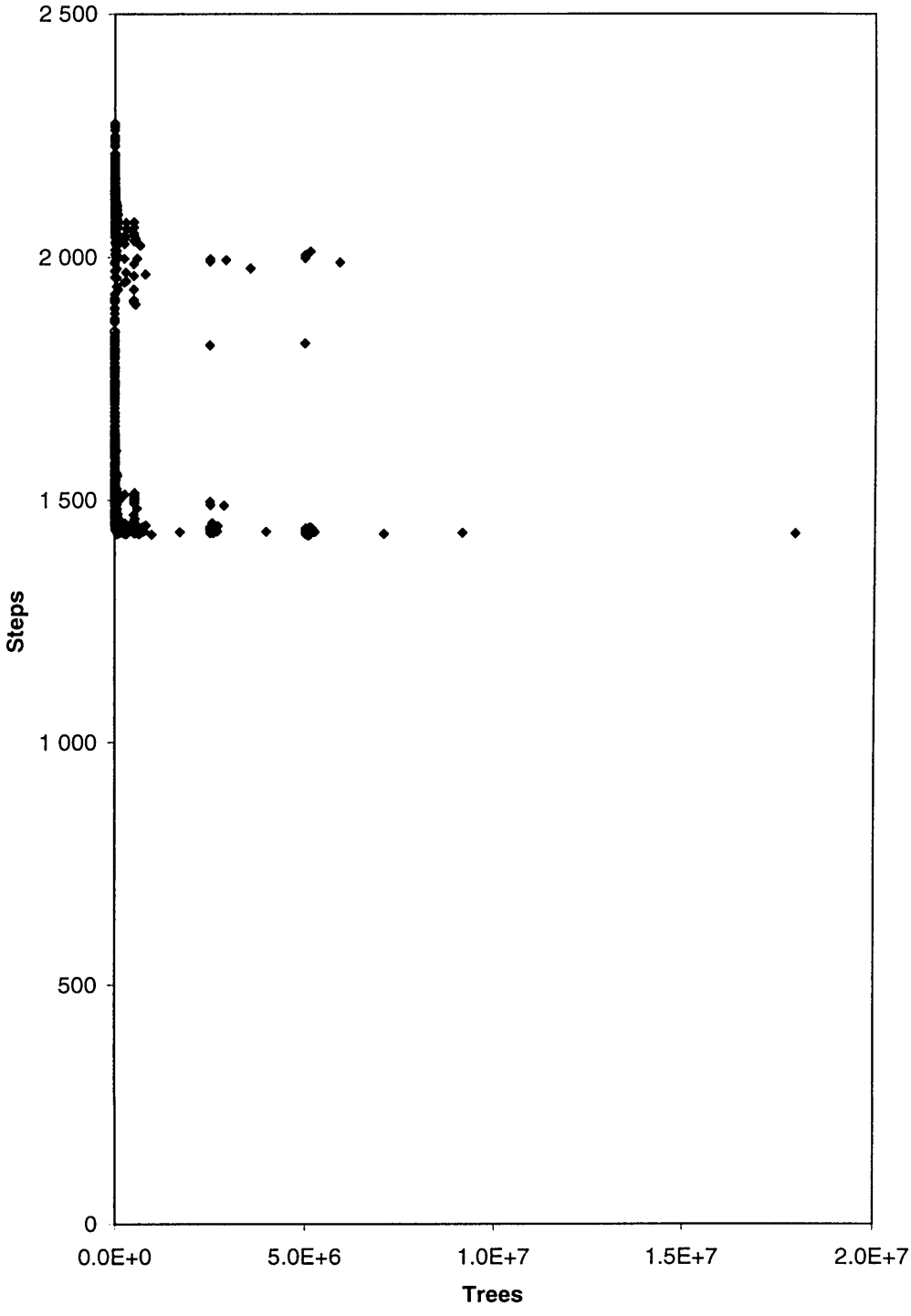


Figure 4.2 The relationship observed between steps on the shortest tree found and number of trees considered, for the monocot data matrix.

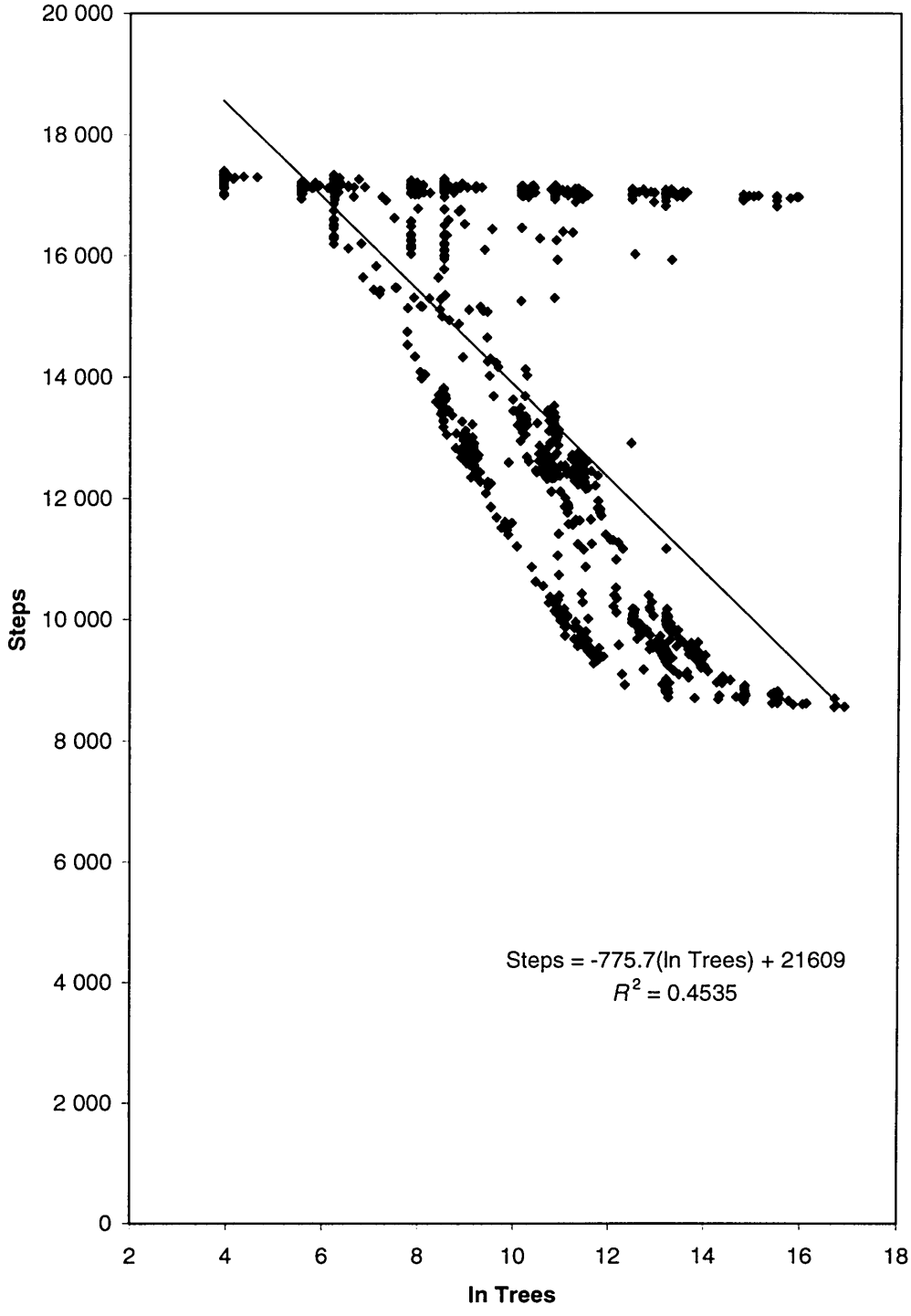


Figure 4.3 Testing for a logarithmic relationship between steps on the shortest tree found and number of trees considered, for the general data matrix. Line: linear regression line, with fitted equation and scatter as shown.

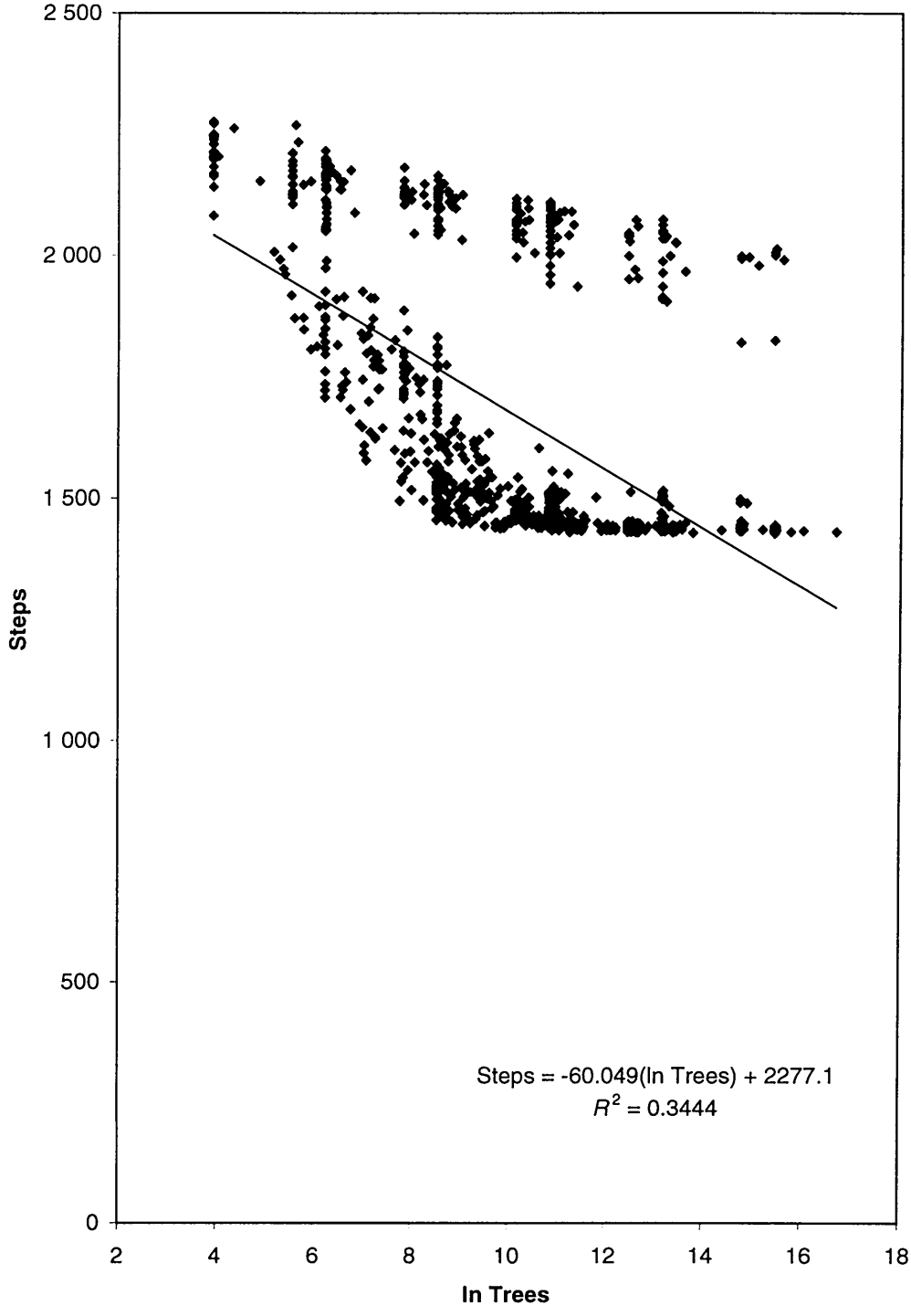


Figure 4.4 Testing for a logarithmic relationship between steps on the shortest tree found and number of trees considered, for the monocot data matrix. Line: linear regression line, with fitted equation and scatter as shown.

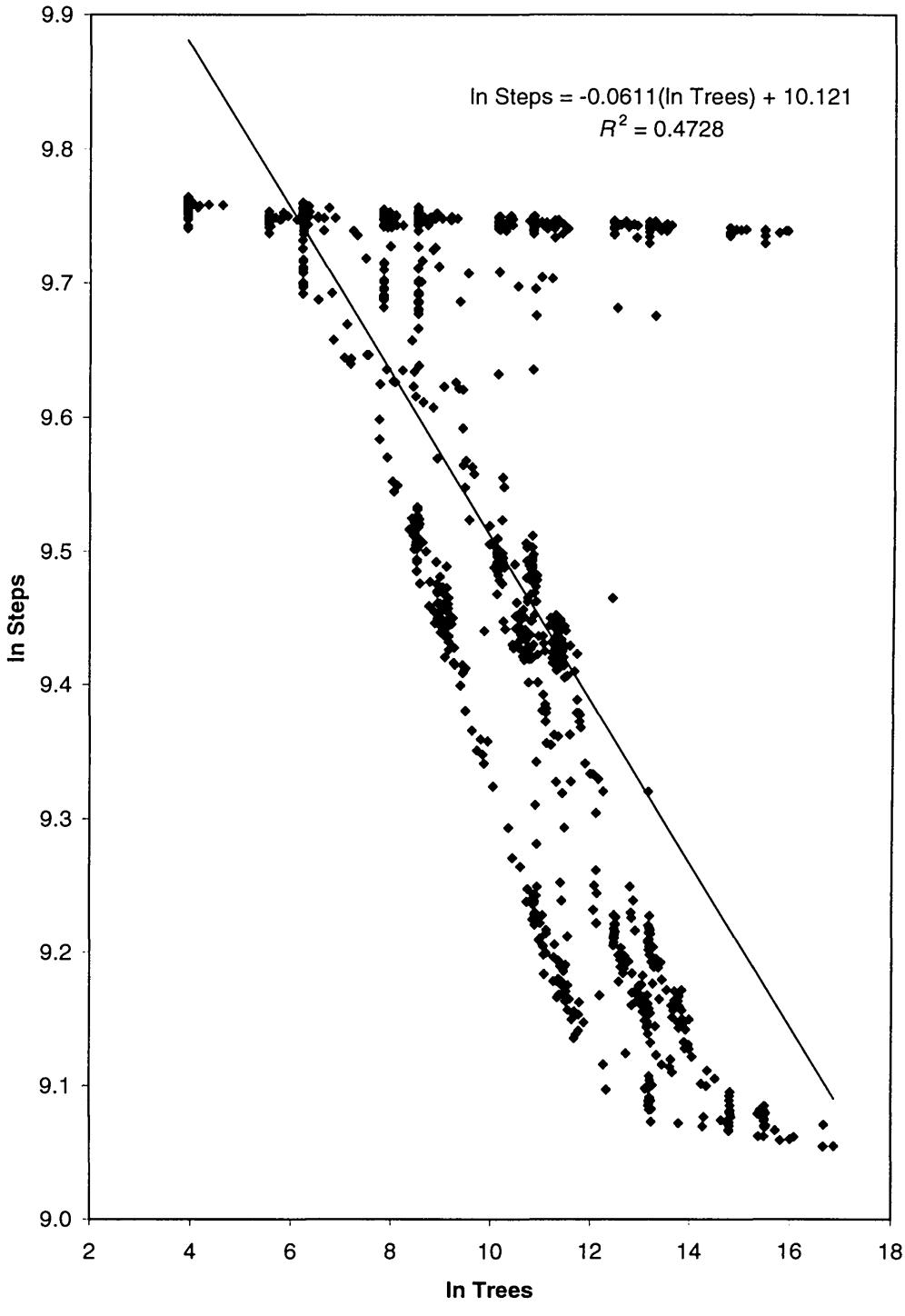


Figure 4.5 Testing for an allometric relationship between steps on the shortest tree found and number of trees considered, for the general data matrix. Line: linear regression line, with fitted equation and scatter as shown.

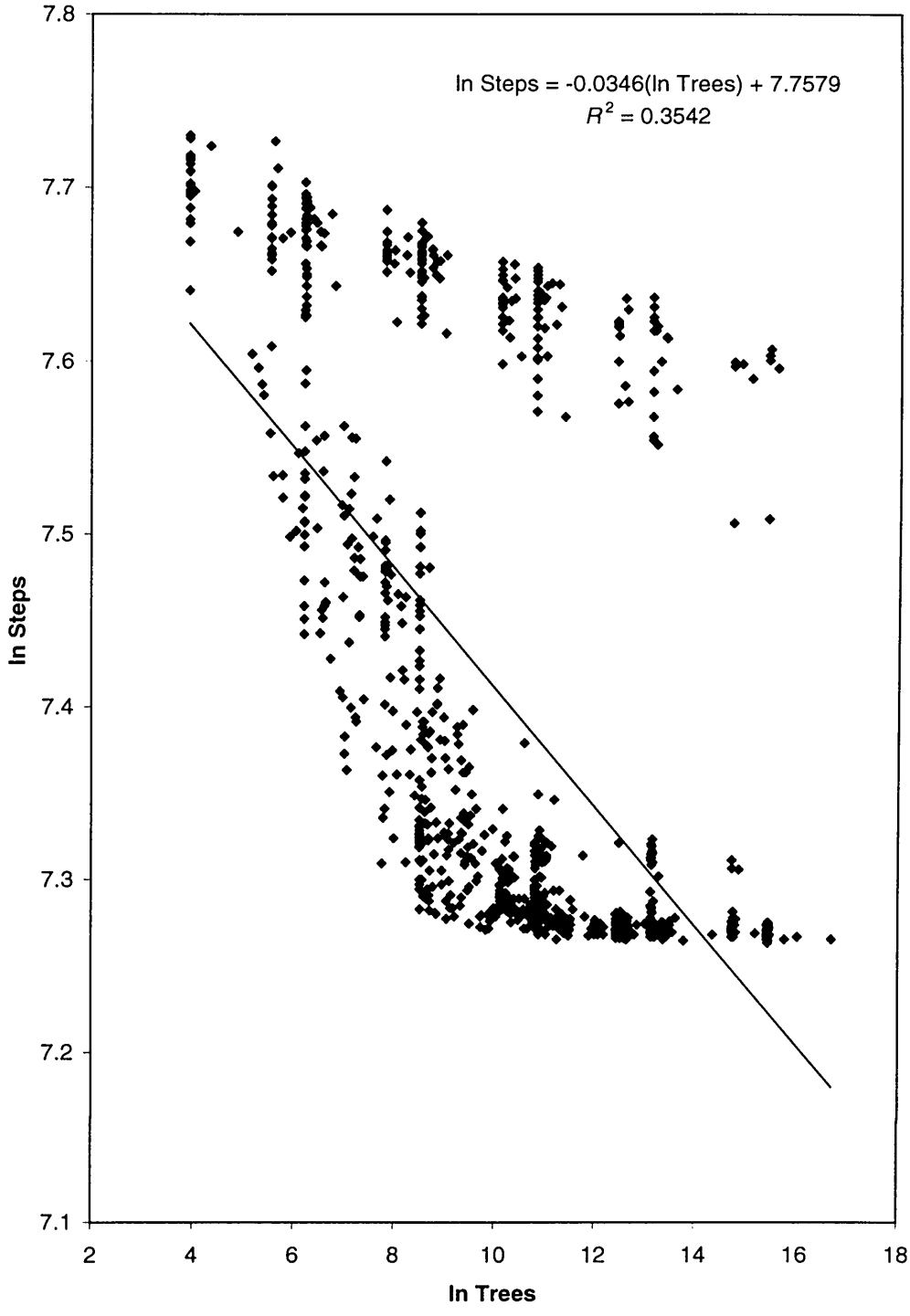


Figure 4.6 Testing for an allometric relationship between steps on the shortest tree found and number of trees considered, for the monocot data matrix. Line: linear regression line, with fitted equation and scatter as shown.

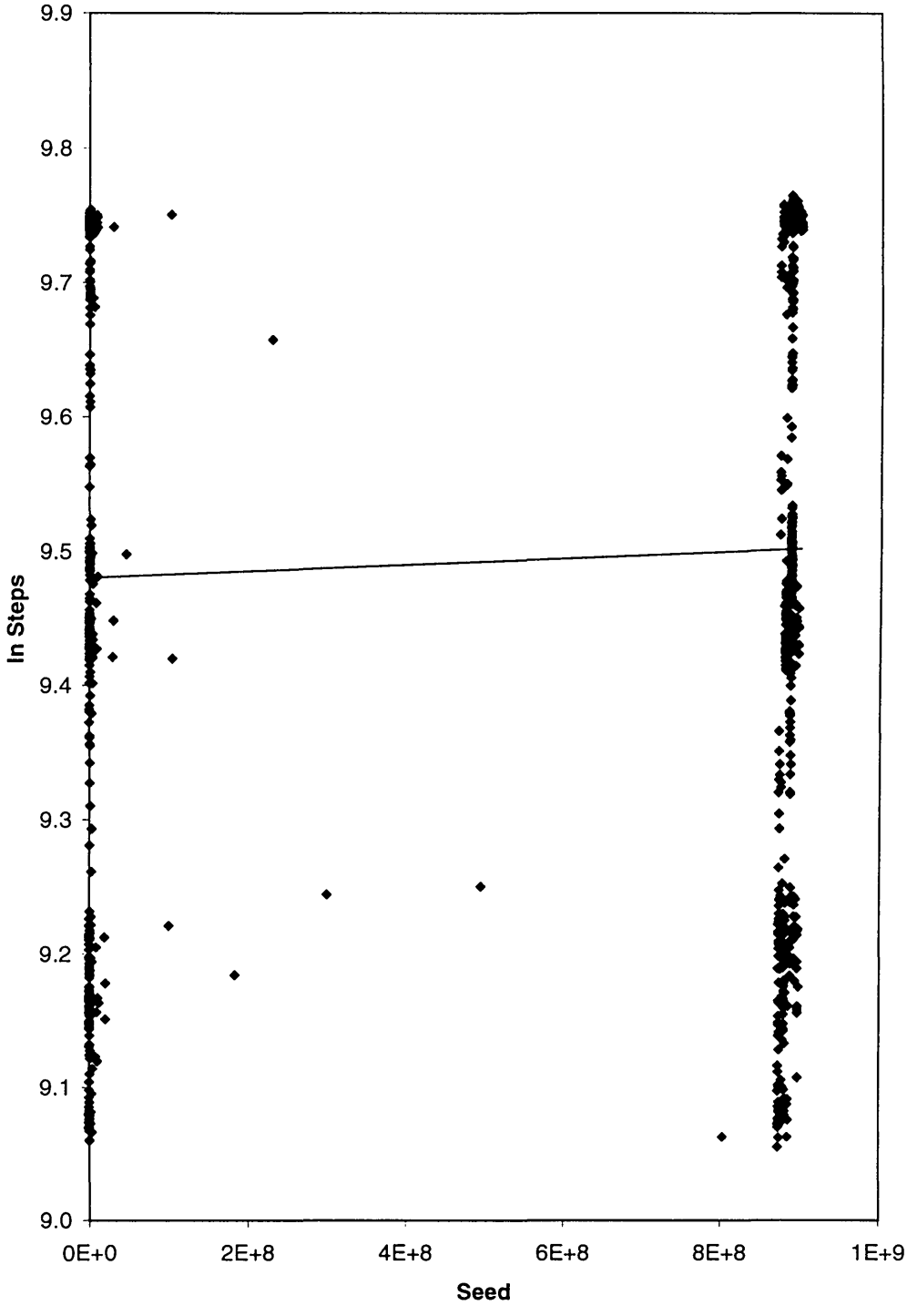


Figure 4.7 The relationship observed between ln(steps on shortest tree found) and random number seed, for the general data matrix. One observation is omitted since it was assembled from two analyses, and consequently used two seeds (Section B.3). Line: linear regression line.

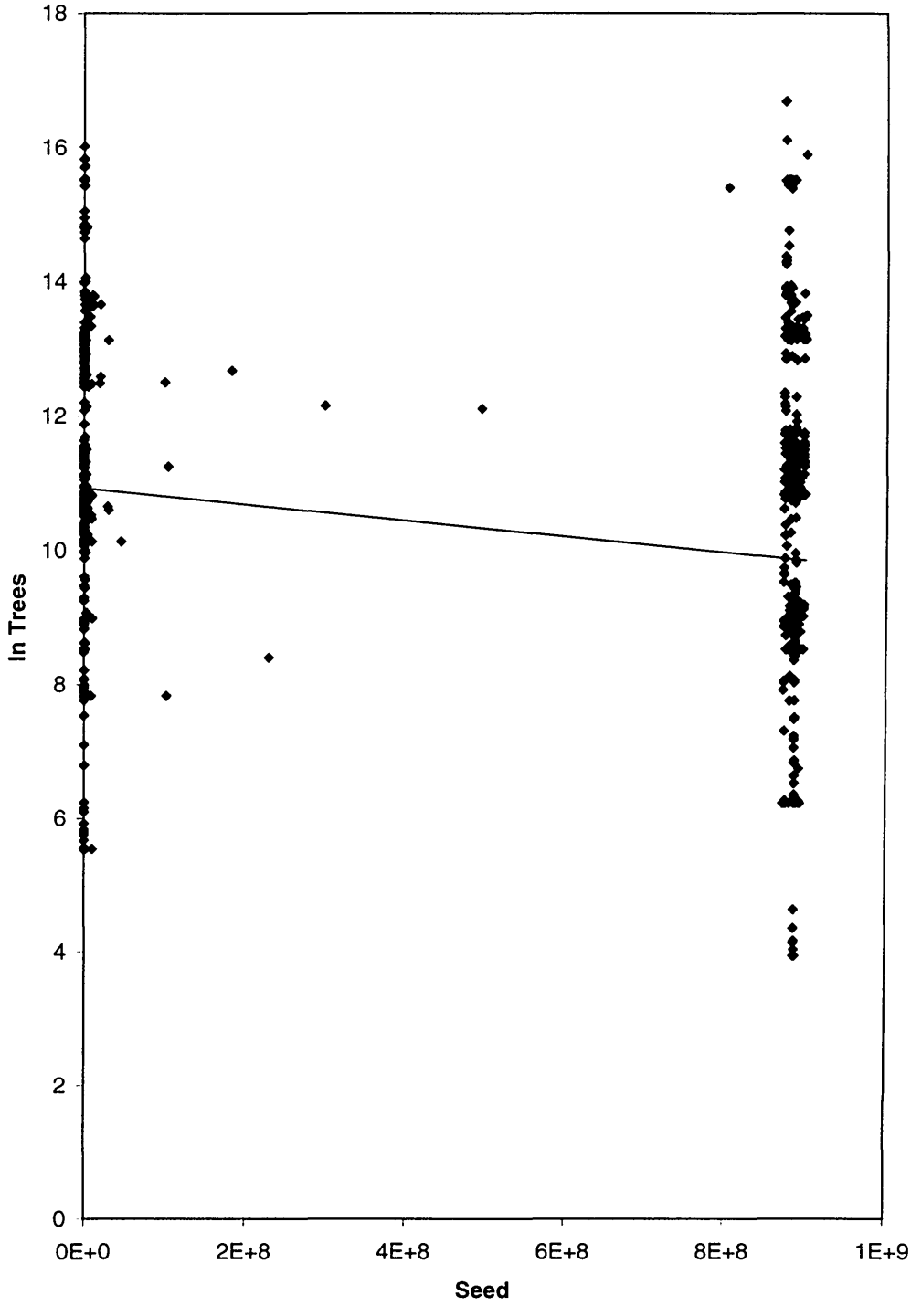


Figure 4.8 The relationship observed between ln(number of trees considered) and random number seed, for the general data matrix. Line: linear regression line.

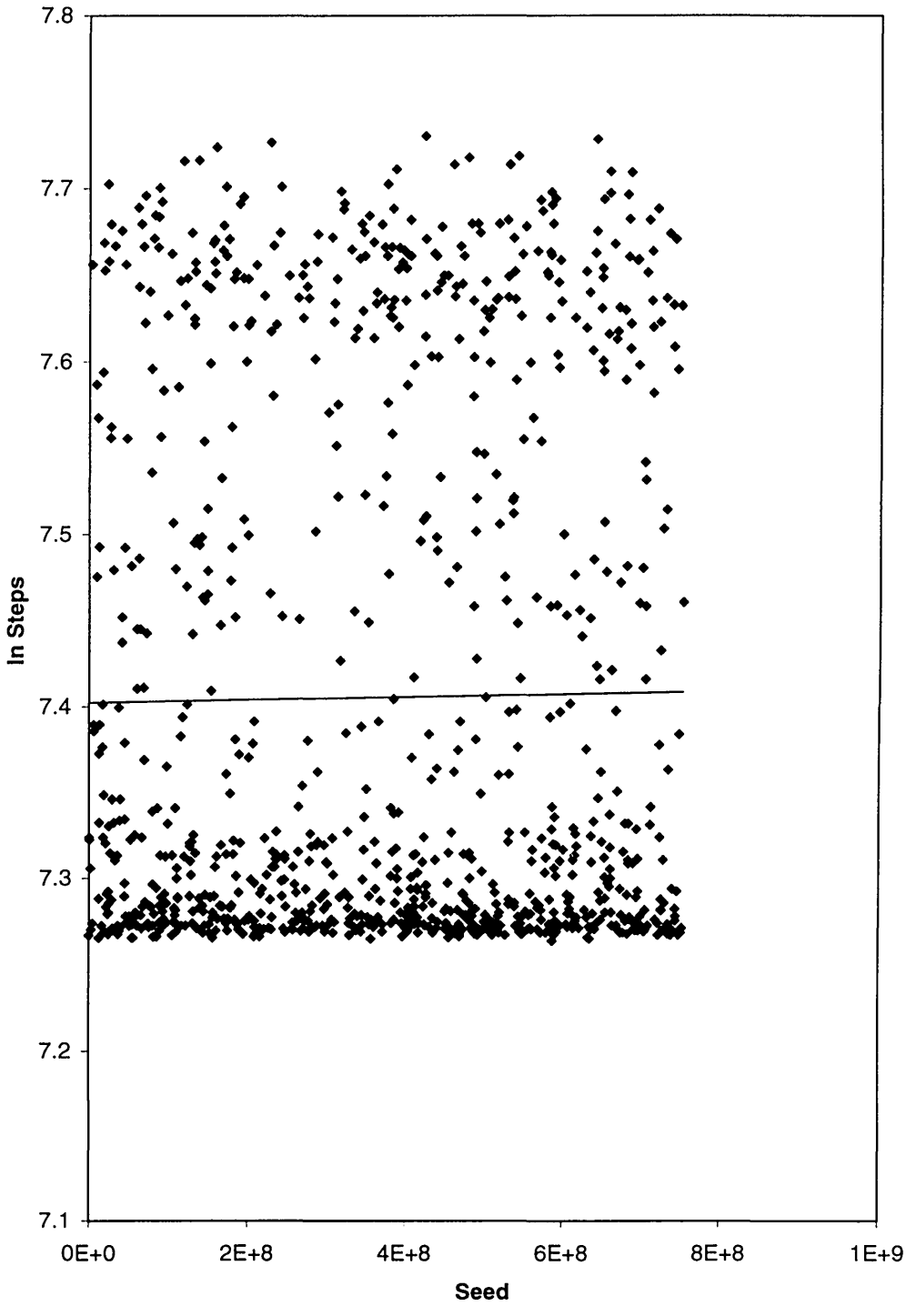


Figure 4.9 The relationship observed between ln(steps on shortest tree found) and random number seed, for the monocot data matrix. Line: linear regression line.

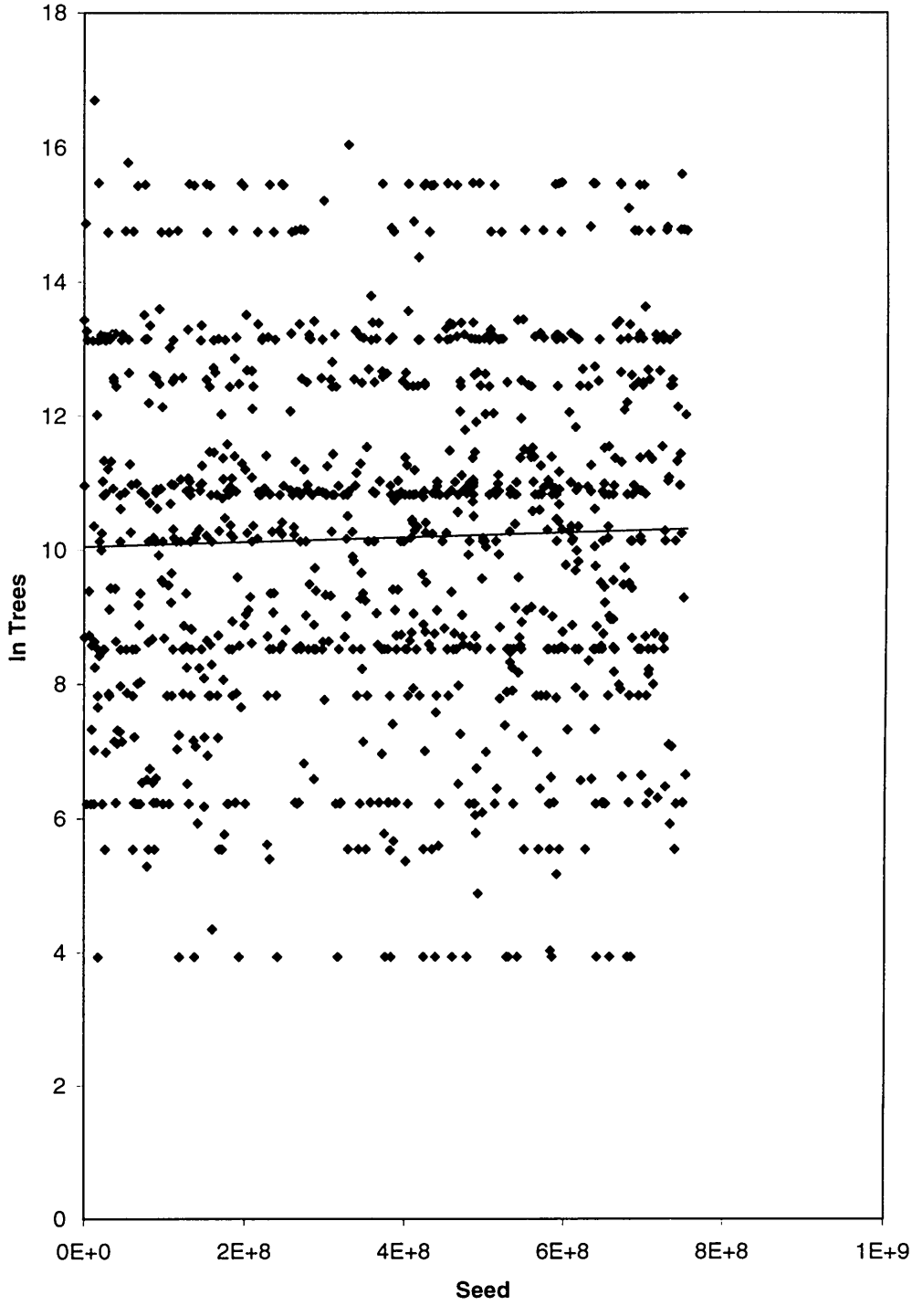


Figure 4.10 The relationship observed between ln(number of trees considered) and random number seed, for the monocot data matrix. Line: linear regression line.

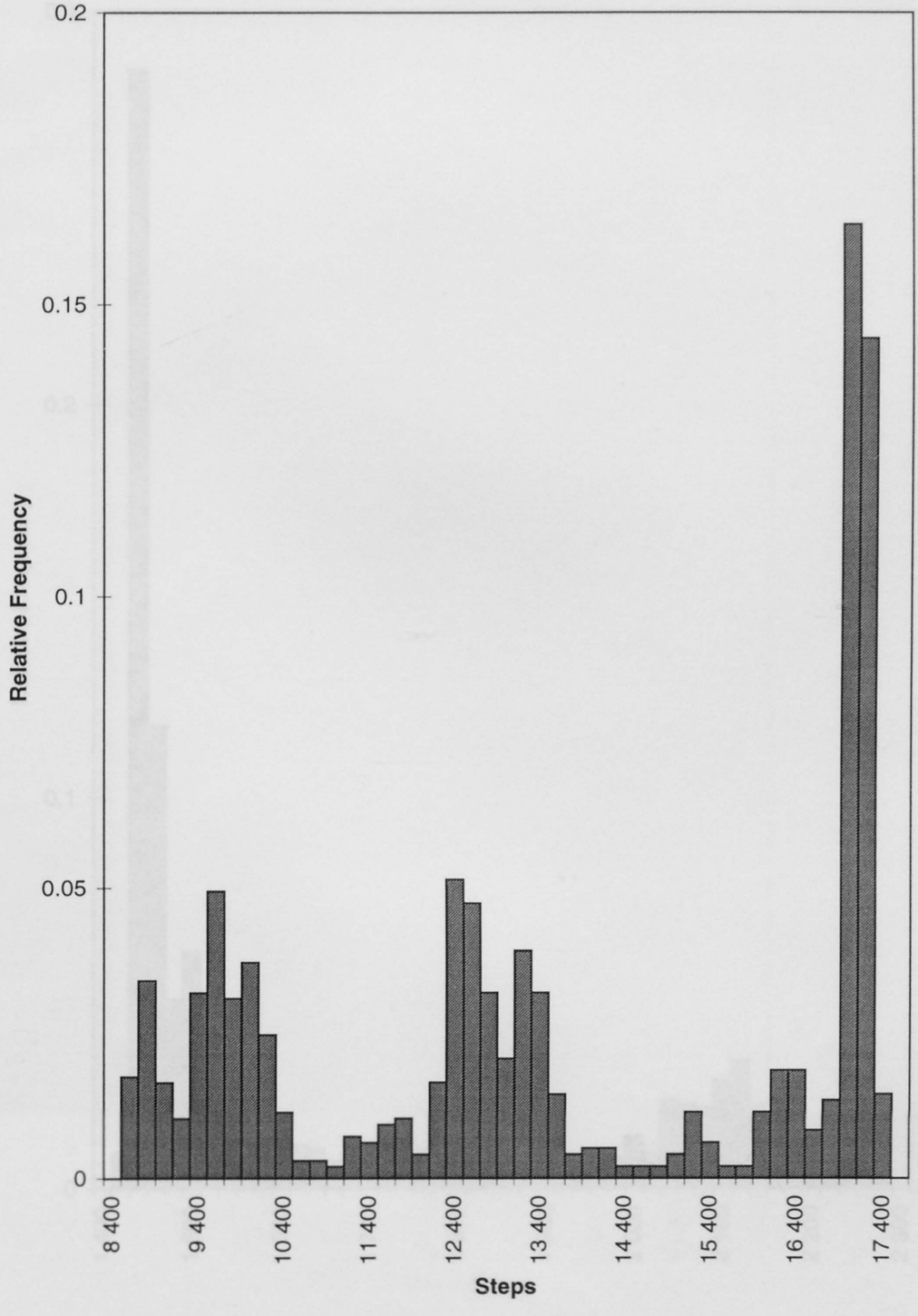


Figure 4.11 Frequency histogram for steps on the shortest tree found, for the general data matrix. The class interval is 200 steps. Where present, class labels give the class mid-point.

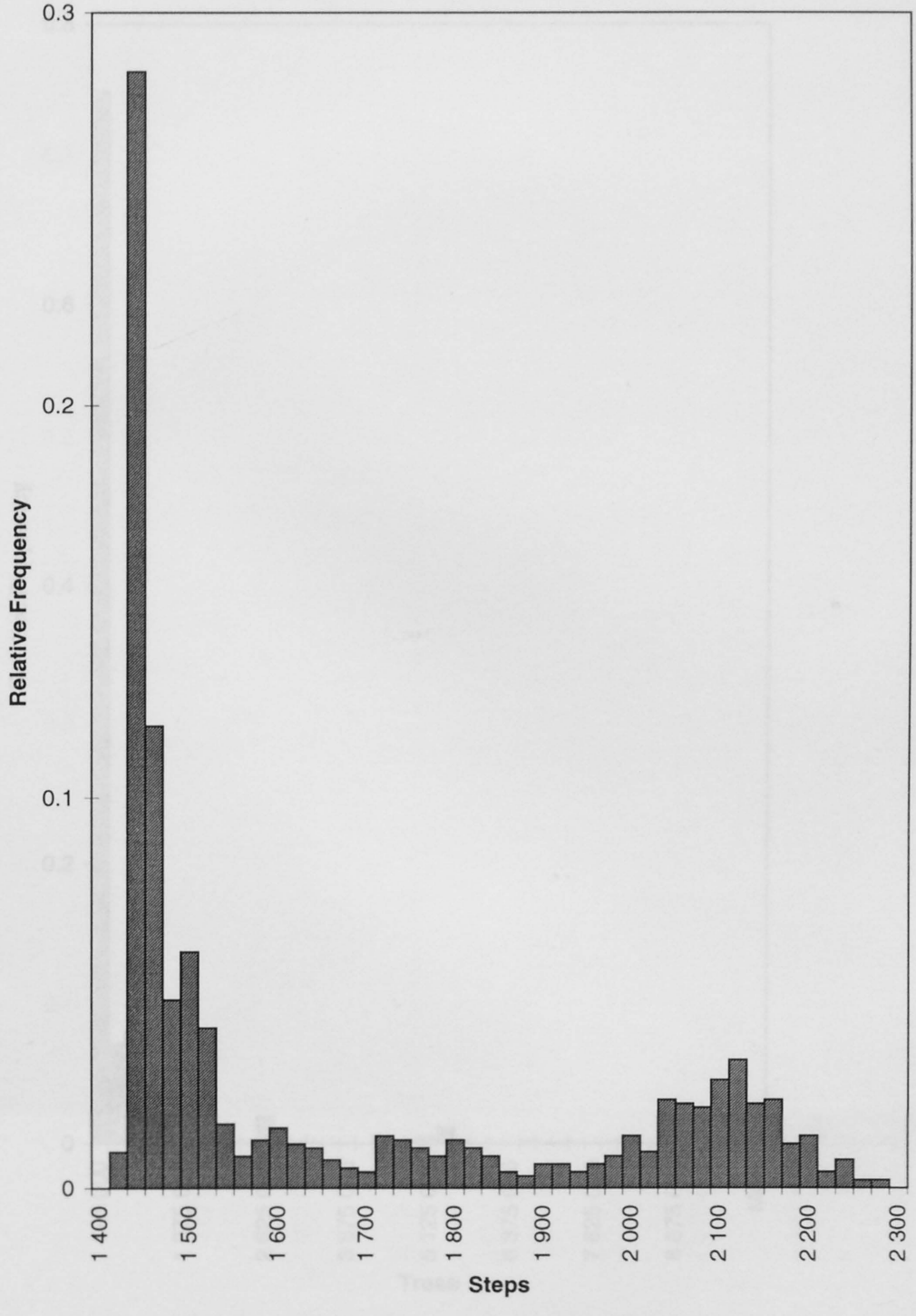


Figure 4.12 Frequency histogram for steps on the shortest tree found, for the monocot data matrix. The class interval is 20 steps. Where present, class labels give the class mid-point.

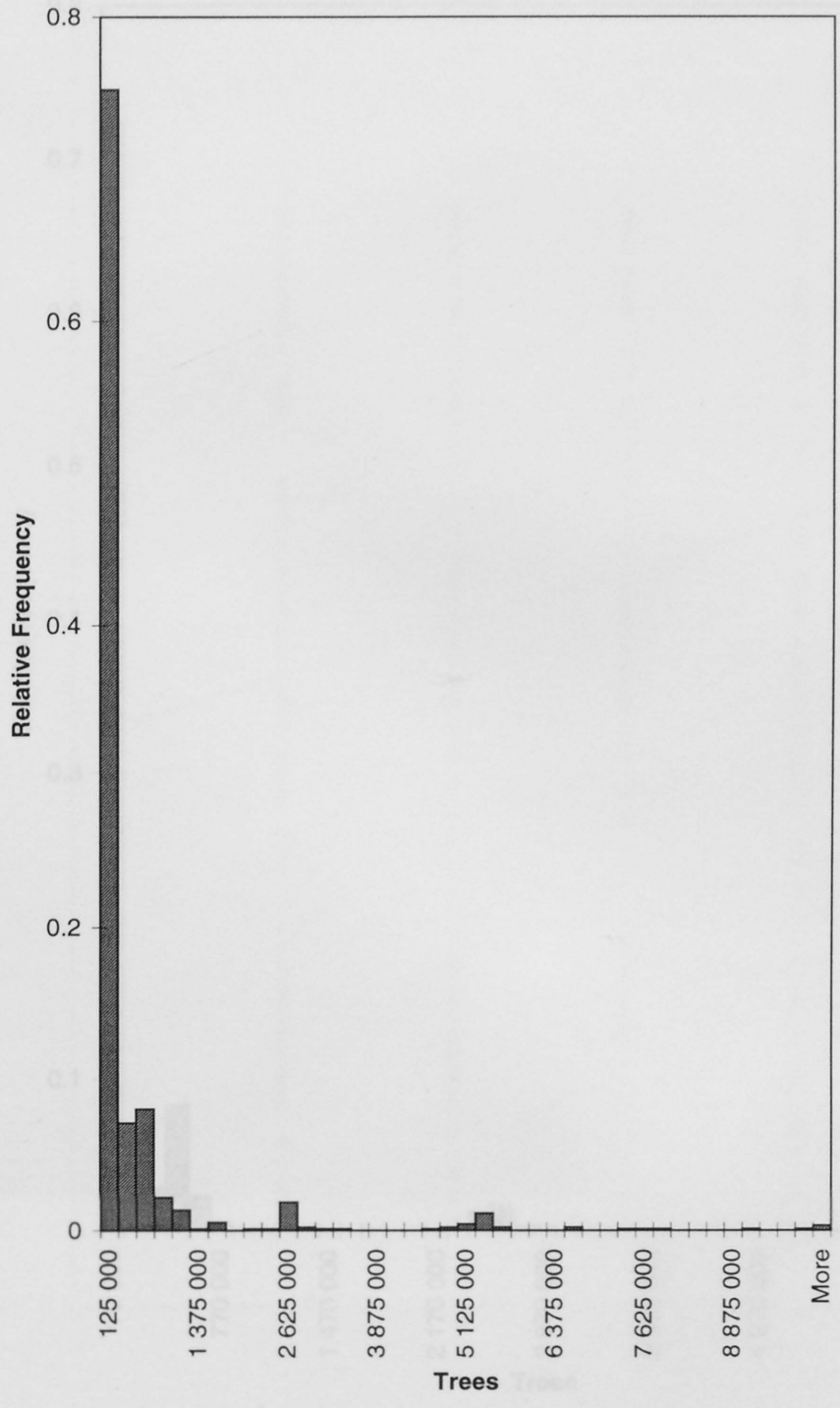


Figure 4.13 Frequency histogram for number of trees considered in the search, for the general data matrix. The class interval is 250 000 trees. Where present, class labels (except 'more') give the class mid-point.

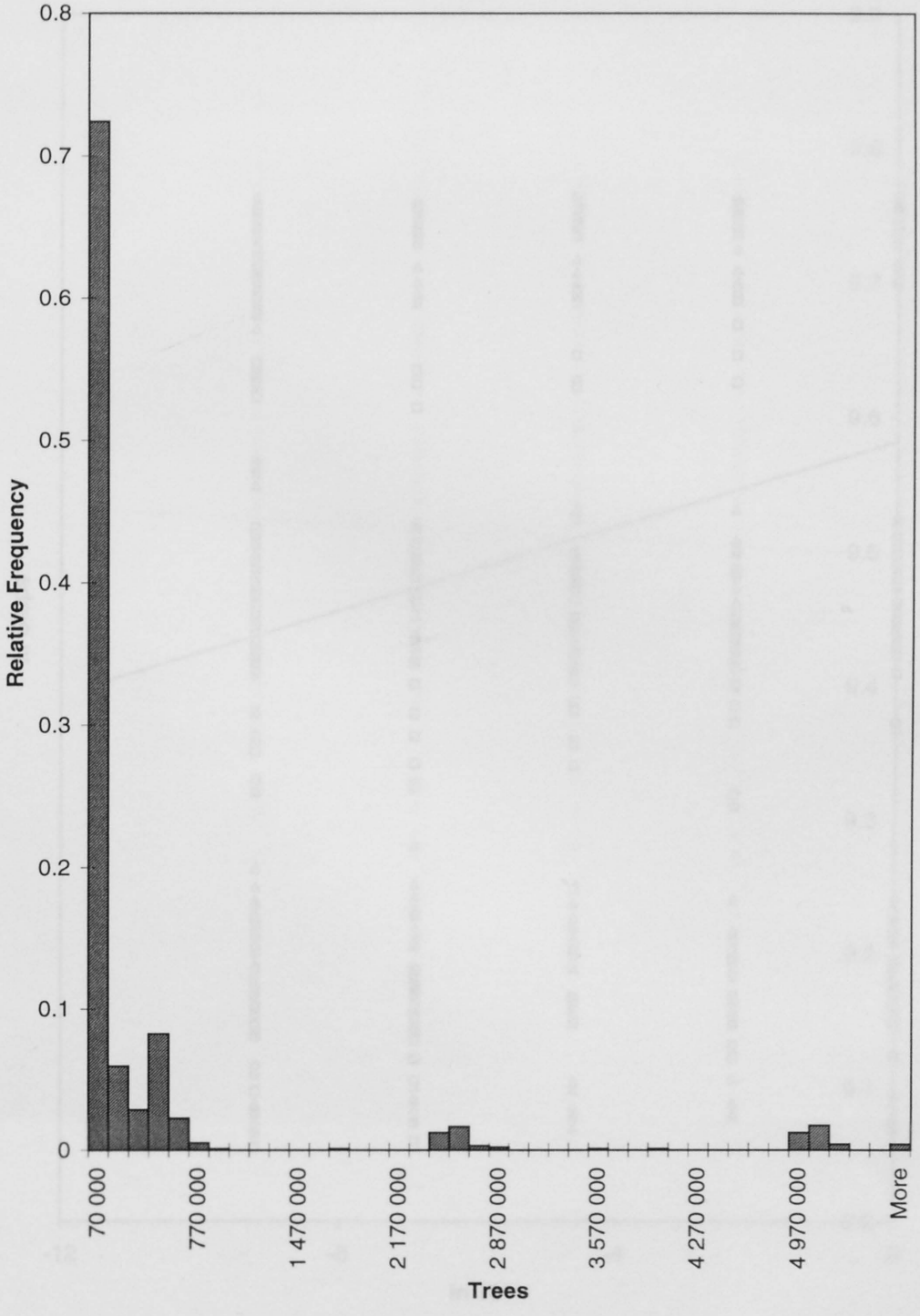


Figure 4.14 Frequency histogram for number of trees considered in the search, for the monocot data matrix. The class interval is 140 000 trees. Where present, class labels (except 'more') give the class mid-point.

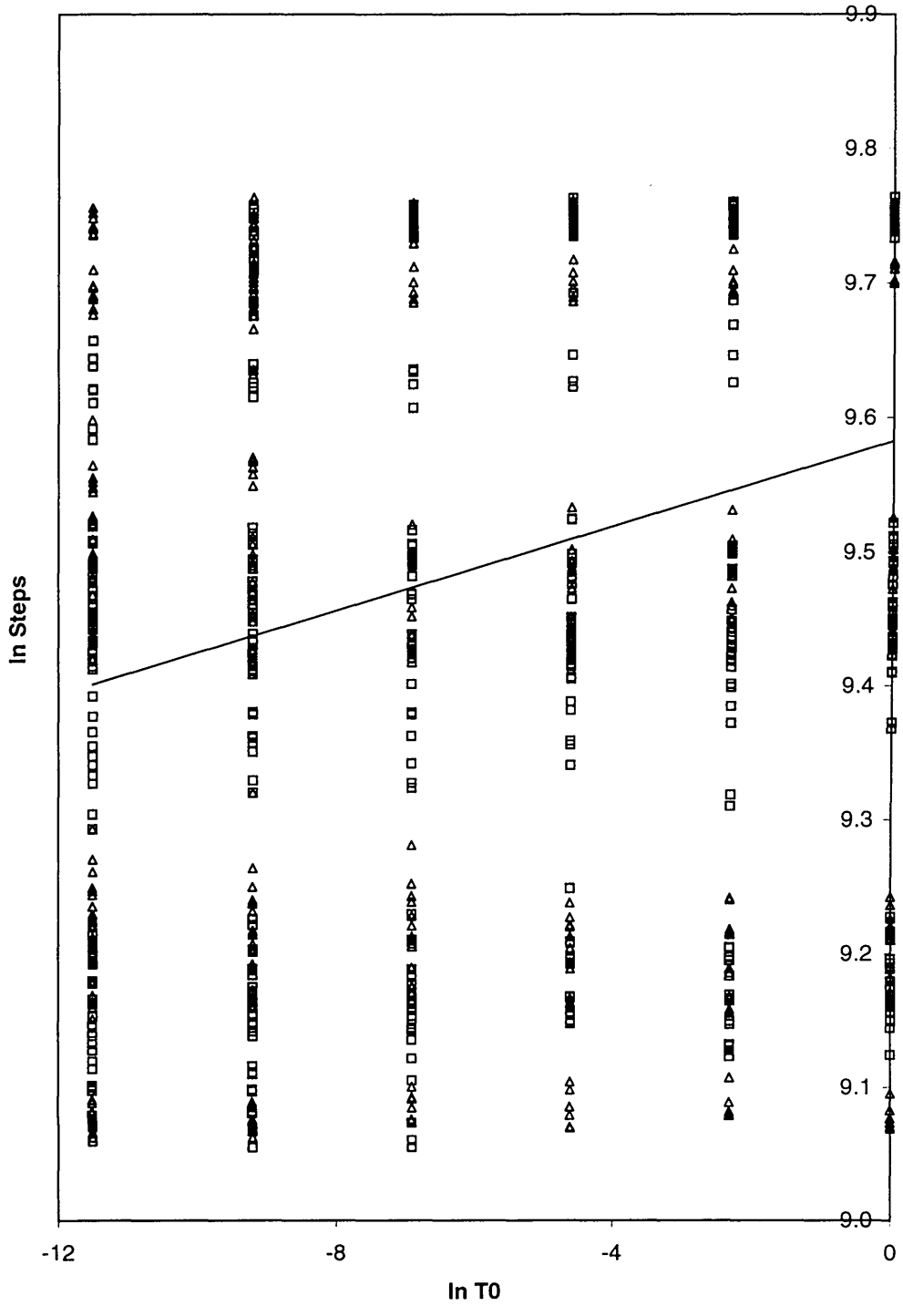


Figure 4.15 The relationship observed between ln(steps on shortest tree found) and $\ln(T_0)$, for the general data matrix. □ `maxaccept` $= 5$. △ `maxaccept` $= 50$. Line: linear regression line for all observations together.

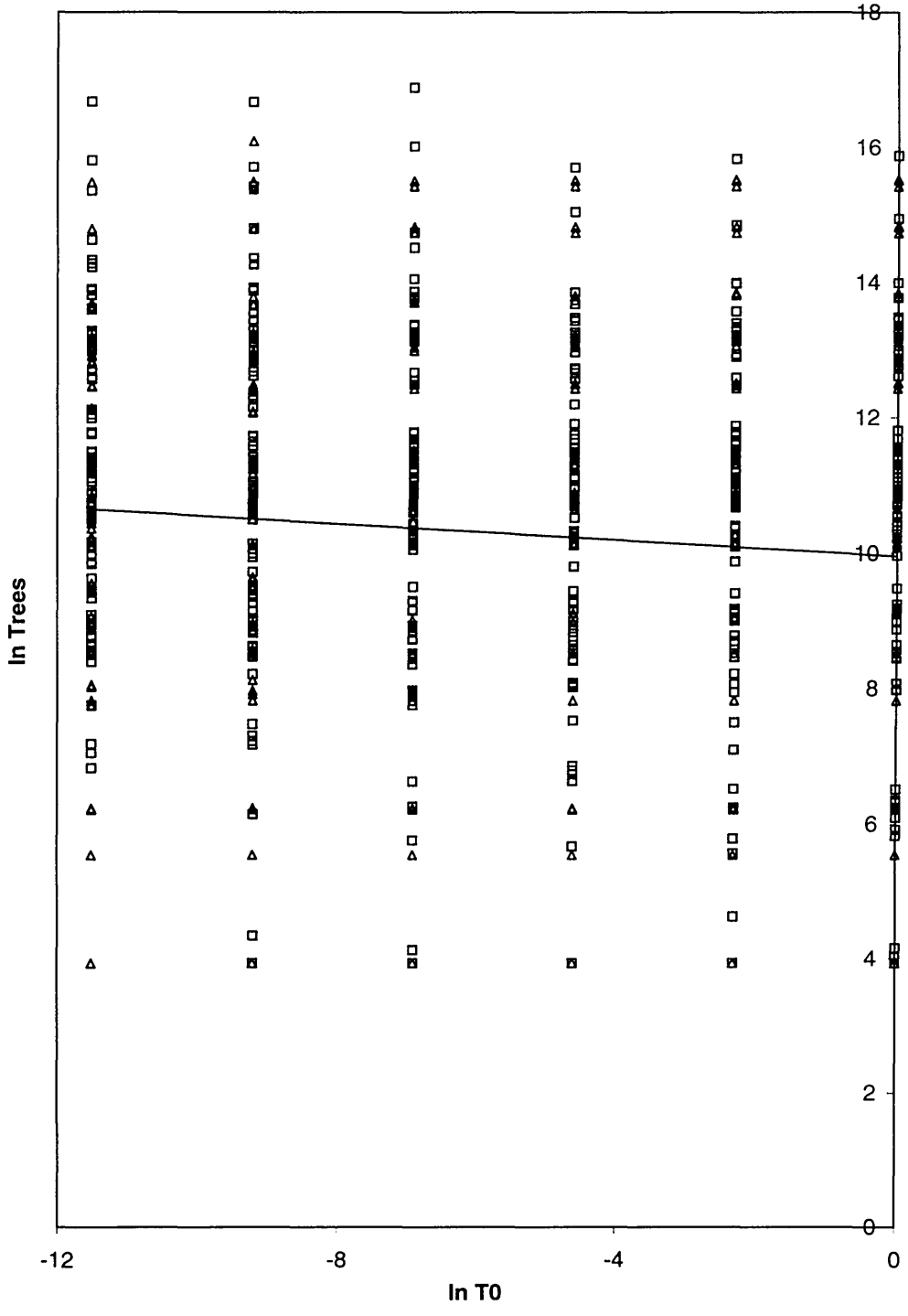


Figure 4.16 The relationship observed between ln(number of trees considered) and $\ln(T_0)$, for the general data matrix. □ `maxaccept` $= 5$. △ `maxaccept` $= 50$. Line: linear regression line for all observations together.

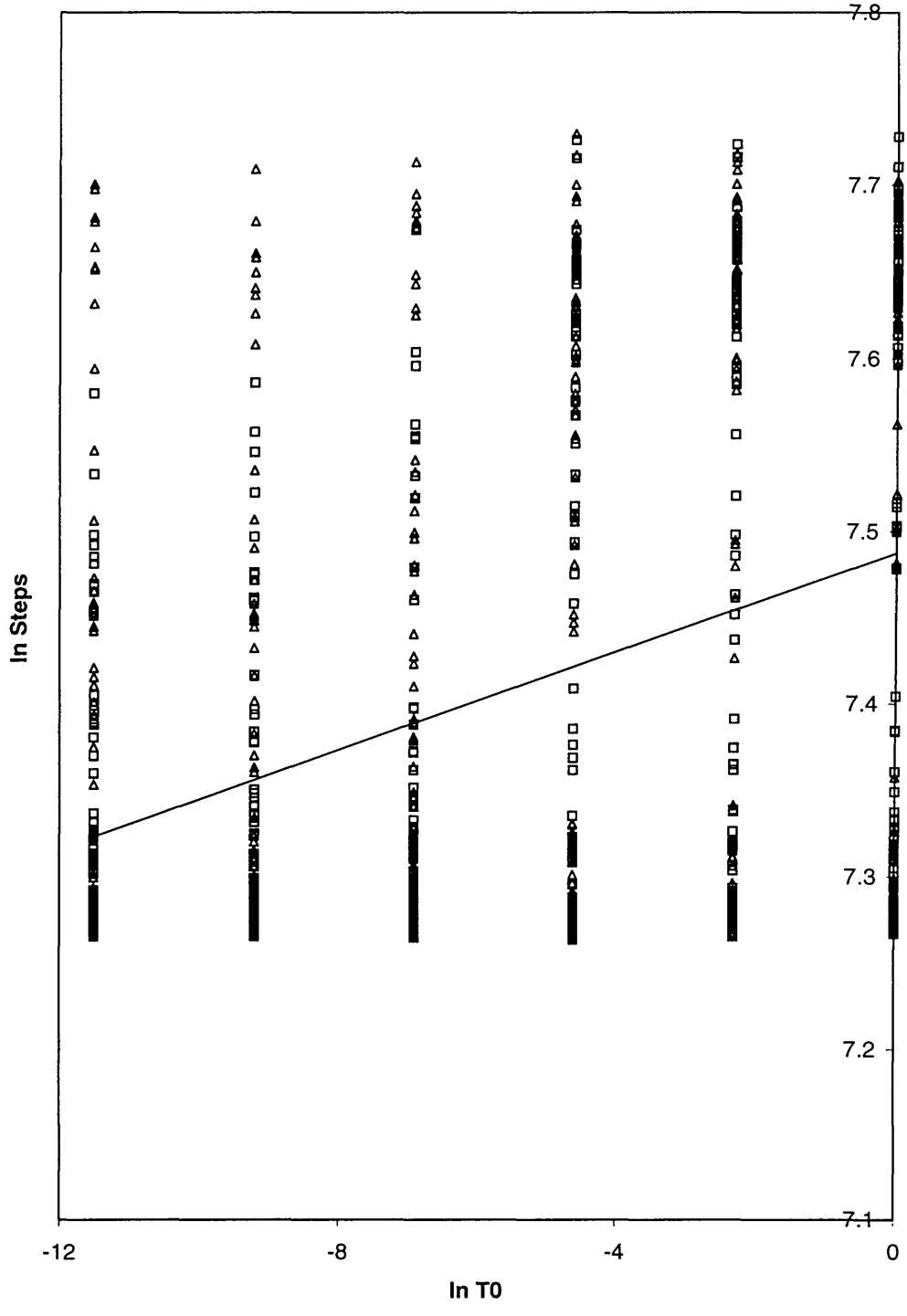


Figure 4.17 The relationship observed between ln(steps on shortest tree found) and $\ln(T_0)$, for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

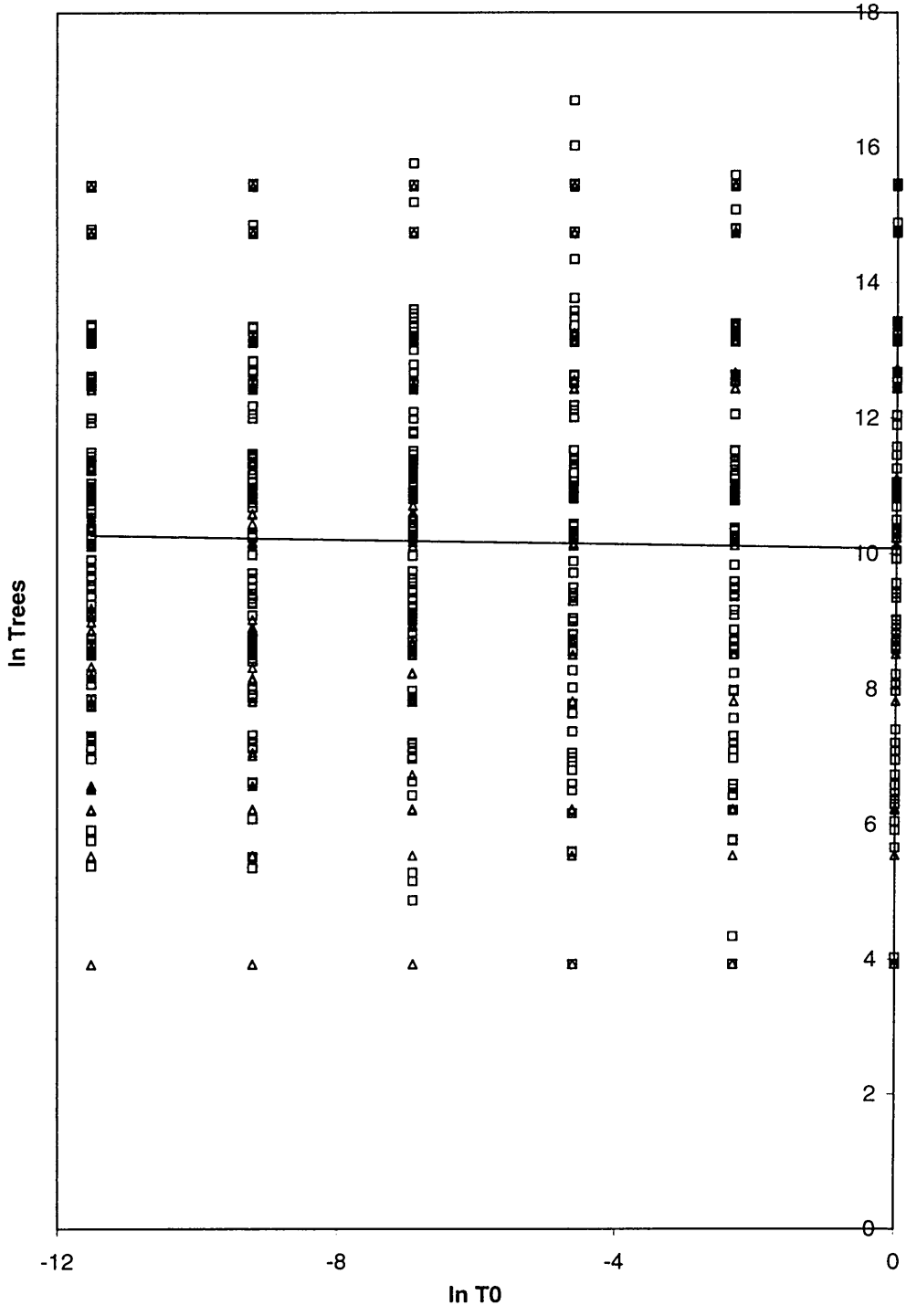


Figure 4.18 The relationship observed between ln(number of trees considered) and $\ln(T_0)$, for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

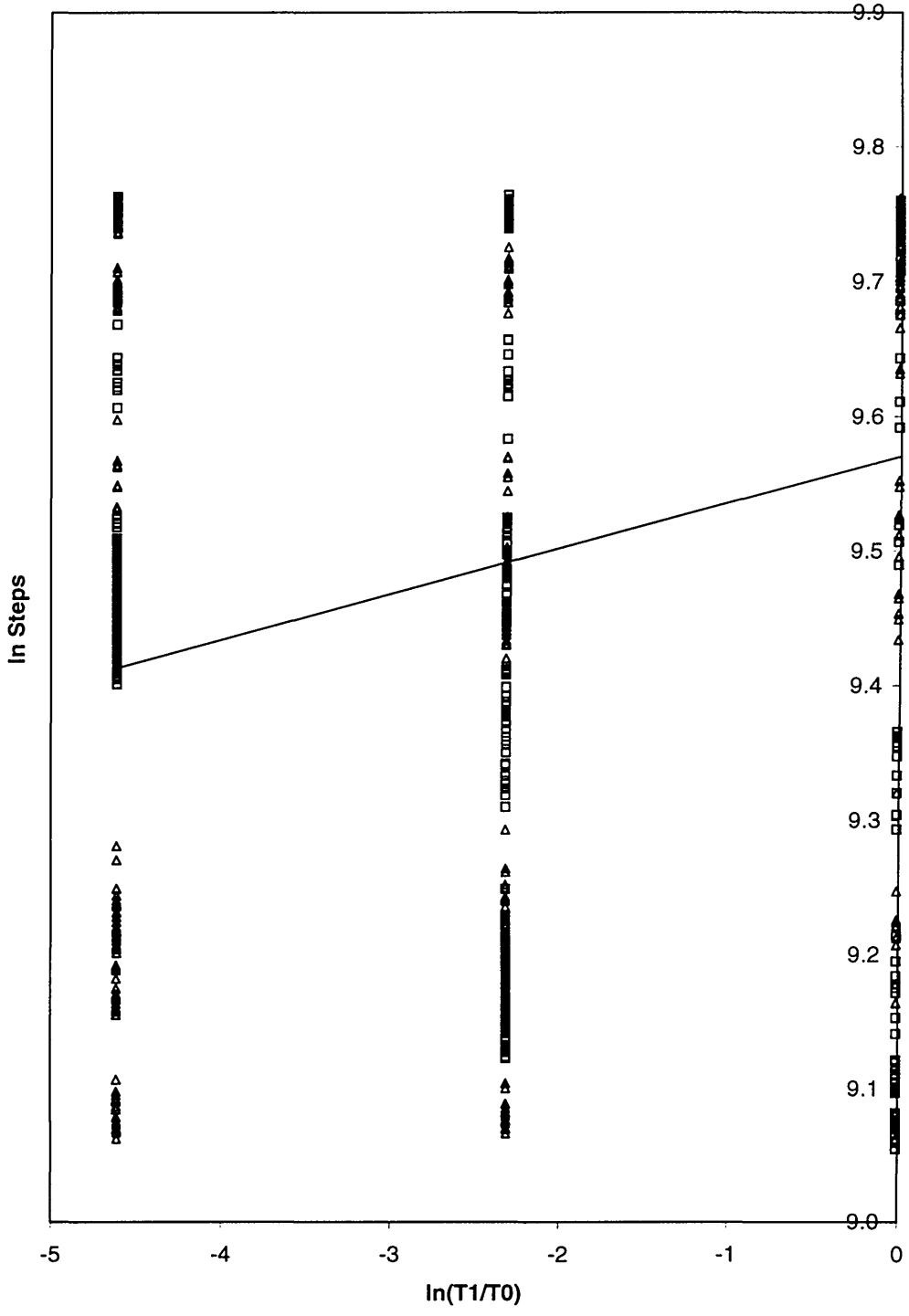


Figure 4.19 The relationship observed between ln(steps on shortest tree found) and $\ln(T_1/T_0)$, for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

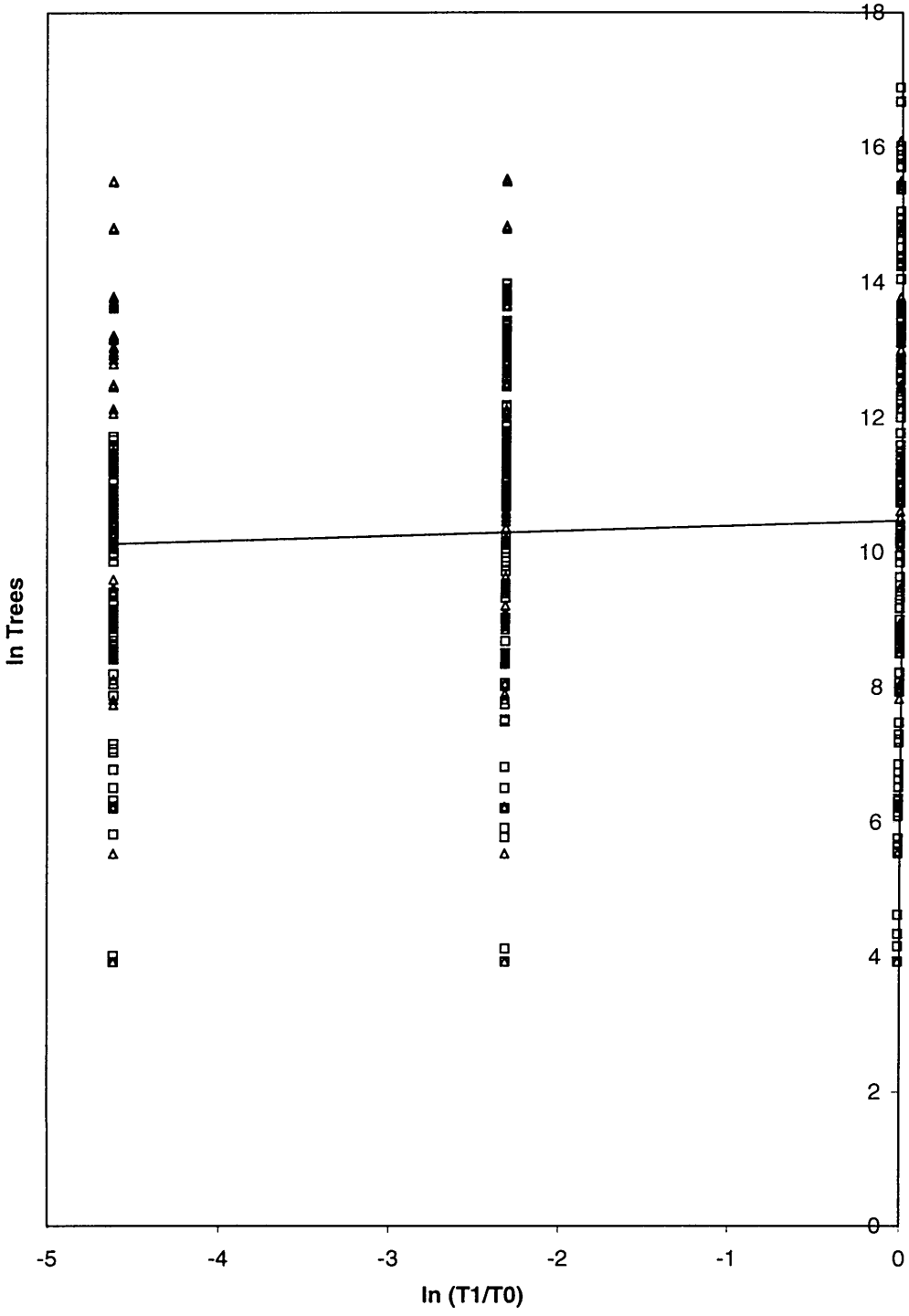


Figure 4.20 The relationship observed between ln(number of trees considered) and $\ln(T_1/T_0)$, for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

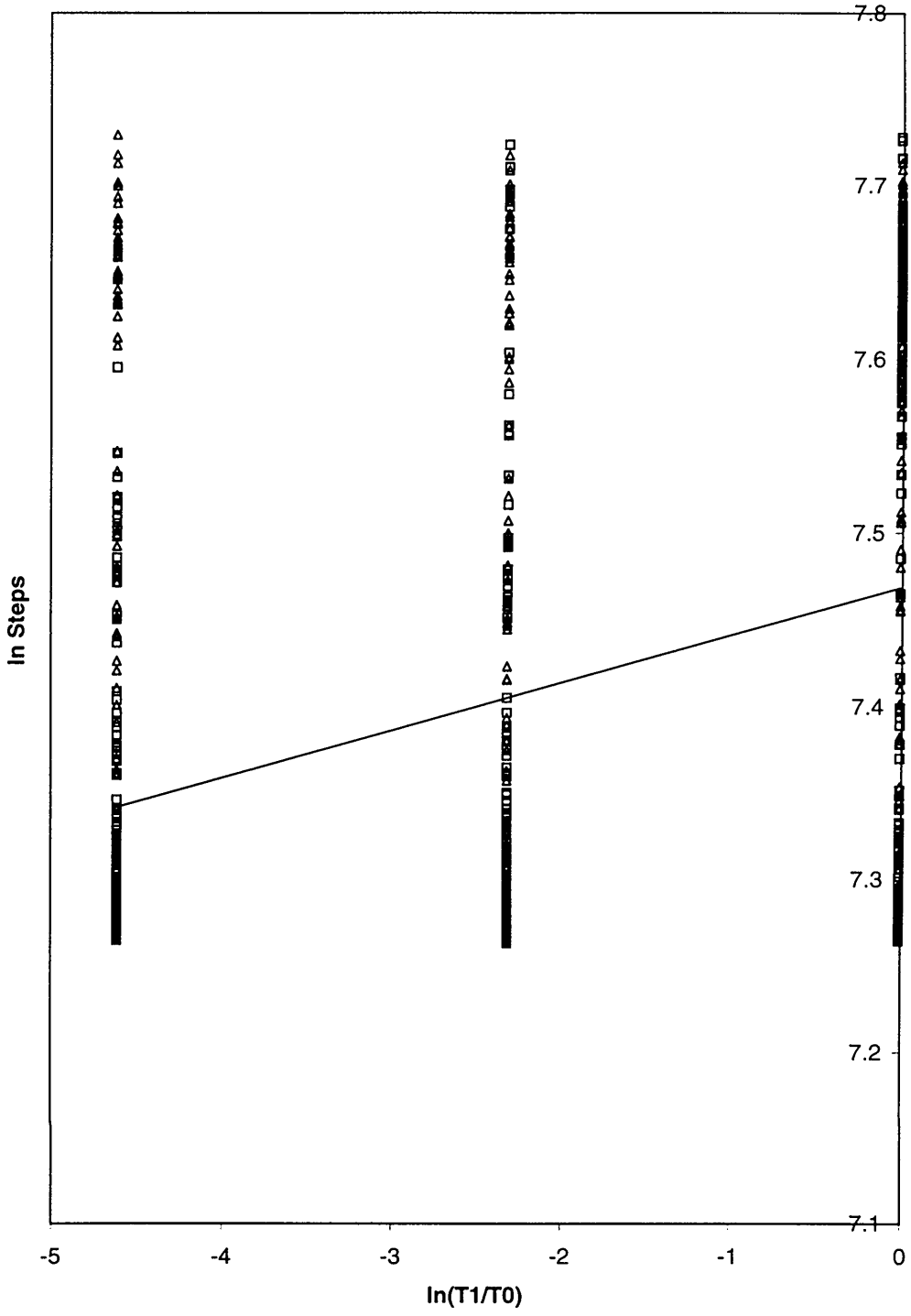


Figure 4.21 The relationship observed between ln(steps on shortest tree found) and $\ln(T_1/T_0)$, for the monocot data matrix. □ `maxaccept` $= 5$. △ `maxaccept` $= 50$. Line: linear regression line for all observations together.

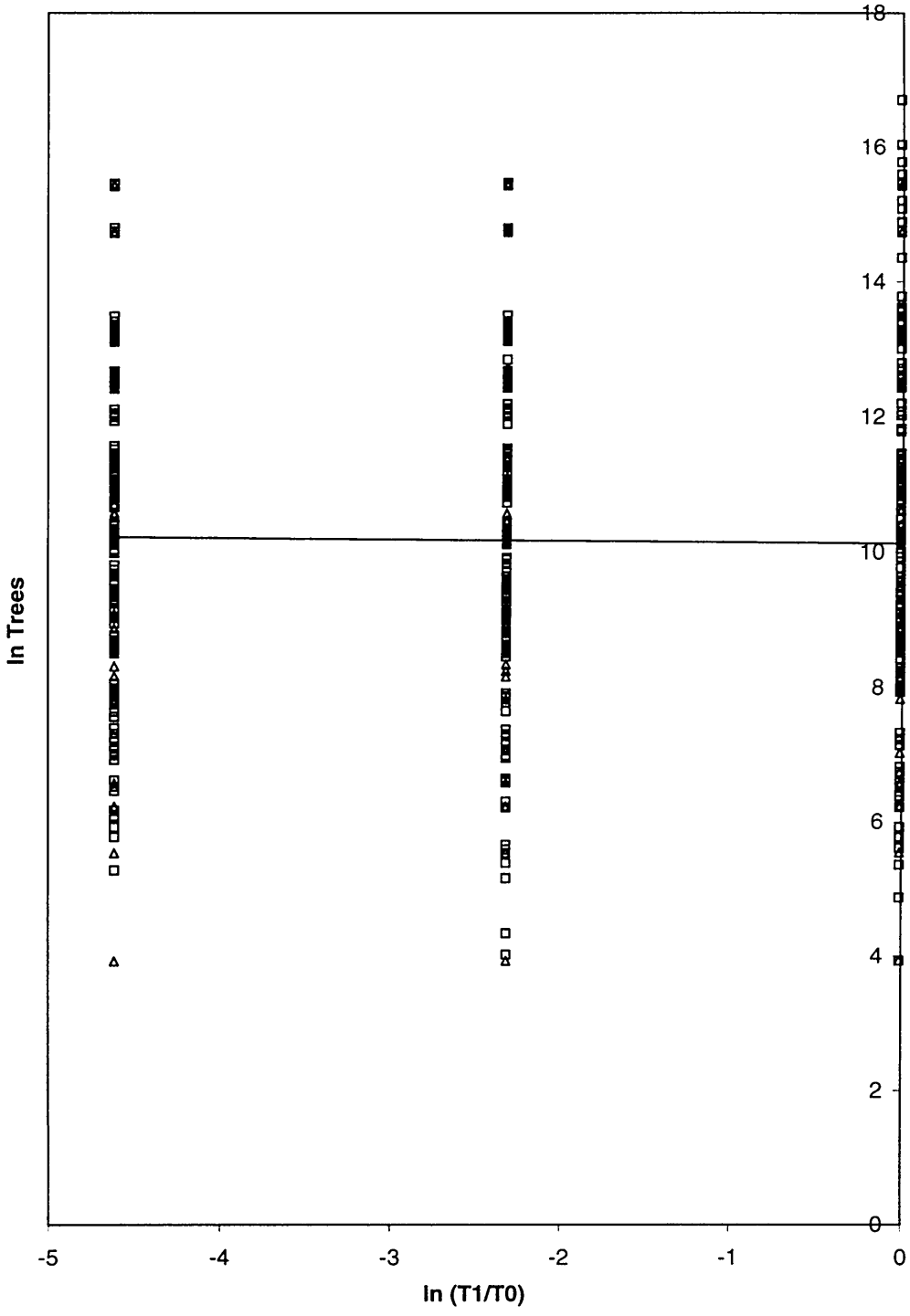


Figure 4.22 The relationship observed between ln(number of trees considered) and $\ln(T_1/T_0)$, for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

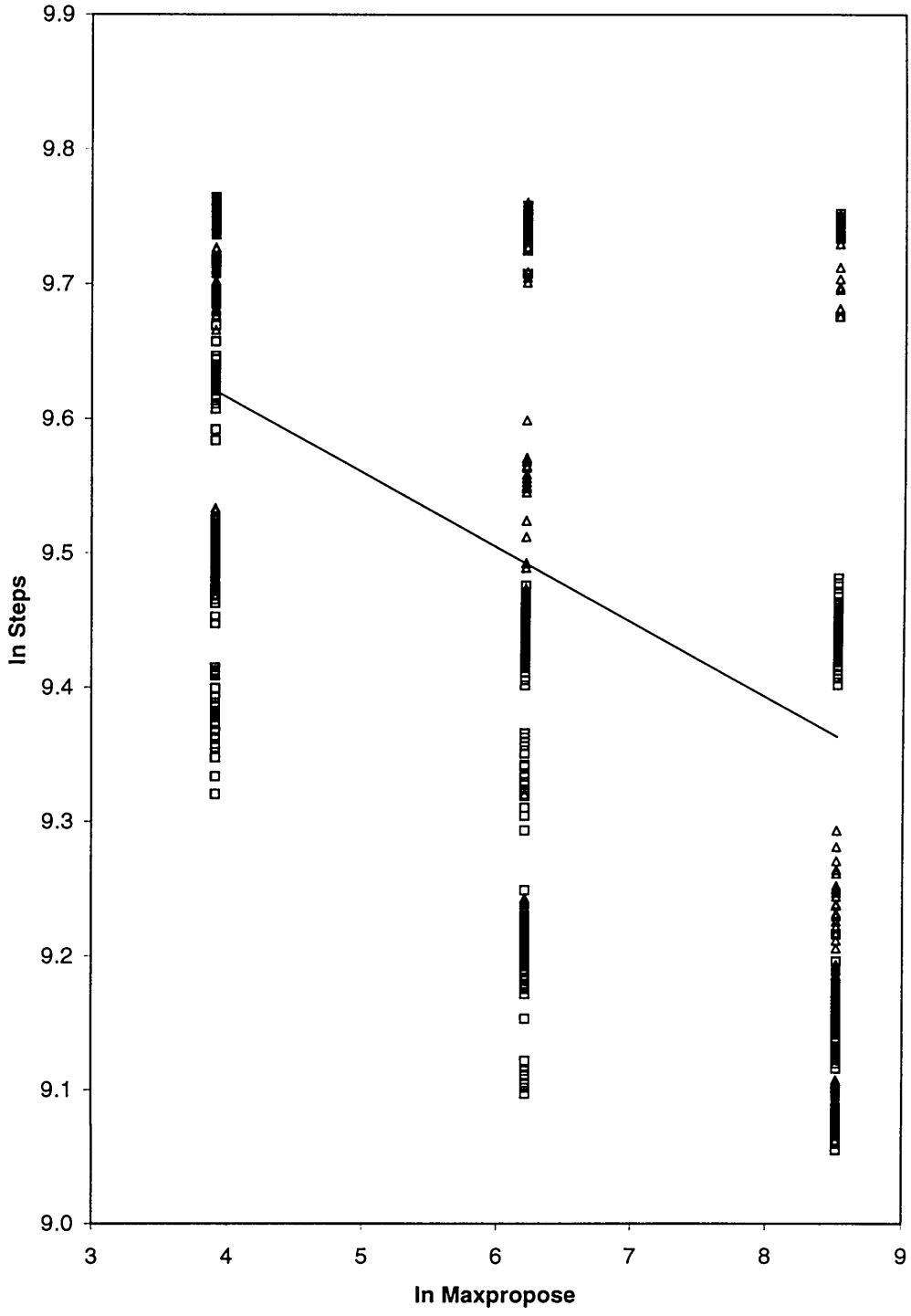


Figure 4.23 The relationship observed between ln(steps on shortest tree found) and ln(`maxpropose`), for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

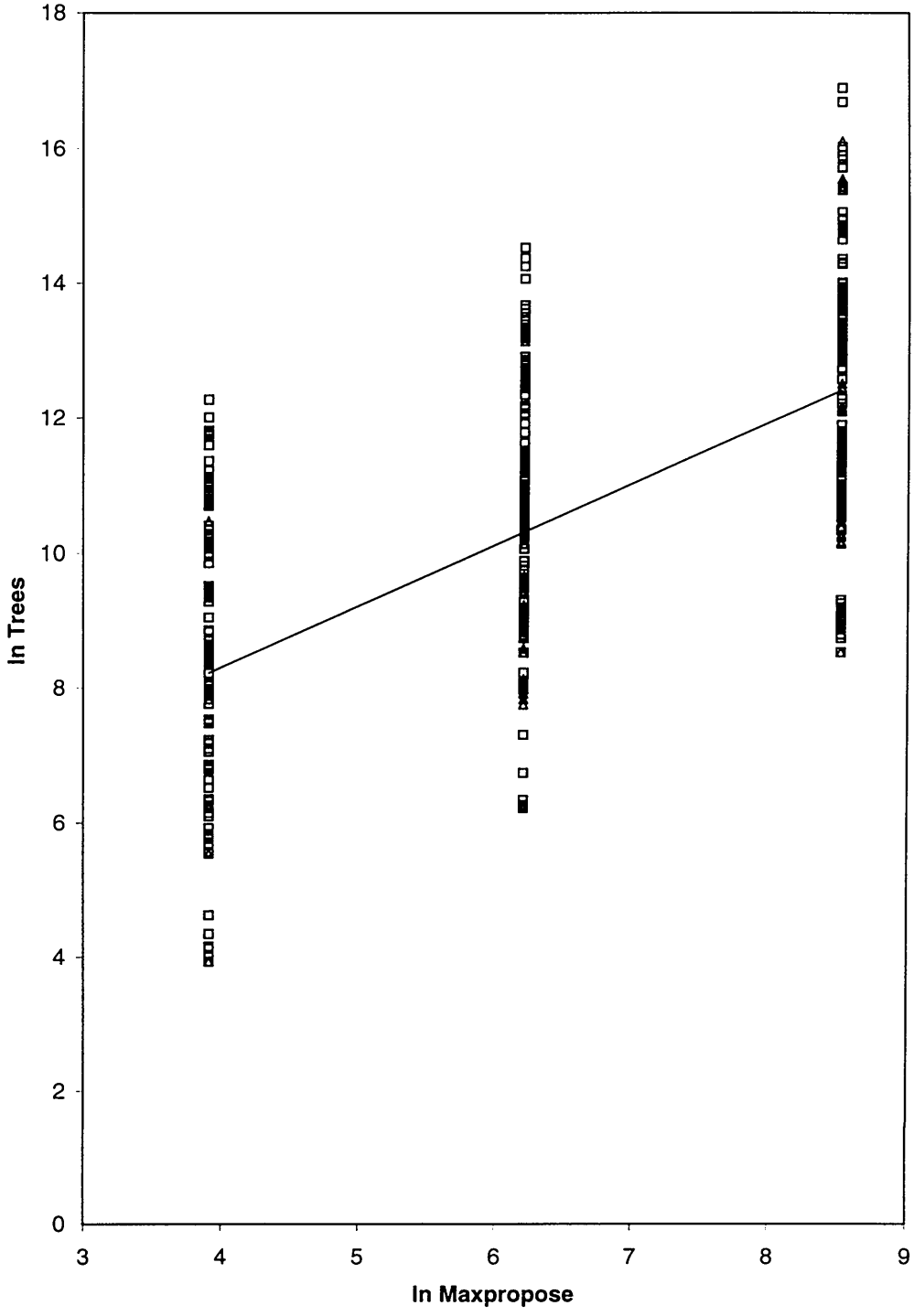


Figure 4.24 The relationship observed between ln(number of trees considered) and ln(`maxpropose`), for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

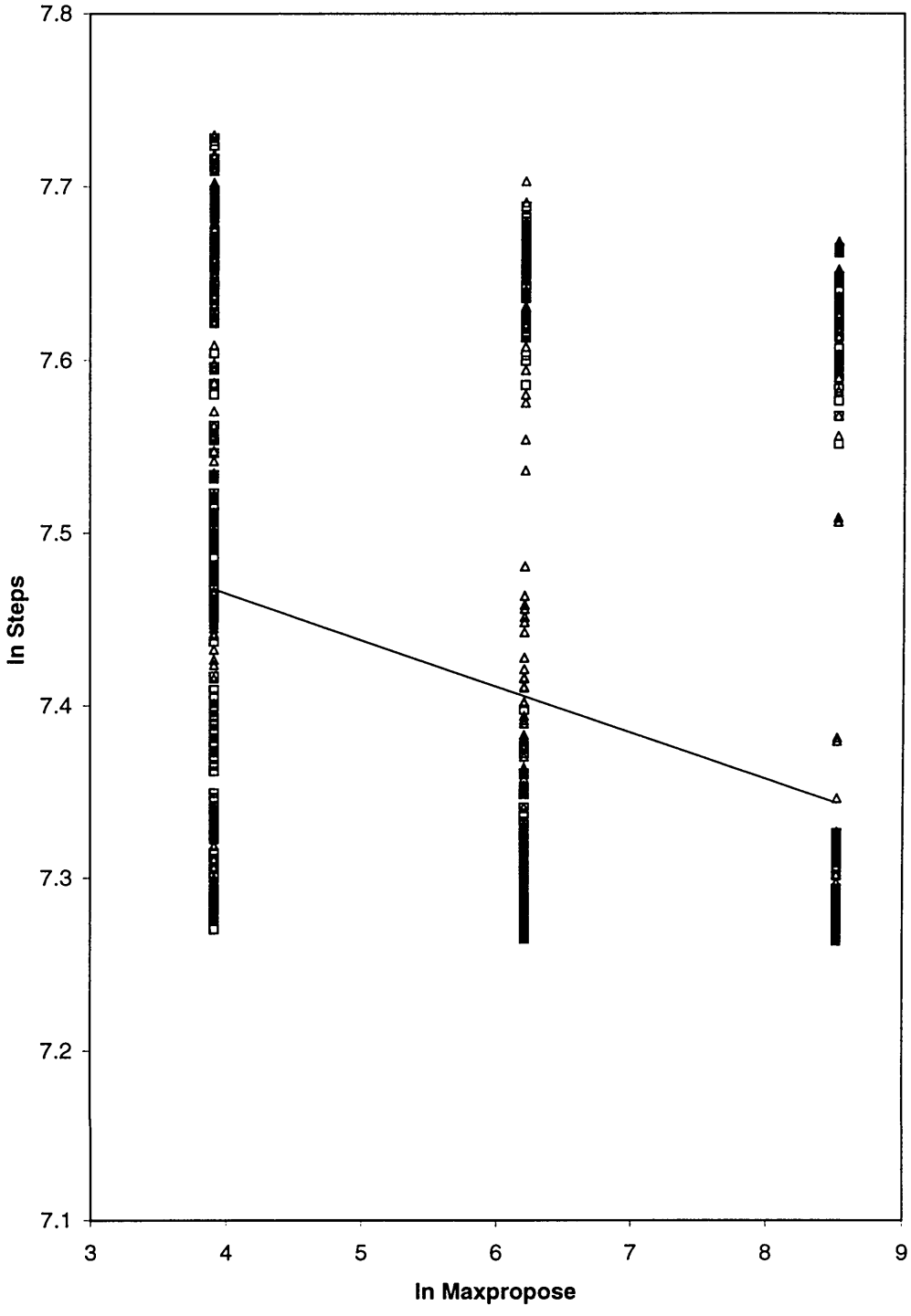


Figure 4.25 The relationship observed between ln(steps on shortest tree found) and ln(`maxpropose`), for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

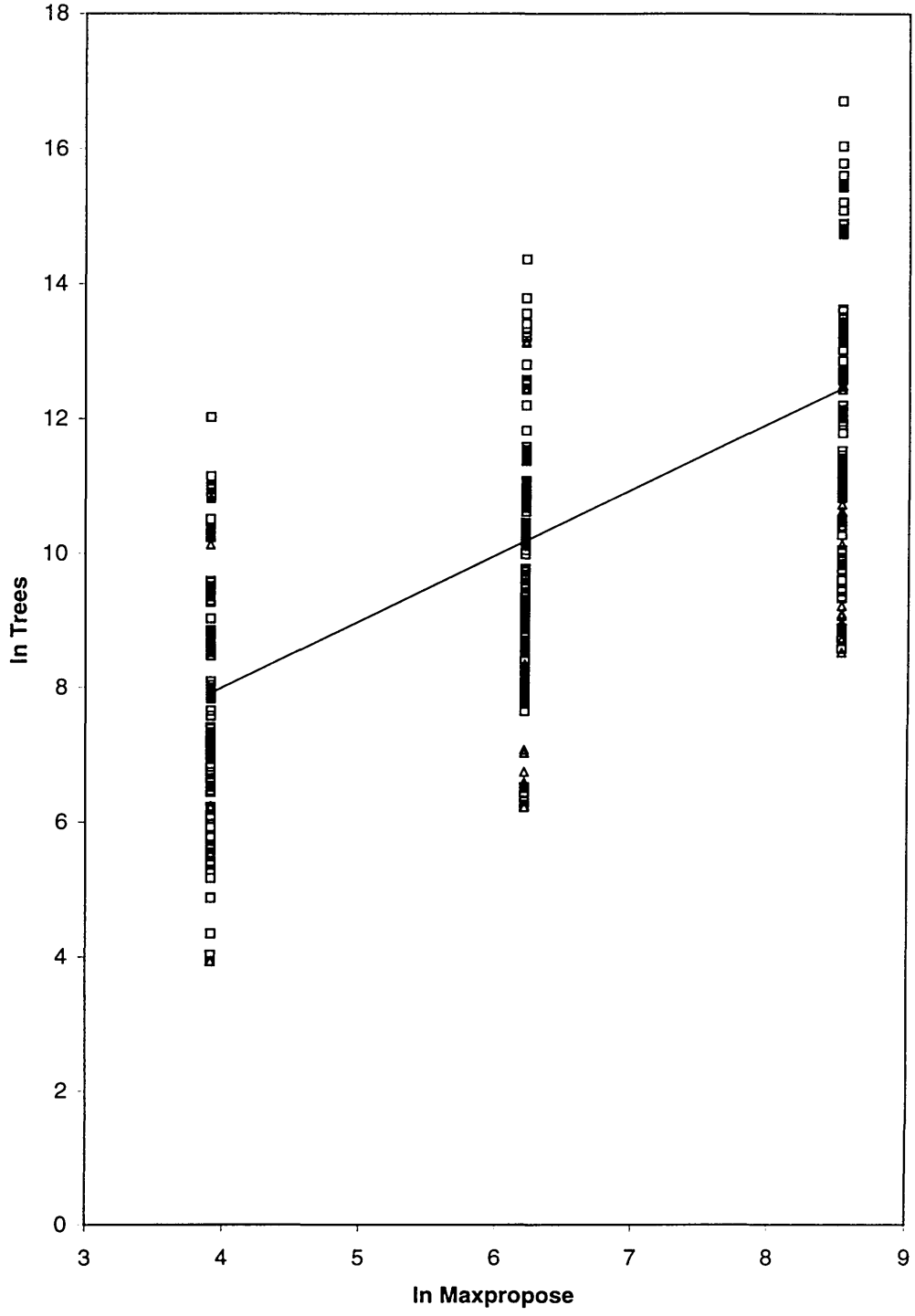


Figure 4.26 The relationship observed between ln(number of trees considered) and ln(`maxpropose`), for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

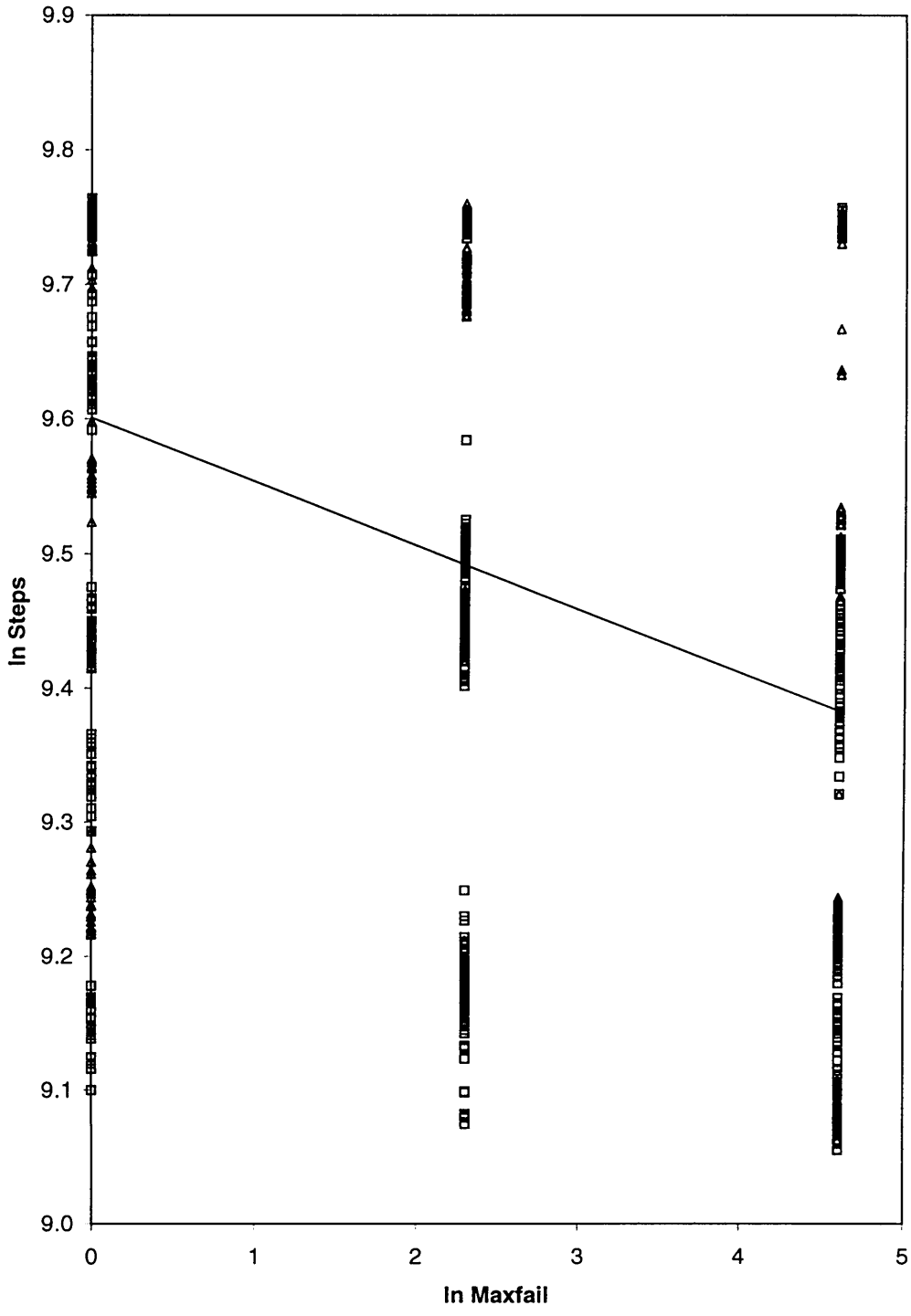


Figure 4.27 The relationship observed between ln(steps on shortest tree found) and ln(`maxfail`), for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

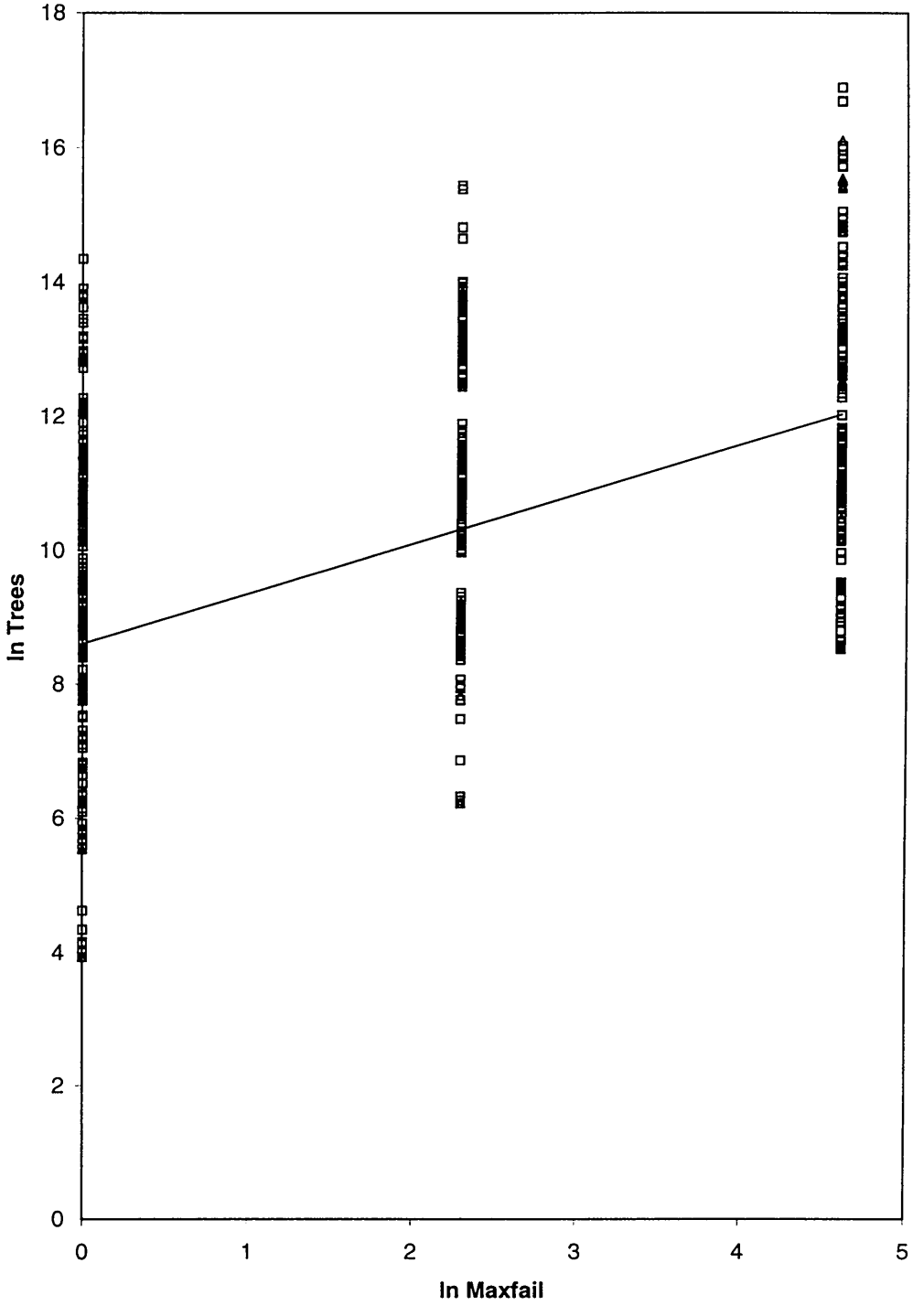


Figure 4.28 The relationship observed between ln(number of trees considered) and ln(`maxfail`), for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

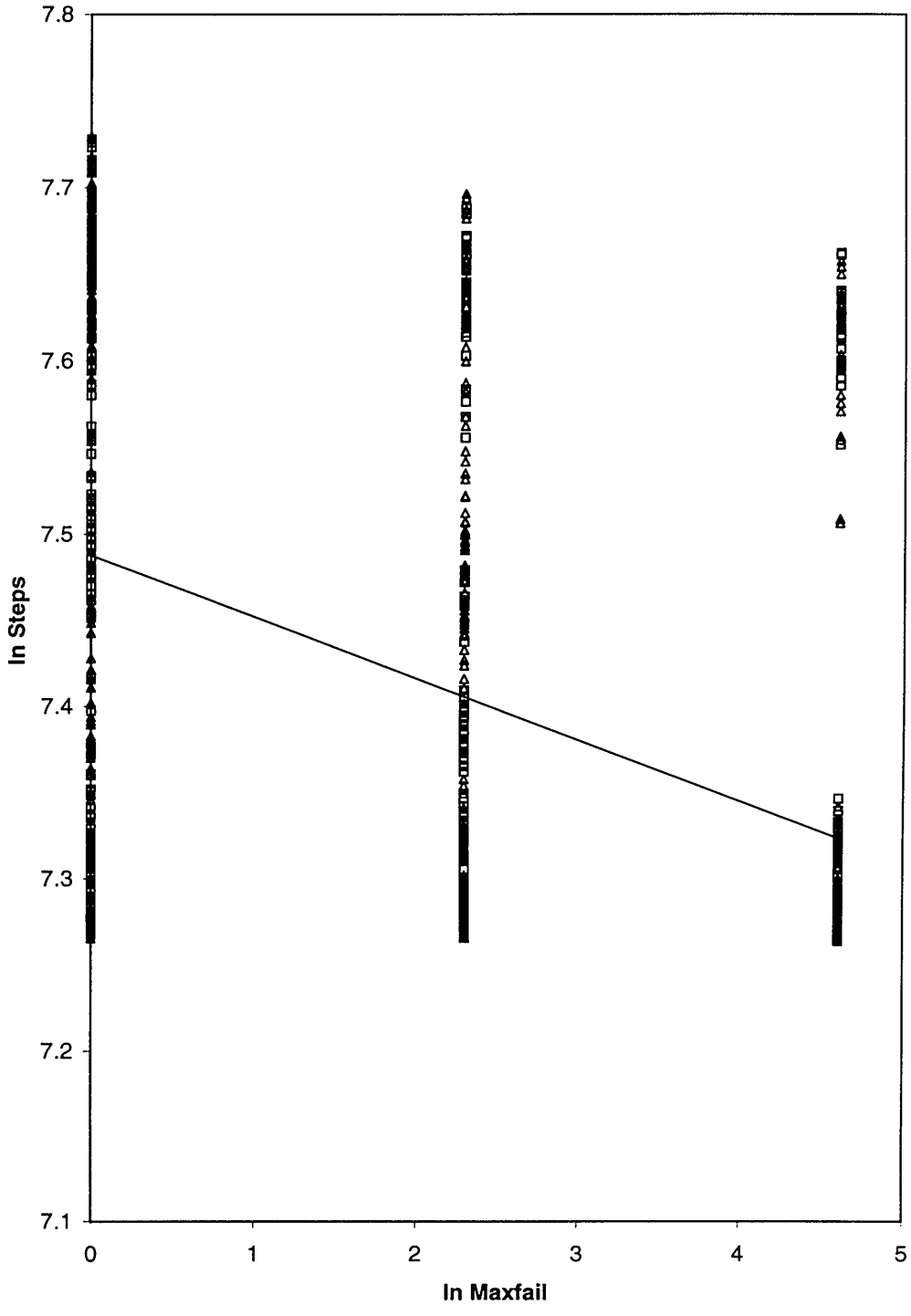


Figure 4.29 The relationship observed between ln(steps on shortest tree found) and ln(`maxfail`), for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

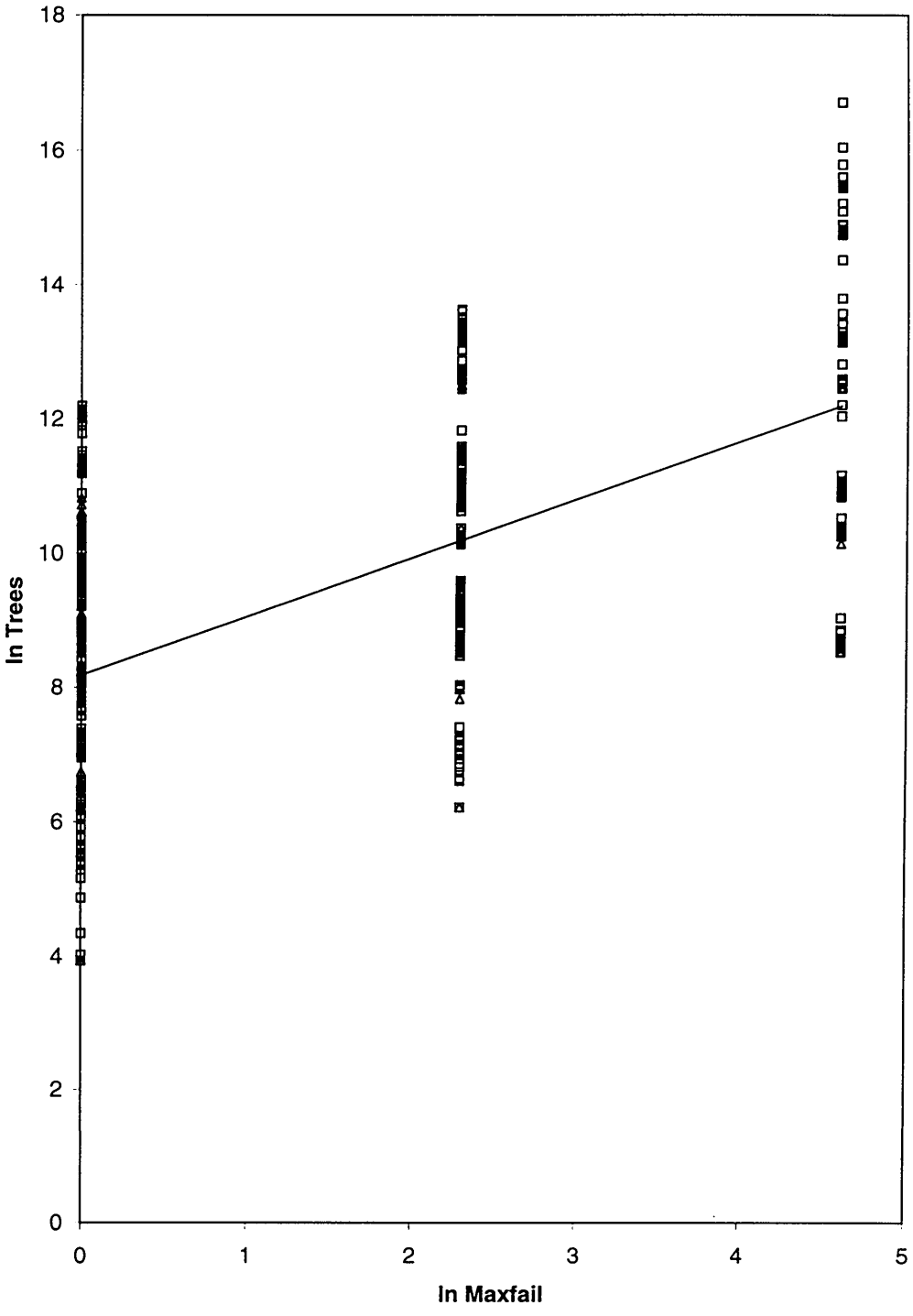


Figure 4.30 The relationship observed between ln(number of trees considered) and ln(`maxfail`), for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

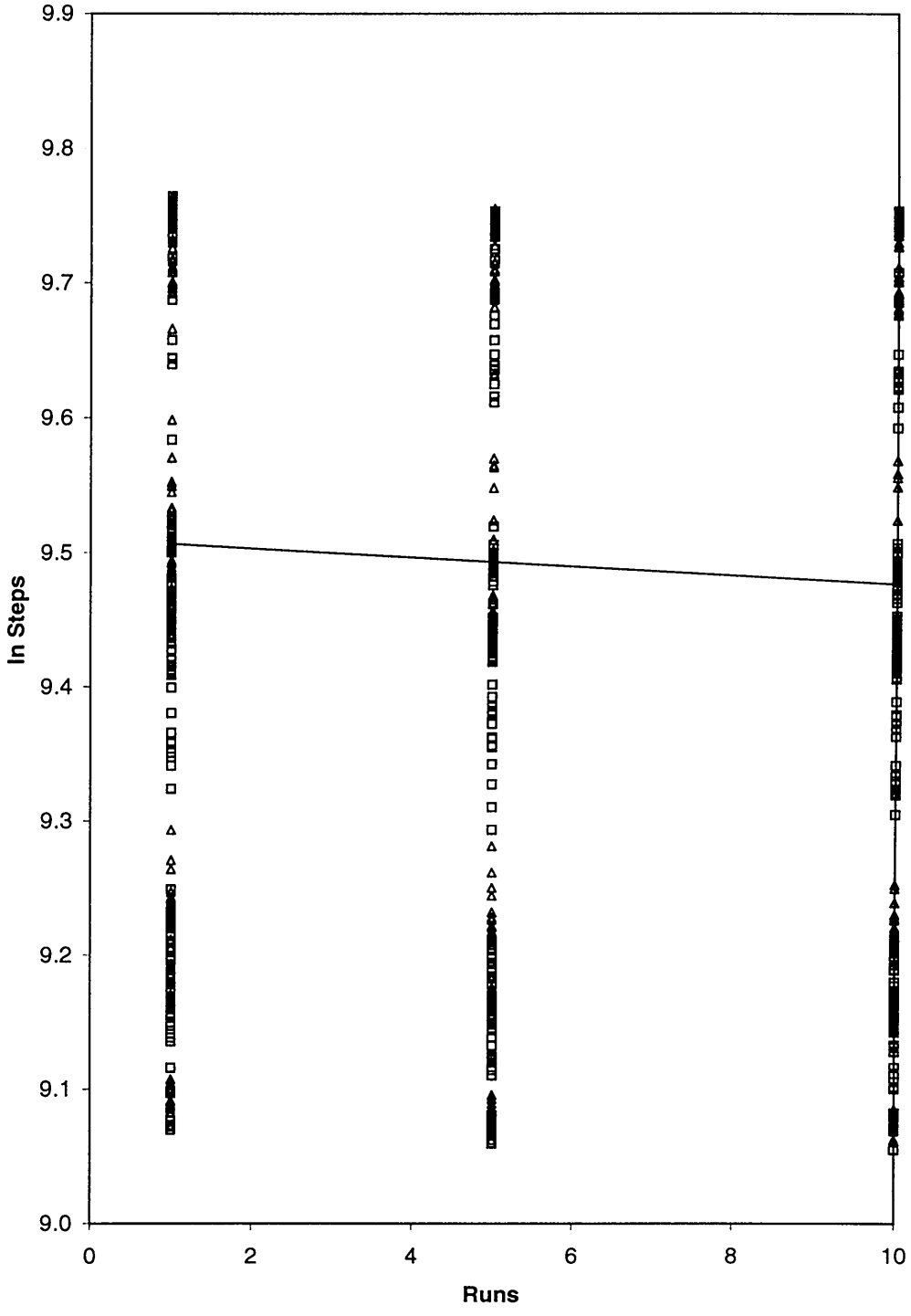


Figure 4.31 The relationship observed between ln(steps on shortest tree found) and `runs`, for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

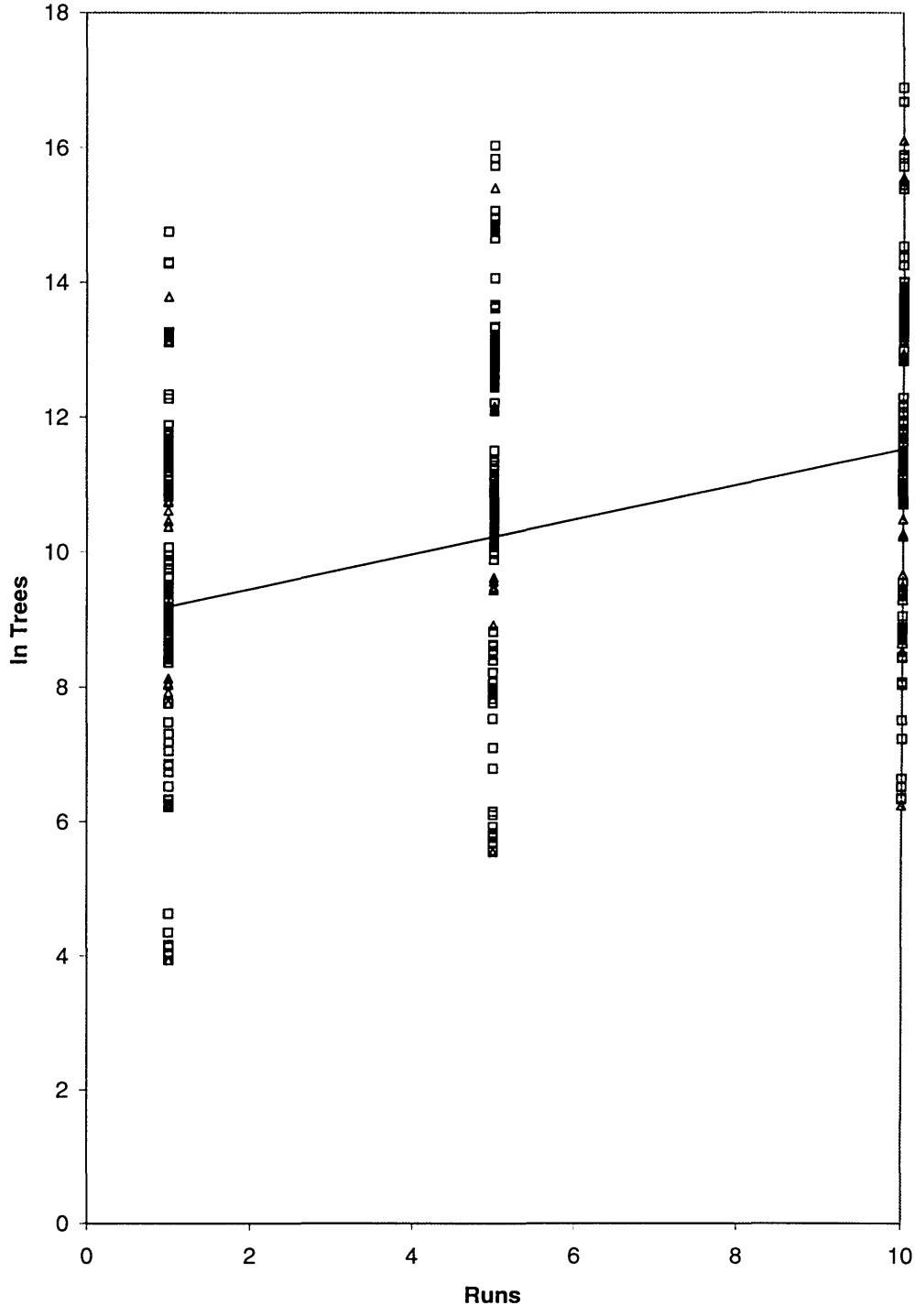


Figure 4.32 The relationship observed between ln(number of trees considered) and `runs`, for the general data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

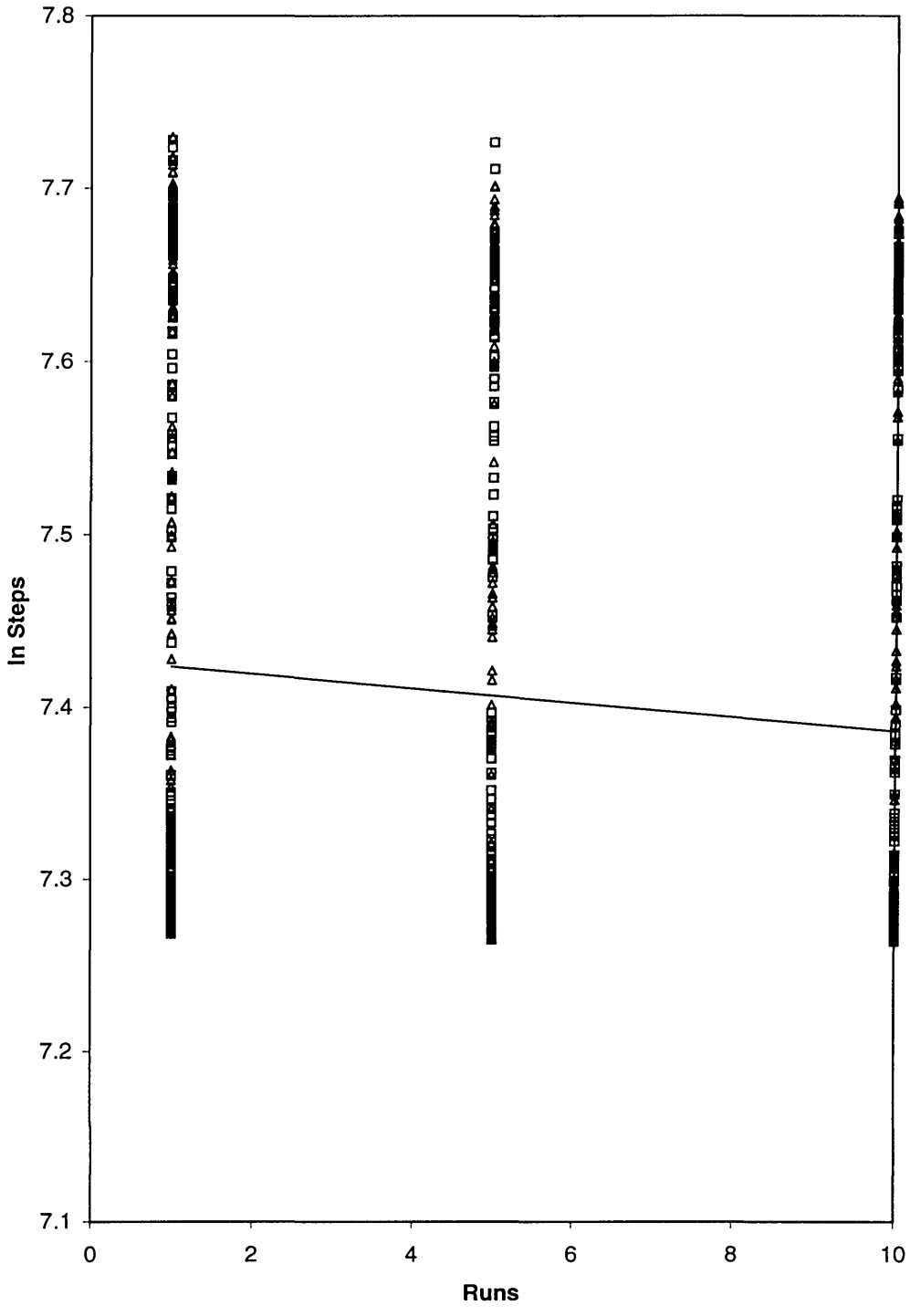


Figure 4.33 The relationship observed between ln(steps on shortest tree found) and `runs`, for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

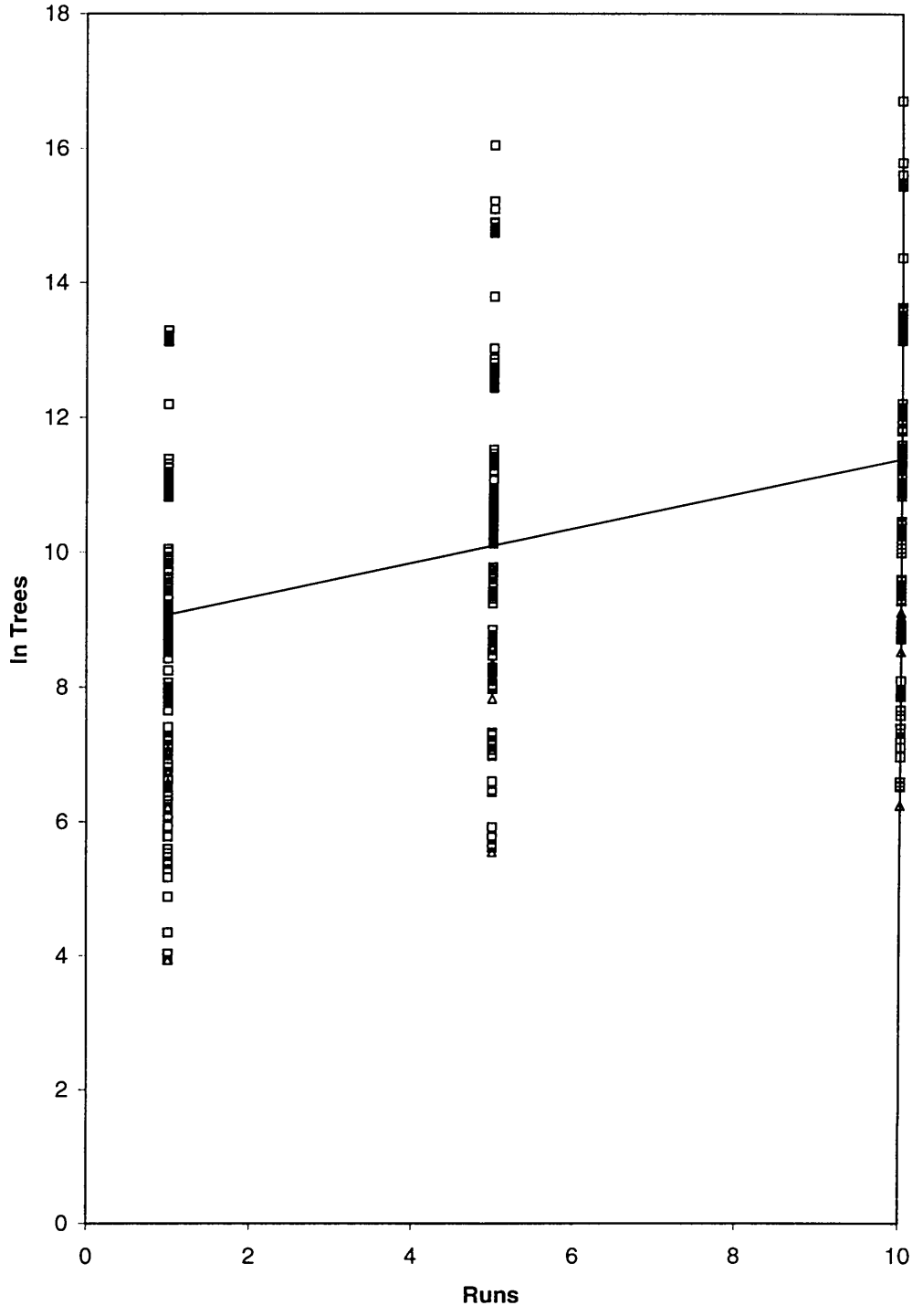


Figure 4.34 The relationship observed between ln(number of trees considered) and `runs`, for the monocot data matrix. □ `maxaccept` = 5. △ `maxaccept` = 50. Line: linear regression line for all observations together.

Chapter 5

Discussion

The fitted equations are good (Section 4.2.5), despite the violation of some assumptions of the model used to obtain them (Section 4.2.1). It would be possible to fine-tune transformations to make the assumptions more legitimate. However, this could only increase the accuracy of the fitted equation *for the levels of factors used.*

Here lies a problem. The data matrix used was necessarily a factor rather than a covariate (Section 3.4). How can results be generalised to data matrices other than the two used in this experiment? Although the data matrices used are subsets of an interesting and frequently reanalysed data matrix (e.g., [57, 58]), the conclusions of the experiment are meant to be more general.

The problem might be attacked in the following ways:

1. add one or more new levels of `matrix` to the current experiment to broaden the scope of the fitted equations;

2. use whatever data matrix is of interest, guess the nearest level of `matrix` in the current experiment, then use the current fitted equations.

LVB analyses of the monocot data matrix were complete within a few months, and it would be feasible to add one more level of `matrix`, perhaps a subset of the monocot data matrix. However, this approach cannot proceed without limit. Even if another 100 levels of `matrix` could be added, (1) would, at some point, have to be abandoned

in favour of (2) for the results to be really useful. Since the computer time invested in the experiment has already been very large, I feel the time is right for (2).

It would be easy to incorporate the fitted equations into a future version of LVB. The program could use simple properties of the data matrix to guess which level of `matrix` to use in the fitted equations, e.g., number of rows or number of cells. The problem of using simulated annealing could then be reduced to *one* question for the user, 'how long do you wish the analysis to run?' The answer, x minutes, could be used as follows:

params $\leftarrow$ values for variables not forming part of a model formula;
$Steps_{min} \leftarrow \infty$;
Trees_wanted $\leftarrow x \times$ (rate of evaluation of trees per minute);
for (each of a factorial combination of annealing parameters)
{
  if ((fit for Trees $\approx$ *Trees_wanted*) **and** (fit for Steps $<$ $Steps_{min}$))
  {
    params $\leftarrow$ *params* $\cup$ (the current combination of annealing parameters);
    $Steps_{min} \leftarrow$ fit for Steps;
  }
}

This leaves in *params* a full set of parameters that is predicted to give the best final results in approximately the requested time. For portability, the program could estimate *Trees_wanted* every time it is run. It might do this by generating a very large chain of updates to a random tree, and counting how many it could evaluate in a minute. This would take account of system type and system load (assuming load varies little throughout the analysis).

Rather than calculate fits for a factorial combination or *grid* of levels of parameters, it would be possible to optimise the annealing parameters using any of the heuristics outlined in Chapter 2. The quantity to minimise would be the fit for

Steps, given that the fit for Trees should approximately equal *Trees_wanted*. This would allow extrapolation to an extent not possible with a grid, since a grid would rapidly become unwieldy as its resolution increased.

The simple approach would be to 'hard-wire' the fitted equations into LVB and apply them in this way to chose parameters for each analysis. More ambitiously, LVB could use feedback from its own analyses to modify its decisions about which parameters to use. For example, if a user's data matrices tend to be of a kind dissimilar to both the general and the monocot data matrix, the fitted equations might suggest parameters that are far from ideal. But if there were a random or user-defined component in the selection of each parameter, and more than one analysis were performed, the effects of the parameters could be recorded and used to help select parameters in future.

Due to the unsystematic nature of this feedback data, its effective use becomes a problem in artificial intelligence rather than statistics. The Boltzmann machine [37], a self-educating, decision-making generalisation of simulated annealing, could be a useful way for LVB to learn. It would be possible for all users of LVB at a given site to share feedback information. LVB Web services (Section 6.1) could learn fast, due to the steady stream of analyses. The knowledge accumulated could then be incorporated into a future general release of LVB.

Chapter 6

Future Directions

There has been considerable interest in the program LVB, and several questions have been raised, both by the author and by others. There are many opportunities for future research on use of simulated annealing and other modern approaches to optimisation in the area of phylogeny. This chapter discusses some of these, along with some work done so far.

6.1 Web Service

Catherine Letondal of the Institut Pasteur has arranged a Web-based LVB service (Section G.4). This has a user-friendly form-based interface to help give values to the variables that control LVB. LVB input files are then created, and the analysis is run on a system at the Institut Pasteur. The results are returned to the user by e-mail or the Web.

The service is much simpler to use than LVB usually is, because the `params` file (Section 3.1.2) is created automatically through the HTML front-end. The user is not left to remember the meaning of each parameter, and then 'program' the analysis in a text editor. Also, the service includes a facility to translate matrices from popular formats, including PHYLIP format, prior to the analysis. Results trees may be post-processed, e.g., by PHYLIP programs to generate consensus trees or graphics files.

6.2 Parallelisation

Since running LVB can be extremely time consuming, it may be worth using a parallel computer [59], at least for exploratory work in which many different analyses are to be done, or for in-depth analysis of especially important data. This is a difficult task due to the inherently serial nature of simulated annealing [37].

With the aim of increasing the scope of the current experiment, I parallelised LVB to run on the Cray T3D distributed-memory system (Silicon Graphics, Inc.) at the Edinburgh Parallel Computing Centre (EPCC) (Section G.6).

6.2.1 Hardware

At the time it was used, the T3D system at EPCC had 512 processor elements, each with one 150 MHz DEC 21064 processor and 64 MB RAM, connected in a high-speed network. The theoretical upper limit on processing speed for the whole T3D was 7.7×10^{10} floating point operations per second.

The system was not time-sharing like a Unix server. Instead, it operated a batch schedule, so that each job had exclusive access to its allotted number of processes for its allotted time. Each processor had a few megabytes of operating system code in its memory, but no processor could perform file input or output directly, nor could be connected to by telnet or FTP. Rather, reading and writing of files was performed through a two-processor Cray Y/MP. Compilation, editing and job submission were performed on a 10-processor Cray J90.

The T3D at EPCC was, for a while, the fastest computer in Europe and the third fastest computer in the world. By mid-1999, though it made number 156 in the world list of most powerful computers (Section G.8), its computing ability seemed no longer very impressive, compared to the high running costs of its cooling system, and the T3D was switched off forever.

The T3D was exceptionally difficult to use effectively. Some problems included:

1. the DEC 21064 processors used had a direct-mapped cache, which meant

memory access could be very slow when working with data structures more complicated than scalars [60];

2. no secondary cache was present, despite the high processor clock speed;

3. parallelisation had to pay close attention to the layout of data across processor elements, since the speed penalty for a processor requiring data held in the memory of a processing element other than its own was great;

4. debugging tools were rudimentary (e.g., the debugger was not recommended for on-line use with programs written in C);

5. profiling tools were rudimentary (e.g., a line-by-line profile could not be generated);

6. automatic parallelisation with anything other than Cray Research Adaptive Fortran (CRAFT) or High Performance Fortran (HPF) [61, 62, 63] was not possible;

7. use of CRAFT was not usually recommended, because it is only portable among Cray systems;

8. automatic parallelisation using HPF was not always recommended, because its distribution of tasks over processors could be poor.

In all, the T3D was a profoundly unbalanced system. The hardware was so complicated and unusual that it required strong support from the programming environment to be used conveniently, but such support was absent. It required many man-months of work on a typical existing program before it was worth using on this system. This was largely due to the low technological level of the compilers, requiring most or all parallelisation to be performed 'by hand', and a lack of useful features in available debugging and profiling tools. This was compounded by the frustrating nature of the DEC 20164 processor. Very fine rearrangement of data layout

in the memory *within* each processor element was sometimes necessary to avoid data cache 'thrashing', which could slow a program down considerably. The lack of a larger, secondary memory cache for each processor element was a serious design flaw (and has been remedied in the more recent T3E).

6.2.2 Software

The parallelisation technique chosen for LVB was to use explicit message-passing with the MPI library [64]. MPI is portable but maps most naturally on to distributed memory systems such as the T3D.

MPI may be called from C, the language of LVB, and also from FORTRAN. MPI is available for a large number of systems, often free. Used carefully, it can allow the same parallel source code to be compiled to run on, for example, time-sharing servers, networks of computers connected by Ethernet, and multiprocessor desktop workstations. It is unlikely that the executable, once compiled, would run efficiently on such a range of systems without some modification of the source code. However, tuning a program that already runs would be easier for users of different systems than translating source code that used Cray-specific libraries.

One immediate advantage of MPI's portability was that a certain amount of development could be performed on the Edinburgh University Computing Service's intermediate server for computational problems, `waverley`. This was unlike the T3D in being a time-sharing system with six UltraSPARC processors (Sun Microsystems). But it proved helpful, due to its comparative ease of use and superior software development environment. The Sun debugger `dbx` for SPARC-based hardware is able to detect reads from uninitialised memory, for example. In the author's experience these are common bugs in the early stages of software development.

With MPI, each process works independently by default. Data is communicated between processes only by explicit calls to library routines. Each process starts execution with an identical copy of the program, but may distinguish itself from others by a unique integer known as its *rank*. For a program with n processes,

processes have ranks from 0 to $n-1$. The rank is used to control execution through normal programming constructs, so that different processes may execute different parts of the program. Hence this approach to parallelisation is 'multiple instruction, multiple data' (MIMD). One process, by convention the process of rank zero, is usually referred to as the *root* process. The root process is given the role of controlling all the others, which may be referred to as *slave* processes. (If, in addition to its controlling role, the root process performs work like the other processes, it is not behaving as a pure *master*. It is arguably incorrect to use 'slave' in this context. However, it simplifies discussion.) Exactly how the processes map on to the hardware and operating system (for example, whether Unix processes or some other approach is used to achieve parallel execution) is hidden from the MPI programmer. On the T3D, one processing element was used for each process.

6.2.3 Algorithm

A simple approach to spreading execution of LVB over many processors would be to distribute different searches. For example, with `runs 300` (Section 3.1.2.2), it would have been possible to perform five searches on each of 60 processors, obtaining results about 60 times more quickly than on serial hardware. Since each search would typically take at least a few hours, and processors would work independently during each search, there would be little overhead from communications between processors.

Unfortunately, this simple approach is inflexible and does not scale well to very large numbers of processors. For example, to perform 300 searches on 300 processors, one search could be performed per processor. But, due to the stochastic element in the simulated annealing search, some searches would finish before others. On the T3D, the processors used by searches finishing early would remain idle until the single most time-consuming search had completed. Also, initial work with LVB suggested that it was a better use of time to perform a small number of slow searches rather than many faster ones. But if few searches are performed, the number of processors that can be used at all is limited. n searches could not be spread over more

than n processors, since each search is executed in a serial manner.

Because of these problems, it was decided to parallelise each search, so that a single search uses many processors. Instead of generating updated trees as required, i.e.,

```
X ← initial tree;
while (the system is not 'frozen')
{
    propose an update, X';
    possibly X ← X';
    possibly decrease the temperature;
}
```

a whole chain of updates is proposed, either on the assumption that they will be accepted or on the assumption that they will be rejected. This assumption is adjusted according to whether updates have tended to be accepted or rejected recently. For example, at the start of a search, temperature will usually be high, in which case updates are mainly accepted.

After it has been created in serial by the root process, the chain of speculative updates is distributed among all processors for evaluation. The lengths of the trees in the chain are then gathered by the root process, and the trees and associated lengths form a buffer from which pre-evaluated trees may be drawn. As soon as the assumption (acceptance or rejection) upon which updates were proposed is violated, unused trees in this buffer must be discarded, and a new chain proposed and evaluated. Because calculation of tree length is more time-consuming than the process of generating updates and controlling the annealing search, the approach has the potential to be efficient. The main drawback is that some processor time is wasted every time it happens that the assumption on which a chain of updates was created proves wrong. This is unlikely to be significant at the very start of a search, where most updates are accepted, or towards the end of the search, when almost all updates

are rejected. For a time between these stages, waste can be considerable.

The amount of waste also depends on the length of the chain. If the chain is of length 10 updates, no more than nine in any chain will ever be wasted. If the chain is of length 1 000 updates, up to 999 updates in a chain could be wasted. However, a longer chain allows more trees to be evaluated by each processor between communication events, reducing the number of communication events required for a given search. On the other hand, the size of each communication is increased. A longer chain also allow more processors to be used without decreasing the number of updates per processor. The interplay of all these factors is complex and difficult to analyse theoretically. It is most easily investigated by test jobs using different combinations of levels of trees per processor and total number of processors.

The following pseudocode outlines the algorithm. First, some initialisations:

```
rank ← process rank;
procs ← total number of processes;
chain_len ← number of updates per chain;
per_proc ← chain_len/procs;      /* updates per processor */
next ← chain_len;                /* next unused update in chain */
X ← initial tree;
```

Then, excluding declarations,

```
if (rank == 0)                          /* root */
{
   while (the system is not 'frozen')
   {
      if (next > chain_len − 1)         /* must refill buffer */
      {
         generate a chain of updates, X'_0, ..., X'_{chain_len−1};
         for (i ← 1, ..., procs − 1)
         {
            lower ← rank × per_proc;
            upper ← lower + per_proc − 1;
            send X'_lower, ..., X'_upper to process i;
         } /* for i */
         evaluate_as(rank, per_proc);
         for (i ← 1, ..., procs − 1)
         {
            lower ← i × per_proc;
            upper ← lower + per_proc − 1;
            receive length_lower, ..., length_upper from process i;
         } /* for i */
         next ← 0;
      } /* if next */
      possibly X ← X'_next;
      if (the assumption used to create the updates was wrong)
         next ← chain_len;              /* clear buffer */
      else
         next ← next + 1;
   } /* while the system */
} /* if rank */
else                                    /* slave process */
   evaluate_as(rank, per_proc);
```

The subroutine *evaluate_as*() takes, as a parameter, the rank of the calling process and the number of updates per processor. If the rank is *root*, *evaluate_as*() evaluates the allotted trees and returns. Otherwise, it waits for trees to arrive, evaluates them and communicates the lengths to the root process in an infinite loop:

```
evaluate_as(rank, per_proc)
{
    lower ← rank × per_process;
    upper ← lower + per_process − 1;
    do
    {
        for (i ← lower, ..., upper)
            length_i ← length of X'_i;
        for (i ← lower, ..., upper)
        {
            if (rank ≠ 0)
                send length_lower, ..., length_upper to root process;
        } /* for i */
    } while (rank ≠ 0);
}
```

The use of **do** ... **while** is something of an abuse, since the value of *rank* is constant throughout the function. However, it is often impossible to parallelise with MPI without some damage to the structure of the source code.

In practice, the approach is more complicated than illustrated above. For example, it is necessary to terminate the slave processes at the end of execution. This may be achieved by sending a special 'terminate now' message along with, or in place of, the usual chain of trees. Upon detecting the message, the slaves would then return from *evaluate_as*() and exit.

6.2.4 Results

It was found that parallel runs would deviate in progress from serial runs with the same data and parameters. This is because random numbers are required both to generate each update and to decide whether to accept an update that is worse than the current tree. The order in which these operations occur is different in the serial and parallel versions (Section 6.2.3), so the random numbers used at each stage tend to differ. The parallel code could not be made to emulate the serial code by requesting many random numbers at once, then drawing from this buffer out of order, because

1. the number of random numbers used to generate an update, in LVB, is indeterminate;

2. the number of speculative updates to be discarded could not be known in advance.

While (1) may be fixed by a small change to the code of LVB's `mutate()` function (Appendix D), there is no fix to (2) other than to make the *serial* version of LVB emulate the parallel version by also generating a series of speculative updates. This would have the undesirable effect of complicating the serial code, almost on a par with the parallel code, and also making the serial code slightly slower. It is easier to accept that the results of a serial and parallel run may vary. Such variation will be no more than if the same analysis was run twice, in serial, but with different seeds for the random number generator (Section 3.1.2.8). However, it does make debugging and testing the parallel code very much more difficult.

In spite of this, LVB was parallelised successfully, and performed several dozen times faster than the serial version when run on several dozen processors. However, almost as soon as this had been achieved, the author thought of a serial optimisation to the code that calculates tree length (Section 6.8). Other optimisations followed. Overall, these allowed tree lengths to be calculated much faster, and altered the balance of calculation and communication in the parallel program. The parallel

program now spent too much time communicating compared to calculating, and was no longer worth using.

The parallel code could, almost certainly, have been made usable again by a further round of optimisation to communications. In particular, the size of LVB's tree data structure, each node of which is of type `Branch` defined in source file `lvb.h` (Appendix D), could be reduced by removing redundant data. This would reduce the size of messages in the parallel version. However, because parallelisation to this point had already proved very time consuming, work on the parallel code was abandoned before carrying out such optimisation.

6.2.5 Future Work

John Parnell of Trinity College Dublin is interested in creating a post-doctoral position to continue work on the parallel version of LVB, perhaps using monocotyledon sequences as a test case. The author is keen to help by supplying existing code, documentation, and general advice. In this way, the parallelisation of LVB could be made successful without duplicated effort. The code would doubtless also benefit from examination by a different programmer, to reduce the incidence of bugs or missed opportunities for optimisation.

Another form of parallelism would be to unite remote computers over the Internet (cf. the Search for Extraterrestrial Intelligence at Home, Section G.9). Although this would be difficult in practice for a single analysis, due to the slow speed of communication over this network, it would be easy to collate statistics on many *different* analyses. Different researchers interested in LVB might perform a few analyses of the data matrices of interest for their own current research, and collation of key details of these analyses would not be difficult.

Unfortunately, the current Web-based LVB service (Section 6.1) cannot provide any such information, since only the command-line for each analysis is logged.

6.3 Other Optimality Criteria

LVB only allows trees to be evaluated using parsimony. It would be easy to extend the program to allow trees to be evaluated using, for example, minimum evolution [65] (cf. [10]) or one of the several maximum likelihood (ML) criteria [66]. Like parsimony, both minimum evolution and ML are properties of the whole tree and ideally require every possible tree to be considered. Because of this, they also lend themselves to a simulated annealing search. ML in particular requires a very time consuming calculation to evaluate each tree.

6.4 Initial Tree

LVB always uses a tree of random arrangement as the initial tree for any search. However, in response to a request from Susumu Nakayama of Nagasaki University, the author produced a modified version of LVB that is able to read in the initial tree from file. The rather cumbersome name of this program is 'LVB version 1.0A 18 August 1997, with Extension 1 written by Daniel Barker, May 1998'. It is available for download from the LVB Web page (Section G.1).

Nakayama used 40 protein sequences. These were hypothetical ancestral sequences for EF-hand domains, inferred using PAML [67], for four domains of each of 10 EF-hand protein subfamilies. EF-hand proteins are calcium-binding proteins [68].

Nakayama constructed an initial tree by neighbour-joining [69] using the `njdist` program of the Molphy package (Section G.3). Neighbour-joining is a quick heuristic search for the minimum evolution tree (Section 6.3). It begins with a star-like tree, consisting of one hypothetical ancestor connected directly to each of the extant objects. A fully resolved tree is developed by recursive elaboration of this structure. Although the construction of the tree is divisive rather than additive in the manner of the greedy algorithm described in Section 2.1, neighbour-joining is still greedy in

spirit: at any point, the solution that gives the immediate best result is chosen. The effect of previous operations may not be reversed in the light of later progress.

The resulting tree, of length 473 changes (evaluated by parsimony), was used as the initial tree in a simulated annealing search with the extended version of LVB. This found 24 trees of length 451. Although the difference was not statistically significant, the consensus of these trees was found, using the Molphy program `protml`, to have a higher likelihood than the tree input to LVB, i.e., the tree found by neighbour-joining. This is no surprise, since the final trees were more parsimonious, and maximum likelihood and parsimony tend to suggest trees that are similar, at least in their broad outline.

Nakayama's work is interesting at various levels. Firstly, it uses neighbour-joining as a quick heuristic to seek a parsimonious tree. This is an unusual use of neighbour-joining, but a reasonable one given that

1. minimum evolution and parsimony are closely related;
2. neighbour-joining is fast;
3. the tree produced was not presented as the final, 'most parsimonious' tree found.

Then, in what may be the first analysis to combine neighbour-joining with simulated annealing and parsimony, this tree was refined by a simulated annealing search for the most parsimonious tree.

Finally, using the maximum likelihood program `protml` from the Molphy package, the tree before simulated annealing was compared to the consensus of trees after. Thus, the whole search for parsimonious trees may be thought of as a heuristic search for the tree of maximum likelihood. Alternatively, this test may be regarded as validation of the results of LVB, since LVB's simple form of parsimony may not be ideally suited for use with protein sequences. Every change is regarded as equal, even though some amino acids may only change from one to the other by more than one

nucleotide mutation. See the documentation of the PHYLIP program `protpars` (Section G.2) for discussion of this point.

The results obtained by Nakayama and co-workers are intended for publication in *Protein Profile*.

6.5 Cooling Cycle

The nature of the cooling cycle used in LVB, particularly the exponential relationship between successive temperatures (Section 2.4), may not be ideal. Temperature falls quite quickly to a very low level, at which point the search is close to hill climbing. This would be less noticeable if the 'energy' of the tree were not so 'quantised'. According to LVB, which does not allow characters to be weighted to increase or decrease their contribution to tree length, tree length is necessarily an integer. When converted to homoplasy index, or 'energy' as I have chosen to define it, there still remain gaps between one possible energy level and the next. At low temperature, moving from one configuration to another may have a very low probability, simply because these gaps are large compared to the temperature.

Cooling schedules have not been investigated in this thesis. However, the author is keen that research proceed in this area. It would be worth incorporating theoretical findings published over the last few years, e.g., [37]. Much of the literature is highly mathematical. Research in this direction would best be performed by, or in collaboration with, someone with a more mathematical background.

6.5.1 Future Work

It would be useful to adapt LVB so it may take an arbitrary expression for the calculation of each successive temperature level, rather than just Equation 2.15. If the expression were to be read at run-time, this would require lexical analysis and parsing of an input expression. Perhaps some of the code to perform this could be taken from

an existing application, e.g., the calculator program gnubc from the Free Software Foundation (Section G.7). Alternatively, coding could be largely automated using a lexical analyser generator (e.g., lex) and compiler-compiler (e.g., yacc). The development of the 'higher order calculator' hoc, described in [70], provides an example of using these tools.

Martin Dietze of Fachhochschule Wedel is interested in arranging diploma projects working with LVB, supervised with assistance from the author. Adapting LVB to allow an arbitrary relation between successive temperatures, and investigating possible relations, might be a suitable project.

6.6 Other Heuristics

6.6.1 Comparison of Simulated Annealing with Other Heuristics

This thesis has only investigated simulated annealing. Unfortunately, due to the time it would take, a comparison with other heuristics has not been attempted. A comparison of simulated annealing with the more usual heuristics in phylogeny, greedy algorithm and hill climbing, would be of interest to many researchers. Perhaps when the next version of PAUP [30] is released, that program could be used for such an investigation.

6.6.2 Genetic Algorithms

Genetic Algorithms were outlined in Section 2.5.

The author initially believed genetic algorithms to be a fruitless approach when we seek to optimise a tree data structure. The problem is that a crossover operator would almost always damage the data structures, for example, by giving a tree ('chromosome') that did not represent each extant object precisely once.

One way to avoid this problem, which only dawned on the author towards the end of the research for this thesis, is to chose the position of crossover considerably *before*

crossover occurs. After making this choice, there should then be many generations *before* the crossover, in which change only occurs by mutation and selection. Mutation could be of the sort used by LVB (Figure 2.6), with the constraint that any mutation event occurs entirely to one side or the other of the future crossover point. In other words, on each side of the proposed crossover, care would be taken that no mutation involved any part of the tree on the other side. Then, when the time came to cross over, *any* tree could crossover with another, and a viable data structure would result. A new crossover position could be selected for the next round of crossover, and the procedure repeated.

This conceptual leap has caused some excitement to the author, who, in the medium term, plans to review literature on optimisation with constraints (e.g., optimisation of tree data structures). It would be interesting to compare genetic algorithms with simulated annealing, as well as investigate mixed approaches.

6.6.3 Threshold Accepting

Threshold accepting [71] is based on simulated annealing, but simpler, since it does not uses probability to determine whether to accept an update or not. For a minimisation problem, any update to the current solution is accepted if it causes a change in energy less than the current temperature.

Despite using the same 'temperature' control parameter as simulated annealing, threshold accepting lacks a clear physical analogy. Its authors' experiments suggest it gives good results in comparatively short runs.

6.6.4 Branch-and-Bound Revisited

Branch-and-bound finds the globally optimal solution, but is impractically slow for large problems (Chapter 2). However, a heuristic based on branch-and-bound is possible: some areas of the search space could be ignored at random, in addition to those ignored because they are known to give a worse solution than the best obtained

so far.

Whether such a heuristic would have any advantages over other heuristics is unclear.

6.7 Bug Fixes

All the experiments in this thesis were performed with the same source code, i.e., LVB 1.0A 18 August 1997. Several bugs have revealed themselves during this time. For the most serious, see Table 3.1.

The author plans to fix all known bugs, and further check the source code by both manual inspection and static analysis, e.g., using the LCLint automatic code checker [72]. More disciplined test coverage of the source code, ideally of every line (cf. [73]), should be performed. Fixes of the known bugs, and any others revealed, will lead to a new minor release of LVB, expected to work smoothly on all forthcoming general purpose computer systems with either simple or optimising compilation (within the bounds of ANSI C).

LCLint not only checks C code, but allows the precision of the language to be extended by means of structured comments. A later version of LVB may be significantly rewritten, in a less 'ad-hoc' manner than the current code, and such comments might be added to ensure an additional degree of safety. The author has begun drawing up a coding standard, specifying which parts of the C language are and are not acceptable for use in the current program [74]. An alternative would be to re-write LVB in a higher-level, less 'dangerous' language, such as Fortran 95 [75] or Ada 95 [76]. However, though portable, neither of these languages is as widely available as C, which perhaps makes C a better choice, since the program and its components are intended to be widely used on whatever systems people have to hand (Section 6.9).

6.8 Code Optimisation

LVB 1.0A was finalised before the author thought of the optimisations mentioned in Section 6.2.4. Partly these optimisations relate to the layout of data, and partly they relate to 'jamming' more operations on small integers into fewer operations on large integers. With all such optimisations implemented, LVB was found to work an order of magnitude faster. The exact speed increase depended on hardware, and was largest for RISC and post-RISC systems. However, particularly the 'jamming' code was very obscure. The optimised version of LVB has not been used for the experiments in this thesis. It is too complicated to understand, and though it seems to work in practice, 'seeming' to work is not good enough.

Some less ferocious optimisation might be worthwhile in the medium term, perhaps coupled with a general 'cleaning up' of the design of LVB in an attempt to make the code itself more useful to others (Section 6.9). The language Fortran 95 (Section 6.7) theoretically allows a compiler to generate a faster executable than C in some circumstances. However, due to low up-take of the language by programmers, and the difficulty of implementing its compiler at all, approximately the same performance may be expected from carefully written C. Not only are compilers for C widely available, but they are often of good quality, within the bounds of the simpler language.

More important than the language are issues such as data layout, which may be optimised in any language, without harm to program structure. LVB is perhaps at a stage where optimisation of data layout should be more of a priority than any dramatic change.

6.9 A Portable Phylogeny Library?

In the LVB manual [31] (Appendix D), all functions and variables with external linkage were documented. The idea was to make it easier for others to understand and

re-use the LVB source code. In fact, though the author has found this documentation useful, it has not been widely used by users. One problem may be the poor modularity of LVB's sub-programs. It would not be easy, for example, to turn important parts of the program into re-usable libraries of object code.

The author is keen to create a genuinely useful library of phylogenetic sub-programs as a 'spin-off' from LVB. This could save untold effort for future workers in the area. In some ways, the current LVB code is an early prototype for such a library. A later version would benefit from:

1. a larger number of 'pure' sub-programs, i.e., functions that have no 'side-effects' and so are functions in the mathematical sense, with no use of external state or anything other than automatic variables;

2. less dependence on external variables;

3. a consistent naming scheme for LVB sub-programs.

(1) and (2) are related, in that less dependence on external variables will likely increase the 'purity' of sub-programs. They are both important if the code is to be re-used in a convenient manner. (1) may also increase opportunities for automatic parallelisation by a compiler that can generate multi-threaded object code [77]. (3) is important to prevent clashes of names with other sub-programs.

Appendix A

*rbc*L Sequences Used

Here I give the names of *rbc*L sequences used in the experiments in this thesis. The names are in the abbreviated form used in the 500-sequence data matrix supplied by Mark Chase, with the exception that the ASCII `Hypoestes` has been used in place of `Hypoëstes` and the inverted commas around `Canella-A` and `Canella-B` have been removed.

Sequence names for each data matrix are given in the order they appear `data` files used in my experiments. This should allow my results to be repeated precisely. Use of the sequences in a different order would not give identical results, due to the random effect on assigning extant objects to leaves in the initial tree. Use of a different value of `seed` would also cause an analysis to differ randomly from my own. (The values of `seed` used for my experiments are given in Appendix B.)

A.1 General Data Matrix

`Nicotiana`, `Galphimia`, `Oenothera`, `Petunia`, `Lycopersi`, `Hydrophyl`, `Eriodicty`, `Digitalis`, `Jasminum`, `Adoxa`, `Pinguicul`, `Pterost`, `Vahlia`, `Salvia`, `Scutellar`, `Valeriana`, `Dipsacus`, `Lobelia`, `Cornuskou`, `Hedera`, `Viburnum`, `Pentas`, `Nyssa`, `Cornuscan`, `Cornuswal`, `Helwingia`, `Davidia`, `Alangium`, `Camptoth`, `Boykinia`, `Tetracarp`, `Gunnera`, `Phyllonom`, `Griselin`, `Corokia`, `Justicodo`, `Aphelandr`,

Thunbergi, Nelsonia, Barleria, Ruttya, Proboscid, Lepidagat, Ruellia, Acanthus, Harpogoph, Hypoestes, Montinia, Ilexcrena, Eucommia, Berzelia, Campanula, Scaevola, Dimorphot, Tagetes, Felicia, Chromolae, Achillea, Gerbera, Cacosmia, Vernonia, Tragopogo, Eupatoriu, Piptocarp, Cichorium, Senecio, Carthamnu, Brasenia, Ceratophy, Canella-A, Canella-B, Magnsalic, Michelia, Manglieti, Magnhypol, Liriodtul, Liriodchi, Eupomatia, Degeneria, Amborella, Knema, Talauma, Talasinga, Drimys, Belliolum, Tasmannia, Cinnamomu, Persea, Hernandia, Idiosperm, Calycchin, Chimonant, Hedycarya, Annona, Gyrocarpu, Asimina, Galbulimi, Asarum, Saururus, Schisandr, Illicium, Austrobai, Piper, Peperomia, Aristoloc, Saruma, Hedyosmum, Chloranth, Sabia, Akebia, Euptelea, Mahonia, Caltha, Xanthorhi, Sanguinar, Cocculus, Trochoden, Sargentod, Coriaria, Cercidiph, Tetracent, Platanus, Daphniphy, Altingia, Rhodoleia, Hamamelis, Celtis, Myrica, Betula, Carya, Casuarina, Chrysolep, Trigonoba, NothofBal, NothoDom, Humulus, Boehmeria, Trema, Photinia, Geum, Prunus, Clavija, Anagallis, Symplocus, Styrax, Manilkara, Chrysophy, Camellia, Impatiens, Diospyros, Ardisia, Cassiope, Chamaedap, Pentachon, Daboecia, Pyrola, Befaria, Arbutus, Epacris, Clethra, Ceratiola, Actinidia, Erica, Rhododend, Cyrilla, Enkianthu, Vaccinium, Sarraceni, Diapensia, Dillenia, Avena, Triticum, Cenchrus, Neurachne, Oryza, Aegilops, Pandanus, Tradesca, Juncus, Oxychloe, Prionium, Elegia, Flagellar, Carex, Xanthorrh, Strelitzi, Stegolepi, Tillandsi, Ananas, Puya, Drymophlo, Nypa, Serenoa, Typha, Anigozant, Pleea, Bowiea, Curculigo, Hypoxis, Aletris, Kniphofia, Danae, Neuwiedia, Oncidium, Dioscorea, Spathiphy, Gymnostac, Tacca, Smilax, Calochort, Colchicum, Burchardi, Lilium, Alstroeme, Anomathec, Chamaelir, Vellozia, Freycinet, Sphaerade, Dianthus, Basella, Mollugo, Phytolacc, Trianthem, Portulaca, Mirabilus, Rivina, Alluaudia, Stegonosp, Atriplex, Amaranthu, Spinacia, Plumbago, Rheum, Nepenthes, D.filifor, D.spathul, Dionaea, Trigonia, Pisum, Medicago, Cephalot, Erythroxy, Humiria, Ochna, Paeonia, Rhamnus, Krameria,

Qualea, Securidac, Polygala, Platythec, Leitneria, Bursera, Acer, Schinus, Cupaniops, Ailanthus, Vitis, Euphorbia, Drypetes, Euonymus, Decumaria, Poncirus, Guaicum, Passiflor, Acridocar, Thryallis, Mascagnia, Dicella, Byrsonima, Spirea, Tetramele, Dcannabin, Luffa, Cucurbita, Cucumis, Lambertia, Bauera, Escalloni, Ceratopet, Pittospor, Lepuropet, Crossosom, Ribes, Myriophyl, Greyia, Francoa, Itea, Carpenter, Brexia, Parnassia, Penthorum, Heuchera, Garrya, Deutzia, Phoradend, Hydrangm, Osyris, Schoepfia, Saxifrag, Philadel, Roridula, Bombax, Tilia, Theobroma, Gossrobin, Thespesia, Shoreazel, Shoreasti, Quisquali, Lythrum, Mouriri, Heteropyx, Terminali, Osbeckia, Ludwigia2, Trapa, Punica, Clarkia, Epilobium, Hauya, Circaea, Lopezia2, Viviania, Geranium, Monsonia, Wendtiagr, Hypseocha, Oxalisdil, Moringa, Bretschne, Stanleya, Cleome, Capparis, Tropaeolu, Reseda, Tovaria, Carica, Akania, Brassica, Limnanthe, Floerkea, Batis, Cycas, Bowenia, Zamia, Chigua, Dioon, Stangeria, Encephala, Abies, Pseudolar, Pseudotsu, Sequoiade, Tsuga, Podocarpu, Keteleeri, Cedrus, Larix, Picea_pun, Pinus_gri, Pinus_rad, Ephedra, Welwitsch.

A.2 Monocot Data Matrix

Brasenia, Avena, Triticum, Cenchrus, Neurachne, Oryza, Aegilops, Pandanus, Tradesca, Juncus, Oxychloe, Prionium, Elegia, Flagellar, Carex, Xanthorrh, Strelitzi, Stegolepi, Tillandsi, Ananas, Puya, Drymophlo, Nypa, Serenoa, Typha, Anigozant, Pleea, Bowiea, Curculigo, Hypoxis, Aletris, Kniphofia, Danae, Neuwiedia, Oncidium, Dioscorea, Spathiphy, Gymnostac, Tacca, Smilax, Calochort, Colchicum, Burchardi, Lilium, Alstroeme, Anomathec, Chamaelir, Vellozia, Freycinet, Sphaerade.

Appendix B

Results

The results analysed in Chapter 4 are given here. They were extracted from `statx` files produced by LVB. The column Steps gives the number of steps on the shortest tree found, and Trees gives the total number of trees considered in the analysis. Other columns give values of LVB parameters. '`maxpr.`' is used here as an abbreviation for `maxpropose`.

B.1 Monocot Data Matrix, `maxaccept` = 5

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	50	1	1	141714483	1805	375
1e-05	9.9e-08	50	1	5	605323366	1725	1513
1e-05	9.9e-08	50	1	10	53186409	1775	2614
1e-05	9.9e-08	50	10	1	470996512	1622	1415
1e-05	9.9e-08	50	10	5	598143083	1631	6475
1e-05	9.9e-08	50	10	10	345154030	1617	10638
1e-05	9.9e-08	50	100	1	99312440	1528	5894
1e-05	9.9e-08	50	100	5	745774450	1469	27986
1e-05	9.9e-08	50	100	10	144161093	1452	58197
1e-05	9.9e-08	500	1	1	300182203	1494	2351
1e-05	9.9e-08	500	1	5	613873719	1483	16111
1e-05	9.9e-08	500	1	10	410702533	1500	32740

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	500	10	1	539107554	1456	9223
1e-05	9.9e-08	500	10	5	592679672	1439	43186
1e-05	9.9e-08	500	10	10	172777608	1442	86656
1e-05	9.9e-08	500	100	1	592689630	1435	53939
1e-05	9.9e-08	500	100	5	270574397	1438	282067
1e-05	9.9e-08	500	100	10	606629267	1439	549096
1e-05	9.9e-08	5000	1	1	617278484	1519	18411
1e-05	9.9e-08	5000	1	5	704012541	1435	89821
1e-05	9.9e-08	5000	1	10	545036592	1439	155015
1e-05	9.9e-08	5000	10	1	64923826	1449	59207
1e-05	9.9e-08	5000	10	5	490368523	1438	308477
1e-05	9.9e-08	5000	10	10	267583710	1436	635434
1e-05	9.9e-08	5000	100	1	613512638	1438	514332
1e-05	9.9e-08	5000	100	5	630985772	1447	2707560
1e-05	9.9e-08	5000	100	10	483877899	1443	5175462
1e-05	9.9e-07	50	1	1	231399033	1959	220
1e-05	9.9e-07	50	1	5	44713726	1794	1463
1e-05	9.9e-07	50	1	10	122988743	1754	2570
1e-05	9.9e-07	50	10	1	503277253	1645	1082
1e-05	9.9e-07	50	10	5	491024223	1605	5164
1e-05	9.9e-07	50	10	10	680870975	1528	13329
1e-05	9.9e-07	50	100	1	12847188	1529	5617
1e-05	9.9e-07	50	100	5	531523514	1458	28927
1e-05	9.9e-07	50	100	10	685803157	1447	57237
1e-05	9.9e-07	500	1	1	520015665	1572	2398
1e-05	9.9e-07	500	1	5	661139380	1536	13869
1e-05	9.9e-07	500	1	10	578398642	1485	29603
1e-05	9.9e-07	500	10	1	31241017	1529	9088
1e-05	9.9e-07	500	10	5	387046417	1459	46001
1e-05	9.9e-07	500	10	10	226579872	1438	89678
1e-05	9.9e-07	500	100	1	173175487	1501	63400
1e-05	9.9e-07	500	100	5	733799036	1441	276539
1e-05	9.9e-07	500	100	10	47297081	1443	548138
1e-05	9.9e-07	5000	1	1	92706189	1452	20313
1e-05	9.9e-07	5000	1	5	648963138	1445	100032
1e-05	9.9e-07	5000	1	10	15784474	1440	164854
1e-05	9.9e-07	5000	10	1	186683	1516	57693

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-07	5000	10	5	486018570	1435	296632
1e-05	9.9e-07	5000	10	10	484381071	1439	646901
1e-05	9.9e-07	5000	100	1	221653137	1515	519615
1e-05	9.9e-07	5000	100	5	751750239	1438	2554782
1e-05	9.9e-07	5000	100	10	75546497	1439	5119550
1e-05	9.9e-06	50	1	1	375046213	1870	321
1e-05	9.9e-06	50	1	5	639542343	1782	1517
1e-05	9.9e-06	50	1	10	149173505	1746	3247
1e-05	9.9e-06	50	10	1	37686253	1635	1272
1e-05	9.9e-06	50	10	5	409114201	1588	6399
1e-05	9.9e-06	50	10	10	5963627	1619	11938
1e-05	9.9e-06	50	100	1	724421585	1516	5968
1e-05	9.9e-06	50	100	5	606372157	1466	28926
1e-05	9.9e-06	50	100	10	560556073	1437	60262
1e-05	9.9e-06	500	1	1	127524190	1495	3810
1e-05	9.9e-06	500	1	5	601626136	1505	17506
1e-05	9.9e-06	500	1	10	654033680	1498	26204
1e-05	9.9e-06	500	10	1	96254355	1499	14035
1e-05	9.9e-06	500	10	5	90738987	1464	40843
1e-05	9.9e-06	500	10	10	161392564	1446	94334
1e-05	9.9e-06	500	100	1	267509834	1450	52352
1e-05	9.9e-06	500	100	5	641274660	1437	270976
1e-05	9.9e-06	500	100	10	589810731	1441	539888
1e-05	9.9e-06	5000	1	1	518210978	1448	20511
1e-05	9.9e-06	5000	1	5	263047769	1445	81768
1e-05	9.9e-06	5000	1	10	510314686	1443	167335
1e-05	9.9e-06	5000	10	1	302720389	1441	76817
1e-05	9.9e-06	5000	10	5	682739610	1433	294797
1e-05	9.9e-06	5000	10	10	285680165	1439	661648
1e-05	9.9e-06	5000	100	1	561702209	1495	525883
1e-05	9.9e-06	5000	100	5	273410815	1445	2583219
1e-05	9.9e-06	5000	100	10	591263236	1444	5130507
0.0001	9.9e-07	50	1	1	498932895	1894	441
0.0001	9.9e-07	50	1	5	9480818	1764	1519
0.0001	9.9e-07	50	1	10	615281119	1766	2806
0.0001	9.9e-07	50	10	1	673503246	1758	752
0.0001	9.9e-07	50	10	5	723781693	1600	5790

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-07	50	10	10	348336999	1534	11506
0.0001	9.9e-07	50	100	1	474250924	1466	5362
0.0001	9.9e-07	50	100	5	168130028	1456	29327
0.0001	9.9e-07	50	100	10	146376364	1466	59369
0.0001	9.9e-07	500	1	1	66324225	1516	2990
0.0001	9.9e-07	500	1	5	684422731	1493	12378
0.0001	9.9e-07	500	1	10	243173340	1478	29511
0.0001	9.9e-07	500	10	1	426379903	1474	6501
0.0001	9.9e-07	500	10	5	81534126	1450	44417
0.0001	9.9e-07	500	10	10	155594340	1447	95059
0.0001	9.9e-07	500	100	1	573404914	1445	55416
0.0001	9.9e-07	500	100	5	705165457	1453	276489
0.0001	9.9e-07	500	100	10	568432146	1437	551715
0.0001	9.9e-07	5000	1	1	21302459	1456	21989
0.0001	9.9e-07	5000	1	5	708533407	1440	84170
0.0001	9.9e-07	5000	1	10	500180403	1443	165318
0.0001	9.9e-07	5000	10	1	208573258	1444	64987
0.0001	9.9e-07	5000	10	5	354033946	1441	323505
0.0001	9.9e-07	5000	10	10	680301001	1444	626133
0.0001	9.9e-07	5000	100	1	441475111	1443	508985
0.0001	9.9e-07	5000	100	5	686187133	1445	2551029
0.0001	9.9e-07	5000	100	10	245632573	1438	5130834
0.0001	9.9e-06	50	1	1	382358391	1916	251
0.0001	9.9e-06	50	1	5	136337546	1803	1282
0.0001	9.9e-06	50	1	10	528916851	1740	2641
0.0001	9.9e-06	50	10	1	697721753	1737	764
0.0001	9.9e-06	50	10	5	533217389	1631	4767
0.0001	9.9e-06	50	10	10	748908568	1610	10688
0.0001	9.9e-06	50	100	1	39329161	1550	5643
0.0001	9.9e-06	50	100	5	724962844	1453	29184
0.0001	9.9e-06	50	100	10	649021137	1461	58620
0.0001	9.9e-06	500	1	1	670756044	1557	2752
0.0001	9.9e-06	500	1	5	108758857	1542	15549
0.0001	9.9e-06	500	1	10	110730860	1489	30042
0.0001	9.9e-06	500	10	1	645455326	1443	13440
0.0001	9.9e-06	500	10	5	217030324	1450	52443
0.0001	9.9e-06	500	10	10	548671660	1433	97653

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-06	500	100	1	360156358	1439	52564
0.0001	9.9e-06	500	100	5	545081721	1435	273133
0.0001	9.9e-06	500	100	10	508134352	1440	541458
0.0001	9.9e-06	5000	1	1	287659167	1440	16811
0.0001	9.9e-06	5000	1	5	411867120	1441	72103
0.0001	9.9e-06	5000	1	10	677129962	1434	196756
0.0001	9.9e-06	5000	10	1	183704713	1445	64175
0.0001	9.9e-06	5000	10	5	186915474	1442	382124
0.0001	9.9e-06	5000	10	10	81499907	1431	625541
0.0001	9.9e-06	5000	100	1	572823573	1453	515631
0.0001	9.9e-06	5000	100	5	268862990	1446	2614916
0.0001	9.9e-06	5000	100	10	371345250	1441	5137088
0.0001	9.9e-05	50	1	1	401709006	1971	213
0.0001	9.9e-05	50	1	5	348944228	1850	1260
0.0001	9.9e-05	50	1	10	411503389	1664	2794
0.0001	9.9e-05	50	10	1	118101173	1626	1391
0.0001	9.9e-05	50	10	5	713182383	1543	6279
0.0001	9.9e-05	50	10	10	206919499	1601	10921
0.0001	9.9e-05	50	100	1	684901575	1528	5886
0.0001	9.9e-05	50	100	5	426147458	1467	28506
0.0001	9.9e-05	50	100	10	58574623	1451	58256
0.0001	9.9e-05	500	1	1	18742133	1554	4557
0.0001	9.9e-05	500	1	5	422235545	1494	15286
0.0001	9.9e-05	500	1	10	57608018	1518	28696
0.0001	9.9e-05	500	10	1	204642903	1479	8991
0.0001	9.9e-05	500	10	5	171949271	1444	47591
0.0001	9.9e-05	500	10	10	743813788	1448	91361
0.0001	9.9e-05	500	100	1	474120792	1449	54106
0.0001	9.9e-05	500	100	5	396691765	1431	271706
0.0001	9.9e-05	500	100	10	38653524	1436	553464
0.0001	9.9e-05	5000	1	1	126890040	1465	11515
0.0001	9.9e-05	5000	1	5	399977923	1438	86988
0.0001	9.9e-05	5000	1	10	673708562	1434	176218
0.0001	9.9e-05	5000	10	1	480309629	1500	59452
0.0001	9.9e-05	5000	10	5	160543576	1436	333048
0.0001	9.9e-05	5000	10	10	215817494	1431	636627
0.0001	9.9e-05	5000	100	1	690150810	1493	510282

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-05	5000	100	5	1673063	1489	2873258
0.0001	9.9e-05	5000	100	10	17898618	1434	5266808
0.001	9.9e-06	50	1	1	78467161	1990	198
0.001	9.9e-06	50	1	5	166635669	1868	1336
0.001	9.9e-06	50	1	10	536133807	1844	2694
0.001	9.9e-06	50	10	1	12628773	1592	1117
0.001	9.9e-06	50	10	5	646343158	1551	5237
0.001	9.9e-06	50	10	10	650204326	1575	12592
0.001	9.9e-06	50	100	1	152806111	1464	5709
0.001	9.9e-06	50	100	5	308742631	1483	28953
0.001	9.9e-06	50	100	10	317041015	1464	57004
0.001	9.9e-06	500	1	1	382226348	1542	2497
0.001	9.9e-06	500	1	5	346124642	1479	15587
0.001	9.9e-06	500	1	10	592182099	1482	33444
0.001	9.9e-06	500	10	1	559161521	1467	9349
0.001	9.9e-06	500	10	5	179973661	1457	50117
0.001	9.9e-06	500	10	10	455604307	1436	96102
0.001	9.9e-06	500	100	1	89849834	1500	54422
0.001	9.9e-06	500	100	5	468532429	1441	269284
0.001	9.9e-06	500	100	10	503746773	1437	540439
0.001	9.9e-06	5000	1	1	638830756	1450	17211
0.001	9.9e-06	5000	1	5	693601954	1438	86327
0.001	9.9e-06	5000	1	10	750479596	1434	163679
0.001	9.9e-06	5000	10	1	402538428	1443	77693
0.001	9.9e-06	5000	10	5	208475322	1431	317117
0.001	9.9e-06	5000	10	10	145491218	1440	626886
0.001	9.9e-06	5000	100	1	31995355	1499	509745
0.001	9.9e-06	5000	100	5	262474317	1444	2570285
0.001	9.9e-06	5000	100	10	151494724	1441	5144892
0.001	9.9e-05	50	1	1	591821110	2006	175
0.001	9.9e-05	50	1	5	27062800	1924	1079
0.001	9.9e-05	50	1	10	30587949	1771	2585
0.001	9.9e-05	50	10	1	752894034	1738	770
0.001	9.9e-05	50	10	5	9365555	1617	5318
0.001	9.9e-05	50	10	10	236583346	1521	11580
0.001	9.9e-05	50	100	1	489236787	1454	6074
0.001	9.9e-05	50	100	5	452193002	1454	28328

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	9.9e-05	50	100	10	485605413	1453	59454
0.001	9.9e-05	500	1	1	189620380	1591	2595
0.001	9.9e-05	500	1	5	281139887	1508	13158
0.001	9.9e-05	500	1	10	415905305	1485	31131
0.001	9.9e-05	500	10	1	411627604	1462	8516
0.001	9.9e-05	500	10	5	584720921	1439	55382
0.001	9.9e-05	500	10	10	352341190	1447	101604
0.001	9.9e-05	500	100	1	628186372	1453	56059
0.001	9.9e-05	500	100	5	109516692	1442	271475
0.001	9.9e-05	500	100	10	20498397	1435	541991
0.001	9.9e-05	5000	1	1	301276010	1458	11246
0.001	9.9e-05	5000	1	5	56764750	1447	79343
0.001	9.9e-05	5000	1	10	209123513	1439	180520
0.001	9.9e-05	5000	10	1	485035388	1497	62235
0.001	9.9e-05	5000	10	5	621356264	1438	322533
0.001	9.9e-05	5000	10	10	74081024	1442	732520
0.001	9.9e-05	5000	100	1	28075634	1462	530621
0.001	9.9e-05	5000	100	5	184824122	1443	2573414
0.001	9.9e-05	5000	100	10	634531828	1429	5144290
0.001	0.00099	50	1	1	492623800	2152	131
0.001	0.00099	50	1	5	571164642	1908	627
0.001	0.00099	50	1	10	548946612	1910	1370
0.001	0.00099	50	10	1	47297626	1911	1268
0.001	0.00099	50	10	5	640809650	1530	7007
0.001	0.00099	50	10	10	542705150	1633	14527
0.001	0.00099	50	100	1	519905119	1451	6952
0.001	0.00099	50	100	5	328764205	1464	36585
0.001	0.00099	50	100	10	339506927	1448	69187
0.001	0.00099	500	1	1	668939875	1632	2938
0.001	0.00099	500	1	5	351189335	1559	10317
0.001	0.00099	500	1	10	615202112	1524	21679
0.001	0.00099	500	10	1	650876663	1512	9973
0.001	0.00099	500	10	5	729484613	1459	64263
0.001	0.00099	500	10	10	613047119	1449	135959
0.001	0.00099	500	100	1	578887073	1434	61846
0.001	0.00099	500	100	5	308367898	1434	361562
0.001	0.00099	500	100	10	403257567	1436	769903

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	0.00099	5000	1	1	104710256	1500	13018
0.001	0.00099	5000	1	5	273935869	1471	73264
0.001	0.00099	5000	1	10	474769330	1501	131054
0.001	0.00099	5000	10	1	739634382	1471	81785
0.001	0.00099	5000	10	5	105980803	1443	448404
0.001	0.00099	5000	10	10	699291298	1448	818594
0.001	0.00099	5000	100	1	128567127	1483	588079
0.001	0.00099	5000	100	5	298207670	1435	3986736
0.001	0.00099	5000	100	10	54307774	1430	7098069
0.01	9.9e-05	50	1	1	149519893	1835	481
0.01	9.9e-05	50	1	5	426391493	1827	1098
0.01	9.9e-05	50	1	10	195360070	1824	2103
0.01	9.9e-05	50	10	1	153673825	1651	1025
0.01	9.9e-05	50	10	5	544346754	1598	5946
0.01	9.9e-05	50	10	10	69654607	1586	11551
0.01	9.9e-05	50	100	1	247221757	1497	5369
0.01	9.9e-05	50	100	5	261370177	1451	30939
0.01	9.9e-05	50	100	10	607125252	1450	59760
0.01	9.9e-05	500	1	1	590852560	1534	2423
0.01	9.9e-05	500	1	5	307738362	1515	11068
0.01	9.9e-05	500	1	10	246925541	1500	33285
0.01	9.9e-05	500	10	1	365402357	1447	8560
0.01	9.9e-05	500	10	5	88836554	1443	51845
0.01	9.9e-05	500	10	10	309923642	1443	91777
0.01	9.9e-05	500	100	1	616366817	1500	55681
0.01	9.9e-05	500	100	5	36752636	1433	273612
0.01	9.9e-05	500	100	10	279570353	1436	542666
0.01	9.9e-05	5000	1	1	32677989	1496	12459
0.01	9.9e-05	5000	1	5	186982551	1441	89168
0.01	9.9e-05	5000	1	10	740265564	1436	183653
0.01	9.9e-05	5000	10	1	278481095	1495	58139
0.01	9.9e-05	5000	10	5	669154726	1440	307906
0.01	9.9e-05	5000	10	10	201364232	1445	731983
0.01	9.9e-05	5000	100	1	363711091	1493	508915
0.01	9.9e-05	5000	100	5	51358210	1440	2562608
0.01	9.9e-05	5000	100	10	595282459	1440	5216948
0.01	0.00099	50	1	1	443567060	1869	268

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.00099	50	1	5	139021905	1797	1176
0.01	0.00099	50	1	10	526757702	1764	1607
0.01	0.00099	50	10	1	584915794	1734	744
0.01	0.00099	50	10	5	6302827	1613	6177
0.01	0.00099	50	10	10	289241600	1575	11967
0.01	0.00099	50	100	1	133794502	1502	6723
0.01	0.00099	50	100	5	412346456	1456	29323
0.01	0.00099	50	100	10	111634291	1463	58144
0.01	0.00099	500	1	1	16927033	1598	2110
0.01	0.00099	500	1	5	675158667	1449	16762
0.01	0.00099	500	1	10	409233802	1455	34452
0.01	0.00099	500	10	1	427263917	1474	13507
0.01	0.00099	500	10	5	477215139	1449	50934
0.01	0.00099	500	10	10	654554742	1463	102138
0.01	0.00099	500	100	1	693847825	1440	52957
0.01	0.00099	500	100	5	152006072	1434	279322
0.01	0.00099	500	100	10	737783078	1432	544046
0.01	0.00099	5000	1	1	335175100	1439	19909
0.01	0.00099	5000	1	5	24308292	1451	83471
0.01	0.00099	5000	1	10	97102016	1443	185293
0.01	0.00099	5000	10	1	129397032	1512	64164
0.01	0.00099	5000	10	5	162485519	1446	309817
0.01	0.00099	5000	10	10	455316836	1430	641184
0.01	0.00099	5000	100	1	279775133	1452	516281
0.01	0.00099	5000	100	5	571921851	1434	2560022
0.01	0.00099	5000	100	10	452566836	1440	5150924
0.01	0.0099	50	1	1	118512010	2243	51
0.01	0.0099	50	1	5	228262771	2267	273
0.01	0.0099	50	1	10	128487973	2152	675
0.01	0.0099	50	10	1	274124738	2086	912
0.01	0.0099	50	10	5	69772237	2043	3072
0.01	0.0099	50	10	10	649170495	2108	6285
0.01	0.0099	50	100	1	545648747	2051	5294
0.01	0.0099	50	100	5	50547927	1448	51969
0.01	0.0099	50	100	10	170417976	1446	166218
0.01	0.0099	500	1	1	468343874	2135	673
0.01	0.0099	500	1	5	158540859	2102	3973

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.0099	500	1	10	132745538	2115	6783
0.01	0.0099	500	10	1	656575761	2030	8192
0.01	0.0099	500	10	5	203366293	2045	28451
0.01	0.0099	500	10	10	440339566	2003	60700
0.01	0.0099	500	100	1	80097275	1440	197263
0.01	0.0099	500	100	5	356845079	1429	966616
0.01	0.0099	500	100	10	416598960	1434	1717295
0.01	0.0099	5000	1	1	193460934	2096	5275
0.01	0.0099	5000	1	5	334579434	2025	28823
0.01	0.0099	5000	1	10	200617308	2040	73016
0.01	0.0099	5000	10	1	561099114	1934	87611
0.01	0.0099	5000	10	5	376754728	1951	303351
0.01	0.0099	5000	10	10	93132049	1965	798803
0.01	0.0099	5000	100	1	311581843	1903	545945
0.01	0.0099	5000	100	5	329114418	1432	9166539
0.01	0.0099	5000	100	10	12481996	1430	17920363
0.1	0.00099	50	1	1	490424363	1846	324
0.1	0.00099	50	1	5	62694327	1783	1356
0.1	0.00099	50	1	10	439952109	1805	1949
0.1	0.00099	50	10	1	41347558	1698	1231
0.1	0.00099	50	10	5	208730516	1622	5479
0.1	0.00099	50	10	10	391878874	1538	12147
0.1	0.00099	50	100	1	228695115	1462	6063
0.1	0.00099	50	100	5	608078205	1460	31164
0.1	0.00099	50	100	10	743207057	1454	57213
0.1	0.00099	500	1	1	468221089	1595	2904
0.1	0.00099	500	1	5	459250865	1520	11743
0.1	0.00099	500	1	10	497941668	1486	26232
0.1	0.00099	500	10	1	67448147	1456	9686
0.1	0.00099	500	10	5	423070123	1456	48291
0.1	0.00099	500	10	10	721074589	1442	101891
0.1	0.00099	500	100	1	127109461	1508	53147
0.1	0.00099	500	100	5	295028837	1440	283639
0.1	0.00099	500	100	10	473285785	1437	542523
0.1	0.00099	5000	1	1	190899091	1511	14650
0.1	0.00099	5000	1	5	33375384	1438	82513
0.1	0.00099	5000	1	10	256459737	1438	173893

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.00099	5000	10	1	592437622	1509	69990
0.1	0.00099	5000	10	5	55487315	1430	309233
0.1	0.00099	5000	10	10	469409230	1433	645847
0.1	0.00099	5000	100	1	75302730	1457	509301
0.1	0.00099	5000	100	5	727057688	1436	2695820
0.1	0.00099	5000	100	10	130289319	1442	5161764
0.1	0.0099	50	1	1	159687013	2261	77
0.1	0.0099	50	1	5	89729630	1913	735
0.1	0.0099	50	1	10	41307345	1723	1498
0.1	0.0099	50	10	1	567090668	1743	1086
0.1	0.0099	50	10	5	462968547	1575	6341
0.1	0.0099	50	10	10	97940692	1580	13520
0.1	0.0099	50	100	1	79816925	1539	5557
0.1	0.0099	50	100	5	616695162	1444	31117
0.1	0.0099	50	100	10	248261997	1456	57245
0.1	0.0099	500	1	1	641129803	2154	501
0.1	0.0099	500	1	5	673760212	1471	12998
0.1	0.0099	500	1	10	234272027	1490	28897
0.1	0.0099	500	10	1	388547359	1506	8960
0.1	0.0099	500	10	5	654771133	1451	49901
0.1	0.0099	500	10	10	559145541	1438	100121
0.1	0.0099	500	100	1	23028672	1450	61418
0.1	0.0099	500	100	5	335933484	1450	278628
0.1	0.0099	500	100	10	692757265	1436	545960
0.1	0.0099	5000	1	1	336525902	1438	18687
0.1	0.0099	5000	1	5	484559584	1437	86027
0.1	0.0099	5000	1	10	468241660	1438	172563
0.1	0.0099	5000	10	1	661274680	1507	56886
0.1	0.0099	5000	10	5	400590482	1449	306485
0.1	0.0099	5000	10	10	663312805	1440	629024
0.1	0.0099	5000	100	1	78105605	1456	513568
0.1	0.0099	5000	100	5	382259149	1442	2661067
0.1	0.0099	5000	100	10	668269453	1438	5167525
0.1	0.099	50	1	1	137452045	2244	51
0.1	0.099	50	1	5	175027052	2144	318
0.1	0.099	50	1	10	85692755	2134	688
0.1	0.099	50	10	1	319674506	2181	501

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.099	50	10	5	711414183	2129	2957
0.1	0.099	50	10	10	167158853	2130	6155
0.1	0.099	50	100	1	401268053	2125	5033
0.1	0.099	50	100	5	538099471	2071	32267
0.1	0.099	50	100	10	485102852	2069	56915
0.1	0.099	500	1	1	516934787	2163	627
0.1	0.099	500	1	5	535548097	2145	3794
0.1	0.099	500	1	10	286712790	2116	7225
0.1	0.099	500	10	1	157008772	2143	5374
0.1	0.099	500	10	5	151978062	2084	27295
0.1	0.099	500	10	10	462694946	2086	60547
0.1	0.099	500	100	1	424081925	2076	52654
0.1	0.099	500	100	5	111883579	1969	284442
0.1	0.099	500	100	10	666916442	2024	658625
0.1	0.099	5000	1	1	199658279	2095	7183
0.1	0.099	5000	1	5	181989792	2095	31965
0.1	0.099	5000	1	10	670409400	2061	80816
0.1	0.099	5000	10	1	470833084	2089	66574
0.1	0.099	5000	10	5	499694832	2058	301371
0.1	0.099	5000	10	10	712829222	2038	540112
0.1	0.099	5000	100	1	504983791	2049	500642
0.1	0.099	5000	100	5	678491023	1977	3554148
0.1	0.099	5000	100	10	744704700	1989	5913480
1	0.0099	50	1	1	489725025	1811	424
1	0.0099	50	1	5	727425725	1814	645
1	0.0099	50	1	10	731609149	1834	1209
1	0.0099	50	10	1	385916711	1643	1642
1	0.0099	50	10	5	431131063	1610	5399
1	0.0099	50	10	10	38529800	1531	12374
1	0.0099	50	100	1	407364018	1471	5688
1	0.0099	50	100	5	143147727	1457	29984
1	0.0099	50	100	10	368653078	1461	61308
1	0.0099	500	1	1	173277821	1573	3179
1	0.0099	500	1	5	232226433	1503	11501
1	0.0099	500	1	10	637676928	1494	23082
1	0.0099	500	10	1	661244802	1488	7784
1	0.0099	500	10	5	107154940	1455	44018

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.0099	500	10	10	177807905	1455	106710
1	0.0099	500	100	1	277907267	1435	54056
1	0.0099	500	100	5	306642922	1445	277996
1	0.0099	500	100	10	34712637	1437	550143
1	0.0099	5000	1	1	501861294	1449	22968
1	0.0099	5000	1	5	144168901	1434	53465
1	0.0099	5000	1	10	604793255	1441	169924
1	0.0099	5000	10	1	120019217	1499	63197
1	0.0099	5000	10	5	717972555	1445	314785
1	0.0099	5000	10	10	547041186	1442	677196
1	0.0099	5000	100	1	166424897	1509	512434
1	0.0099	5000	100	5	726020705	1453	2569129
1	0.0099	5000	100	10	492177257	1432	5156074
1	0.099	50	1	1	583343777	2203	56
1	0.099	50	1	5	386677737	2232	287
1	0.099	50	1	10	372235059	1838	1051
1	0.099	50	10	1	149739717	1770	1349
1	0.099	50	10	5	325405364	1611	6014
1	0.099	50	10	10	497207512	1555	14275
1	0.099	50	100	1	390234770	1488	6135
1	0.099	50	100	5	654385598	1481	31097
1	0.099	50	100	10	539269881	1459	59983
1	0.099	500	1	1	718262948	2182	546
1	0.099	500	1	5	385814692	1537	12214
1	0.099	500	1	10	182491756	1513	27726
1	0.099	500	10	1	327147617	1521	8217
1	0.099	500	10	5	282144258	1457	51384
1	0.099	500	10	10	487026179	1441	94048
1	0.099	500	100	1	587301998	1523	55170
1	0.099	500	100	5	89138000	1436	288339
1	0.099	500	100	10	257253157	1437	551845
1	0.099	5000	1	1	479889664	1445	20391
1	0.099	5000	1	5	555818046	1437	94225
1	0.099	5000	1	10	488588162	1433	147007
1	0.099	5000	10	1	108384678	1453	58732
1	0.099	5000	10	5	202379352	1448	320462
1	0.099	5000	10	10	540397542	1440	672105

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.099	5000	100	1	455670869	1502	508346
1	0.099	5000	100	5	743555271	1444	2587838
1	0.099	5000	100	10	195096848	1433	5193554
1	0.99	50	1	1	641519407	2271	51
1	0.99	50	1	5	733593337	2151	371
1	0.99	50	1	10	287011944	2150	723
1	0.99	50	10	1	81829695	2174	844
1	0.99	50	10	5	45206015	2113	2901
1	0.99	50	10	10	438001747	2123	6314
1	0.99	50	100	1	472453702	2123	8334
1	0.99	50	100	5	596638285	2068	29766
1	0.99	50	100	10	632384120	2079	52204
1	0.99	500	1	1	707520713	2168	589
1	0.99	500	1	5	347317160	2123	3732
1	0.99	500	1	10	251214738	2099	6695
1	0.99	500	10	1	305203462	2146	5628
1	0.99	500	10	5	210335784	2112	31608
1	0.99	500	10	10	338063157	2036	57356
1	0.99	500	100	1	516079732	2072	58771
1	0.99	500	100	5	424275490	2027	254573
1	0.99	500	100	10	506297664	1997	582606
1	0.99	5000	1	1	396353803	2130	6223
1	0.99	5000	1	5	485984184	2003	36315
1	0.99	5000	1	10	146675445	2088	77350
1	0.99	5000	10	1	362899979	2079	51322
1	0.99	5000	10	5	371642730	2071	290180
1	0.99	5000	10	10	358820163	2025	648243
1	0.99	5000	100	1	228291461	2033	524071
1	0.99	5000	100	5	410252254	1994	2918568
1	0.99	5000	100	10	636680029	2011	5134809

B.2 Monocot Data Matrix, maxaccept = 50

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	50	1	1	528112358	2168	51
1e-05	9.9e-08	50	1	5	329726347	2131	255
1e-05	9.9e-08	50	1	10	749184481	2063	510

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	50	10	1	489192482	1896	501
1e-05	9.9e-08	50	10	5	17072098	1638	2505
1e-05	9.9e-08	50	10	10	593263867	1735	5010
1e-05	9.9e-08	50	100	1	637109570	1517	5001
1e-05	9.9e-08	50	100	5	362273333	1471	25005
1e-05	9.9e-08	50	100	10	243697406	1465	50010
1e-05	9.9e-08	500	1	1	72458201	1707	688
1e-05	9.9e-08	500	1	5	662774906	1671	3561
1e-05	9.9e-08	500	1	10	68451316	1654	7191
1e-05	9.9e-08	500	10	1	482378389	1480	5240
1e-05	9.9e-08	500	10	5	138921309	1463	26286
1e-05	9.9e-08	500	10	10	292099408	1463	52864
1e-05	9.9e-08	500	100	1	328485768	1442	50224
1e-05	9.9e-08	500	100	5	618145038	1447	251511
1e-05	9.9e-08	500	100	10	174927775	1436	502826
1e-05	9.9e-08	5000	1	1	553054509	1520	8915
1e-05	9.9e-08	5000	1	5	466430162	1453	38630
1e-05	9.9e-08	5000	1	10	196287692	1444	79853
1e-05	9.9e-08	5000	10	1	224331511	1483	55195
1e-05	9.9e-08	5000	10	5	690811519	1437	267291
1e-05	9.9e-08	5000	10	10	464155766	1437	527211
1e-05	9.9e-08	5000	100	1	3920111	1442	503834
1e-05	9.9e-08	5000	100	5	235398900	1497	2508792
1e-05	9.9e-08	5000	100	10	156176087	1430	5027524
1e-05	9.9e-07	50	1	1	241061648	2210	51
1e-05	9.9e-07	50	1	5	26358908	2163	255
1e-05	9.9e-07	50	1	10	650590474	1987	510
1e-05	9.9e-07	50	10	1	178706178	1760	501
1e-05	9.9e-07	50	10	5	60337699	1711	2505
1e-05	9.9e-07	50	10	10	647592136	1662	5010
1e-05	9.9e-07	50	100	1	110444955	1480	5001
1e-05	9.9e-07	50	100	5	97455103	1459	25005
1e-05	9.9e-07	50	100	10	583894270	1462	50010
1e-05	9.9e-07	500	1	1	635449506	1722	724
1e-05	9.9e-07	500	1	5	631226201	1596	4220
1e-05	9.9e-07	500	1	10	585252318	1626	8146
1e-05	9.9e-07	500	10	1	298136807	1467	5595

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-07	500	10	5	111656461	1464	26321
1e-05	9.9e-07	500	10	10	696268237	1469	52521
1e-05	9.9e-07	500	100	1	570799698	1457	50180
1e-05	9.9e-07	500	100	5	731299393	1437	251590
1e-05	9.9e-07	500	100	10	356965504	1445	502098
1e-05	9.9e-07	5000	1	1	107921697	1449	10011
1e-05	9.9e-07	5000	1	5	175206797	1446	35454
1e-05	9.9e-07	5000	1	10	344380521	1449	79493
1e-05	9.9e-07	5000	10	1	35608818	1500	55048
1e-05	9.9e-07	5000	10	5	345529947	1441	264905
1e-05	9.9e-07	5000	10	10	24734478	1433	533852
1e-05	9.9e-07	5000	100	1	516211862	1458	503742
1e-05	9.9e-07	5000	100	5	215334274	1438	2516007
1e-05	9.9e-07	5000	100	10	464200240	1433	5035563
1e-05	9.9e-06	50	1	1	316597929	2204	51
1e-05	9.9e-06	50	1	5	628061491	2104	255
1e-05	9.9e-06	50	1	10	389642001	2107	510
1e-05	9.9e-06	50	10	1	105856188	1820	501
1e-05	9.9e-06	50	10	5	228744093	1747	2505
1e-05	9.9e-06	50	10	10	335791565	1729	5010
1e-05	9.9e-06	50	100	1	52270388	1514	5001
1e-05	9.9e-06	50	100	5	514274831	1468	25005
1e-05	9.9e-06	50	100	10	24081283	1469	50010
1e-05	9.9e-06	500	1	1	622034222	1730	709
1e-05	9.9e-06	500	1	5	705963925	1662	3685
1e-05	9.9e-06	500	1	10	123962096	1638	7118
1e-05	9.9e-06	500	10	1	269767590	1562	5248
1e-05	9.9e-06	500	10	5	215907827	1467	26053
1e-05	9.9e-06	500	10	10	70720084	1460	52753
1e-05	9.9e-06	500	100	1	165171487	1457	50226
1e-05	9.9e-06	500	100	5	468963823	1431	251865
1e-05	9.9e-06	500	100	10	220708931	1437	502236
1e-05	9.9e-06	5000	1	1	574527050	1515	8838
1e-05	9.9e-06	5000	1	5	426273573	1464	32961
1e-05	9.9e-06	5000	1	10	543855469	1448	86332
1e-05	9.9e-06	5000	10	1	517794937	1437	52772
1e-05	9.9e-06	5000	10	5	696858711	1442	265241

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-06	5000	10	10	343498805	1435	529040
1e-05	9.9e-06	5000	100	1	415789687	1470	501277
1e-05	9.9e-06	5000	100	5	29431794	1438	2513289
1e-05	9.9e-06	5000	100	10	697691021	1437	5033017
0.0001	9.9e-07	50	1	1	438795976	2081	51
0.0001	9.9e-07	50	1	5	739650896	2015	255
0.0001	9.9e-07	50	1	10	268161923	2100	510
0.0001	9.9e-07	50	10	1	265803504	1721	501
0.0001	9.9e-07	50	10	5	455850099	1758	2505
0.0001	9.9e-07	50	10	10	243898740	1724	5010
0.0001	9.9e-07	50	100	1	17610653	1516	5001
0.0001	9.9e-07	50	100	5	609357935	1454	25005
0.0001	9.9e-07	50	100	10	505458374	1446	50010
0.0001	9.9e-07	500	1	1	78351045	1874	720
0.0001	9.9e-07	500	1	5	532854841	1573	4127
0.0001	9.9e-07	500	1	10	201493534	1588	8417
0.0001	9.9e-07	500	10	1	126996728	1511	5217
0.0001	9.9e-07	500	10	5	694465196	1464	26591
0.0001	9.9e-07	500	10	10	188866834	1468	52777
0.0001	9.9e-07	500	100	1	413489665	1501	50256
0.0001	9.9e-07	500	100	5	39698803	1439	251563
0.0001	9.9e-07	500	100	10	588752433	1432	502829
0.0001	9.9e-07	5000	1	1	473141141	1467	6021
0.0001	9.9e-07	5000	1	5	568338206	1455	39784
0.0001	9.9e-07	5000	1	10	662351967	1451	85831
0.0001	9.9e-07	5000	10	1	434530230	1446	51149
0.0001	9.9e-07	5000	10	5	276862046	1444	267926
0.0001	9.9e-07	5000	10	10	587623898	1434	532558
0.0001	9.9e-07	5000	100	1	237344361	1497	504412
0.0001	9.9e-07	5000	100	5	430231918	1439	2512176
0.0001	9.9e-07	5000	100	10	247540141	1435	5039211
0.0001	9.9e-06	50	1	1	585392869	2163	51
0.0001	9.9e-06	50	1	5	171900335	2123	255
0.0001	9.9e-06	50	1	10	263215677	2073	510
0.0001	9.9e-06	50	10	1	652755976	1821	501
0.0001	9.9e-06	50	10	5	353894502	1718	2505
0.0001	9.9e-06	50	10	10	64111463	1711	5010

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-06	50	100	1	45081902	1532	5001
0.0001	9.9e-06	50	100	5	725668308	1462	25005
0.0001	9.9e-06	50	100	10	44462692	1465	50010
0.0001	9.9e-06	500	1	1	735153473	1577	1175
0.0001	9.9e-06	500	1	5	543244850	1717	3525
0.0001	9.9e-06	500	1	10	547245001	1663	7508
0.0001	9.9e-06	500	10	1	661505518	1480	5134
0.0001	9.9e-06	500	10	5	407766563	1481	26271
0.0001	9.9e-06	500	10	10	330078213	1446	52810
0.0001	9.9e-06	500	100	1	438397237	1496	50605
0.0001	9.9e-06	500	100	5	590100244	1442	251238
0.0001	9.9e-06	500	100	10	452172847	1442	502904
0.0001	9.9e-06	5000	1	1	450439445	1452	6819
0.0001	9.9e-06	5000	1	5	589658391	1451	34794
0.0001	9.9e-06	5000	1	10	632494035	1430	77326
0.0001	9.9e-06	5000	10	1	230670636	1447	51103
0.0001	9.9e-06	5000	10	5	424054084	1441	262196
0.0001	9.9e-06	5000	10	10	591746331	1434	530893
0.0001	9.9e-06	5000	100	1	45954684	1448	503447
0.0001	9.9e-06	5000	100	5	257747662	1490	2514970
0.0001	9.9e-06	5000	100	10	435166460	1434	5029919
0.0001	9.9e-05	50	1	1	658149699	2229	51
0.0001	9.9e-05	50	1	5	595155222	2118	255
0.0001	9.9e-05	50	1	10	379786291	2051	510
0.0001	9.9e-05	50	10	1	487701581	1734	501
0.0001	9.9e-05	50	10	5	440788721	1791	2505
0.0001	9.9e-05	50	10	10	724785644	1690	5010
0.0001	9.9e-05	50	100	1	415331030	1472	5001
0.0001	9.9e-05	50	100	5	44717259	1476	25005
0.0001	9.9e-05	50	100	10	157685526	1465	50010
0.0001	9.9e-05	500	1	1	115876046	1608	1126
0.0001	9.9e-05	500	1	5	705706643	1734	3449
0.0001	9.9e-05	500	1	10	610364415	1639	7144
0.0001	9.9e-05	500	10	1	597297545	1495	5168
0.0001	9.9e-05	500	10	5	327338145	1455	26014
0.0001	9.9e-05	500	10	10	646731793	1468	52287
0.0001	9.9e-05	500	100	1	700526845	1439	50273

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-05	500	100	5	210190482	1433	251475
0.0001	9.9e-05	500	100	10	417640464	1442	503315
0.0001	9.9e-05	5000	1	1	424288835	1516	7242
0.0001	9.9e-05	5000	1	5	637359522	1439	40425
0.0001	9.9e-05	5000	1	10	557175874	1449	86989
0.0001	9.9e-05	5000	10	1	264549456	1503	52634
0.0001	9.9e-05	5000	10	5	192525644	1435	261351
0.0001	9.9e-05	5000	10	10	188605855	1437	521932
0.0001	9.9e-05	5000	100	1	110291343	1435	502767
0.0001	9.9e-05	5000	100	5	95415287	1441	2516066
0.0001	9.9e-05	5000	100	10	136476027	1438	5038175
0.001	9.9e-06	50	1	1	459848682	2238	51
0.001	9.9e-06	50	1	5	342755503	2163	255
0.001	9.9e-06	50	1	10	648751868	2097	510
0.001	9.9e-06	50	10	1	200777211	1807	501
0.001	9.9e-06	50	10	5	624915673	1704	2505
0.001	9.9e-06	50	10	10	379569525	1767	5010
0.001	9.9e-06	50	100	1	618713143	1508	5001
0.001	9.9e-06	50	100	5	481241148	1459	25005
0.001	9.9e-06	50	100	10	269477716	1451	50010
0.001	9.9e-06	500	1	1	582421336	2049	501
0.001	9.9e-06	500	1	5	366895569	1622	5345
0.001	9.9e-06	500	1	10	441107050	1578	8948
0.001	9.9e-06	500	10	1	84630636	1472	5820
0.001	9.9e-06	500	10	5	434704140	1459	28004
0.001	9.9e-06	500	10	10	478245888	1461	54661
0.001	9.9e-06	500	100	1	231445852	1490	50471
0.001	9.9e-06	500	100	5	504517234	1438	250236
0.001	9.9e-06	500	100	10	616198015	1443	503099
0.001	9.9e-06	5000	1	1	235585781	1501	8663
0.001	9.9e-06	5000	1	5	484953419	1448	45182
0.001	9.9e-06	5000	1	10	569637233	1455	76627
0.001	9.9e-06	5000	10	1	614115682	1496	53389
0.001	9.9e-06	5000	10	5	527141679	1432	266458
0.001	9.9e-06	5000	10	10	589490143	1432	533991
0.001	9.9e-06	5000	100	1	486711146	1440	507679
0.001	9.9e-06	5000	100	5	506532234	1436	2518926

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	9.9e-06	5000	100	10	66602248	1438	5033475
0.001	9.9e-05	50	1	1	193232599	2197	51
0.001	9.9e-05	50	1	5	351974467	2173	255
0.001	9.9e-05	50	1	10	39770292	2155	510
0.001	9.9e-05	50	10	1	537555887	1847	501
0.001	9.9e-05	50	10	5	419610637	1801	2505
0.001	9.9e-05	50	10	10	643633492	1675	5010
0.001	9.9e-05	50	100	1	403830146	1455	5001
0.001	9.9e-05	50	100	5	88461161	1457	25005
0.001	9.9e-05	50	100	10	389650530	1480	50010
0.001	9.9e-05	500	1	1	344761766	2057	501
0.001	9.9e-05	500	1	5	13358143	1619	3834
0.001	9.9e-05	500	1	10	276566170	1604	8292
0.001	9.9e-05	500	10	1	695001769	1523	5001
0.001	9.9e-05	500	10	5	185496465	1482	26596
0.001	9.9e-05	500	10	10	457975445	1452	53199
0.001	9.9e-05	500	100	1	24724228	1459	50001
0.001	9.9e-05	500	100	5	415545291	1447	252359
0.001	9.9e-05	500	100	10	642395814	1441	503493
0.001	9.9e-05	5000	1	1	658222803	1476	7783
0.001	9.9e-05	5000	1	5	559816164	1450	39054
0.001	9.9e-05	5000	1	10	582874446	1441	87440
0.001	9.9e-05	5000	10	1	297137827	1454	51946
0.001	9.9e-05	5000	10	5	93480617	1446	262568
0.001	9.9e-05	5000	10	10	721691652	1432	534044
0.001	9.9e-05	5000	100	1	160944371	1499	502087
0.001	9.9e-05	5000	100	5	385559762	1445	2513256
0.001	9.9e-05	5000	100	10	691999483	1442	5026053
0.001	0.00099	50	1	1	382784134	2182	51
0.001	0.00099	50	1	5	551095797	2160	255
0.001	0.00099	50	1	10	62241688	2086	510
0.001	0.00099	50	10	1	514343970	1872	501
0.001	0.00099	50	10	5	703091320	1885	2505
0.001	0.00099	50	10	10	537233317	1830	5010
0.001	0.00099	50	100	1	264820050	1543	5001
0.001	0.00099	50	100	5	590153153	1489	25005
0.001	0.00099	50	100	10	218499913	1471	50010

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	0.00099	500	1	1	491506398	1682	848
0.001	0.00099	500	1	5	143258338	1743	3779
0.001	0.00099	500	1	10	701662099	1773	6032
0.001	0.00099	500	10	1	60127126	1653	5001
0.001	0.00099	500	10	5	86138565	1542	26427
0.001	0.00099	500	10	10	178217266	1555	53545
0.001	0.00099	500	100	1	531776364	1512	50302
0.001	0.00099	500	100	5	155872918	1451	251043
0.001	0.00099	500	100	10	296141841	1439	503720
0.001	0.00099	5000	1	1	184533996	1605	7589
0.001	0.00099	5000	1	5	45087690	1602	40753
0.001	0.00099	5000	1	10	29220761	1550	73837
0.001	0.00099	5000	10	1	131718358	1518	53977
0.001	0.00099	5000	10	5	361443887	1512	267068
0.001	0.00099	5000	10	10	383928631	1503	520213
0.001	0.00099	5000	100	1	491809316	1437	504249
0.001	0.00099	5000	100	5	105041176	1432	2509184
0.001	0.00099	5000	100	10	510727634	1432	5028717
0.01	9.9e-05	50	1	1	423772653	2275	51
0.01	9.9e-05	50	1	5	87693232	2209	255
0.01	9.9e-05	50	1	10	380747047	2134	510
0.01	9.9e-05	50	10	1	130473097	1706	501
0.01	9.9e-05	50	10	5	184423417	1723	2505
0.01	9.9e-05	50	10	10	465671619	1774	5010
0.01	9.9e-05	50	100	1	279571657	1519	5001
0.01	9.9e-05	50	100	5	80177799	1475	25005
0.01	9.9e-05	50	100	10	394370222	1451	50010
0.01	9.9e-05	500	1	1	67363086	2135	501
0.01	9.9e-05	500	1	5	340979197	2120	2505
0.01	9.9e-05	500	1	10	529382771	2098	5010
0.01	9.9e-05	500	10	1	470664	1514	5992
0.01	9.9e-05	500	10	5	261822743	1468	27637
0.01	9.9e-05	500	10	10	532021267	1462	54024
0.01	9.9e-05	500	100	1	301191528	1493	51189
0.01	9.9e-05	500	100	5	733646398	1433	254327
0.01	9.9e-05	500	100	10	54829228	1441	507730
0.01	9.9e-05	5000	1	1	122796328	2096	5001

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	9.9e-05	5000	1	5	120147024	2064	25005
0.01	9.9e-05	5000	1	10	466582089	2024	50010
0.01	9.9e-05	5000	10	1	181780513	1501	56667
0.01	9.9e-05	5000	10	5	121667937	1440	287217
0.01	9.9e-05	5000	10	10	337611170	1436	574046
0.01	9.9e-05	5000	100	1	596875858	1440	506795
0.01	9.9e-05	5000	100	5	705712624	1437	2539806
0.01	9.9e-05	5000	100	10	669107681	1438	5072692
0.01	0.00099	50	1	1	478537316	2247	51
0.01	0.00099	50	1	5	80386952	2145	255
0.01	0.00099	50	1	10	188589634	2188	510
0.01	0.00099	50	10	1	704280723	1866	501
0.01	0.00099	50	10	5	165679027	1715	2505
0.01	0.00099	50	10	10	180460013	1794	5010
0.01	0.00099	50	100	1	712310390	1527	5001
0.01	0.00099	50	100	5	682583560	1466	25005
0.01	0.00099	50	100	10	259699334	1475	50010
0.01	0.00099	500	1	1	3352600	2113	501
0.01	0.00099	500	1	5	693476388	2118	2505
0.01	0.00099	500	1	10	270591858	2113	5010
0.01	0.00099	500	10	1	24820771	1526	5001
0.01	0.00099	500	10	5	610406475	1464	25811
0.01	0.00099	500	10	10	257070727	1443	53819
0.01	0.00099	500	100	1	566624336	1460	50001
0.01	0.00099	500	100	5	402877104	1435	252372
0.01	0.00099	500	100	10	521439817	1440	507156
0.01	0.00099	5000	1	1	390565108	2133	5001
0.01	0.00099	5000	1	5	132170738	2041	25005
0.01	0.00099	5000	1	10	284511322	2001	50010
0.01	0.00099	5000	10	1	381887979	1482	57281
0.01	0.00099	5000	10	5	36172905	1440	285716
0.01	0.00099	5000	10	10	2910012	1437	577920
0.01	0.00099	5000	100	1	20533618	1511	507534
0.01	0.00099	5000	100	5	690670631	1444	2538617
0.01	0.00099	5000	100	10	587078039	1427	5093663
0.01	0.0099	50	1	1	17531294	2140	51
0.01	0.0099	50	1	5	583264133	2124	255

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.0099	50	1	10	357789257	2140	510
0.01	0.0099	50	10	1	649862879	2194	501
0.01	0.0099	50	10	5	565916960	2129	2505
0.01	0.0099	50	10	10	235368836	2041	5010
0.01	0.0099	50	100	1	614209516	2049	5001
0.01	0.0099	50	100	5	695397144	1994	25005
0.01	0.0099	50	100	10	302008239	1940	50010
0.01	0.0099	500	1	1	444378834	2159	501
0.01	0.0099	500	1	5	578523940	2103	2505
0.01	0.0099	500	1	10	508503373	2059	5010
0.01	0.0099	500	10	1	592842934	2091	5001
0.01	0.0099	500	10	5	269585199	2049	25005
0.01	0.0099	500	10	10	684524096	2013	50010
0.01	0.0099	500	100	1	485387543	1958	50001
0.01	0.0099	500	100	5	313403916	1949	250005
0.01	0.0099	500	100	10	144663934	1908	500010
0.01	0.0099	5000	1	1	711433999	2069	5001
0.01	0.0099	5000	1	5	361766518	2066	25005
0.01	0.0099	5000	1	10	539316266	1977	50010
0.01	0.0099	5000	10	1	382554659	2049	50001
0.01	0.0099	5000	10	5	557303426	1997	250005
0.01	0.0099	5000	10	10	10648118	1934	500010
0.01	0.0099	5000	100	1	26652372	1912	500001
0.01	0.0099	5000	100	5	519531200	1819	2500005
0.01	0.0099	5000	100	10	422965164	1823	5000010
0.1	0.00099	50	1	1	541748907	2249	51
0.1	0.00099	50	1	5	168152657	2161	255
0.1	0.00099	50	1	10	368616079	2162	510
0.1	0.00099	50	10	1	12497358	1795	501
0.1	0.00099	50	10	5	109010066	1772	2505
0.1	0.00099	50	10	10	317974475	1680	5010
0.1	0.00099	50	100	1	587846030	1543	5001
0.1	0.00099	50	100	5	159028866	1490	25005
0.1	0.00099	50	100	10	405709668	1449	50010
0.1	0.00099	500	1	1	482303335	2163	501
0.1	0.00099	500	1	5	155110935	2139	2505
0.1	0.00099	500	1	10	345633426	2153	5010

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.00099	500	10	1	287708103	1509	5001
0.1	0.00099	500	10	5	139173809	1455	28184
0.1	0.00099	500	10	10	512028585	1475	52448
0.1	0.00099	500	100	1	562483078	1505	50001
0.1	0.00099	500	100	5	409394435	1444	251409
0.1	0.00099	500	100	10	498261047	1436	509441
0.1	0.00099	5000	1	1	639414396	2127	5001
0.1	0.00099	5000	1	5	399188384	2069	25005
0.1	0.00099	5000	1	10	452571889	2099	50010
0.1	0.00099	5000	10	1	440186748	1438	56540
0.1	0.00099	5000	10	5	85859514	1431	295446
0.1	0.00099	5000	10	10	450831982	1437	593161
0.1	0.00099	5000	100	1	678296296	1503	510590
0.1	0.00099	5000	100	5	60809101	1440	2547134
0.1	0.00099	5000	100	10	230409418	1437	5093362
0.1	0.0099	50	1	1	684762763	2228	51
0.1	0.0099	50	1	5	569453957	2193	255
0.1	0.0099	50	1	10	320753476	2189	510
0.1	0.0099	50	10	1	8587939	1972	501
0.1	0.0099	50	10	5	132540132	1799	2505
0.1	0.0099	50	10	10	145865663	1740	5010
0.1	0.0099	50	100	1	696246210	1497	5001
0.1	0.0099	50	100	5	354466686	1465	25005
0.1	0.0099	50	100	10	410482814	1451	50010
0.1	0.0099	500	1	1	741007412	2144	501
0.1	0.0099	500	1	5	31239823	2136	2505
0.1	0.0099	500	1	10	375317375	2123	5010
0.1	0.0099	500	10	1	289460328	1512	5001
0.1	0.0099	500	10	5	22037171	1467	28373
0.1	0.0099	500	10	10	391205501	1469	61558
0.1	0.0099	500	100	1	395127369	1442	51951
0.1	0.0099	500	100	5	493863795	1437	253361
0.1	0.0099	500	100	10	696474490	1436	506720
0.1	0.0099	5000	1	1	546609588	2125	5001
0.1	0.0099	5000	1	5	447454149	2099	25005
0.1	0.0099	5000	1	10	649480747	1999	50010
0.1	0.0099	5000	10	1	579464179	1498	61433

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.0099	5000	10	5	703073915	1438	317832
0.1	0.0099	5000	10	10	366296850	1444	642872
0.1	0.0099	5000	100	1	587425523	1442	514965
0.1	0.0099	5000	100	5	548985060	1448	2574101
0.1	0.0099	5000	100	10	424887492	1440	5131700
0.1	0.099	50	1	1	530627406	2238	51
0.1	0.099	50	1	5	171345431	2210	255
0.1	0.099	50	1	10	86545143	2172	510
0.1	0.099	50	10	1	404757583	2168	501
0.1	0.099	50	10	5	157302532	2116	2505
0.1	0.099	50	10	10	23820512	2117	5010
0.1	0.099	50	100	1	579662926	2099	5001
0.1	0.099	50	100	5	395090519	2115	25005
0.1	0.099	50	100	10	131964297	2048	50010
0.1	0.099	500	1	1	64668153	2163	501
0.1	0.099	500	1	5	239288788	2152	2505
0.1	0.099	500	1	10	694132968	2119	5010
0.1	0.099	500	10	1	692517658	2118	5001
0.1	0.099	500	10	5	647702784	2060	25005
0.1	0.099	500	10	10	443822118	2091	50010
0.1	0.099	500	100	1	384063361	2070	50001
0.1	0.099	500	100	5	308405297	2044	250005
0.1	0.099	500	100	10	17262127	1986	500010
0.1	0.099	5000	1	1	184250753	2103	5001
0.1	0.099	5000	1	5	276527171	2072	25005
0.1	0.099	5000	1	10	311986534	2095	50010
0.1	0.099	5000	10	1	537122674	2104	50001
0.1	0.099	5000	10	5	684777441	2042	250005
0.1	0.099	5000	10	10	713612913	1962	500010
0.1	0.099	5000	100	1	668689505	2033	500001
0.1	0.099	5000	100	5	152181206	1996	2500005
0.1	0.099	5000	100	10	197891857	1998	5000010
1	0.0099	50	1	1	375868035	2213	51
1	0.0099	50	1	5	424437254	2144	255
1	0.0099	50	1	10	588398772	2195	510
1	0.0099	50	10	1	314453069	1848	501
1	0.0099	50	10	5	655587067	1769	2505

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.0099	50	10	10	286206139	1811	5010
1	0.0099	50	100	1	454235778	1476	5001
1	0.0099	50	100	5	208800252	1476	25005
1	0.0099	50	100	10	323454824	1468	50010
1	0.0099	500	1	1	583652454	2187	501
1	0.0099	500	1	5	405025556	2123	2505
1	0.0099	500	1	10	529182904	2073	5010
1	0.0099	500	10	1	297460219	1509	5001
1	0.0099	500	10	5	246518185	1479	27909
1	0.0099	500	10	10	131224040	1452	60845
1	0.0099	500	100	1	26502413	1502	51333
1	0.0099	500	100	5	553366671	1452	255516
1	0.0099	500	100	10	519002317	1434	506737
1	0.0099	5000	1	1	705426026	2103	5001
1	0.0099	5000	1	5	114301648	2093	25005
1	0.0099	5000	1	10	309398148	2066	50010
1	0.0099	5000	10	1	728539529	1496	61052
1	0.0099	5000	10	5	370885886	1435	313982
1	0.0099	5000	10	10	458789020	1437	629317
1	0.0099	5000	100	1	481915857	1502	512440
1	0.0099	5000	100	5	116152118	1440	2568375
1	0.0099	5000	100	10	403835135	1438	5123479
1	0.099	50	1	1	680300361	2200	51
1	0.099	50	1	5	435185127	2126	255
1	0.099	50	1	10	682268612	2169	510
1	0.099	50	10	1	179766568	1924	501
1	0.099	50	10	5	681621206	1775	2505
1	0.099	50	10	10	601230314	1808	5010
1	0.099	50	100	1	533134101	1520	5001
1	0.099	50	100	5	684235098	1496	25005
1	0.099	50	100	10	660309897	1473	50010
1	0.099	500	1	1	98385542	2052	501
1	0.099	500	1	5	682223491	2122	2505
1	0.099	500	1	10	489925267	2163	5010
1	0.099	500	10	1	434119769	1568	5001
1	0.099	500	10	5	12189138	1463	31518
1	0.099	500	10	10	95154576	1467	54942

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.099	500	100	1	589204684	1510	50001
1	0.099	500	100	5	696392164	1438	254924
1	0.099	500	100	10	281299807	1439	507270
1	0.099	5000	1	1	581744636	2128	5001
1	0.099	5000	1	5	500659112	2092	25005
1	0.099	5000	1	10	390220736	2038	50010
1	0.099	5000	10	1	516561542	1441	66870
1	0.099	5000	10	5	636375796	1437	335874
1	0.099	5000	10	10	6816	1432	679375
1	0.099	5000	100	1	347521083	1505	516645
1	0.099	5000	100	5	747358400	1432	2593697
1	0.099	5000	100	10	634284564	1439	5184192
1	0.99	50	1	1	657788490	2202	51
1	0.99	50	1	5	60894585	2184	255
1	0.99	50	1	10	90422864	2191	510
1	0.99	50	10	1	69917758	2199	501
1	0.99	50	10	5	571745907	2179	2505
1	0.99	50	10	10	231211380	2136	5010
1	0.99	50	100	1	461653123	2074	5001
1	0.99	50	100	5	738437097	2064	25005
1	0.99	50	100	10	400134770	2108	50010
1	0.99	500	1	1	22774835	2214	501
1	0.99	500	1	5	102879379	2126	2505
1	0.99	500	1	10	371926567	2134	5010
1	0.99	500	10	1	133217528	2104	5001
1	0.99	500	10	5	18936941	2106	25005
1	0.99	500	10	10	75557927	2080	50010
1	0.99	500	100	1	220183251	2075	50001
1	0.99	500	100	5	628555556	2037	250005
1	0.99	500	100	10	722099133	2044	500010
1	0.99	5000	1	1	663508964	2138	5001
1	0.99	5000	1	5	498003813	2033	25005
1	0.99	5000	1	10	677777202	2058	50010
1	0.99	5000	10	1	513907290	2070	50001
1	0.99	5000	10	5	180921996	2039	250005
1	0.99	5000	10	10	729691033	2072	500010
1	0.99	5000	100	1	380410742	2061	500001

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.99	5000	100	5	594086679	1991	2500005
1	0.99	5000	100	10	431723453	2004	5000010

B.3 General Data Matrix, maxaccept = 5

For one 10-run analysis, the runs were spread over two computer systems to obtain results more quickly. Seven runs were performed on one system with seed 9020755 and three runs were performed on a different system with seed 100192408. For the analysis in question, an asterisk is placed in the seed column. This causes the value to be treated as missing data by Minitab.

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	50	1	1	886725098	15431	1155
1e-05	9.9e-08	50	1	5	39501	15341	5107
1e-05	9.9e-08	50	1	10	886725199	15070	12365
1e-05	9.9e-08	50	10	1	886905030	13474	4987
1e-05	9.9e-08	50	10	5	391205	13231	24697
1e-05	9.9e-08	50	10	10	886906833	13244	47309
1e-05	9.9e-08	50	100	1	886907460	13637	5246
1e-05	9.9e-08	50	100	5	149810	13074	26111
1e-05	9.9e-08	50	100	10	887337016	13010	54242
1e-05	9.9e-08	500	1	1	880969025	12850	7607
1e-05	9.9e-08	500	1	5	100143	12631	38817
1e-05	9.9e-08	500	1	10	881002805	12708	72773
1e-05	9.9e-08	500	10	1	881127339	12340	8891
1e-05	9.9e-08	500	10	5	1905102	12707	38325
1e-05	9.9e-08	500	10	10	881180340	12419	86390
1e-05	9.9e-08	500	100	1	881255100	12646	8340
1e-05	9.9e-08	500	100	5	900312	12330	44692
1e-05	9.9e-08	500	100	10	881197200	12266	89905
1e-05	9.9e-08	5000	1	1	881125938	12928	8889
1e-05	9.9e-08	5000	1	5	392104	12692	40523
1e-05	9.9e-08	5000	1	10	881172006	12639	72121
1e-05	9.9e-08	5000	10	1	881172006	12822	6459
1e-05	9.9e-08	5000	10	5	83085	12611	39105

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	5000	10	10	881174401	12503	84173
1e-05	9.9e-08	5000	100	1	881605805	13059	6534
1e-05	9.9e-08	5000	100	5	8126202	12855	37090
1e-05	9.9e-08	5000	100	10	881876400	12317	87007
1e-05	9.9e-07	50	1	1	886724303	15637	924
1e-05	9.9e-07	50	1	5	230502212	15630	4437
1e-05	9.9e-07	50	1	10	886724514	15084	11361
1e-05	9.9e-07	50	10	1	886905153	14525	2343
1e-05	9.9e-07	50	10	5	139340	13161	24932
1e-05	9.9e-07	50	10	10	887336851	12738	52905
1e-05	9.9e-07	50	100	1	887346028	12240	13475
1e-05	9.9e-07	50	100	5	1014963	11998	63747
1e-05	9.9e-07	50	100	10	887357954	11817	132104
1e-05	9.9e-07	500	1	1	874513878	11397	19444
1e-05	9.9e-07	500	1	5	290021	11241	82979
1e-05	9.9e-07	500	1	10	874519310	11306	172945
1e-05	9.9e-07	500	10	1	880855809	9890	67376
1e-05	9.9e-07	500	10	5	2900202	9839	300495
1e-05	9.9e-07	500	10	10	880765638	9823	558896
1e-05	9.9e-07	500	100	1	874514179	9876	64052
1e-05	9.9e-07	500	100	5	195202	9868	327404
1e-05	9.9e-07	500	100	10	874519634	9933	580195
1e-05	9.9e-07	5000	1	1	874604817	9682	91463
1e-05	9.9e-07	5000	1	5	29101	9371	514813
1e-05	9.9e-07	5000	1	10	874631473	9383	1006204
1e-05	9.9e-07	5000	10	1	881129647	9699	86128
1e-05	9.9e-07	5000	10	5	192502	9565	440244
1e-05	9.9e-07	5000	10	10	881564598	9252	1076124
1e-05	9.9e-07	5000	100	1	874606883	9537	99374
1e-05	9.9e-07	5000	100	5	195201	9310	523227
1e-05	9.9e-07	5000	100	10	874607861	9209	1100038
1e-05	9.9e-06	50	1	1	886725458	15420	1320
1e-05	9.9e-06	50	1	5	389202	14929	5575
1e-05	9.9e-06	50	1	10	886725564	14644	12344
1e-05	9.9e-06	50	10	1	887346124	13451	4926
1e-05	9.9e-06	50	10	5	2003232	13619	21248
1e-05	9.9e-06	50	10	10	887358168	13224	47299

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-06	50	100	1	887665710	11471	19000
1e-05	9.9e-06	50	100	5	569002	11558	74330
1e-05	9.9e-06	50	100	10	887665952	11310	163678
1e-05	9.9e-06	500	1	1	873814378	11681	15259
1e-05	9.9e-06	500	1	5	2990202	10864	98088
1e-05	9.9e-06	500	1	10	874062743	10985	185035
1e-05	9.9e-06	500	10	1	880764928	10035	67571
1e-05	9.9e-06	500	10	5	20199356	9681	290964
1e-05	9.9e-06	500	10	10	880765117	9847	591863
1e-05	9.9e-06	500	100	1	874063107	9443	129647
1e-05	9.9e-06	500	100	5	3925021	9082	811176
1e-05	9.9e-06	500	100	10	874063628	8969	1527192
1e-05	9.9e-06	5000	1	1	874510862	9331	129400
1e-05	9.9e-06	5000	1	5	9879696	9133	825489
1e-05	9.9e-06	5000	1	10	874511083	8955	1699057
1e-05	9.9e-06	5000	10	1	880766936	8936	494096
1e-05	9.9e-06	5000	10	5	203033	8729	2279803
1e-05	9.9e-06	5000	10	10	881564668	8769	4742802
1e-05	9.9e-06	5000	100	1	874511298	8750	1610287
1e-05	9.9e-06	5000	100	5	30020	8600	7417918
1e-05	9.9e-06	5000	100	10	874511457	8699	17526170
0.0001	9.9e-07	50	1	1	886724943	15361	1303
0.0001	9.9e-07	50	1	5	792235	15286	3687
0.0001	9.9e-07	50	1	10	886907704	15150	10666
0.0001	9.9e-07	50	10	1	887336723	13605	5122
0.0001	9.9e-07	50	10	5	23590	13198	23516
0.0001	9.9e-07	50	10	10	887657099	13261	49109
0.0001	9.9e-07	50	100	1	887666301	13448	5589
0.0001	9.9e-07	50	100	5	490002	13289	25647
0.0001	9.9e-07	50	100	10	887666542	13073	55382
0.0001	9.9e-07	500	1	1	880967493	13035	7487
0.0001	9.9e-07	500	1	5	29002923	12346	42215
0.0001	9.9e-07	500	1	10	881622601	12315	86995
0.0001	9.9e-07	500	10	1	881605805	12685	8043
0.0001	9.9e-07	500	10	5	490998	12726	36535
0.0001	9.9e-07	500	10	10	881955440	12377	86157
0.0001	9.9e-07	500	100	1	881873116	12809	8406

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-07	500	100	5	4902	12569	40539
0.0001	9.9e-07	500	100	10	881841480	12220	82784
0.0001	9.9e-07	5000	1	1	881125810	12850	7990
0.0001	9.9e-07	5000	1	5	3945544	12507	41198
0.0001	9.9e-07	5000	1	10	886164787	12286	76813
0.0001	9.9e-07	5000	10	1	881125669	12957	7549
0.0001	9.9e-07	5000	10	5	202049	12421	39224
0.0001	9.9e-07	5000	10	10	881182805	12515	79701
0.0001	9.9e-07	5000	100	1	881706720	12762	9304
0.0001	9.9e-07	5000	100	5	2355440	12410	43903
0.0001	9.9e-07	5000	100	10	881655384	12239	87540
0.0001	9.9e-06	50	1	1	886724652	17243	51
0.0001	9.9e-06	50	1	5	90202	14992	4815
0.0001	9.9e-06	50	1	10	886724741	15102	8445
0.0001	9.9e-06	50	10	1	887336625	13548	4757
0.0001	9.9e-06	50	10	5	708960	13435	22063
0.0001	9.9e-06	50	10	10	887407929	12937	48635
0.0001	9.9e-06	50	100	1	887408020	12194	12724
0.0001	9.9e-06	50	100	5	10250	11855	62860
0.0001	9.9e-06	50	100	10	887408209	11834	124636
0.0001	9.9e-06	500	1	1	873578187	11509	16871
0.0001	9.9e-06	500	1	5	5970	11643	79002
0.0001	9.9e-06	500	1	10	873713996	11266	192641
0.0001	9.9e-06	500	10	1	880490114	10159	57857
0.0001	9.9e-06	500	10	5	700303	9774	323993
0.0001	9.9e-06	500	10	10	880744553	9930	549509
0.0001	9.9e-06	500	100	1	873714266	10113	60267
0.0001	9.9e-06	500	100	5	56005	9929	304010
0.0001	9.9e-06	500	100	10	873716922	9786	591059
0.0001	9.9e-06	5000	1	1	873733263	9307	123216
0.0001	9.9e-06	5000	1	5	740223	9600	381143
0.0001	9.9e-06	5000	1	10	873734067	9399	1087212
0.0001	9.9e-06	5000	10	1	880060344	9456	115189
0.0001	9.9e-06	5000	10	5	30290	9519	445888
0.0001	9.9e-06	5000	10	10	880060455	9339	1113046
0.0001	9.9e-06	5000	100	1	873734213	9553	107584
0.0001	9.9e-06	5000	100	5	90	9403	491745

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-06	5000	100	10	873745050	9384	1066949
0.0001	9.9e-05	50	1	1	886723866	17298	77
0.0001	9.9e-05	50	1	5	250	17121	467
0.0001	9.9e-05	50	1	10	886724012	16960	1375
0.0001	9.9e-05	50	10	1	886724107	16614	1766
0.0001	9.9e-05	50	10	5	965050	16583	5462
0.0001	9.9e-05	50	10	10	887336374	16090	11611
0.0001	9.9e-05	50	100	1	887344886	11587	20935
0.0001	9.9e-05	50	100	5	3590	11629	85833
0.0001	9.9e-05	50	100	10	887406244	11163	213086
0.0001	9.9e-05	500	1	1	874613103	16902	1487
0.0001	9.9e-05	500	1	5	56902	16722	6764
0.0001	9.9e-05	500	1	10	873242801	16434	13619
0.0001	9.9e-05	500	10	1	880389097	9654	91776
0.0001	9.9e-05	500	10	5	235900	9740	379707
0.0001	9.9e-05	500	10	10	880389203	9620	761395
0.0001	9.9e-05	500	100	1	873243021	8930	226312
0.0001	9.9e-05	500	100	5	820005	9046	853848
0.0001	9.9e-05	500	100	10	873571228	9058	1720500
0.0001	9.9e-05	5000	1	1	873245061	9100	213267
0.0001	9.9e-05	5000	1	5	35930	15925	52923
0.0001	9.9e-05	5000	1	10	873247079	9098	693825
0.0001	9.9e-05	5000	10	1	880389652	8943	500677
0.0001	9.9e-05	5000	10	5	1039	8784	2693989
0.0001	9.9e-05	5000	10	10	881564812	8795	4999888
0.0001	9.9e-05	5000	100	1	873710406	8689	1577983
0.0001	9.9e-05	5000	100	5	1009320	8664	6693952
0.0001	9.9e-05	5000	100	10	873297104	8559	17401540
0.001	9.9e-06	50	1	1	886717062	17292	51
0.001	9.9e-06	50	1	5	40339	15299	2672
0.001	9.9e-06	50	1	10	200331	14870	6823
0.001	9.9e-06	50	10	1	887398316	13422	4867
0.001	9.9e-06	50	10	5	30228	13267	26094
0.001	9.9e-06	50	10	10	887398193	13444	44248
0.001	9.9e-06	50	100	1	887398446	13442	5011
0.001	9.9e-06	50	100	5	100403	13292	28104
0.001	9.9e-06	50	100	10	887656624	13330	51196

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	9.9e-06	500	1	1	880964666	12900	9489
0.001	9.9e-06	500	1	5	697007	12457	34292
0.001	9.9e-06	500	1	10	880965007	12396	65558
0.001	9.9e-06	500	10	1	880977601	12947	7605
0.001	9.9e-06	500	10	5	3900723	12338	45247
0.001	9.9e-06	500	10	10	880965104	12374	85777
0.001	9.9e-06	500	100	1	880988400	12428	10704
0.001	9.9e-06	500	100	5	3927461	12106	47599
0.001	9.9e-06	500	100	10	880999201	12304	89949
0.001	9.9e-06	5000	1	1	881125432	12410	9598
0.001	9.9e-06	5000	1	5	20041	12422	46475
0.001	9.9e-06	5000	1	10	881125502	12462	86872
0.001	9.9e-06	5000	10	1	881128135	12535	9580
0.001	9.9e-06	5000	10	5	97961	12566	41434
0.001	9.9e-06	5000	10	10	881157601	12388	92330
0.001	9.9e-06	5000	100	1	881175601	12481	9564
0.001	9.9e-06	5000	100	5	300791	12347	49591
0.001	9.9e-06	5000	100	10	881146801	12338	95534
0.001	9.9e-05	50	1	1	886714268	17260	62
0.001	9.9e-05	50	1	5	309790	15132	2352
0.001	9.9e-05	50	1	10	886714438	15271	4651
0.001	9.9e-05	50	10	1	886714355	13582	4274
0.001	9.9e-05	50	10	5	300793	13126	25572
0.001	9.9e-05	50	10	10	886714537	12951	52758
0.001	9.9e-05	50	100	1	886714654	11853	13477
0.001	9.9e-05	50	100	5	3004922	11838	67779
0.001	9.9e-05	50	100	10	886714762	11834	129629
0.001	9.9e-05	500	1	1	876344489	11204	23374
0.001	9.9e-05	500	1	5	208949	11409	55547
0.001	9.9e-05	500	1	10	876344599	11244	110987
0.001	9.9e-05	500	10	1	879783000	10193	54302
0.001	9.9e-05	500	10	5	200634	10002	265632
0.001	9.9e-05	500	10	10	879783118	9667	574770
0.001	9.9e-05	500	100	1	876344722	9949	63157
0.001	9.9e-05	500	100	5	183488272	9740	317403
0.001	9.9e-05	500	100	10	876344891	9781	642465
0.001	9.9e-05	5000	1	1	878532107	9585	87457

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	9.9e-05	5000	1	5	975810	9446	533606
0.001	9.9e-05	5000	1	10	879946906	9352	970078
0.001	9.9e-05	5000	10	1	879947103	9530	131476
0.001	9.9e-05	5000	10	5	300218	9487	534037
0.001	9.9e-05	5000	10	10	879947217	9388	957467
0.001	9.9e-05	5000	100	1	878532761	9279	118307
0.001	9.9e-05	5000	100	5	37773	9356	508463
0.001	9.9e-05	5000	100	10	878532284	9364	1045949
0.001	0.00099	50	1	1	886711330	17213	51
0.001	0.00099	50	1	5	25920	17105	317
0.001	0.00099	50	1	10	886711451	16965	759
0.001	0.00099	50	10	1	886711543	17126	522
0.001	0.00099	50	10	5	39042	17049	2795
0.001	0.00099	50	10	10	886711646	17109	6974
0.001	0.00099	50	100	1	886711838	17145	5001
0.001	0.00099	50	100	5	900232	17056	29231
0.001	0.00099	50	100	10	887345323	11645	108419
0.001	0.00099	500	1	1	876180825	17280	525
0.001	0.00099	500	1	5	1390	17124	2917
0.001	0.00099	500	1	10	876181280	17118	6196
0.001	0.00099	500	10	1	878683090	17114	10947
0.001	0.00099	500	10	5	102409	16960	28861
0.001	0.00099	500	10	10	879273609	16986	65742
0.001	0.00099	500	100	1	876182404	16875	76532
0.001	0.00099	500	100	5	1003273	9151	1266326
0.001	0.00099	500	100	10	877551799	9003	2012878
0.001	0.00099	5000	1	1	876250082	17056	6163
0.001	0.00099	5000	1	5	54334	17039	30643
0.001	0.00099	5000	1	10	876250237	17049	74960
0.001	0.00099	5000	10	1	878691052	16995	76108
0.001	0.00099	5000	10	5	364	17036	284408
0.001	0.00099	5000	10	10	878684900	16965	630824
0.001	0.00099	5000	100	1	877551950	8720	2528484
0.001	0.00099	5000	100	5	58958	8606	9025517
0.001	0.00099	5000	100	10	*	8562	21484373
0.01	9.9e-05	50	1	1	887778744	17383	51
0.01	9.9e-05	50	1	5	340630	16193	891

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	9.9e-05	50	1	10	887778832	15104	4580
0.01	9.9e-05	50	10	1	887779334	13042	5343
0.01	9.9e-05	50	10	5	32020	13251	24888
0.01	9.9e-05	50	10	10	888289316	12902	46038
0.01	9.9e-05	50	100	1	888289853	13687	5205
0.01	9.9e-05	50	100	5	300100	13342	27687
0.01	9.9e-05	50	100	10	888289519	13107	51788
0.01	9.9e-05	500	1	1	887344994	12572	8115
0.01	9.9e-05	500	1	5	90030	12601	29643
0.01	9.9e-05	500	1	10	887657556	12380	77651
0.01	9.9e-05	500	10	1	887657640	12534	9550
0.01	9.9e-05	500	10	5	90202	12729	37376
0.01	9.9e-05	500	10	10	888297056	12152	97517
0.01	9.9e-05	500	100	1	888297141	12289	10564
0.01	9.9e-05	500	100	5	4020204	12345	45818
0.01	9.9e-05	500	100	10	888308409	12603	80859
0.01	9.9e-05	5000	1	1	9753	12271	10860
0.01	9.9e-05	5000	1	5	90071	12325	47940
0.01	9.9e-05	5000	1	10	72011	12443	92016
0.01	9.9e-05	5000	10	1	13460	12660	7229
0.01	9.9e-05	5000	10	5	393	12500	44930
0.01	9.9e-05	5000	10	10	214011	12169	102611
0.01	9.9e-05	5000	100	1	10989	12703	7569
0.01	9.9e-05	5000	100	5	93733	12534	57705
0.01	9.9e-05	5000	100	10	922311	12402	93461
0.01	0.00099	50	1	1	887340495	17264	51
0.01	0.00099	50	1	5	40020	15464	1867
0.01	0.00099	50	1	10	887340626	15164	3062
0.01	0.00099	50	10	1	887340725	13694	4507
0.01	0.00099	50	10	5	300064	12676	28459
0.01	0.00099	50	10	10	887340845	13099	45022
0.01	0.00099	50	100	1	888276667	12267	12647
0.01	0.00099	50	100	5	100039	11873	67836
0.01	0.00099	50	100	10	888276743	11950	126181
0.01	0.00099	500	1	1	888347397	11605	18180
0.01	0.00099	500	1	5	2004	11570	68211
0.01	0.00099	500	1	10	888366335	11397	148077

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.00099	500	10	1	888817610	10393	56691
0.01	0.00099	500	10	5	200053	9875	290243
0.01	0.00099	500	10	10	888363046	9978	525772
0.01	0.00099	500	100	1	888366609	9986	58423
0.01	0.00099	500	100	5	790045	9831	336168
0.01	0.00099	500	100	10	888817974	9820	676689
0.01	0.00099	5000	1	1	897681330	17186	5001
0.01	0.00099	5000	1	5	20203	9581	198265
0.01	0.00099	5000	1	10	7826	9585	427635
0.01	0.00099	5000	10	1	920	9858	78609
0.01	0.00099	5000	10	5	2968	9394	517280
0.01	0.00099	5000	10	10	9301521	9475	869505
0.01	0.00099	5000	100	1	897681064	9467	118333
0.01	0.00099	5000	100	5	272756	9454	548948
0.01	0.00099	5000	100	10	33722	9413	1033882
0.01	0.0099	50	1	1	887340231	17322	51
0.01	0.0099	50	1	5	39073	17130	290
0.01	0.0099	50	1	10	887340320	17122	762
0.01	0.0099	50	10	1	887340405	17123	953
0.01	0.0099	50	10	5	306340	17130	3182
0.01	0.0099	50	10	10	887340944	17031	6088
0.01	0.0099	50	100	1	888262239	17259	5001
0.01	0.0099	50	100	5	30069	17091	25658
0.01	0.0099	50	100	10	888017480	17071	58429
0.01	0.0099	500	1	1	892607769	17251	845
0.01	0.0099	500	1	5	200	17030	3240
0.01	0.0099	500	1	10	892608661	17132	6400
0.01	0.0099	500	10	1	892609261	17181	7256
0.01	0.0099	500	10	5	30470	17077	30692
0.01	0.0099	500	10	10	892960981	17023	66200
0.01	0.0099	500	100	1	892641542	17042	89871
0.01	0.0099	500	100	5	400230	17079	314936
0.01	0.0099	500	100	10	897331346	17022	679512
0.01	0.0099	5000	1	1	897836947	17036	5001
0.01	0.0099	5000	1	5	20332	17080	46262
0.01	0.0099	5000	1	10	702773	17096	81595
0.01	0.0099	5000	10	1	897706650	17070	79181

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.0099	5000	10	5	200898	17013	332566
0.01	0.0099	5000	10	10	7226645	16959	709189
0.01	0.0099	5000	100	1	9021	17020	572820
0.01	0.0099	5000	100	5	30493	16978	3417435
0.01	0.0099	5000	100	10	10032	16935	6608280
0.1	0.00099	50	1	1	887344244	16111	680
0.1	0.00099	50	1	5	10239	15818	1211
0.1	0.00099	50	1	10	887347904	15154	3178
0.1	0.00099	50	10	1	887662801	13383	4729
0.1	0.00099	50	10	5	240	13402	24301
0.1	0.00099	50	10	10	887748604	13401	45286
0.1	0.00099	50	100	1	887781344	13423	5288
0.1	0.00099	50	100	5	3920	13197	28830
0.1	0.00099	50	100	10	887781461	13125	54976
0.1	0.00099	500	1	1	892623602	17180	515
0.1	0.00099	500	1	5	302956	12585	19498
0.1	0.00099	500	1	10	892749181	12546	48823
0.1	0.00099	500	10	1	892785023	12690	8507
0.1	0.00099	500	10	5	4970003	12402	46553
0.1	0.00099	500	10	10	893960586	12261	95790
0.1	0.00099	500	100	1	894947690	12677	9045
0.1	0.00099	500	100	5	399063	12323	50075
0.1	0.00099	500	100	10	894963154	12433	80932
0.1	0.00099	5000	1	1	897496823	12627	8193
0.1	0.00099	5000	1	5	575444	12315	43464
0.1	0.00099	5000	1	10	897502006	12455	95525
0.1	0.00099	5000	10	1	897683629	12800	9379
0.1	0.00099	5000	10	5	10042	12108	57154
0.1	0.00099	5000	10	10	897682479	12369	124906
0.1	0.00099	5000	100	1	897496644	12612	9086
0.1	0.00099	5000	100	5	1337	12537	63043
0.1	0.00099	5000	100	10	897501891	12445	108516
0.1	0.0099	50	1	1	887338678	17340	51
0.1	0.0099	50	1	5	20004	17112	326
0.1	0.0099	50	1	10	887338930	15465	1812
0.1	0.0099	50	10	1	887340526	13360	5993
0.1	0.0099	50	10	5	395	13177	25809

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.0099	50	10	10	887666401	13184	49090
0.1	0.0099	50	100	1	887742729	12080	12174
0.1	0.0099	50	100	5	90030	11911	66466
0.1	0.0099	50	100	10	887745417	11759	131298
0.1	0.0099	500	1	1	887751432	17199	501
0.1	0.0099	500	1	5	910289	11051	54731
0.1	0.0099	500	1	10	887750734	11148	93876
0.1	0.0099	500	10	1	887945991	9733	64693
0.1	0.0099	500	10	5	9000233	9946	260792
0.1	0.0099	500	10	10	887828063	9877	538683
0.1	0.0099	500	100	1	888258670	9938	63521
0.1	0.0099	500	100	5	2059004	9869	293139
0.1	0.0099	500	100	10	888259955	9848	612752
0.1	0.0099	5000	1	1	23441	9570	101253
0.1	0.0099	5000	1	5	2	9595	412913
0.1	0.0099	5000	1	10	1007	9555	654943
0.1	0.0099	5000	10	1	70020	9389	143951
0.1	0.0099	5000	10	5	7782700	9164	618124
0.1	0.0099	5000	10	10	93302	9235	1174934
0.1	0.0099	5000	100	1	90021	9411	112796
0.1	0.0099	5000	100	5	200893	9248	549605
0.1	0.0099	5000	100	10	1033329	9205	1190923
0.1	0.099	50	1	1	887338301	17289	102
0.1	0.099	50	1	5	37362	17017	262
0.1	0.099	50	1	10	887338211	17123	679
0.1	0.099	50	10	1	887338438	17180	501
0.1	0.099	50	10	5	594	17195	2828
0.1	0.099	50	10	10	887338534	17056	5968
0.1	0.099	50	100	1	887660264	17107	5001
0.1	0.099	50	100	5	503932	17031	33003
0.1	0.099	50	100	10	887705010	17026	60851
0.1	0.099	500	1	1	892768869	17150	501
0.1	0.099	500	1	5	293	17028	3709
0.1	0.099	500	1	10	893961643	17100	6500
0.1	0.099	500	10	1	894301118	17094	9639
0.1	0.099	500	10	5	39906	17148	31980
0.1	0.099	500	10	10	894927721	17068	64826

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.099	500	100	1	894992401	16986	99265
0.1	0.099	500	100	5	39003	17027	396551
0.1	0.099	500	100	10	895161610	17043	591149
0.1	0.099	5000	1	1	2342348	17114	8107
0.1	0.099	5000	1	5	300872	17055	48199
0.1	0.099	5000	1	10	12752	17091	76050
0.1	0.099	5000	10	1	5190	16924	88881
0.1	0.099	5000	10	5	907	16992	516146
0.1	0.099	5000	10	10	302278	17030	777797
0.1	0.099	5000	100	1	7015	17022	504060
0.1	0.099	5000	100	5	23	16975	2810154
0.1	0.099	5000	100	10	73301	16960	7520392
1	0.0099	50	1	1	887337782	17320	56
1	0.0099	50	1	5	90969	17192	340
1	0.0099	50	1	10	887337891	17136	562
1	0.0099	50	10	1	887338015	13653	4738
1	0.0099	50	10	5	34602	13433	21336
1	0.0099	50	10	10	887338105	13268	45290
1	0.0099	50	100	1	887658233	13490	5115
1	0.0099	50	100	5	38129	13254	27866
1	0.0099	50	100	10	887704911	13118	55438
1	0.0099	500	1	1	892732921	12827	8937
1	0.0099	500	1	1	892755451	17286	516
1	0.0099	500	1	5	9037372	12423	35044
1	0.0099	500	1	10	892625859	12681	49472
1	0.0099	500	1	10	892758462	12456	52160
1	0.0099	500	10	1	892758736	12842	9393
1	0.0099	500	10	5	9803	12429	47301
1	0.0099	500	10	10	2096561	12479	81722
1	0.0099	500	100	1	892771051	12538	9223
1	0.0099	500	100	5	9000293	12419	49437
1	0.0099	500	100	10	892774994	12443	86792
1	0.0099	5000	1	1	103744	12665	7910
1	0.0099	5000	1	5	99686	12442	38154
1	0.0099	5000	1	10	4234	12379	76552
1	0.0099	5000	10	1	93015	12708	10351
1	0.0099	5000	10	5	373	12467	63882

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.0099	5000	10	10	920222	12207	118955
1	0.0099	5000	100	1	10033546	13108	7956
1	0.0099	5000	100	5	2673566	12364	56550
1	0.0099	5000	100	10	10203	12687	89309
1	0.099	50	1	1	887057478	17396	51
1	0.099	50	1	5	3061	17148	372
1	0.099	50	1	10	887057563	17140	675
1	0.099	50	10	1	887337639	13518	4661
1	0.099	50	10	5	4091001	13040	27646
1	0.099	50	10	10	887340026	12866	54636
1	0.099	50	100	1	887650449	12216	13079
1	0.099	50	100	5	30194	11760	67006
1	0.099	50	100	10	887650675	11708	134803
1	0.099	500	1	5	5045	17157	3202
1	0.099	500	10	1	892830182	9998	59906
1	0.099	500	10	5	70399	9786	301899
1	0.099	500	10	10	893779206	9857	571365
1	0.099	500	100	1	892663685	10174	63025
1	0.099	500	100	5	6239	9826	354689
1	0.099	500	100	10	894352320	9693	689519
1	0.099	5000	1	1	897834207	10061	67599
1	0.099	5000	1	5	300277	9176	334814
1	0.099	5000	1	10	897834125	9511	376527
1	0.099	5000	10	1	898611227	9653	104182
1	0.099	5000	10	5	33729	9363	604314
1	0.099	5000	10	10	732	9410	1194553
1	0.099	5000	100	1	481	9477	104276
1	0.099	5000	100	5	9872	9589	439533
1	0.099	5000	100	10	11923688	9541	957439
1	0.99	50	1	1	887056686	17282	64
1	0.99	50	1	5	3887	17104	444
1	0.99	50	1	10	887056841	17199	572
1	0.99	50	10	1	887057224	17038	554
1	0.99	50	10	5	20098	17005	2926
1	0.99	50	10	10	887337529	17128	5650
1	0.99	50	100	1	887394862	17221	5020
1	0.99	50	100	5	203345	17135	30918

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.99	50	100	10	887400004	17044	59422
1	0.99	500	1	1	887056935	17272	563
1	0.99	500	1	5	390002	17083	2948
1	0.99	500	1	10	887057364	17151	7158
1	0.99	500	10	1	887339938	17174	5001
1	0.99	500	10	5	39920	17107	32937
1	0.99	500	10	10	887648273	17017	72228
1	0.99	500	100	1	887651309	17081	80802
1	0.99	500	100	5	3023	17041	367673
1	0.99	500	100	10	887653827	17014	583623
1	0.99	5000	1	1	898373464	17133	9710
1	0.99	5000	1	5	30402	17147	50855
1	0.99	5000	1	10	898388196	16990	75635
1	0.99	5000	10	1	898396986	17052	80949
1	0.99	5000	10	5	30023	16876	390530
1	0.99	5000	10	10	899820313	17044	714975
1	0.99	5000	100	1	20445	17078	561497
1	0.99	5000	100	5	99697	16969	3103518
1	0.99	5000	100	10	899887040	16957	7864005

B.4 General Data Matrix, maxaccept = 50

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	50	1	1	886907009	17194	51
1e-05	9.9e-08	50	1	5	1900103	16933	255
1e-05	9.9e-08	50	1	10	886907093	16918	510
1e-05	9.9e-08	50	10	1	886907332	16266	501
1e-05	9.9e-08	50	10	5	190032	16167	2505
1e-05	9.9e-08	50	10	10	887337143	16001	5010
1e-05	9.9e-08	50	100	1	887344541	13489	5001
1e-05	9.9e-08	50	100	5	100149	13344	25005
1e-05	9.9e-08	50	100	10	887344520	13111	50010
1e-05	9.9e-08	500	1	1	881127635	14738	2329
1e-05	9.9e-08	500	1	5	975021	14250	12491
1e-05	9.9e-08	500	1	10	881128588	14019	28351
1e-05	9.9e-08	500	10	1	2399001	12545	8646
1e-05	9.9e-08	500	10	5	195320	12610	36460

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-08	500	10	10	104592002	12331	76136
1e-05	9.9e-08	500	100	1	881956501	10142	51776
1e-05	9.9e-08	500	100	5	1131	10003	260727
1e-05	9.9e-08	500	100	10	881652660	10097	524619
1e-05	9.9e-08	5000	1	1	881609401	10620	34679
1e-05	9.9e-08	5000	1	5	300121125	10343	188412
1e-05	9.9e-08	5000	1	10	888297314	10396	366179
1e-05	9.9e-08	5000	10	1	881692205	9953	79994
1e-05	9.9e-08	5000	10	5	803048	9593	416757
1e-05	9.9e-08	5000	10	10	881956815	9508	836295
1e-05	9.9e-08	5000	100	1	885594753	8874	543469
1e-05	9.9e-08	5000	100	5	39068	8691	2686868
1e-05	9.9e-08	5000	100	10	885342781	8625	5357521
1e-05	9.9e-07	50	1	1	887382774	16994	51
1e-05	9.9e-07	50	1	5	399065	17130	255
1e-05	9.9e-07	50	1	10	887396405	17022	510
1e-05	9.9e-07	50	10	1	887396405	16291	501
1e-05	9.9e-07	50	10	5	29140	16152	2505
1e-05	9.9e-07	50	10	10	887396405	15941	5010
1e-05	9.9e-07	50	100	1	887380211	13648	5001
1e-05	9.9e-07	50	100	5	1940	13253	25005
1e-05	9.9e-07	50	100	10	887421601	13200	50010
1e-05	9.9e-07	500	1	1	874514376	13971	3154
1e-05	9.9e-07	500	1	5	1300212	13678	14156
1e-05	9.9e-07	500	1	10	874603835	14118	27603
1e-05	9.9e-07	500	10	1	880942329	12769	7105
1e-05	9.9e-07	500	10	5	1050123	12464	37427
1e-05	9.9e-07	500	10	10	880765758	12477	74616
1e-05	9.9e-07	500	100	1	874632400	10254	52173
1e-05	9.9e-07	500	100	5	1994101	10175	263172
1e-05	9.9e-07	500	100	10	874603990	9986	524337
1e-05	9.9e-07	5000	1	1	874605824	10863	31915
1e-05	9.9e-07	5000	1	5	2592301	10523	185502
1e-05	9.9e-07	5000	1	10	874607989	10195	373251
1e-05	9.9e-07	5000	10	1	880877858	9682	78203
1e-05	9.9e-07	5000	10	5	1920465	9532	405627
1e-05	9.9e-07	5000	10	10	20049201	9426	853133

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1e-05	9.9e-07	5000	100	1	874631119	8854	529663
1e-05	9.9e-07	5000	100	5	3436003	8658	2667156
1e-05	9.9e-07	5000	100	10	874607172	8725	5350396
1e-05	9.9e-06	50	1	1	887665518	17259	51
1e-05	9.9e-06	50	1	5	10011133	17002	255
1e-05	9.9e-06	50	1	10	887665613	16991	510
1e-05	9.9e-06	50	10	1	887665922	16482	501
1e-05	9.9e-06	50	10	5	3996321	16129	2505
1e-05	9.9e-06	50	10	10	887666084	16170	5010
1e-05	9.9e-06	50	100	1	887771255	13720	5001
1e-05	9.9e-06	50	100	5	59002	12938	25005
1e-05	9.9e-06	50	100	10	887774033	13299	50010
1e-05	9.9e-06	500	1	1	874064759	14077	3070
1e-05	9.9e-06	500	1	5	2302	14013	12973
1e-05	9.9e-06	500	1	10	874509977	13677	27309
1e-05	9.9e-06	500	10	1	880766778	12750	7733
1e-05	9.9e-06	500	10	5	3938699	12505	40436
1e-05	9.9e-06	500	10	10	880765346	12697	75270
1e-05	9.9e-06	500	100	1	874510197	10136	52168
1e-05	9.9e-06	500	100	5	19247001	10020	263934
1e-05	9.9e-06	500	100	10	874510486	9968	525064
1e-05	9.9e-06	5000	1	1	874511650	10373	46998
1e-05	9.9e-06	5000	1	5	2301112	10113	187126
1e-05	9.9e-06	5000	1	10	874511930	10155	377709
1e-05	9.9e-06	5000	10	1	880891201	9683	83395
1e-05	9.9e-06	5000	10	5	102002	9549	445622
1e-05	9.9e-06	5000	10	10	880765540	9546	889124
1e-05	9.9e-06	5000	100	1	874512106	8798	536445
1e-05	9.9e-06	5000	100	5	1935350	8714	2685398
1e-05	9.9e-06	5000	100	10	874961014	8790	5373850
0.0001	9.9e-07	50	1	1	887666437	17387	51
0.0001	9.9e-07	50	1	5	393012	17132	255
0.0001	9.9e-07	50	1	10	887771342	17153	510
0.0001	9.9e-07	50	10	1	887771475	16447	501
0.0001	9.9e-07	50	10	5	5980808	16022	2505
0.0001	9.9e-07	50	10	10	887773923	15982	5010
0.0001	9.9e-07	50	100	1	887773830	13436	5001

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-07	50	100	5	30059	13254	25005
0.0001	9.9e-07	50	100	10	887773753	13061	50010
0.0001	9.9e-07	500	1	1	881835960	14035	3378
0.0001	9.9e-07	500	1	5	29501	14230	14904
0.0001	9.9e-07	500	1	10	881906962	14295	13173
0.0001	9.9e-07	500	10	1	881856937	12970	7579
0.0001	9.9e-07	500	10	5	29990292	12687	39994
0.0001	9.9e-07	500	10	10	881683322	12413	79689
0.0001	9.9e-07	500	100	1	881837400	10287	52986
0.0001	9.9e-07	500	100	5	699904	10072	263790
0.0001	9.9e-07	500	100	10	881126248	9913	529029
0.0001	9.9e-07	5000	1	1	881957012	10275	45881
0.0001	9.9e-07	5000	1	5	324333	10215	176207
0.0001	9.9e-07	5000	1	10	881971200	10287	390777
0.0001	9.9e-07	5000	10	1	882130135	9819	87800
0.0001	9.9e-07	5000	10	5	23434	9583	421020
0.0001	9.9e-07	5000	10	10	886089758	9514	870339
0.0001	9.9e-07	5000	100	1	885346882	8840	536192
0.0001	9.9e-07	5000	100	5	302396	8668	2655420
0.0001	9.9e-07	5000	100	10	886091425	8738	5363351
0.0001	9.9e-06	50	1	1	887747187	17028	51
0.0001	9.9e-06	50	1	5	5002	17123	255
0.0001	9.9e-06	50	1	10	887747056	17045	510
0.0001	9.9e-06	50	10	1	887747280	16186	501
0.0001	9.9e-06	50	10	5	40057	16107	2505
0.0001	9.9e-06	50	10	10	887747367	16069	5010
0.0001	9.9e-06	50	100	1	888277806	13258	5001
0.0001	9.9e-06	50	100	5	2936002	13340	25005
0.0001	9.9e-06	50	100	10	887964764	13228	50010
0.0001	9.9e-06	500	1	1	873730935	14331	2738
0.0001	9.9e-06	500	1	5	200570	14321	7429
0.0001	9.9e-06	500	1	10	873732148	14156	15560
0.0001	9.9e-06	500	10	1	880744736	12735	7784
0.0001	9.9e-06	500	10	5	2904	12754	39466
0.0001	9.9e-06	500	10	10	880744707	12338	77429
0.0001	9.9e-06	500	100	1	874495115	10301	51897
0.0001	9.9e-06	500	100	5	30930	9969	263868

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.0001	9.9e-06	500	100	10	873732639	10041	530028
0.0001	9.9e-06	5000	1	1	873735387	10551	40503
0.0001	9.9e-06	5000	1	5	496033334	10403	179354
0.0001	9.9e-06	5000	1	10	873745461	10057	407541
0.0001	9.9e-06	5000	10	1	880231394	9565	84341
0.0001	9.9e-06	5000	10	5	20025	9631	429526
0.0001	9.9e-06	5000	10	10	880389008	9502	864386
0.0001	9.9e-06	5000	100	1	873812997	8824	530656
0.0001	9.9e-06	5000	100	5	235325	8857	2692213
0.0001	9.9e-06	5000	100	10	873813257	8743	5335814
0.0001	9.9e-05	50	1	1	887406346	17173	51
0.0001	9.9e-05	50	1	5	499040	17128	255
0.0001	9.9e-05	50	1	10	100342	16958	510
0.0001	9.9e-05	50	10	1	887406532	17056	501
0.0001	9.9e-05	50	10	5	334600	16562	2505
0.0001	9.9e-05	50	10	10	887406733	16758	5010
0.0001	9.9e-05	50	100	1	887406830	15768	5001
0.0001	9.9e-05	50	100	5	469003	15244	25005
0.0001	9.9e-05	50	100	10	887406911	15295	50010
0.0001	9.9e-05	500	1	1	873243537	16845	501
0.0001	9.9e-05	500	1	5	23060	16766	2888
0.0001	9.9e-05	500	1	10	873243709	16747	7056
0.0001	9.9e-05	500	10	1	880389346	16329	5299
0.0001	9.9e-05	500	10	5	2390	16451	25212
0.0001	9.9e-05	500	10	10	880389433	16386	59106
0.0001	9.9e-05	500	100	1	873244213	13517	50001
0.0001	9.9e-05	500	100	5	100404	12899	250869
0.0001	9.9e-05	500	100	10	873244445	11161	523375
0.0001	9.9e-05	5000	1	1	873391624	16513	7644
0.0001	9.9e-05	5000	1	5	109255	16273	36752
0.0001	9.9e-05	5000	1	10	873393293	16370	72533
0.0001	9.9e-05	5000	10	1	880392831	16245	51953
0.0001	9.9e-05	5000	10	5	1052	16013	266078
0.0001	9.9e-05	5000	10	10	880396086	15921	570901
0.0001	9.9e-05	5000	100	1	873398146	8709	968197
0.0001	9.9e-05	5000	100	5	803292000	8625	4794068
0.0001	9.9e-05	5000	100	10	874494856	8621	9755130

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	9.9e-06	50	1	1	887656774	17297	51
0.001	9.9e-06	50	1	5	100293	17135	255
0.001	9.9e-06	50	1	10	887657197	17106	510
0.001	9.9e-06	50	10	1	887657302	16323	501
0.001	9.9e-06	50	10	5	20034	16120	2505
0.001	9.9e-06	50	10	10	887657402	16076	5010
0.001	9.9e-06	50	100	1	887964646	13280	5001
0.001	9.9e-06	50	100	5	203	13357	25005
0.001	9.9e-06	50	100	10	887964557	13304	50010
0.001	9.9e-06	500	1	1	881020800	17148	501
0.001	9.9e-06	500	1	5	2362001	17036	2505
0.001	9.9e-06	500	1	10	881013601	17087	5010
0.001	9.9e-06	500	10	1	881031601	13254	7370
0.001	9.9e-06	500	10	5	300259	12552	40280
0.001	9.9e-06	500	10	10	881565218	12734	81618
0.001	9.9e-06	500	100	1	881044200	10179	53744
0.001	9.9e-06	500	100	5	102004	10023	265457
0.001	9.9e-06	500	100	10	881060400	9971	528672
0.001	9.9e-06	5000	1	1	881575204	17133	5001
0.001	9.9e-06	5000	1	5	272622	10732	56245
0.001	9.9e-06	5000	1	10	881586000	10285	92986
0.001	9.9e-06	5000	10	1	881614801	9608	100528
0.001	9.9e-06	5000	10	5	105871	9651	437710
0.001	9.9e-06	5000	10	10	882043057	9610	890687
0.001	9.9e-06	5000	100	1	881859638	8879	540547
0.001	9.9e-06	5000	100	5	100369	8888	2695451
0.001	9.9e-06	5000	100	10	881172000	8819	5390627
0.001	9.9e-05	50	1	1	886714944	17292	51
0.001	9.9e-05	50	1	5	100539	17207	255
0.001	9.9e-05	50	1	10	886715035	17175	510
0.001	9.9e-05	50	10	1	886715221	16512	501
0.001	9.9e-05	50	10	5	292490	16323	2505
0.001	9.9e-05	50	10	10	886715332	16198	5010
0.001	9.9e-05	50	100	1	887388502	13635	5001
0.001	9.9e-05	50	100	5	45936800	13328	25005
0.001	9.9e-05	50	100	10	887358333	13230	35288
0.001	9.9e-05	500	1	1	876346055	17265	501

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	9.9e-05	500	1	5	39502	17002	2724
0.001	9.9e-05	500	1	10	876346219	17181	5010
0.001	9.9e-05	500	10	1	879274077	12821	8204
0.001	9.9e-05	500	10	5	3936013	12562	41088
0.001	9.9e-05	500	10	10	879946720	12557	84448
0.001	9.9e-05	500	100	1	876346365	10331	52953
0.001	9.9e-05	500	100	5	100382385	10105	266728
0.001	9.9e-05	500	100	10	876346711	9992	533154
0.001	9.9e-05	5000	1	1	878528648	17177	5001
0.001	9.9e-05	5000	1	5	30065	17056	25005
0.001	9.9e-05	5000	1	10	878532933	10426	90888
0.001	9.9e-05	5000	10	1	879947797	9792	90021
0.001	9.9e-05	5000	10	5	36626	9658	461376
0.001	9.9e-05	5000	10	10	879956483	9557	927510
0.001	9.9e-05	5000	100	1	878533131	8959	571142
0.001	9.9e-05	5000	100	5	240111	8743	2726622
0.001	9.9e-05	5000	100	10	878533253	8737	5411430
0.001	0.00099	50	1	1	886711750	17319	51
0.001	0.00099	50	1	5	249202	17137	255
0.001	0.00099	50	1	10	886712040	17160	510
0.001	0.00099	50	10	1	886712435	17152	501
0.001	0.00099	50	10	5	1000493	17230	2505
0.001	0.00099	50	10	10	886712731	17143	5010
0.001	0.00099	50	100	1	887345208	17148	5001
0.001	0.00099	50	100	5	90340	17012	25005
0.001	0.00099	50	100	10	886713299	16958	50010
0.001	0.00099	500	1	1	876182778	17256	501
0.001	0.00099	500	1	5	29600	17116	2929
0.001	0.00099	500	1	10	876182951	17071	5010
0.001	0.00099	500	10	1	879273735	17034	5001
0.001	0.00099	500	10	5	2006	17079	25005
0.001	0.00099	500	10	10	879780467	16980	50432
0.001	0.00099	500	100	1	876183144	17069	50001
0.001	0.00099	500	100	5	4868709	16923	250005
0.001	0.00099	500	100	10	876249952	16886	500010
0.001	0.00099	5000	1	1	876250630	17112	5001
0.001	0.00099	5000	1	5	73004	17031	25005

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.001	0.00099	5000	1	10	876250747	17011	51046
0.001	0.00099	5000	10	1	879782086	17106	50001
0.001	0.00099	5000	10	5	40032	17009	250005
0.001	0.00099	5000	10	10	878691205	17001	500010
0.001	0.00099	5000	100	1	876344220	16807	500001
0.001	0.00099	5000	100	5	1023	16896	2500005
0.001	0.00099	5000	100	10	876344361	16802	5001169
0.01	9.9e-05	50	1	1	888289405	17108	51
0.01	9.9e-05	50	1	5	140	17043	255
0.01	9.9e-05	50	1	10	888289607	17119	510
0.01	9.9e-05	50	10	1	888289737	16442	501
0.01	9.9e-05	50	10	5	23590	16255	2505
0.01	9.9e-05	50	10	10	888290165	16093	5010
0.01	9.9e-05	50	100	1	888290276	13810	5001
0.01	9.9e-05	50	100	5	24020	13385	25005
0.01	9.9e-05	50	100	10	888351784	13274	50010
0.01	9.9e-05	500	1	1	888347595	17151	501
0.01	9.9e-05	500	1	5	23590	17104	2505
0.01	9.9e-05	500	1	10	888817700	17077	5010
0.01	9.9e-05	500	10	1	888815790	13007	7935
0.01	9.9e-05	500	10	5	390	12731	41857
0.01	9.9e-05	500	10	10	888311140	12630	80670
0.01	9.9e-05	500	100	1	888339600	10281	53624
0.01	9.9e-05	500	100	5	293353	10025	266907
0.01	9.9e-05	500	100	10	888354000	9940	535054
0.01	9.9e-05	5000	1	5	10043	17117	25005
0.01	9.9e-05	5000	1	10	83013	17001	50010
0.01	9.9e-05	5000	10	1	897838592	9786	97407
0.01	9.9e-05	5000	10	5	9904	9517	463263
0.01	9.9e-05	5000	10	10	302207	9545	922240
0.01	9.9e-05	5000	100	1	60114	8939	531908
0.01	9.9e-05	5000	100	5	90003	8770	2730184
0.01	9.9e-05	5000	100	10	91172	8694	5429320
0.01	0.00099	50	1	1	888284027	17134	51
0.01	0.00099	50	1	5	20050	17111	255
0.01	0.00099	50	1	10	888277262	17223	510
0.01	0.00099	50	10	1	888277404	16603	501

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.00099	50	10	5	390052	16144	2505
0.01	0.00099	50	10	10	888359703	16334	5010
0.01	0.00099	50	100	1	888277651	13164	5001
0.01	0.00099	50	100	5	68686	13383	25005
0.01	0.00099	50	100	10	888277018	13264	50010
0.01	0.00099	500	1	1	888837738	17145	501
0.01	0.00099	500	1	5	20062	17117	2505
0.01	0.00099	500	1	10	888846079	17036	5010
0.01	0.00099	500	10	1	888856388	13209	9078
0.01	0.00099	500	10	5	20203	12599	43913
0.01	0.00099	500	10	10	888858634	12504	91409
0.01	0.00099	500	100	1	894221461	10099	53829
0.01	0.00099	500	100	5	39222	10111	269455
0.01	0.00099	500	100	10	896221442	10169	536499
0.01	0.00099	5000	1	1	897496949	17120	5001
0.01	0.00099	5000	1	5	93478	17069	25005
0.01	0.00099	5000	1	10	897501777	17100	50010
0.01	0.00099	5000	10	1	897712433	9833	86665
0.01	0.00099	5000	10	5	9090	9572	478620
0.01	0.00099	5000	10	10	5335	9503	979072
0.01	0.00099	5000	100	1	24	8992	542685
0.01	0.00099	5000	100	5	90290	8825	2718453
0.01	0.00099	5000	100	10	9902	8687	5457812
0.01	0.0099	50	1	1	888260249	17353	51
0.01	0.0099	50	1	5	20059	17132	255
0.01	0.0099	50	1	10	888260415	17194	510
0.01	0.0099	50	10	1	888276583	17309	501
0.01	0.0099	50	10	5	30070	17113	2505
0.01	0.0099	50	10	10	888261019	16959	5010
0.01	0.0099	50	100	1	888260628	17212	5001
0.01	0.0099	50	100	5	56948	17060	25005
0.01	0.0099	50	100	10	888260538	17074	50010
0.01	0.0099	500	1	1	892624562	17258	501
0.01	0.0099	500	1	5	902261	17115	2505
0.01	0.0099	500	1	10	892954081	17039	5010
0.01	0.0099	500	10	1	892632362	17189	5001
0.01	0.0099	500	10	5	400312	17071	25005

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.01	0.0099	500	10	10	892639622	17064	50010
0.01	0.0099	500	100	1	892807181	17104	50001
0.01	0.0099	500	100	5	304432	16913	250005
0.01	0.0099	500	100	10	895961822	17012	500010
0.01	0.0099	5000	1	1	82110	17177	5001
0.01	0.0099	5000	1	1	900434	17148	5001
0.01	0.0099	5000	1	5	3933	17072	25005
0.01	0.0099	5000	1	10	9001323	17059	50010
0.01	0.0099	5000	10	1	300043	17073	50001
0.01	0.0099	5000	10	5	476466	16982	250005
0.01	0.0099	5000	10	10	30022312	17007	500010
0.01	0.0099	5000	100	1	937772	17008	500001
0.01	0.0099	5000	100	5	97898	16939	2500005
0.01	0.0099	5000	100	10	1004	16894	5000010
0.1	0.00099	50	1	1	887781262	17195	51
0.1	0.00099	50	1	5	350303	17112	255
0.1	0.00099	50	1	10	887781537	17063	510
0.1	0.00099	50	10	1	887782971	16310	501
0.1	0.00099	50	10	5	249929	16230	2505
0.1	0.00099	50	10	10	887783231	16178	5010
0.1	0.00099	50	100	1	887946081	13786	5001
0.1	0.00099	50	100	5	3615	13487	25005
0.1	0.00099	50	100	10	887947328	13191	50010
0.1	0.00099	500	1	1	894370615	17238	501
0.1	0.00099	500	1	5	499967	17191	2505
0.1	0.00099	500	1	10	894587892	17118	5010
0.1	0.00099	500	10	1	894587809	13007	8379
0.1	0.00099	500	10	5	100593	12720	43132
0.1	0.00099	500	10	10	894588012	12708	82291
0.1	0.00099	500	100	1	894946594	10303	53183
0.1	0.00099	500	100	5	392	10041	266582
0.1	0.00099	500	100	10	898361840	10078	534476
0.1	0.00099	5000	1	1	897439184	17172	5001
0.1	0.00099	5000	1	5	100043	17050	28605
0.1	0.00099	5000	1	10	897443231	17025	50010
0.1	0.00099	5000	10	1	897496093	9788	90415
0.1	0.00099	5000	10	5	29490	9466	470082

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.00099	5000	10	10	897438786	9491	991506
0.1	0.00099	5000	100	1	898062142	9020	534971
0.1	0.00099	5000	100	5	738830	8770	2732947
0.1	0.00099	5000	100	10	823073	8767	5446426
0.1	0.0099	50	1	1	887743142	17347	51
0.1	0.0099	50	1	5	20482	17101	255
0.1	0.0099	50	1	10	887744961	17078	510
0.1	0.0099	50	10	1	887745047	16738	501
0.1	0.0099	50	10	5	47261	16476	2505
0.1	0.0099	50	10	10	887747543	16341	5010
0.1	0.0099	50	100	1	887748269	13401	5001
0.1	0.0099	50	100	5	9686	13345	25005
0.1	0.0099	50	100	10	887748496	13151	50010
0.1	0.0099	500	1	1	888260077	17131	501
0.1	0.0099	500	1	5	102847222	17159	2505
0.1	0.0099	500	1	10	888264195	17063	5010
0.1	0.0099	500	10	1	888263407	12839	8530
0.1	0.0099	500	10	5	10392	12868	46022
0.1	0.0099	500	10	10	888310301	12630	96462
0.1	0.0099	500	100	1	888276484	10317	53931
0.1	0.0099	500	100	5	923411	10049	270484
0.1	0.0099	500	100	10	888017391	10061	543178
0.1	0.0099	5000	1	1	701472	17164	5001
0.1	0.0099	5000	1	5	1005777	17145	25467
0.1	0.0099	5000	1	10	427744	17009	50010
0.1	0.0099	5000	10	1	100263	9754	95972
0.1	0.0099	5000	10	5	58839	9582	515462
0.1	0.0099	5000	10	10	712	9474	1027356
0.1	0.0099	5000	100	1	289457	8856	553685
0.1	0.0099	5000	100	5	2204932	8792	2737979
0.1	0.0099	5000	100	10	109938	8776	5522970
0.1	0.099	50	1	1	887705178	17238	51
0.1	0.099	50	1	5	30921	17111	255
0.1	0.099	50	1	10	887705250	17135	510
0.1	0.099	50	10	1	887705427	17148	501
0.1	0.099	50	10	5	49670	17194	2505
0.1	0.099	50	10	10	887705638	17034	5010

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
0.1	0.099	50	100	1	887707023	17173	5001
0.1	0.099	50	100	5	9400220	17130	25005
0.1	0.099	50	100	10	887715474	16922	50010
0.1	0.099	500	1	1	894301459	17257	501
0.1	0.099	500	1	5	20042	17096	2505
0.1	0.099	500	1	10	894302314	17025	5010
0.1	0.099	500	10	1	894314397	17042	5001
0.1	0.099	500	10	5	30342	17136	25005
0.1	0.099	500	10	10	894311233	16978	50010
0.1	0.099	500	100	1	895441892	17048	50001
0.1	0.099	500	100	5	49	17062	250005
0.1	0.099	500	100	10	897406004	16945	500010
0.1	0.099	5000	1	1	51228	17077	5001
0.1	0.099	5000	1	5	200763	17028	25005
0.1	0.099	5000	1	10	82003	17046	50010
0.1	0.099	5000	10	1	449402	17033	50001
0.1	0.099	5000	10	5	3843	16998	250005
0.1	0.099	5000	10	10	2004721	17027	500009
0.1	0.099	5000	100	1	90223	17015	500001
0.1	0.099	5000	100	5	8687	16917	2500005
0.1	0.099	5000	100	10	90127	16971	5000010
1	0.0099	50	1	1	887741522	17315	51
1	0.0099	50	1	5	1006003	17130	255
1	0.0099	50	1	10	887741684	17078	510
1	0.0099	50	10	1	887741775	16490	501
1	0.0099	50	10	5	10048	16350	2505
1	0.0099	50	10	10	887742093	16317	5010
1	0.0099	50	100	1	887741977	13384	5001
1	0.0099	50	100	5	1002005	13308	25005
1	0.0099	50	100	10	887741420	13400	50010
1	0.0099	500	1	1	892768603	17097	501
1	0.0099	500	1	5	9234747	17156	2505
1	0.0099	500	1	10	893093761	17132	5010
1	0.0099	500	10	1	893093652	12998	9121
1	0.0099	500	10	5	2673	12790	42347
1	0.0099	500	10	10	893101921	12601	82886
1	0.0099	500	100	1	893095011	10262	53386

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.0099	500	100	5	144482	10103	270292
1	0.0099	500	100	10	893099664	10066	538783
1	0.0099	5000	1	1	1014	17101	5001
1	0.0099	5000	1	5	3000	16984	25005
1	0.0099	5000	1	10	54002	17009	50010
1	0.0099	5000	10	1	2388	9802	99820
1	0.0099	5000	10	5	30234	9723	470493
1	0.0099	5000	10	10	10463340	9575	972346
1	0.0099	5000	100	1	71310	8719	558687
1	0.0099	5000	100	5	3005010	8914	2725510
1	0.0099	5000	100	10	10502	8681	5470569
1	0.099	50	1	1	887650746	17330	51
1	0.099	50	1	5	10	17069	255
1	0.099	50	1	10	887650857	17210	510
1	0.099	50	10	1	887659207	16580	501
1	0.099	50	10	5	503	16550	2505
1	0.099	50	10	10	887655219	16493	5010
1	0.099	50	100	1	887656505	13705	5001
1	0.099	50	100	5	409	13176	25005
1	0.099	50	100	10	887653739	13214	50010
1	0.099	500	1	1	892953181	17324	501
1	0.099	500	1	5	50394	17158	2505
1	0.099	500	1	10	892661867	17160	5010
1	0.099	500	10	1	892935181	12650	10101
1	0.099	500	10	5	4	12681	50361
1	0.099	500	10	10	892662224	12587	99254
1	0.099	500	100	1	892775407	10325	55703
1	0.099	500	100	5	5069	10159	273611
1	0.099	500	100	10	897401828	10033	548536
1	0.099	5000	1	1	924001	17085	5001
1	0.099	5000	1	5	7440	16972	25770
1	0.099	5000	1	10	910	17061	50010
1	0.099	5000	10	1	1284412	10011	104756
1	0.099	5000	10	5	3068	9521	523640
1	0.099	5000	10	10	85501	9619	1026748
1	0.099	5000	100	1	300232	8801	560193
1	0.099	5000	100	5	767534	8749	2777537

continued on next page

t0	t1	maxpr.	maxfail	runs	seed	Steps	Trees
1	0.099	5000	100	10	34	8694	5571912
1	0.99	50	1	1	887390950	17202	51
1	0.99	50	1	5	5885	17166	255
1	0.99	50	1	10	887387317	17138	510
1	0.99	50	10	1	887399075	17239	501
1	0.99	50	10	5	898	17129	2505
1	0.99	50	10	10	887399075	17131	5010
1	0.99	50	100	1	887395535	17199	5001
1	0.99	50	100	5	39945	16986	25005
1	0.99	50	100	10	887387525	17105	50010
1	0.99	500	1	1	887653948	17311	501
1	0.99	500	1	5	203989	17013	2505
1	0.99	500	1	5	203989	17013	2505
1	0.99	500	1	10	887654142	17077	5010
1	0.99	500	10	1	887654226	17151	5001
1	0.99	500	10	10	887662805	17056	50010
1	0.99	500	100	1	887654507	17066	50001
1	0.99	500	100	5	9274	16997	250005
1	0.99	500	100	10	887654057	17017	500010
1	0.99	5000	1	1	897838198	17035	5001
1	0.99	5000	1	5	321238	16999	27721
1	0.99	5000	1	10	899823384	17021	50010
1	0.99	5000	10	1	899849656	17145	50001
1	0.99	5000	10	5	20014	17088	250005
1	0.99	5000	10	10	92271	17075	500010
1	0.99	5000	100	1	899889967	16991	500001
1	0.99	5000	100	5	902115	16996	2500005
1	0.99	5000	100	10	1002326	16960	5000010

Appendix C

LVB Test Suites

Two test analyses are used to test LVB on new systems. I have called these 'Test 1' and 'Test 2'. Both use the 500-sequence *rbc*L data matrix of Chase et al. [53], with the three changes to names mentioned in Appendix A.

The `data` file and files of expected results are available from the author on request.

C.1 Test 1

This test completes within a few minutes on current machines. The `params` file is given below. The instructions in the `params` file to 'change the value of `seed` before every run' are intended for users who modify this `params` file for their own analyses. `seed` should not be changed when the `params` file is used in the test suite.

This is the `params` file used:

```
method anneal
t0 1.0
t1 0.7
maxaccept 15
maxpropose 1000
maxfail 2
equivalent none
matchchar .
ignorechars KYMSWRN-
outgroup none
runs 1
titleprec none
verbose 1
% CHANGE THE VALUE OF SEED BEFORE EVERY RUN
seed 1789
```

C.2 Test 2

This test completes within a few hours on current machines. It is more complete than Test 1, especially in testing arithmetic underflow (Table 3.1). The `params` file is given below. See Section C.1 regarding `seed`.

This is the `params` file used:

```
method anneal
t0 0.0001
t1 0.00009
maxaccept 30
maxpropose 3000
maxfail 2
matchchar .
ignorechars KYMSWRN-
equivalent none
outgroup none
runs 1
verbose 1
titleprec #
% CHANGE THE VALUE OF SEED BEFORE EVERY RUN
seed 9227
```

Appendix D

LVB Manual

The LVB manual was originally available in printed form [31]. In 1999, it was made available in corrected form as plain (ASCII) text, downloadable from the LVB Web page (Appendix G). This plain text version of the manual follows.

LVB 1.0

RECONSTRUCTING EVOLUTION WITH PARSIMONY AND SIMULATED ANNEALING

Daniel Barker
Institute of Cell and Molecular Biology
University of Edinburgh

First published by Daniel Barker, 1997
Reprinted 1997 (twice), 1998
ISBN 0-9531042-0-6

Released electronically as PostScript text, 1997
Released electronically as ASCII text with corrections and minor changes, 1999

Edinburgh

> '. . . men are strangely inclined to worship what they do not understand. A grand secret, upon which several imposers on mankind have totally relied for the success of their frauds.'
>
> Henry Fielding, 'Tom Jones'

CONTENTS

6.2.14 "clnclose": close file

6.2.15 "clnopen": open file

6.2.16 "clnremove": remove file

6.2.17 "crash": crash verbosely

6.2.18 "epause": pause simulated annealing

6.2.19 "f2str": copy file to string

6.2.20 "getlmask": get branch length mask

6.2.21 "getminlen": get minimum tree length

6.2.22 "getparam": get configurable parameters

6.2.23 "getplen": get tree length

6.2.24 "getroot": find root of tree

6.2.25 "getstatev": get list of states for character

6.2.26 "hillclimb": hill climbing

6.2.27 "main": LVB

6.2.28 "makebin": get integer-encoded matrix

6.2.29 "matchange": alter data matrix

6.2.30 "matrin": read data matrix

6.2.31 "memfree": free memory to prevent leaks

6.2.32 "mempush": register memory to prevent leaks

6.2.33 "mutate": mutate tree

6.2.34 "nextnonwspc": find next non-white space in string

6.2.35 "objreroot": reroot tree on object

6.2.36 "pausenow": check if it is time to pause

6.2.37 "randomwalk": random walk

6.2.38 "randpint": get random non-negative integer

6.2.39 "randtree": get random tree

6.2.40 "restart": get restart information

6.2.41 "rinit": seed random number generator

6.2.42 "rowfree": deallocate matrix rows

6.2.43 "salloc": string memory allocation

6.2.44 "scream": log a warning

6.2.45 "settraps": set signal traps

6.2.46 "stopnow": check if it is time to stop

6.2.47 "strcicmp": case-insensitive string comparison

6.2.48 "supper": convert string to upper case

6.2.49 "treealloc": tree allocation

6.2.50 "treeclear": clear tree

6.2.51 "treecmp": compare trees

6.2.52 "treecopy": copy tree

6.2.53 "treedump": dump tree

6.2.54 "treeprint": print tree

6.2.55 "treeswap": swap trees

6.2.56 "trundump": read dumped tree

6.2.57 "uni": get random number

PREFACE TO THE ASCII EDITION

This ASCII version of the manual is as close as seemed reasonable to the printed first edition,

> Barker D. 1997. LVB 1.0: reconstructing evolution with parsimony and simulated annealing (Edinburgh: Daniel Barker). ISBN 0-9531042-0-6.

In preparing the ASCII edition from this, there have been necessary typographical changes, reemoval of page numbers from the Contents, a minor change to the author's address reflecting an office move and a change to the expiry date of the author's e-mail address (Section 7.2), and the addition of this preface. This manual is for LVB 1.0, which contains a few bugs that were fixed in LVB 1.0A. The instructions for use were not affected. A list of these bug fixes is given in a separate document, entitled 'Bug fixes in LVB Version 1.0A 18 August 1997'.

This manual should be viewed using a fixed-width font, such as Courier, with margins and font size set to allow 80 machine characters per line. Tab settings are not important, since all tabs have been expanded to the correct number of spaces.

The most important typographical change is use of double quote marks around non-display text as it would be shown by or input to the computer. These quote marks are purely to delimit such text, and are not part of it. Other important changes are to mathematical formulae in Section 3.2, where '^' is used to indicate 'to the power of' and '*' is used to indicate multiplication. (This lone '*' is not to be confused with symbols "/*" and "*/", which delimit comments in the C programming language). Also, some text that was printed in italics is now delimited by single quote marks, and some which was printed in bold is now given in capitals.

Here is a list of all changes other than the above, and the sections affected:

Preface. From 'enough to be capture' to 'enough to capture'

Chapter 1. From 'in italics' to 'in capitals'

Chapter 5. From 'in this book' to 'in this manual'

Chapter 6. From 'The the code' to 'The code'

Section 6.2.2. From 'Synopsis. "const Branch"' to 'Synopsis. "Lint anneal(const branch"'

Section 6.2.2. From '"proposed" (number of changes accepted)' to '"proposed" (number of changes proposed)'

Section 6.2.18. From '"proposed" is the number of changes accepted' to '"proposed" is the number of changes proposed'

Section 6.2.26. From 'Assumes the random number generator "uni" has been initialized by calling "rinit", external "nbranches"' to 'Assumes external "nbranches"'

Section 6.2.26. From 'calling "bstrestore" (e.g. from the function "bstrestore")' to 'calling "bstrestore" (e.g. from the function "restart")'

Section 6.2.29. From 'less then "MIN_M"' to 'less than "MIN_M"'

Section 6.2.37. From 'Assumes the random number generator "uni" has been initialized by calling "rinit", external "nbranches"' to 'Assumes external "nbranches"'

Section 6.2.37. From 'calling "bstrestore" (e.g. from the function "bstrestore")' to 'calling "bstrestore" (e.g. from the function "restart")'

Section 6.2.42. From 'Description. deallocate' to 'Description. Deallocate'

Section 6.2.51. From '"tree_1" and "tree_2" (of "root_1" and "root_2" respectively)' to '"tree_1" and "tree_2" (of roots "root_1" and "root_2" respectively)'

Appendix C. From 'fix them meaning' to 'fix the meaning'

Most of these changes are corrections, but some were necessary due to the different nature of the ASCII file and the printed document.

I am very grateful to Joern Scharlemann for comments on a draft of this version of the manual. John Pannell also helped by spotting some mistakes in the printed version. Remaining uncorrected mistakes are my fault, and I would be very glad to hear about them. I would also be grateful for ideas on how to make the manual easier to understand. I remain happy to help anyone use the program LVB.

Daniel Barker, 1999.
Institute of Cell and Molecular Biology,
University of Edinburgh.

PREFACE

I wrote LVB to examine the use of simulated annealing in the search for optimal classification trees. I chose parsimony as the optimality criterion because it is used so often in evolutionary studies, and I wanted to understand it better before proceeding with my own, new criteria. Though it may currently be 'the best we can do' in many situations, parsimony is not really suitable for reconstructing evolution. Evolutionary history cannot usually be represented as a tree. But tree-based methods such as parsimony are with us until we have a universal model of evolution based on a realistic data structure, such as the directed acyclic graph, in which each node may have more than one ancestor. We can only pretend evolutionary history can be represented as a tree by reference to non-operational fictions such as the population or the species. But however wrong it may be, the tree of life is fairly easy to understand and implement, and does generate neat boxes within boxes that can be translated into formal taxa. These advantages have no relevance to parsimony in reconstructing evolution.

By using simulated annealing, LVB casts parsimony as a problem not fundamentally different from other problems of optimisation, for example in biochemistry, engineering, electronics and geography as well as biological classification. Simulated annealing can be used to guess the solution of almost any problem that is too time-consuming to solve analytically. As its name suggests, it is based on the physical process of annealing, whereby a liquid cools to a crystalline solid. During this process, some particles will occasionally move in such a way that the stability of the developing crystal is reduced. By virtue of such moves, the substance can eventually reach a more stable state than if its particles only moved in the direction of immediately increased stability. A parallel can be drawn with the survival of the fittest, which is only a general tendency and not a fixed rule. Like annealing and evolution, simulated annealing is a probabilistic process, in which change over a short period of time only tends to be in the direction of apparent improvement. Early in simulated annealing, we move from one solution to another almost at random. As the run progresses, 'temperature' falls and, like the developing crystal, the system can no longer change so freely. Eventually, we end up with a fixed set of solutions to our problem, which does not change at all. This is analogous to the freezing of the liquid in annealing. The more slowly we decrease temperature, the longer the process takes, and the more stable our end result is likely to be.

In fact, the analogy with true annealing and evolution is not perfect, but it is close enough to capture the imagination. Evolution is faithfully simulated in a different group of optimisation techniques, known as genetic algorithms. With genetic algorithms, solutions coded as chromosomes mutate, cross over and die probabilistically, according to their quality (fitness) and

the selection pressure we choose to apply. Simulated annealing is closer to true annealing, which allows exciting possibilities of a different kind. We can monitor specific heat as the run progresses, a large change indicating a major rearrangement of the order of condensation of a gas or freezing of a liquid in nature. As well as specific heat, we can monitor entropy, an indicator of stability. In simulated annealing entropy is the stability of the quality of our solution, which will be at a maximum when the run is over. With LVB, the quality of the solution is the shortness of the tree.

LVB 1.0 does not itself estimate specific heat and entropy. LVB cannot do a great deal, but it is careful and efficient, and will work well on many modern computers. It is written in standard C, and can be compiled to run on any computer for which a C compiler conforming to the ANSI standard is available. This includes all current general purpose desktops, servers and mainframes. However, LVB will not run significantly faster on multiprocessor architectures, since automatic parallelisation of the current code will be crippled by my optimisations for serial execution. Development of an explicitly parallel version is being considered. As it is, trivial parallelisation by running LVB several times with different random number seeds is a simple way to use more than one processor at once. In many institutions, there are large numbers of computers idle at weekends and at night, and no investment is necessary for quite large-scale parallelisation. For example, you may be able to use two dozen desktop machines daily from 10 p.m. to 8 a.m. without any impact on normal use during office hours. You could stop and re-start LVB manually, or you may be able to arrange automatic scheduling using the computers' operating system. Every day, you would be doing ten times as much processing as if your own single machine were working non-stop, or you bought a computer ten times faster than your current one just for running LVB.

In the hope that it will be used widely, the code of LVB is specifically released from copyright and may be freely used for any purpose whatsoever. However, it was not developed with re-use by other programmers as a major aim. I hope to spin off a library of more modular sub-programs based on LVB, and would appreciate suggestions from interested programmers about the form this should take. Like LVB, this library will be portable and free.

In addition to my own institution, the following have been useful in the development of LVB: the Computing Services at the University of Edinburgh, the Department of Biochemistry at the University of Edinburgh, the Edinburgh Parallel Computing Centre and the Royal Botanic Garden Edinburgh. Jon Bennett made useful comments on a draft of the manual. Richard Gault, Joe Felsenstein, John Collins, Tom Barker, Martin Pullan, Quentin Cronk, Roger Hyam and Dima Alexeev gave useful advice, Steve Yewdall set up the LVB Internet site and Mark Chase supplied the ultimate data matrix for testing parsimony programs. My thanks to all of these. I am the only one who has checked the source code of LVB 1.0 and tested the program. I am confident that there are no serious errors or ambiguities, but it is impossible to be sure until the program has

been tested more widely. If any mistakes crop up, please let me know, so I can fix them immediately and also for the next general release.

The following free software has been useful in developing LVB and writing this booklet: the Dmalloc library by Gray Wilson, DVIPS, EMACS, GDB, Ghostview, GNU C ("gcc"), GNUPLOT, LaTeX (and hence TeX), MicroEMACS ("ue"), the PHYLIP package and XDVI.

My work is funded by a Biotechnology and Biological Sciences Research Council PhD research studentship, without which it would be impossible.

Daniel Barker, 1997.
Institute of Cell and Molecular Biology,
University of Edinburgh.

Chapter 1. INTRODUCTION

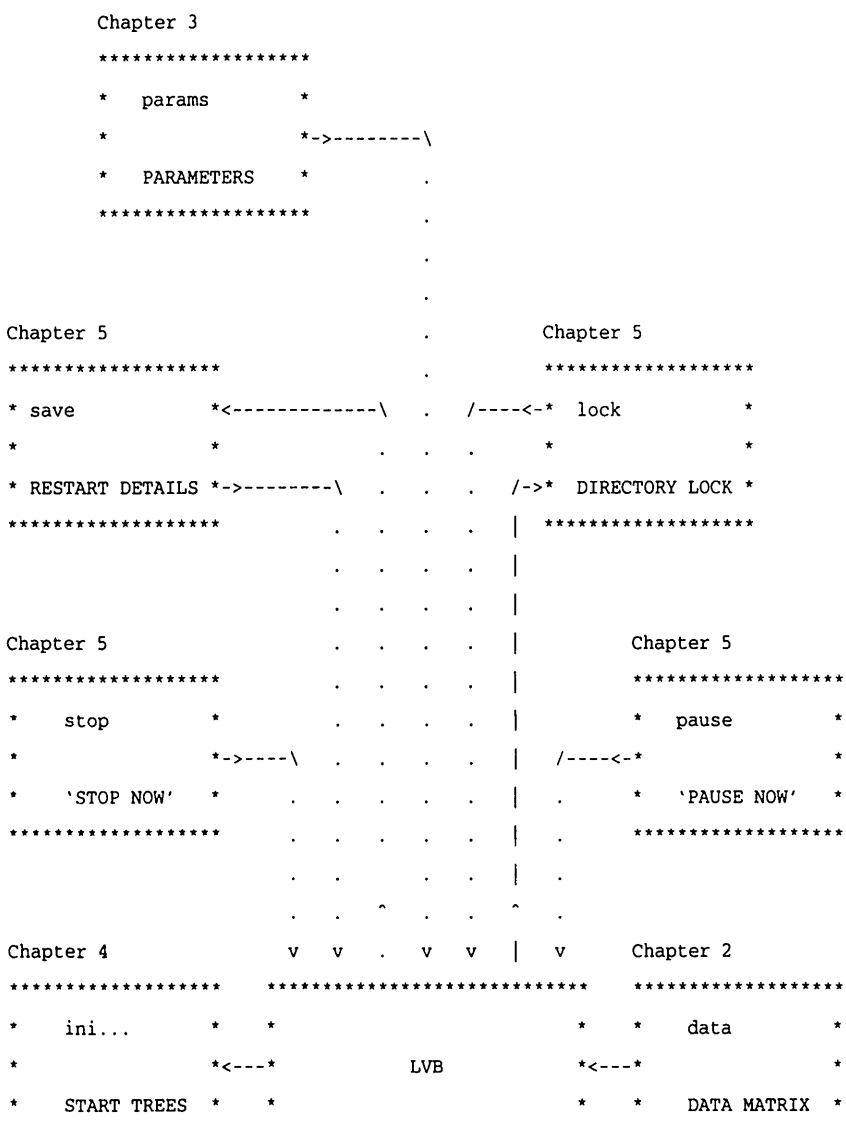
LVB, named in honour of Ludwig van Beethoven, searches for optimal classifications of rows in a matrix of discrete data. The optimality criterion is unweighted reversible parsimony (Fitch 1970, 1971), currently popular in biological systematics. The idea is to produce a tree for which the amount of evolutionary change required to explain the known biological characters (columns) of the organisms (rows) is as low as possible. First, we assume evolutionary history may be shown by a tree whose leaves (terminal branches) are the organisms. We then assume that each biological character can only have the states known in the data we have gathered, that all change is equally possible at all times, and that the tree based on our data compatible with the least amount of evolutionary change is correct. This tree is known as the most parsimonious tree. Sometimes, there is more than one equally parsimonious tree. The minimum number of changes that can have occurred on a tree is known as its length. The most parsimonious tree (or trees) is (are) the tree (trees) of minimum length for our data.

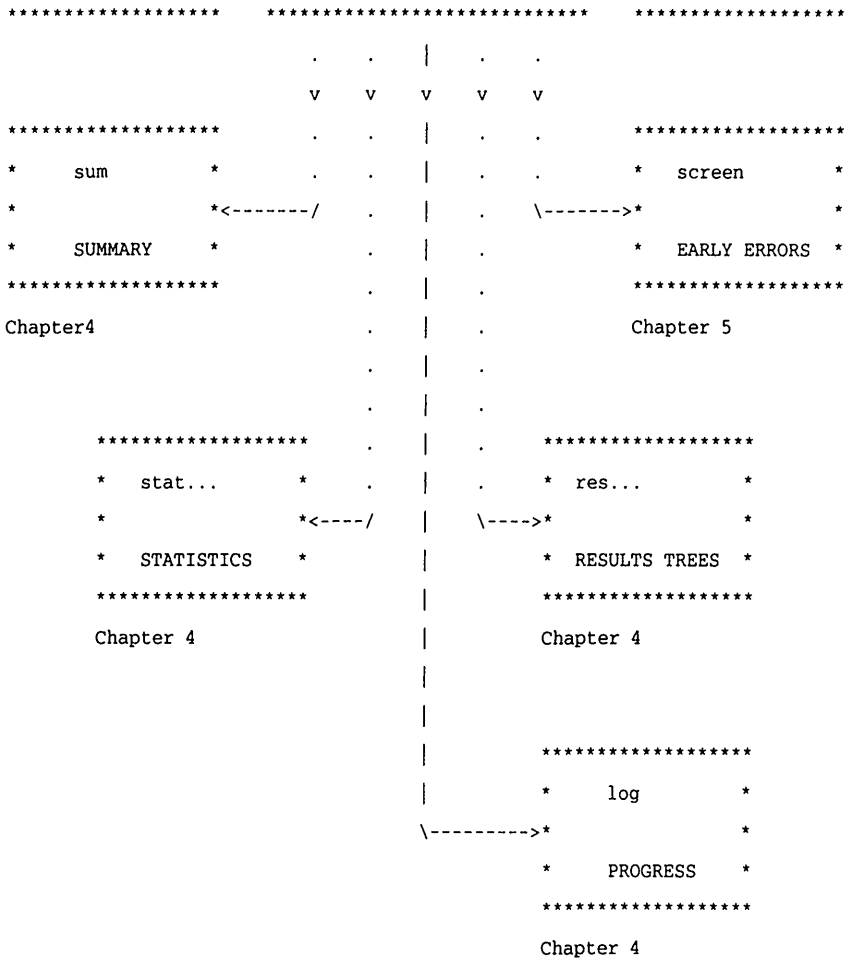
One of the methods LVB uses to search for parsimonious trees, simulated annealing, will give a shorter tree than many other methods in a given amount of time if the data matrix is large. LVB has been developed with very large data matrices in mind. Though there is nothing in the program code to prevent it running with smaller data matrices, if the matrix contains less than 50 rows, performance will often be poor compared to that obtained with other parsimony programs. In particular, if the matrix is so small that optimal trees can be found using an exhaustive or a branch and bound search, LVB could be extremely inefficient.

The program will run (after correct compilation) on any recent general

purpose computer. It is recommended that there should be at least 10 MB memory free for LVB to use when it runs. Note that some operating systems, such as MS-DOS, do not necessarily make more than a small amount of the computer's memory available. LVB may run out of memory, log an error message, and exit prematurely when run on such systems. However, when compiled for and run under Windows 95, Windows NT, MacOS System 7 and recent versions of Unix, LVB will rarely or never run out of memory.

LVB is non-interactive. All input is read from, and output written to, files in the current directory (folder of some computer systems) whose names are fixed. Since all these files consist of plain text, they can be manipulated using many other programs, including word processors, text editors, spreadsheets, text filters and graphics packages. Progress can be followed by examining output files while LVB is running. The flow of data as LVB runs is summed up below:





Boxes represent files, apart from 'LVB', which represents a program image or process. 'Screen' is the standard error, which will usually be the computer screen. Dotted connections show rare or optional data flow. Summaries are given in capitals, and the chapters given should be consulted for details.

At several points, the manual refers to limits which are not fixed, but vary according to the 'system'. These limits may even vary on the same computer with the same operating system, according to the compiler or compiler settings used. Where no upper limit for integers is mentioned, it is always at least 2147483646.

Chapter 2. THE DATA MATRIX

2.1 Format of the matrix file

The input data matrix must be in a text file called "data". The matrix must be in the format described below.

Blank lines in the file are ignored. Apart from these, all lines must contain an object name, followed by the row of the matrix representing that object. The object is the thing we wish to classify, and the name of a species will often serve as its name.

The title and cell contents (character states) for the row must be separated by at least some white space. White space machine characters are space, formfeed, newline, carriage return, tab and vertical tab. Space, tab and newline are the only ones likely to be used in a text file. (Newline cannot separate the title and cell contents, because then they would not be on the same line.) Any white space at the start or end of the line, and between columns in the row, is ignored.

No row may have as its title the word "none", and no row may have the same title as another row. The decision about what is identical is case-insensitive, so, for example, there could not be a row called "Ceratophyllum" as well as one called "CEratophyllum", and there could not be a row called "NONE". Row titles must be from 1 to 120 machine characters long.

Row titles and matrix cells may contain any valid machine characters, other than those which come under the heading of white space or any of ,:;()[]`'" or the null machine character. (The null character will not normally be found in text files.) Treatment of cell contents is case-insensitive, so, for example, the character states "A" and "a" are regarded as identical. Apart from this, in unweighted reversible parsimony all differences are regarded as equally different: "A" is no more similar to "B" than it is to "Z", and "2" is no more similar to "3" than it is to "9".

There is a limit to the number of different character states that can occur in any given column (character). The exact number is system-dependent, but is always at least 16. On 32-bit systems, including many modern computers, it will usually be 32. On 64-bit systems, it might be 32 or 64. LVB will log an error message and exit early if the limit is exceeded.

Rows of the input matrix must be of equal length. The number of rows in the input must be between 4 and 16383 inclusive. The number of characters (columns) must be between 1 and 32766 inclusive. There is a separate limit to the total number of cells (rows x columns) in the matrix, which can vary a great deal. LVB will usually log an error message and exit if it is exceeded. Be warned, it can only do this if the standard library function "malloc" always harmlessly returns "NULL" on failure. If "malloc" has undefined effect when a large amount of memory is requested, then LVB too may behave in an undefined way. Please check the documentation of your standard library, and ensure "MAX_ALLOC" in the file "lvb.h" (Section C.1) does not exceed the maximum safe request in bytes per "malloc" call. If you do not understand the above, please talk to someone who does, for example the author (Section 7.2).

An input file might contain the following:

```
Plant                   CTG-1
Animal                  GTC-0
Fungus                  GTAC0
Bacterium_sp._1         GTA--
```

In this case, there is no space between rows or matrix cells, and row titles are separated from rows by one or more tabs. If exported from a spreadsheet as space-delimited text, the same matrix would look like this:

```
Plant C T G - 1
Animal G T C - 0
Fungus G T A C 0
Bacterium_sp._1 G T A - -
```

and if exported from a spreadsheet as tab-delimited text, it might look like this:

```
Plant  C        T       G       -       1
Animal G        T       C       -       0
Fungus G        T       A       C       0
Bacterium_sp._1         G       T       A       -       -
```

All of the above matrix formats are acceptable to LVB.

In some matrix files, there may be a special symbol in the data matrix meaning 'this cell contains the same as the cell in the first row in this column'. This can be indicated to LVB (Section 3.3.2).

LVB checks the matrix format carefully. It will log an error message and exit early if it detects a mistake.

2.2 Missing data

LVB itself has no concept of missing data. The user should look carefully at his data beforehand, decide himself the best way to handle any given instance of missing data, and modify the matrix accordingly.

By default, symbols often used for missing data, such as "?" and "-", will be treated as character states in their own right. LVB does provide a way to ignore whole columns, which can be used to deal with missing data (Section 3.3.3). It also provides a way to treat different character states as if they were the same (Section 3.3.1).

Chapter 3. CONFIGURING LVB

The method used to search for optimal classifications, and other parameters, may be set in a run time configuration file called "params". This must be a text file. It configures LVB by assigning values to variables. If there is no "params" file, or where it only gives values for some of the possible features, defaults will be used for the undefined values. It is recommended that you do not rely on these defaults. They may not give good results with your matrix, especially if you configure some things but rely on the defaults for others.

LVB's variables are "method", "trees", "t0", "t1", "maxaccept", "maxpropose", "maxfail" (Section 3.2), "equivalent" (Section 3.3.1), "matchchar" (Section 3.3.2), "ignorechars" (Section 3.3.3), "outgroup" (Section 3.4), "runs" (Section 3.5), "seed" (Section 3.6), "titleprec" and "verbose" (Section 3.7 and Section 4.3).

3.1 Format of the configuration file

Any line beginning with "%", possibly preceded by white space, will be ignored. This is to allow the user to put comments in the file.

On lines not beginning with "%", a name followed by a value assigns the value to the variable. There must be at least one space or tab between the name and the value. There can only be one assignment per line. Depending on the variable named, the value may be an integer, a number including a decimal point (also known as a floating point or real number), or a string of letters or other machine characters not containing white space. On any line, spaces and tabs before the name, or after the value, are ignored. Reading of the configuration file is case-insensitive, except for the value of "titleprec" (Section 3.7). The null machine character and white space (Section 2.1) other than space or tab will cause LVB to exit with an error message.

Values given to variables which are not relevant in the context of the whole file will not be used. If a variable is given a value more than once in the configuration file, the last value given is used. LVB will log an error and exit on detecting an error in "params".

Since the file "params" is read at run time, it is not necessary or useful to re-compile LVB every time the file is altered.

3.2 Search method

The method used to search for parsimonious trees is specified with the variable "method". Possible values are "anneal", for simulated annealing, "hillclimb", for hill climbing (also known as steepest descent), and "randomwalk", for random walk. For example, to use hillclimbing with default settings for the associated parameters, you could use a "params" file consisting of the following:

```
method hillclimb
```

All methods begin the search for an optimal classification with an unrooted bifurcating tree whose leaves (terminal branches) are objects in the data matrix, and whose shape is random. They try and improve on this tree by making random changes to it. Each change consists of choosing one leaf at random, removing it, and inserting it in a new, random position on the tree in such a way that the tree remains bifurcating. All methods keep a record of the shortest trees found, which is output at the end as the solution or solutions. The default is "method anneal".

The random walk is a simple and ineffective method that is given for comparison only. The method of searching through possible trees is:

```
WHILE (the number of trees examined is less than the number requested)
{
        propose a random change to the current tree;
        accept the change;
}
```

One worse tree may follow another, and few trees better than the original will be found, even if many trees are examined. But since the search is unbiased, if an infinite number of trees is examined, it is guaranteed that the best tree will be found. The number of trees to look at is set using the variable "trees", which must have an integral value of at least 1. The default is "trees 20000".

Hill climbing is better in practice, when an infinite amount of trees cannot be examined. The method of searching through possible trees is:

```
WHILE (the number of trees examined is less than the maximum requested)
{
        propose a random change to the current tree;
        IF (the changed tree is no longer than the current tree)
                accept the change;
}
```

This approach is like trying to reach the top of a mountain range by only ever walking up hill. The result may be much better than the initial, random tree, but is also unlikely to be the best tree possible. To reach the highest point of a mountain range from most starting points, you will sometimes have to walk downhill. To find the shortest tree of all possible trees, usually you will have to make some changes that increase the tree's length on the way. So hill climbing is not guaranteed to find the best tree. But if hill climbing is repeated many times, with different, random initial trees (Section 3.5), it becomes more likely that one initial tree is close to the best tree, and the chance of finding the best tree increases. The number of trees to examine is set using the variable "trees", as with "method randomwalk" (above).

Simulated annealing (Kirkpatrick et al. 1983) is better in that it will occasionally accept worse trees, with the aim of obtaining even shorter ones in the long run. The probability of accepting a change that increases tree length depends on two things:

1. how much longer the tree would be after the change;

2. the 'temperature', a control parameter which changes during the run.

Simulated annealing is the recommended method when there is a large number of objects. Like the random walk, it is guaranteed to find the shortest tree if run for long enough, no matter what the initial tree is. But unlike the random walk, it does tend to move in the direction of short trees. Unlike hill climbing, it can avoid becoming stuck in a local optimum by occasionally making changes that increase the length of the current tree. However, there is still no guarantee that any given run has even come close to finding the shortest tree.

The method of searching through possible trees is basically this:

```
initialize T to the specified starting temperature;
FOR (;;)                                /* infinite loop */
{
        propose a change to the current tree;
        IF (the changed tree is no longer than the current tree)
                accept the change;
        ELSE
                accept the change with probability e^(-1/T * delta H);
        IF (changes made at this temperature >= "maxaccept")
                decrease T;
        ELSE IF (changes proposed at this temperature >= "maxpropose")
        {
                IF (the system appears 'frozen')
```

```
                        BREAK;          /* end the cooling cycle */
                ELSE
                        decrease T;
        }
}
```

where e is the usual physical constant (e approx.= 2.72), H is the homoplasy index (H.I.) (Swofford 1993), delta H is the increase in H.I. the change to the tree would cause, "maxaccept" and "maxpropose" are constant natural numbers, T is the current temperature and 0 <= T <= 1.

In simulated annealing, 'energy' of some kind indicates the quality of the solution to our problem: the higher the energy, the worse the solution. In parsimony, we seek short trees, so energy must increase as length increases. In LVB, I have chosen to represent energy by the H.I. For matrices acceptable to LVB, the H.I. is the ratio of the length of a tree with no homoplasy to the observed length of the tree, subtracted from 1. The tree with no homoplasy is a theoretical 'ideal' tree, where each biological character changes just enough to give all the states it shows in our data matrix. For example, a character with 4 states must always change at least 3 times, and would contribute 3 to the length of the tree without homoplasy. This tree will not usually exist for a real data matrix, since the tree with the minimum number of changes for one character may differ from the tree with the minimum number of changes for another character. This is expected because of parallel changes and reversal of changes in evolutionary history, and also because of errors in the data matrix such as poor assessment of homology. For a given data matrix, a longer tree has higher H.I. than a short tree. The H.I. can never be less than 0 or greater than 1 and has no units, avoiding the requirement for a constant like the Boltzmann constant of thermodynamics when calculating the probability of accepting a longer tree.

LVB's cooling schedule is based on, but not identical to, the logarithmic cooling of Kirkpatrick et al. (1983). The initial temperature, T0, is the LVB variable "t0". The second temperature, T1, is the LVB variable "t1". "t0" and "t1" must have values greater than 0.0 but no more than 1.0, and the value of "t1" must be less than or equal to the value of "t0". Temperatures other than the initial temperature are obtained by Tn = (T1/T0)^n * T0, where n = 1 for the second temperature and increases by 1 with each temperature change. At each temperature, enough changes are considered that either "maxaccept" new trees are accepted or "maxpropose" changes are considered, whichever occurs sooner. If less than "maxaccept" new trees are accepted in each of "maxfail" successive temperatures, the system is considered frozen, and the run stops. LVB's use of new solutions for comparison with "maxaccept" differs from Kirkpatrick et al.'s use of accepted solutions, since some solutions may be accepted that have already been encountered. If accepted solutions were not compared against those already

found, we might endlessly move from one equally parsimonious tree to another and back again, and the run would only terminate with some sort of error.

"maxaccept", "maxpropose" and "maxfail" must have integral values of at least 0. "t0" and "t1" should not be given very low values, since the effect would be similar to hillclimbing (above), and LVB might have trouble with the arithmetic in extreme cases.

The defaults are "t0 1.0", "t1 0.9", "maxaccept 50", "maxpropose 5000" and "maxfail 2".

Both the random walk and hill climbing can be thought of as special cases of simulated annealing: in the random walk, T remains fixed at 1, and in hill climbing, T remains fixed at 0. For example,

```
method randomwalk
trees 10000
```

would have the same effect as

```
method anneal
t0 1.0
maxaccept 20000
maxpropose 10000
maxfail 0
t1 0.1
```

but this is not recommended because it makes unnecessary use of floating point arithmetic, which might not be accurate. Also,

```
method hillclimb
trees 10000
```

is conceptually the same as

```
method anneal
t0 0.0
maxaccept 20000
maxpropose 10000
maxfail 0
t1 0.0
```

but here, the "method anneal" version would not work at all, since LVB insists "t0" and "t1" must be greater than zero. In both these annealing schedules, "t1" is useless and "maxaccept" is higher than "maxpropose" so can never be reached. "maxfail 0" ensures T never falls below its initial value. It works because LVB does not check if the system is frozen before "maxpropose" changes

have been considered at the current temperature, and the current temperature is initially "t0".

A similar approach can be used to examine properties of the system of candidate solution trees at any non-zero fixed temperature using the Metropolis algorithm (Metropolis et al. 1953). LVB can save details as it runs (Section 3.7 and Section 4.3) from which statistics such as the entropy and specific heat (Kirkpatrick et al. 1983) can be estimated. However, given the way trees are sampled by LVB, it may be that these estimates are not meaningful.

3.3 Modifying the matrix

Three variables are used to alter LVB's interpretation of the data matrix. Only LVB's internal working copy of the matrix is affected. The contents of the matrix file are not changed.

The order in which the specified actions are taken could sometimes affect the final result. The changes are made in the order of the three sections below, irrespective of where the variables occur in the file "params".

3.3.1 Equivalent character states

It is possible to tell LVB that some biological character states are equivalent to others, in which case those states will be treated as if they were the same. This is done using the variable "equivalent". Its value should be one or more lists of states, or "none". Multiple lists should be separated by a comma, without any space. Character states in the same list are converted to the first state in the list before LVB starts to analyse the matrix. "none" indicates that no states are equivalent to any others.

There may be no more than 32 lists of states, and no more than 32 states within each list. LVB will log an error message and exit prematurely if either of these limits is exceeded.

As an example, imagine you have a DNA sequence matrix, containing character states "A" for adenine, "C" for cytosine, "G" for guanine, "T" for thymine, "-" for missing due to alignment, "?" for missing because that particular region was not sequenced and "N" for missing due to uncertainty in sequencing. To treat all missing data as the same, and reduce the other character states to two states, signifying purine and pyrimidine, you could use the following:

```
equivalent -?N,AG,CT
```

This will cause any occurrences of "?" and "N" to be replaced with "-", "G" with "A" and "T" with "C". There must be no space within or between lists (for

example, between "A" and "G" above or after the commas).

The lists are read left to right, and each set of replacements is made before the next list is examined. All such changes are made before those outlined in Section 3.3.2 and Section 3.3.3.

This is a convenience option. The same effect could be achieved using 'search and replace' with a text editor or spreadsheet.

The default is "equivalent none".

3.3.2 Rows relative to the first row

Sometimes, only the first row of the matrix is given in full. In subsequent rows, a fixed symbol is then used to indicate that the cell contains the same character state as in the first row. This is indicated by assigning a value to the variable "matchchar". Possible values for "matchchar" are "none", to indicate that all states are exactly as recorded in the matrix file, or a single machine character. Instances of this symbol in LVB's internal copy of the matrix will be replaced by the symbol in that column in the first row before LVB starts to analyse the matrix.

I have included this variable to allow reasonable compatibility with the matrix format of some other programs. The name "matchchar" is borrowed from PAUP (Swofford 1993).

Any replacements are made after those mentioned in Section 3.3.1, but before those mentioned in Section 3.3.3.

The default is "matchchar none".

3.3.3 Ignoring whole columns in the matrix

To tell LVB to ignore columns containing one or more specific character states, use the variable "ignorechars". This can have a value of "none", to use all columns, or a string of at least one machine character, consisting of symbols whose presence will cause a column to be ignored.

If the symbol "-" indicates missing data, and you wish to ignore all columns containing missing data, you could use the following:

```
ignorechars -
```

If the matrix consists of a mixture of molecular characters with states "A", "C", "G" and "T", and morphological characters with states "0" and "1", you could ignore all morphological characters as follows:

```
ignorechars 01
```

There must be no space within the list of character states (for example, between the "0" and "1" above).

The decision as to which columns to ignore is made after the changes to the matrix outlined in Section 3.3.1 and Section 3.3.2 have been made.

The default is "ignorechars none".

3.4 Tree rooting and outgroup

The outgroup, or absence of an outgroup, is set by assigning a value to the variable "outgroup". Possible values are the title of any row in the matrix, or "none".

When "outgroup" is set to the title of a row in the matrix, results trees will be rooted, and have the named object as outgroup. An outgroup consisting of more than one object is not possible with LVB. The outgroup will appear as sister group to a clade consisting of all other objects. With "outgroup none", results trees will be unrooted.

The unrooted trees will always be bifurcating, and could therefore include branches on which no character state changes are ever supposed to occur. In rooted trees, branches within the ingroup clade on which no changes can ever occur under the accelerated transformation model of character change (Maddison & Maddison 1992) will be collapsed, giving trichotomies or polychotomies. This difference between output of rooted and unrooted trees is because without a root, it is not generally possible to locate all changes, so specific branch lengths cannot be inferred. The search for parsimonious trees is not affected, since this simply involves working out how many changes seem to have occurred, not where they have occurred. Working out tree length is much faster than working out the length of every branch on a tree, and may be done by hand for a few biological characters without too much trouble. Many text books give the impression that it is a deeply involved process, by introducing the calculation of tree length as based on reconstructions of character state change. Since there are often many reconstructions of change that fit any one tree of fixed length and topology, this increases the amount of work considerably.

To return to LVB, I emphasize that the overall length of the results trees is always independent of the value of "outgroup". It is only the shape of the trees that is affected.

The default is "outgroup none".

3.5 The number of runs

The number of times you wish to apply the search method is set with the variable "runs". Each run starts with a different random tree, and produces

different statistics and tree files. The value of "runs" must be an integer greater than zero and less than a limit which is always at least 32766. LVB exits with an error message if "runs" is out of range.

This is a convenience option. It saves the user from having to start the program several times when several runs are desired. Multiple runs are performed one after the other.

The default is "runs 1".

3.6 Seed for the random number generator

Computer 'random number' generators actually generate a sequence of numbers that is entirely predictable, based on an initial number called the seed. The important thing is that the sequence of numbers is statistically indistinguishable from a random sequence throughout the program run.

The random number seed may be set by assigning a value to the variable "seed". The value of "seed" must be an integer from 0 to some limit that is always at least 32767 and no more than 900000000. LVB exits with an error message if the value of "seed" is unsuitable.

The default value of "seed" varies with time. If the value of "seed" is not specified, LVB tries to take its value from the system clock, and exits with an error message if this value is unavailable or unsuitable.

3.7 Statistics

The amount of output generated by LVB is controlled by the variable "verbose", which can have value "0" or "1". With "verbose 1", a large amount of disk space can be used up by statistics logged as LVB runs. With "verbose 0", fewer statistics are written and less disk space is used. See Section 4.3 for details. The default is "verbose 1".

A machine character, or string of up to 32 machine characters, can be introduced at the start of lines containing column titles in tabulated statistics. The string will appear at the start of the line, and be followed by a space then the usual column titles. This is useful if you wish to 'comment out' title lines. The variable is "titleprec". The default is "titleprec none", causing no text to precede the titles. The case of "titleprec" is preserved. To avoid having to edit statistics files before using them with GNUPLOT, you might include the following line in "params":

```
titleprec #
```

Chapter 4. OUTPUT

4.1 Trees

Trees are output as plain text files conforming to the unpublished 'Newick Standard'. Trees in this form are acceptable to several programs by other authors, including some programs in the PHYLIP package (Section A.2, Appendix B).

The Newick format allows many possibilities, not all of which are used by LVB. With LVB's trees, tree length is not given, labels of length 1 to 120 machine characters are only and always present for leaves of the tree, each tree begins at the start of a line, and lines only and always end after commas (clade delimiters) and semicolons (tree terminators). There is no other white space. Since LVB does not allow object names to contain white space or any of ,:;()[]`'", these will never occur within labels. Trees are not necessarily rooted, but when they are rooted they are not necessarily bifurcating (Section 3.4).

The initial, random tree is output in bifurcating unrooted form to a text file called "inix", where "x" is the number of the run. The first run is number 0. The results tree or trees are output to a text file called "resx", where "x" is the number of the run. If there is more than one tree, the can easily be separated because each tree always ends with a semicolon, and semicolons will never be found at any other point. If trees are rooted (Section 3.4), some may be included more than once in the "resx" file (Section 4.3). This gives an incorrect impression of the number of different results trees, and may lead to incorrect majority-rule consensus trees by effectively weighting some trees more than others. If this would be a problem, choose unrooted trees (Section 3.4), in which case the trees saved for a run are all different. But remember that, with unrooted trees, some branches may have zero length for any reconstruction of character state change.

The problem with rooted trees is a problem with LVB alone, and is likely to vanish in later versions.

4.2 Log

Parameters, pertinent information, non-tabulated progress reports and errors (with exceptions mentioned in Section 5.1) are logged to a file called "log".

If, after LVB has finished, the file does not end with a line either the form

```
End time:  Wed Jul 23 11:09:44 1997
```

or consisting of

```
End of execution
```

then something has gone badly wrong, for example a write error on the log file, and all the output should be regarded as suspect.

4.3 Statistics

With "verbose 1" and "method anneal", the number of the current tree examined (in a column headed "Tree"), the best tree length so far ("BestLen"), the number of the current temperature ("TempNo"), the current temperature ("Temp"), the current tree length ("CurrLen"), the length of the tree after the proposed change ("PropLen") and the probability of accepting the proposed change ("ProbAcc") are output to a file called "statx", where "x" is the number of the run. The first run, tree and temperature are numbered 0. With "verbose 1" and "method hillclimb" or "method randomwalk", only the number of the current tree ("Tree"), the best length so far ("BestLen"), the current length ("CurrLen") and the length of the tree after the proposed change ("PropLen") are output to "statx". With "verbose 0", "no statx" file is created or altered with any method.

Whatever the value of "verbose", the run ("Run"), length of the initial tree ("InitLen"), length of the most parsimonious trees ("BestLen") and number of most parsimonious trees for each run ("Trees") are always output in tab-delimited form to the summary file "sum". Note that the number of trees given is the number of bifurcating unrooted trees discovered by LVB. If trees are rooted, some branches may have been collapsed on output, possibly changing trees that did appear different into trees that are identical. However, LVB does not check this, and so may count and output the same tree more than once (Section 4.1).

Many users will be more interested in the results than the progress of the search, and they can save disk space, and often execution time, with "verbose 0". With this setting, only initial trees, results trees, the log file and the summary file will be written. None of these files is likely to be large. Since no temporary files are created by LVB (unless it is paused, see Section 5.2), "verbose 0" allows it to run where very little disk space is available. However, if disk space becomes exhausted while LVB runs, the effect will be fatal for the program. It may only be possible to detect the error

indirectly (Section 4.2, Section 5.4).

Chapter 5. RUNNING LVB

5.1 Basics

Put a matrix file called "data" in the current directory. Put a configuration file called "params" in the current directory if required. Rename or move to another directory any files in the current directory that were output by a previous run, unless LVB is being restarted (Section 5.2). Now, do whatever is necessary to run the executable file. At its simplest, this may involve entering the command

```
lvb
```

or double-clicking an icon labelled "lvb" with the mouse. Some systems have useful commands that will let you run LVB at low priority, when the system is not very busy, at a fixed later time, or immediately but detached from your terminal so you may log out. (In Unix, these are "nice", "batch", "at" and "nohup" respectively.) These are not relevant to the program LVB itself, and are not mentioned further in this manual.

There will be no screen output, unless LVB has detected another copy of itself running in the current directory or cannot open the log file, in which case it will display a brief message to the standard error (usually the screen) and exit before writing to or creating any files.

5.2 Pausing LVB

With simulated annealing (Section 3.2), the user can pause LVB by putting a readable text file called "pause" in its working directory. A note of the pause is sent to the log file, and LVB terminates after saving its current internal state to a file called "save". LVB will remove the file called "pause" before exit. The next time LVB is run in the current directory, it will use "save" to restart where it left off. It will then remove "save" before exit. On restarting, LVB saves the unrooted bifurcating current tree at the time of the pause to a file called "inixr", where "x" is the number of the run.

The ability to pause is useful if a power cut or shut-down is expected, or when running LVB on computers that are only available some of the time. On a stable, well-maintained machine, judicial pauses can allow LVB to continue indefinitely.

After pausing, it is even possible to restart LVB on a different computer. To do this, put copies of the files "data", "params" and "save" on the system where you want to run LVB. Then, simply start LVB in the normal way on that system. This is useful if LVB has started on a slow computer but is taking too long. The operation can be moved to a faster machine without having to restart from scratch. "save" is a plain text file, and is portable to other computers using the same character set.

The "save" file can be large. LVB will crash with no save if there is not enough disk space for it.

Altering the contents of the files "data", "params" or "save" between pausing and restarting has an undefined effect. LVB may produce incorrect results, behave unexpectedly or crash obscurely. However, removing "save" entirely is allowed. If "save" is removed, a restart is prevented, and the other files can be changed as usual. The next time LVB runs it will start from scratch.

Since LVB only checks for the presence of the file "pause" once per 5000 trees examined, it may take some time for the message to take effect.

Pausing has not been implemented for "method hillclimb" or "method randomwalk", because each run will usually be shorter when using these methods. To achieve the effect of pausing LVB with these, stop LVB at the end of the current run (Section 5.3), move or rename the output files, and start again from scratch later.

5.3 Stopping LVB early

The user can tell LVB to stop early by putting a readable text file called "stop" in its working directory.

A note of the early stop is sent to the log file. All the output files that can be generated at that point in the analysis will be written. This allows the user to record the best trees found so far and their length, then stop prematurely. LVB will remove the file called "stop" on exit.

Since LVB only checks for the presence of the file "stop" once per 5000 trees examined, it may take some time for the message to take effect.

To terminate LVB quickly and without saving any trees is also possible, but the way to do this depends on your computer system. For example, when running LVB from a Unix or MS-DOS command prompt, pressing "c" while holding down the "CONTROL" key will have this effect. With MacOS, press "ESCAPE" while holding down the "OPTION" and "COMMAND" keys, and click "Force Quit" when

prompted.

5.4 Warning about files

If there is a file called "log" in the directory when LVB starts, it will normally discard its original contents without warning, unless this is a restart after a pause (Section 5.2).

Also, any files whose names begin with "res", "ini" or "stat" are at risk of being erased or overwritten. Files called "pause", "stop" and "save" have special meaning for LVB, and should only be present if the user is aware of, and intends, their effect (Section 5.2 and Section 5.3). They will be deleted by LVB after they have served their purpose.

A file called "lock" will appear wherever LVB is running. This text file containing a single space is used by LVB to detect if another copy of the program is already running. It will be removed by LVB on normal exit. If this file is still there after LVB has finished, something has gone badly wrong, for example switching off the computer. Deleting the file will allow LVB to run again in that directory.

Chapter 6. NOTES FOR PROGRAMMERS

Notes on all identifiers with external linkage are given here to help programmers understand, modify and re-use the program source code. The code was written as quickly as seemed consistent with efficient and correct execution, and ease of re-use was not the main priority. However, it will be reasonably clear to programmers, even those unfamiliar with C, because of low reliance on the more confusing aspects of the language. The notes in this chapter are far from a complete guide to the LVB internal interface, because the time it would take to write is not currently justified. The next version of LVB may differ significantly. However, I have found the notes useful myself. The full program listing, including a moderate number of comments for explanation, is given in Appendix C. Please report any errors to the author (Section 7.2).

All macros mentioned other than those from the standard library are defined in "lvb.h", apart from "MAX_SEED", which is defined in "myuni.h". I have used many of my own derived data types, which are also defined in "lvb.h", with the exception of "Uni_float", which is defined in "myuni.h". The purpose of these types is clear from these headers, and is not discussed in this

chapter.

Use of upper and lower case for LVB's identifiers follows Kernighan & Ritchie (1988): variables, functions and macros with arguments are lower case, defined and enumerated constants are upper case, and derived data types are lower case with an initial capital letter.

6.1 LVB external variables

All LVB's variables with external linkage are defined in "main.c", and made visible by "extern" declarations within functions outside this file. Care should be taken when changing their value outside the function "main".

6.1.1 "logfp": log file

Synopsis. "FILE *logfp"

Description. The log file, obtained by opening the file whose name is given by "LOGFNAM" for writing (on normal start) or appending (on restart after pause).

6.1.2 "matrix": data matrix

Synopsis. "Dataptr matrix"

Description. The data matrix, read into memory from the file whose name is given by "MATFNAM" by calling "matrin", then modified by calling "matchange".

6.1.3 "nbranches": branches per tree

Synopsis. "Branchno nbranches"

Description. The number of branches in any unrooted, bifurcating tree based on the data matrix, obtained by calling "brcnt" after the data matrix has been put into the external "matrix".

6.1.4 "tbinmat:" encoded matrix

Synopsis. "Uint **tbinmat"

Description. A transposed version of the matrix proper, with each state in a biological character coded as a different power of 2 to allow calculation of intersection and union by fast bitwise operations. It is obtained from the external "matrix" by calling "makebin".

Notes. "tbinmat" is transposed in memory with respect to the original because it is accessed by biological characters, and there would usually be a loss of speed if it were stored by objects. To hide the fact that it is transposed, elements should be accessed using the macro "binmat". "binmat(i, j)" expands to "tbinmat[j][i]", the value based on "matrix->row[i][j]". Also for speed, its rows are usually contiguous in memory, so over-running bounds will not always be a detectable error: if the matrix has been allocated contiguously (which is not always the case, see Section 6.2.28), it is legal to dereference all cells using only the first dimension. For clarity this is never recommended.

6.2 LVB external functions

Sections are in alphabetical order of function name. All LVB's functions with external linkage are declared in "lvb.h", except "rinit" and "uni", which are declared in "myuni.h".

Many of the functions are non-reentrant could be awkward to use in multi-threaded programs. Notes on this are not given.

Some "static" functions, especially those in the file "trops.c", may also be of interest, but are not described here. There is reasonable documentation in the source code itself (Section C).

6.2.1 "alloc": basic memory allocation

Synopsis. "void *alloc(const size_t bytes, const char *const msg)"

Description. Return pointer to "bytes" bytes of uninitialized memory, to be used for the object mentioned in string "msg"; or, if "bytes" exceeds "MAX_ALLOC", crash verbosely without attempting allocation. Assumes external "logfp" is open for writing. Exits with an error message based on "msg" on failure.

Notes. The memory to which a pointer is returned may be deallocated using the standard library function "free".

Source file. "mem.c"

6.2.2 "anneal": simulated annealing

Synopsis. "Lint anneal(const Branch *const inittree, Branchno root, const Real t0, const Real t1, const Lint maxaccept, const Lint maxpropose,

const Lint maxfail, const Integer run, FILE *const lenfp,
const char *const titleprec, Lint lenbest, Lint proposed, Lint accepted,
Lint failedcnt, Real t, Lint n, Lint iter)"

Description. Use simulated annealing to optimize classification tree "inittree" (of root "root"). Leave best tree(s) on best tree stack, and return best length or 0 - best length if stopped prematurely. Log statistics to "lenfp" if it is not "NULL". "t0", "t1", "maxaccept", "maxpropose", "maxfail" and "titleprec" correspond to LVB variables of the same names, and have the effects noted in Chapter 3. "run" is the ordinal number of the current run, counting from 0. "lenbest" (best length so far), "proposed" (number of changes proposed at the current temperature), "accepted" (number of changes "accepted" at the current temperature), "failedcnt" (consecutive number of changes just before the current temperature at which maxaccept changes were not accepted), "t" (current temperature), "n" (ordinal number of the current temperature, counting from 0) and "iter" (ordinal number of the current tree, counting from 0) should be set to the value of the corresponding elements of the "struct finnegan" returned by "restart" if this is a restart, or to "UNSET" (cast to the correct type) if this is not a restart. Assumes the random number generator "uni" has been initialized by calling "rinit", external "nbranches" is the number of branches in the tree, external "matrix->m" and "matrix->n" are the number of biological characters and objects respectively, the best tree stack has been initialized by calling "bstinit" and, on restart, restored by calling "bstrestore" (e.g. from the function "restart"), and external "logfp" is open for writing.

Source file. "solve.c"

6.2.3 "brcnt": tree branch count

Synopsis. "Branchno brcnt(void)"

Description. Return number of branches in an unrooted bifurcating tree. Assumes external "matrix->n" is the number of leaves in the tree.

Source file. "trops.c"

6.2.4 "bstclear": clear best tree stack

Synopsis. "void bstclear(void)"

Description. Clear the best tree stack and reduce its allocation to "STACKBLOCK" trees, if it has increased beyond this size. Assumes external "logfp" is open for writing.

Source file. "bstack.c"

6.2.5 "bstcnt": size of best tree stack

Synopsis. "Lint bstcnt(void)"

Description. Return number of trees on best tree stack.

Source file. "bstack.c"

6.2.6 "bstdump": dump best tree stack

Synopsis. "Lint bstdump(FILE *const outfp)"

Description. Pop and dump all trees on the best tree stack to "outfp". Assumes external "nbranches" is the number of branches per tree and external "logfp" is open for writing.

Source file. "bstack.c"

6.2.7 "bstfree": deallocate best tree stack

Synopsis. "void bstfree(void)"

Description. Deallocate the best tree stack.

Source file. "bstack.c"

6.2.8 "bstinit": initialize best tree stack

Synopsis. "void bstinit(Params rcstruct)"

Description. Initialize best tree stack. This must be called once, before "bstpush" or "bstrestore" (which is called by "restart"). "rcstruct.og" should be "UNSET" if trees output by "bstprint" should be unrooted, or the number of the outgroup object if not. Assumes external "matrix->n" is the number of leaves per tree, external "nbranches" is the number of branches per tree and external "logfp" is open for writing.

Source file. "bstack.c"

6.2.9 "bstprint": print best tree stack

Synopsis. "Lint bstprint(FILE *const outfp)"

Description. Pop and print all trees on the best tree stack to "stream". Assumes external "nbranches" is the number of branches per tree, the best tree stack has been initialized by calling "bstinit" and external "logfp" is open for writing.

Source file. "bstack.c"

6.2.10 "bstpush": push tree on to best tree stack

Synopsis. "Integer bstpush(const Branch *const barray, const Branchno root)"

Description. Push tree "barray" (of root "root") on to the best tree stack if its topology is not stored there already, increasing the stack's allocation by "STACKBLOCK" trees as necessary. Return 1 on push, or 0 on no push. Assumes external "nbranches" is the number of branches per tree, external "matrix->n" is the number of leaves per tree, the best tree stack has been initialized by calling "bstinit" and external "logfp" is open for writing.

Source file. "bstack.c"

6.2.11 "bstrestore": restore best tree stack

Synopsis. "void bstrestore(FILE *const stream,
const struct finnegan reparams)"

Description. Restore the best tree stack by reading "reparams.bst_trees" dumped trees from "stream". Assumes external "logfp" is open for writing, the best tree stack has been initialized by calling "bstinit" and the tree structure within the dumped data is sound.

Source file. "bstack.c"

6.2.12 "childadd": add child to branch

Synopsis. "Branchno childadd(Branch *const tree, const Branchno destination,
const Branchno newchild)"

Description. Make branch number "newchild" a child of branch number "destination" in tree "tree". Exits with an error message if branch "destination" already has two children not equal to "UNSET". Assumes external "logfp" is open for writing.

Source file. "trops.c"

6.2.13 "cleanup": clean up for exit

Synopsis. "void cleanup(void)"

Description. Log end time if possible, close external log file "logfp", free all memory registered with "mempush" and in the best tree stack, and remove lock file (whose name is given by "LOCKFNAM"). Assumes external "logfp" is open for writing on call, or "NULL" if no output is desired. "cleanup" is called once on exit, either after registration early in the function "main" using the standard library function "atexit", or directly late in "main" if "atexit" fails.

Source file. "cleanup.c"

6.2.14 "clnclose": close file

Synopsis. "void clnclose(FILE *const fp, const char *const fnam)"

Description. If "fp" is not "NULL", close the file "fp" whose name is given by string "fnam". Assumes external "logfp" is open for writing. Exits with an error message including "fnam" on failure.

Source file. "fops.c"

6.2.15 "clnopen": open file

Synopsis. "FILE *clnopen(const char *const nam, const char *const mod)"

Description. Return pointer to file of name "nam", opened with mode "mod". Assumes external "logfp" is open for writing. Exits with an error message including "fnam" on failure.

Source file. "fops.c"

6.2.16 "clnremove": remove file

Synopsis. "void clnremove(const char *const fnam)"

Description. Attempt to remove file whose name is given by string "fnam". Assumes external "logfp" is open for writing. Logs a warning including "fnam" on failure.

Source file. "fops.c"

6.2.17 "crash": crash verbosely

Synopsis. "void crash(const char *const fmt, ...)"

Description. Log message consisting of "FATAL ERROR: " followed by the message given in the argument list then newline, and exit abnormally. If "IAMDANIEL" is defined, this message is followed by another, giving the latest system error. The argument list must be acceptable to the standard library function "vfprintf", i.e. consist of a string followed by 0 or more associated variables which should only be pointers or of type "int", "unsigned int" or "double". Assumes external "logfp" is open for writing.

Source file. "err.c"

6.2.18 "epause": pause simulated annealing

Synopsis. "void epause(const Branch *const x, const Integer run, const Lint blen, const Lint proposed, const Lint accepted, const Lint failedcnt, const Real t, const Lint n, const Lint iter)"

Description. Save details of current state of simulated annealing to the file whose name is given by "SAVEFNAM", log a message and exit normally. "x" is the current tree, "run" is the ordinal number of the current run (counting from 0), "blen" is the best length so far, "proposed" is the number of changes proposed at the current temperature, "accepted" is the number of changes accepted at the current temperature, "failedcnt" is the consecutive number of changes just before the current temperature at which "maxaccept" changes were not accepted, "t" is the current temperature, "n" is the ordinal number of the current temperature (counting from 0) and "iter" is the ordinal number of the current tree (counting from 0). Assumes the random number generator "uni" has been seeded by calling "rinit", external matrix may be passed to "rowfree", external "nbranches" is the number of branches per tree and external "logfp" is open for writing.

Source file. "epause.c"

6.2.19 "f2str": copy file to string

Synopsis. "char *f2str(FILE *const stream)"

Description. Return new string containing a copy of file read from "stream", with newline added before terminating null byte. Assumes external "logfp" is

open for writing. Exits with an error message if file is more than "ULONG_MAX" - 3 machine characters long, memory required exceeds "MAX_ALLOC" bytes or allocation fails.

Notes. The string returned may be deallocated using the standard library function "free".

Source file. "fops.c"

6.2.20 "getlmask": get branch length mask

Synopsis. "Boolean *getlmask(const Branch *const tree, const Branchno root)"

Description. Return pointer to static array "TRUE" where the corresponding branch of tree "tree" (of root and outgroup "root") may have length under an accelerated transformation reconstruction of character state change and for "root", and "FALSE" elsewhere. Assumes external "nbranches" is the number of branches per tree, externals "matrix->m" and "matrix->n" are the number of characters and objects respectively, external "tbinmat" is a transposed encoded version of the data matrix and external "logfp" is open for writing.

Source file. "parsim.c"

6.2.21 "getminlen": get minimum tree length

Synopsis. "Integer getminlen(const Dataptr matrix)"

Description. Return minimum length of any tree based on "matrix". Assumes external "logfp" is open for writing. Exits with an error message if any column of the matrix contains more than "MAXSTATES" different character states.

Source file. "datops.c"

6.2.22 "getparam": get configurable parameters

Synopsis. "Params *getparam(void)"

Description. Return pointer to configurable parameters set to values supplied by user in the file whose name is given by "RCFNAM", using defaults where necessary; though "ognam" is set, "og" is always "D_OG" in the structure whose address is returned. Assumes external "logfp" is open for writing. Exits with a message on error.

Notes. Pointers should not be freed explicitly.

Source file. "getparam.c"

6.2.23 "getplen: get tree length"

Synopsis. "Lint getplen(const Branch *const tree, const Branchno root)"

Description. Return minimum number of character state changes on tree "tree" (of root "root"). Assumes external "nbranches" is the number of branches per tree, externals "matrix->m" and "matrix->n" are the number of characters and objects respectively, external "tbinmat" is a transposed encoded version of the data matrix and external "logfp" is open for writing.

Source file. "parsim.c"

6.2.24 "getroot": find root of tree

Synopsis. "Branchno getroot(const Branch *const barray)"

Description. Return root branch number for tree stored in "barray". Assumes external "nbranches" is the number of branches in the tree and external "logfp" is open for writing. Exits with an error message if no root is found.

Source file. "trops.c"

6.2.25 "getstatev": get list of states for character

Synopsis. "getstatev(const Dataptr matrix, const Charno k)"

Description. Return internal static string containing one instance of each character state (machine character) in column "k" of "matrix->row". Assumes external "logfp" is open for writing. Exits with an error message if column "k" contains more than "MAXSTATES" character states.

Notes. String returned is overwritten on subsequent calls, and should not be freed explicitly.

Source file. "datops.c"

6.2.26 "hillclimb": hill climbing

Synopsis. "Lint hillclimb(const Branch *const inittree, Branchno root, const Lint maxtrees, FILE *const lenfp, const char *const titleprec)"

Description. Use hill climbing to optimize classification tree "inittree" (of root "root"), looking at maxtrees trees. Log statistics to "lenfp" if it is not "NULL", preceding column titles with "titleprec" then a space if "titleprec" has length greater than zero. Leave best tree(s) on best tree stack, and return best length or 0 - best length if stopped prematurely. Assumes external "nbranches" is the number of branches in the tree, external "matrix->m" and "matrix->n" are the number of biological characters and objects respectively, the best tree stack has been initialized by calling "bstinit" and, on restart, restored by calling "bstrestore" (e.g. from the function "restart"), the random number generator "uni" has been seeded by calling "rinit", and external "logfp" is open for writing.

Source file. "solve.c"

6.2.27 "main": LVB

Synopsis. "main()"

Description. Run LVB.

Notes. Exits using the standard library function "exit" rather than a return statement. An exit status of 0 indicates successful execution.

Source file. "main.c"

6.2.28 "makebin": get integer-encoded matrix

Synopsis. "Uint **makebin(const Dataptr matrix)"

Description. Return new, transposed version of the data matrix, in which each different character state for a given biological character is represented by a different power of 2. Assumes external "logfp" is open for writing. See Section 6.1.4.

Notes. Allocates for the whole matrix in contiguous memory, unless the space required would exceed "MAX_ALLOC" bytes or allocation fails, in which case it allocates for each row of the returned matrix individually. Pointers should not be freed explicitly.

Source file. "makebin.c"

6.2.29 "matchange": alter data matrix

Synopsis. "void matchange(Dataptr matrix, const Params rcstruct, const Boolean verbose)"

Description. "If rcstruct.matchchar" is not the null byte, change instances of "rcstruct.matchchar" in "matrix->row[1]" and subsequent rows to the character state in the corresponding column of "matrix->row[0]"; cut columns containing any character in the string "rcstruct.badstates"; remove constant columns. Details are logged if "verbose" is "TRUE". Assumes external "logfp" is open for writing. Exits with an error message if less than "MIN_M" columns remain. If any change is made, the value of "matrix->row" is changed and "matrix->m" is updated.

Notes. If changed, the rows and array of rows may still be deallocated by calling rowfree.

Source file. "datops.c"

6.2.30 "matrin": read data matrix

Synopsis. "Dataptr matrin(const char *const fnam)"

Description. Return a pointer to a data matrix read from the file whose name is given by "fnam". Assumes external "logfp" is open for writing. Checks the matrix format and file carefully, and exits with a message on error.

Notes. The array of pointers to rows and the rows themselves may be deallocated by calling "rowfree". No other pointers should be freed explicitly.

Source file. "datops.c"

6.2.31 "memfree": free memory to prevent leaks

Synopsis. "void memfree(void)"

Description. Free the stack of pointers to memory created by calling "mempush". Pointers are freed in reverse order of their registration with "mempush".

Notes. Called on program exit, to prevent perceived leaks of dynamic memory retained in allocation across function calls.

Source file. "mem.c"

6.2.32 "mempush": register memory to prevent leaks

Synopsis. "Lint mempush(void *const mem)"

Description. Push "mem" on to a static dynamic stack of pointers to memory to be deallocated by calling "memfree" (on exit, to prevent leaks), and return the offset of the pushed pointer from the bottom of the stack. "memfree" will free pointers in reverse order of registration with "mempush". Assumes external "logfp" is open for writing. Exits with an error message on internal allocation failure.

Source file. "mem.c"

6.2.33 "mutate": mutate tree

Synopsis. "Branchno mutate(Branch *const desttree,
const Branch *const sourcetree, Branchno root)"

Description. Make a copy of the tree "sourcetree" in "desttree", with one change in topology caused by moving a random leaf to a random, different position, and return the root number of the new tree. Assumes the random number generator "uni" has been initialized by calling "rinit", external "nbranches" gives the number of branches per tree, "desttree" is big enough and external "logfp" is open for writing.

Source file. "trops.c"

6.2.34 "nextnonwspc": find next non-white space in string

Synopsis. "char *nextnonwspc(const char *string)"

Description. Return pointer to next non-white space machine character in string "string", or "NULL" if none is found. The string may be of any length.

Source file. "sops.c"

6.2.35 "objreroot": reroot tree on object

Synopsis. "Branchno objreroot(Branch *const barray, const Branchno oldroot,
const Objno newrobj)"

Description. Change root of tree barray from "oldroot" to the branch whose object is "newrobj", and return the number of the new root branch. Assumes external "nbranches" gives the number of branches in "barray" and external

"logfp" is open for writing.

Source file. "trops.c"

6.2.36 "pausenow": check if it is time to pause

Synopsis. "Boolean pausenow(void)"

Description. Return "TRUE" if the user has indicated a desire to pause by placing a readable text file whose name is given by "PAUSEFNAM" in the current directory, "FALSE" if not.

Source file. "com.c"

6.2.37 "randomwalk": random walk

Synopsis. "Lint randomwalk(const Branch *const inittree, Branchno root, const Lint maxtrees, FILE *const lenfp, const char *const titleprec)"

Description. Use a random walk to optimize classification tree "inittree" (of root "root"), looking at "maxtrees" trees. Log statistics to "lenfp" if it is not "NULL", preceding column titles with "titleprec" then a space if "titleprec" has length greater than zero. Leave best tree(s) on best tree stack, and return best length or 0 - best length if stopped prematurely. Assumes external "nbranches" is the number of branches in the tree, external "matrix->m" and "matrix->n" are the number of biological characters and objects respectively, the best tree stack has been initialized by calling "bstinit" and, on restart, restored by calling "bstrestore" (e.g. from the function "restart"), the random number generator "uni" has been seeded by calling "rinit", and external "logfp" is open for writing.

Source file. "solve.c"

6.2.38 "randpint": get random non-negative integer

Synopsis. "randpint(const Integer upper)"

Description. Return an integer in the range 0 to "upper" inclusive. Assumes "upper" is positive and the random number generator "uni" has been initialized by calling "rinit".

Source file. randpint.c

6.2.39 "randtree": get random tree

Synopsis. "void randtree(Branch *const barray)"

Description. Fill "barray" with a bifurcating tree of random topology, where "barray[0]" is the root and leaves have objects numbered from 0. Assumes the random number generator "uni" has been initialized by calling "rinit" and external "matrix->n" is the number of leaves required.

Notes. The root is a leaf from which the tree may be conveniently traversed. Interior branches have children and a parent but no object, the root has left and right children and an object but no parent, and other leaves have a parent and an object but no children. Absent elements of branches are "UNSET", which is negative.

Source file. "trops.c"

6.2.40 "restart": get restart information

Synopsis. "struct finnegan restart(void)"

Description. Return restart details, and restore the best tree stack, using information in the save file whose name is given by "SAVEFNAM". Assumes the best tree stack has been initialized with a call to "bstinit", the user has not tampered with the files "save", "data" or "params", external "nbranches" is the number of branches per tree and external "logfp" is open for writing.

Source file. "epause.c"

6.2.41 "rinit": seed random number generator

Synopsis. "void rinit(int ijkl)"

Description. Seed the random number generator "uni" with "ijkl", which must be at least 0 and no more than "MAX_SEED". "rinit" should be called once, before the first call to "uni".

Notes. Supplied by the Edinburgh Parallel Computing Centre (Section A.3).

Source file. "myuni.c"

6.2.42 "rowfree": deallocate matrix rows

Synopsis. void rowfree(Dataptr matrix)

Description. Deallocate the 2-dimensional array "matrix->row" and set it to "NULL", or do nothing if it is "NULL" already. "matrix->n" must give the number of rows.

Source file. "datops.c"

6.2.43 "salloc": string memory allocation

Synopsis. "char *salloc(const Integer len, const char *const msg)"

Description. Return pointer to enough uninitialized memory to store a string of length "len" machine characters (i.e. "len" + 1 bytes), to be used for the object mentioned in the string "msg". Assumes external "logfp" is open for writing. Exits with an error message based on "msg" on failure.

Notes. The memory to which a pointer is returned may be deallocated using the standard library function "free".

Source file. "sops.c"

6.2.44 "scream": log a warning

Synopsis. "void scream(const char *const fmt, ...)"

Description. Log message consisting of "ERROR: " followed by the message given in the argument list then newline. The argument list must be acceptable to the standard library function "vfprintf", i.e. consist of a string followed by 0 or more associated variables which should only be pointers or of type "int", "unsigned int" or "double". Assumes external "logfp" is open for writing.

Source file. "err.c"

6.2.45 "settraps": set signal traps

Synopsis. "void settraps(void)"

Description. If they are defined, trap any of the signals "SIGFPE", "SIGHUP", "SIGINT" and "SIGTERM" with functions that exit with an error message. In the error message, the signals are assumed to have meanings similar to 'arithmetic error', 'hangup', 'interrupt' and 'termination request' respectively. Success of registration is not checked. Assumes external "logfp" is open for writing. "settraps" is called once, early in the function "main".

Source file. "signal.c"

6.2.46 "stopnow": check if it is time to stop

Synopsis. "Boolean stopnow(void)"

Description. Return "TRUE" if the user has indicated a desire to stop by placing a readable text file whose name is given by "STOPFNAM" in the current directory, "FALSE" if not.

Source file. "com.c"

6.2.47 "strcicmp": case-insensitive string comparison

Synopsis. "Integer strcicmp(const char *const s1, const char *const s2)"

Description. Compare strings "s1" and "s2" in a case-insensitive way, returning 0 if they are identical and non-zero otherwise. The strings may be no longer than "MAX_INTEGER" machine characters long. Either or both strings may have zero length. Strings differing in length are always considered different.

Source file. "sops.c"

6.2.48 "supper": convert string to upper case

Synopsis. "char *supper(char *const s)"

Description. Convert lower case letters in string "s" to upper case; return "s". The string may be of any length.

Source file. "sops.c"

6.2.49 "treealloc": tree allocation

Synopsis. "Branch *treealloc(void)"

Description. Return pointer to space for a new tree, with all structure members initialized to "UNSET", or return "NULL" if allocation fails. Assumes external "nbranches" is the number of branches required.

Notes. Pointer should not be freed explicitly.

Source file. "trops.c"

6.2.50 "treeclear": clear tree

Synopsis. "void treeclear(Branch *const barray)"

Description. Set all structure members in "barray" to "UNSET". Assumes external "nbranches" is the number of branches in "barray".

Source file. "trops.c"

6.2.51 "treecmp": compare trees

Synopsis. "Integer treecmp(const Branch *const tree_1, const Branchno root_1, const Branch *const tree_2, const Branchno root_2)"

Description. Return 0 if the topology of trees "tree_1" and "tree_2" (of roots "root_1" and "root_2" respectively) are the same, or non-zero if different. Assumes external "nbranches" is the number of branches per tree, external "matrix->n" is the number of leaves per tree and external "logfp" is open for writing.

Source file. "trops.c"

6.2.52 "treecopy": copy tree

Synopsis. "void treecopy(Branch *const dest, const Branch *const source)"

Description. Copy tree from "source" to "dest". Assumes external "nbranches" gives the number of branches per tree and "dest" is big enough.

Source file. "trops.c"

6.2.53 "treedump": dump tree

Synopsis. "void treedump(FILE *const stream, const Branch *const tree)"

Description. Dump tree "tree" to file stream as a tab-delimited table of integers with column headings, in which each row is one branch. The dumped tree may be read by calling "trundump". Assumes external "nbranches" is the number of branches in tree and external "logfp" is open for writing. Exits with a message on file error.

Source file. "trops.c"

6.2.54 "treeprint": print tree

Synopsis. "void treeprint(FILE *const stream, const Branch *const barray, const Branchno root, const Boolean *const haslen)"

Description. Write tree "barray" (of root "root") as bracketed 'Newick Standard' text (Section 4.1) to file "stream". If "haslen" is "NULL", the tree is written in unrooted bifurcating form. Otherwise, "haslen" is an array "FALSE" where branches are to be collapsed and "TRUE" elsewhere, and the tree is written in rooted form with clades collapsed as indicated. Assumes external "logfp" is open for writing. Exits with a message on file error.

Source file. "trops.c"

6.2.55 "treeswap": swap trees

Synopsis. "treeswap(Branch **const tree1, Branchno *const root1, Branch *const tree2, Branchno *const root2)"

Description. Swap trees pointed to by "tree1" and "tree2" and their roots pointed to by "root1" and "root2" respectively.

Source file. "trops.c"

6.2.56 "trundump": read dumped tree

Synopsis. "void trundump(FILE *const stream, Branch *const barray)"

Description. Read a tree written as text by "treedump" into "barray" from "stream", re-rooting where necessary to ensure "barray[0]" is the root. Assumes external "nbranches" is the number of branches in the tree, external "logfp" is open for writing and "barray" is big enough to hold the tree. Exits with a message on data format or file error.

Source file. "trops.c"

6.2.57 "uni": get random number

Synopsis. "Uni_float uni(void)"

Description. Return a pseudo-random number in the range 0.0 to 1.0 inclusive. Assumes the seed has been set by calling "rinit".
Notes. Supplied by the Edinburgh Parallel Computing Centre (Section A.3).

Source file. "myuni.c"

Chapter 7. COMPILATION AND SUPPORT

7.1 Compiling LVB for your computer

The source code is written in ANSI (ISO) C (Kernighan & Ritchie 1988), and is in the ASCII character set. A version in the ISO invariant code character set is available from the author (Section 7.2).

The most important thing to ensure when compiling is that the constant "MAX_ALLOC", defined in "lvb.h", has a suitable value for your standard library's function "malloc" (Section 2.1). If its value is too high, this will not be detected at compile time, and will lead to unpredictable problems when LVB runs.

LVB will work if compiled with an ANSI C compiler (unless one of the data types it requires is not available, see below). Compilers whose features are an extension of the ANSI standard should also be acceptable. Non-ANSI compilers, however, may not be able to compile LVB, and if they can compile it the limits of variables may differ from those given in this manual, with undefined effects at run-time.

Avoid compiler optimisations that might alter standard arithmetic or the return values of standard functions. Such optimisations would have no noticeable effect on LVB's speed in any case. The most time-consuming parts of the program are those using integer operations to calculate tree length. Since the function central to this, "ssupdate" in the file "parsim.c", is called once per biological character every time tree length is calculated, there may be a speed advantage if it the compiler expands it to inline code. This generally will not happen by default. "ssupdate" is only called by functions in the same file, so global optimisation is not required (though it may bring other advantages). Compilers vary so much that precise instructions on how to ensure "ssupdate" is inlined cannot be given here.

If a data type required by LVB is not obviously available, it will detect this when it is being compiled. The compiler will give a message including the phrase "LVB WARNING: suitable data type not found", and LVB will fail to compile. Anyone encountering this should contact the author for a solution (Section 7.2). The problem is so rare that I have never known it occur.

Defining "IAMDANIEL" when compiling will cause error messages written to the log file to include notice of the latest system error, which may not be recent or have anything to do with the problem. This is not recommended for

general use.

7.2 Availability and support

LVB itself is available on the Internet (Section A.1), and also by e-mail or on disk from the author (see below). While stocks last, this book is available from the author or through book shops.

Users may register to receive LVB news as it becomes available (Section A.1), though this is not compulsory. LVB will be fully supported at least until the end of September 1998. Problems will usually be solved within 48 hours of my knowing about them.

I am particularly keen to know of problems with, or comments on, the following: compilation of LVB, portability, efficiency, the matrix input format, use of LVB with other programs, and suggestions for new features.

Questions and opinions may be given in writing to:

Daniel Barker,
Biocomputing Research Unit,
Institute of Cell and Molecular Biology,
University of Edinburgh,
King's Buildings,
Mayfield Road,
Edinburgh
EH9 3JR
United Kingdom

or, at least until the end of April 1999, by e-mail to the author, sokal@holyrood.ed.ac.uk.

Appendix A. INTERNET SITES

Addresses of some Internet sites that might be useful to LVB users are given below.

A.1 LVB

"http://www.icmb.ed.ac.uk/sokal.html". C source code that can be compiled by the user is available here. A ready to run version for the Power Macintosh will be available before the end of 1997. A ready to run version for IBM-compatible PCs might appear during 1997 or 1998. It is also possible to e-mail the author from this address, and register to receive LVB news as it becomes available. Answers to frequent questions and notes of bugs will be posted here when it becomes clear what (if any) the questions and bugs are.

A.2 PHYLIP

"http://evolution.genetics.washington.edu/phylip.html". Ready-to-run versions for various microcomputers and C source code that can be compiled by the user are available here. See Appendix B for recommended changes to the code for using PHYLIP with LVB. People in Europe may be able to download the files more quickly from the Scottish mirror site, "ftp://ftp.bioss.sari.ac.uk/pub/phylogeny/phylip/". For details see "http://evolution.genetics.washington.edu/phylip/getmeeurope.html". The Scottish site is also available through this address, but more slowly.

A.3 Edinburgh Parallel Computing Centre

"http://www.epcc.ed.ac.uk/". Much information useful to those interested in high-performance computing is available here, including course notes, times of courses, and technical reports. The course and notes 'Introduction to Computational Science: Monte Carlo Methods' have been especially useful for developing LVB. 'Introduction to Genetic Algorithms' was also helpful.

Appendix B. USING PHYLIP WITH LVB

PHYLIP (Section A.2) is a free package of many programs for reconstructing phylogeny and related tasks. Various PHYLIP programs work well with LVB. "drawgram" and "drawtree" can display and print rooted and unrooted trees respectively, and "consense" can make a consensus if LVB finds more than one equally parsimonious tree (but see the warning in Section 4.1 about making majority-rule consensus trees from LVB's rooted trees).

However, by default, some of these programs' limits are lower than

those of LVB. Trees produced by LVB may exceed the PHYLIP limits, with undefined effects. Recompiling PHYLIP 3.572c after making the changes to its source code described below should avoid any trouble with "drawgram", "drawtree" and "consense", providing you have no more than 8191 objects in your trees (Section B.1).

B.1 Modifying drawgram and drawtree

Line 4 of the file "drawgraphics.h" and line 10 of the file "drawtree.c" define the maximum number of machine characters in an object name. The value is originally 30:

```
#define maxnch                  30
```

To raise this to LVB's own limit of 120 characters, change the line in both files to:

```
#define  maxnch                 120     /* CHANGED */
```

Line 3 of the file "drawgraphics.h" defines a number which is divided by 4 to give the maximum number of objects allowed. The value is originally 1200:

```
#define  maxnodes               1200
```

This limits the number of objects in trees handled by "drawgram" and "drawtree" to 300. To increase the maximum number of objects to 8191, change the line to:

```
#define  maxnodes               32764   /* CHANGED */
```

If there are more than 8191 objects, more extensive changes (not given here) must be made before using "drawgram" or "drawtree".

B.2 Modifying "consense"

Line 9 of the file consense.c defines the maximum number of machine characters in an object name. The value is originally 20:

```
#define  nmlngth                20      /* max. no. of
```

```
        characters in species name */
```

(The two lines above fit on to one line in the file.) To raise this to LVB's own limit of 120, change the line to:

```
        #define  nmlngth                120     /* CHANGED */
```

Appendix C. SOURCE CODE

LVB source code is available in electronic form on the Internet (Section A.1) and from the author (Section 7.2). It is given here for reference and to permanently fix the meaning of 'LVB 1.0'.

The program has been written from scratch by the author, apart from the random number functions contained in the file "myuni.c". This file is based on "uni.c", kindly supplied by the Edinburgh Parallel Computing Centre.

The full program text follows. Files appear below in the following order: "lvb.h", "myuni.h", "main.c", "bstack.c", "cleanup.c", "com.c", "datops.c", "epause.c", "err.c", "fops.c", "getparam.c", "makebin.c", "mem.c", "mops.c", "myuni.c", "parsim.c", "randpint.c", "signal.c", "solve.c", "sops.c", "trops.c". Within a file, function declarations are in approximate alphabletical order of function name, but function definitions are in no particular order.

C.1 "lvb.h"

```
/* ********** lvb.h - main header for lvb ********** */

#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <math.h>
#include <signal.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

```
#include "myuni.h"

/* the program */
#define PROGNAM "lvb"                   /* program file name */
#define VERSION "1.0"                   /* version of program */
#define SUBVERSION "24 July 1997"       /* very precise version of version */

/* value that may have to be decreased to suit some versions of malloc() */
#define MAX_ALLOC (~((size_t) 0))       /* max. bytes contiguous allocation */

/* values some people may feel the urge to change */
#define CHECK_INTERVAL ((Lint) 5000)    /* check stop/pause every ... trees */
#define COMMENT '%'                     /* comment line in config. file */
#define FORBIDDEN_CHRS ",:;()[]`'\""    /* not allowed in matrix file */
#define LVB_FNAMSIZE 128                /* maximum bytes for file names */
#define MAX_EQUIVALENT 32               /* max. eq. st. lists & sts per list */
#define STACKBLOCK 16                   /* block sz for (re)alloc. of stacks */
#define UNSET (-1)                      /* value of integral vars when unset */
#define UNSETNAM "none"                 /* value for names if unset */

/* implementation-independent limits */
#define MIN_MAX_SEED 32767      /* min. max. rand. no. seed on any system */
#define MIN_M 1                 /* min. no. of characters for any analysis */
#define MAX_N 16383             /* to allow 14-bit packed object numbers */
#define MIN_N 4                 /* min. no. of objs, for rearrangeable tree */
#define MAX_TITLEPRECLEN 32     /* max. length of titleprec string */
#define MAX_HNAMLEN 15          /* max. length of method name string */

/* limits that reassure the user */
#define MAX_M 32766             /* max. columns in data matrix */
#define MAX_TITLELEN 120        /* max. length of each object name string */

/* unchangeable types */
typedef unsigned char Byte;             /* byte for packed objects */
typedef enum { FALSE, TRUE } Boolean;   /* boolean digit */

/* types that might be changeable in a later version */
typedef int Branchno;           /* branch no. (array offset, count or UNSET) */
typedef int Charno;             /* biol. char. no. (offset, count or UNSET) */
typedef int Integer;            /* general purpose signed integer */
typedef int Objno;              /* object no. (array offset, count or UNSET) */
typedef double Real;            /* general purpose floating point variable */
typedef unsigned Uint;          /* general purpose unsigned integer */
```

```
/* get long signed integral type of at least 32 bits */
#if INT_MAX < 2147483647
typedef long Lint;      /* int not big enough, use long */
#define MAX_LINT ((long) LONG_MAX)
#else
typedef int Lint;       /* int is big enough, use int */
#define MAX_LINT INT_MAX
#endif  /* if INT_MAX < 2147483647 */

/* limits that can vary with system or typedefs */
#define MAX_INTEGER INT_MAX             /* maximum value for Integer */
/* max. states per character */
#define MAXSTATES ((Integer) sizeof(Uint) * CHAR_BIT)

/* matrix and associated information */
typedef struct data
{
        char **row;             /* array of row strings */
        Charno m;               /* number of columns */
        Objno n;                /* number of rows */
        char **rowtitle;        /* array of row title strings */
} *Dataptr;

/* branch of tree */
typedef struct
{
        Branchno parent;        /* parent branch number, UNSET in root */
        Branchno left;          /* child 1 number */
        Branchno right;         /* child 2 number */
        Objno object;           /* object number if leaf, otherwise UNSET */
} Branch;

/* configurable parameters */
typedef struct
{
        char *hnam;             /* all: heuristic method name */
        char matchchar;         /* all: indicates 'same state as in 1st row' */
        char *titleprec;        /* all: to precede stats title lines */
        char *badstates;        /* all: ignore cols containing these symbols */
        char **equivalent;      /* all: lists of equivalent character states */
        char *ognam;            /* all: outgroup object's name */
        Objno og;               /* all: outgroup object number */
        int seed;               /* all: seed for random number generator */
        Integer verbose;        /* all: verboseness level */
```

```
        Integer runs;           /* all: number of runs */
        Lint trees;             /* H.C., R.W.: changes to consider */
        Real t0;                /* S.A.: initial temperature */
        Real t1;                /* S.A.: second temperature */
        Lint maxaccept;         /* S.A.: max. changes at one temperature */
        Lint maxpropose;        /* S.A.: max. changes considered at a temp. */
        Lint maxfail;           /* S.A.: max. consecutive temps w/ no change */
} Params;

/* restart details */
struct finnegan
{
        long bst_trees;         /* number of best trees */
        long run;               /* current run */
        long blen;              /* best length */
        long proposed;          /* changes proposed at this temperature */
        long accepted;          /* changes accepted at this temperature */
        long failedcnt;         /* consecutive temps with no accepted change */
        long n;                 /* ordinal number of this temperature */
        Real t;                 /* current temperature */
        long iter;              /* current tree number */
        long seed;              /* random number seed */
        Branch *x;              /* current tree */
};

/* fixed file names */
#define LOCKFNAM "lock"         /* lock message file */
#define LOGFNAM "log"           /* progress log file */
#define MATFNAM "data"          /* matrix file name */
#define PAUSEFNAM "pause"       /* pause execution semaphore file name */
#define RCFNAM "params"         /* run-time configuration parameters file */
#define SAVEFNAM "save"         /* save file generated on pausing */
#define STOPFNAM "stop"         /* stop semaphore file name */

/* file name bases (run-specific suffixes will be used) */
#define LENFNAM "stat"          /* current tree and length file name prefix */
#define RESFNAM "res"           /* results file name prefix */
#define SUMFNAM "sum"           /* summary of trees per run file name */
#define TREE1FNAM "ini"         /* initial tree file name prefix */

/* method names */
#define SAHNAM "anneal"         /* simulated annealing */
#define HCHNAM "hillclimb"      /* hill climbing */
#define RWHNAM "randomwalk"     /* random walk */
```

```
#define D_HNAM SAHNAM           /* default method */

/* constants used by all methods */
#define NO_OGNAM UNSETNAM       /* unrooted tree */
#define D_RUNS 1                /* number of runs */
#define D_EQUIVALENT '\0'       /* no state is equivalent to another */
#define D_OG UNSET              /* outgroup object number */
#define D_OGNAM NO_OGNAM        /* outgroup object name */
#define D_BADSTATES ""          /* use all columns */
#define D_MATCHCHAR '\0'        /* states aren't relative to first row */
#define D_TITLEPREC '\0'        /* to precede all lines of stats titles */
#define D_VERBOSE 1             /* verbose output */

/* default hill climbing and randomwalk parameter */
#define D_TREES ((Lint) 20000)  /* number of trees to examine */

/* default simulated annealing parameters */
#define D_T0 ((Real) 1.0)               /* initial temperature */
#define D_T1 ((Real) 0.9)               /* second temperature */
#define D_MAXACCEPT ((Lint) 50)         /* max. changes at a temperature */
#define D_MAXPROPOSE ((Lint) 5000)      /* max. proposals at a temperature */
#define D_MAXFAIL ((Lint) 2)            /* max. temperatures with no change */

/* data-hiding macro to access integer-encoded matrix */
#define binmat(i, j) tbinmat[j][i]

/* LVB global functions */
void *alloc(const size_t bytes, const char *const msg);
Lint anneal(const Branch *const inittree, Branchno root, const Real t0,
 const Real t1, const Lint maxaccept, const Lint maxpropose,
 const Lint maxfail, const Integer run, FILE *const lenfp,
 const char *const titleprec, Lint lenbest, Lint proposed,
 Lint accepted, Lint failedcnt, Real t, Lint n, Lint iter);
Branchno brcnt(void);
void bstclear(void);
Lint bstcnt(void);
Lint bstdump(FILE *const outfp);
void bstfree(void);
void bstinit(Params rcstruct);
Lint bstprint(FILE *const outfp);
Integer bstpush(const Branch *const barray, const Branchno root);
void bstrestore(FILE *const stream, const struct finnegan reparams);
Branchno childadd(Branch *const tree, const Branchno destination,
 const Branchno newchild);
```

```
void cleanup(void);
void clnclose(FILE *const fp, const char *const fnam);
FILE *clnopen(const char *const nam, const char *const mod);
void clnremove(const char *const fnam);
void crash(const char *const fmt, ...);
void epause(const Branch *const x, const Integer run, const Lint blen,
 const Lint proposed, const Lint accepted, const Lint failedcnt, const Real t,
 const Lint n, const Lint iter);
char *f2str(FILE *const stream);
Boolean *getlmask(const Branch *const tree, const Branchno root);
Lint getminlen(const Dataptr matrix);
Params *getparam(void);
Lint getplen(const Branch *const tree, const Branchno root);
Branchno getroot(const Branch *const barray);
char *getstatev(const Dataptr matrix, const Charno k);
Lint hillclimb(const Branch *const inittree, Branchno root,
 const Lint maxtrees, FILE *const lenfp, const char *const titleprec);
Uint **makebin(const Dataptr matrix);
void matchange(Dataptr matrix, const Params rcstruct, const Boolean verbose);
Dataptr matrin(const char *const fnam);
void memfree(void);
Lint mempush(void *const mem);
Branchno mutate(Branch *const desttree, const Branch *const sourcetree,
 Branchno root);
char *nextnonwspc(const char *string);
Branchno objreroot(Branch *const barray, const Branchno oldroot,
 const Objno newrobj);
Boolean pausenow(void);
void randtree(Branch *const barray);
Lint randomwalk(const Branch *const inittree, Branchno root,
 const Lint maxtrees, FILE *const lenfp, const char *const titleprec);
Integer randpint(const Integer upper);
struct finnegan restart(void);
void rowfree(Dataptr matrix);
char *salloc(const Integer len, const char *const msg);
void scream(const char *const fmt, ...);
void settraps(void);
Branch *treealloc(void);
Boolean stopnow(void);
Integer strcicmp(const char *const s1, const char *const s2);
char *supper(char *const s);
void treeclear(Branch *const barray);
void treecopy(Branch *const dest, const Branch *const source);
Integer treecmp(const Branch *const tree_1, const Branchno root_1,
```

```
 const Branch *const tree_2, Branchno root_2);
void treedump(FILE *const stream, const Branch *const tree);
Branch *treepop(void);
void treeprint(FILE *const stream, const Branch *const barray,
 const Branchno root, const Boolean *const haslen);
void treeswap(Branch **const tree1, Branchno *const root1,
 Branch **const tree2, Branchno *const root2);
void trundump(FILE *const stream, Branch *const barray);
```

C.2 "myuni.h"

```
/* ********** myuni.h - header for RNG functions in myuni.c ********** */

#include <float.h>
#include <limits.h>

/* check double is big enough for uni() */
#if DBL_MANT_DIG < 24                   /* too small */
#error LVB WARNING: suitable data type not found for Uni_float
#else                                   /* big enough */
typedef double Uni_float;
#endif

/* set max. random number seed value suitable for both machine and rinit() */
#if 900000000 < INT_MAX
#define MAX_SEED 900000000      /* rinit() max. is less than int type max. */
#else
#define MAX_SEED INT_MAX        /* use max. value that will fit in int type */
#endif  /* if 900000000 < INT_MAX */

/* external uni functions */
Uni_float uni(void);
void rinit(int ijkl);
```

C.3 "main.c"

```
/* ********** main.c - LVB ********** */

#include "lvb.h"
```

```
/* external variables for the whole program */
FILE *logfp = NULL;             /* log file */
Dataptr matrix = NULL;          /* data matrix */
Uint **tbinmat = NULL;          /* transposed, integer-encoded matrix */
Branchno nbranches = 0;         /* number of branches in each tree */

static void getsoln(const Params rcstruct, struct finnegan *reparams);
static void logstim(void);
static void logtree1(const Branch *const barray, const Integer run,
 const Boolean isrestart);
static void setlock(void);
static void smessg(const Boolean isrestart, const Integer run);
static void writeinf(Params prms);
static Params setog(Params rcstruct);

main()
{
        const char *logmod;             /* log file opening mode */
        FILE *savefp;                   /* save file with restart info. */
        Params rcstruct;                /* configurable parameters */
        Boolean rstflag;                /* execution is a restart */
        Boolean verbstup;               /* verbose start-up */
        Boolean manual_cleanup = TRUE;  /* call cleanup() manually at end */
        struct finnegan reparams;       /* restart details */

        setlock();
        if (atexit(cleanup) == 0)
                manual_cleanup = FALSE;

        /* decide if this is a restart according to presence of save file */
        if ((savefp = fopen(SAVEFNAM, "r")) != NULL)    /* is restart */
        {
                rstflag = TRUE;
                verbstup = FALSE;
                logmod = "a";
                clnclose(savefp, SAVEFNAM);
        }
        else                            /* isn't restart */
        {
                rstflag = FALSE;
                verbstup = TRUE;
                logmod = "w";
        }
```

```
/* global files: part 1 */
logfp = fopen(LOGFNAM, logmod);         /* can't use clnopen() here */
if (logfp == NULL)
{
        fprintf(stderr, "FATAL ERROR: %s: cannot open log file\n",
         PROGNAM);
        exit(1);
}
clnclose(stdin, "stdin");       /* no standard input */

settraps();
rcstruct = *(getparam());

/* entitle log file */
fprintf(logfp, "This is %s version %s %s\n", PROGNAM, VERSION,
 SUBVERSION);

/* global files: part 2 */
clnclose(stderr, "stderr");
clnclose(stdout, "stdout");

matrix = matrin(MATFNAM);       /* get data matrix */
nbranches = brcnt();
rcstruct = setog(rcstruct);     /* get outgroup object number */

bstinit(rcstruct);              /* set up best tree stack */
if (rstflag == FALSE)   /* echo configurable information */
        writeinf(rcstruct);
else                    /* get restart info. */
        reparams = restart();
matchange(matrix, rcstruct, verbstup);  /* cut columns from matrix */

tbinmat = makebin(matrix);              /* set up encoded matrix */
fprintf(logfp, "Minimum theoretical tree length for this matrix is "
"%ld\n", (long) getminlen(matrix));
if (rstflag == FALSE)   /* not a restart */
{
        logstim();
        rinit(rcstruct.seed);
        getsoln(rcstruct, NULL);
}
else                    /* is a restart */
{
        fprintf(logfp, "Resuming\n");
```

```
                logstim();
                fprintf(logfp, "Seed for random number generator (based on "
                 "last random number) = %d\n", (int) reparams.seed);
                rinit((int) reparams.seed);
                getsoln(rcstruct, &reparams);
        }
        rowfree(matrix);
        if (manual_cleanup == TRUE)
                cleanup();
        exit(0);

}       /* end main() */

static void setlock(void)
/* lock directory, or crash with message to stderr if lock already present;
 * N.B. only ever writes to stderr */
{
        FILE *lockfp;   /* lock file */

        lockfp = fopen(LOCKFNAM, "r");  /* can't use clnopen() here */
        if (lockfp != NULL)     /* directory is locked */
        {
                fprintf(stderr, "%s: FATAL ERROR: ", PROGNAM);
                fprintf(stderr, "%s apparently already running in ", PROGNAM);
                fprintf(stderr, "current directory.\n");
                fprintf(stderr, "if not, delete file %s and restart\n",
                 LOCKFNAM);
                exit(1);
        }

        /* set lock: make a file of name LOCKFNAM containing 1 space only */
        lockfp = fopen(LOCKFNAM, "w");
        if (lockfp == NULL)
        {
                fprintf(stderr, "FATAL ERROR: %s: cannot create lock file\n",
                 PROGNAM);
                exit(1);
        }
        if (fputc(' ', lockfp) == EOF)
        {
                fprintf(stderr, "FATAL ERROR: %s: cannot write to lock file\n",
                 PROGNAM);
                exit(1);
        }
```

```
        fclose(lockfp);         /* don't check value, failure is not serious */

        return;

}       /* end setlock() */

static void logstim(void)
/* log start time with message, if possible */
{
        time_t stim;    /* time */

        stim = time(NULL);
        if (stim != -1)         /* time is available */
                fprintf(logfp, "Start time: %s", ctime(&stim));
        return;

}       /* end logstim() */

static void writeinf(Params prms)
/* write initial details to log file */
{
        Integer i;      /* loop counter */

        /* heuristic method details */
        fprintf(logfp, "Search method used, method = %s\n", prms.hnam);
        if (strcicmp(prms.hnam, SAHNAM) == 0)
        {
                fprintf(logfp, "Initial temperature, t0 = %g\n",
                 (double) prms.t0);
                fprintf(logfp, "Second temperature, t1 = %g\n",
                 (double) prms.t1);
                fprintf(logfp, "Maximum changes accepted before "
                 "decreasing temperature, maxaccept = %ld\n",
                 (long) prms.maxaccept);
                fprintf(logfp, "Maximum changes considered before "
                 "decreasing temperature, maxpropose = %ld\n",
                 (long) prms.maxpropose);
                fprintf(logfp, "Maximum consecutive temperatures with no "
                 "change accepted before\n"
                 "ending cooling, maxfail = %ld\n", (long) prms.maxfail);
        }
        else if ((strcicmp(prms.hnam, HCHNAM) == 0)
         || (strcicmp(prms.hnam, RWHNAM) == 0))
                fprintf(logfp, "Number of trees to examine, trees = %ld\n",
```

```
         (long) prms.trees);
else    /* impossible condition */
        crash("internal error: unknown method '%s'", prms.hnam);

/* universally applicable details */
fprintf(logfp, "Seed for random number generator, seed = %d\n",
 prms.seed);
fprintf(logfp, "Runs, runs = %d\n", (int) prms.runs);
fprintf(logfp, "Verboseness, verbose = %d\n", (int) prms.verbose);
fprintf(logfp, "Outgroup, outgroup = %s", prms.ognam);
if (strcicmp(prms.ognam, NO_OGNAM) == 0)
        fprintf(logfp, " (will output unrooted trees)");
putc('\n', logfp);
fprintf(logfp, "'Same as row 1' state, matchchar = ");
if (prms.matchchar == '\0')
        fputs("none\n", logfp);
else
        fprintf(logfp, "%c\n", prms.matchchar);
if (strlen(prms.badstates) > 0)
        fprintf(logfp, "Will ignore columns containing any of: %s\n",
         prms.badstates);
else
        fprintf(logfp, "Not requested to ignore any columns\n");
for (i = 0; prms.equivalent[i][0] != '\0'; ++i)
{
        fprintf(logfp, "Equivalent character states, equivalent ");
        if (prms.equivalent[1][0] != '\0')      /* > 1 list */
                fprintf(logfp, "(%d) ", (int) i + 1);
        fprintf(logfp, "= %s\n", prms.equivalent[i]);
}
if (i == 0)
        fputs("Equivalent character states, equivalent = none\n",
         logfp);
fprintf(logfp, "String to precede titles in statistics files, "
 "titleprec = ");
if (prms.titleprec[0] == '\0')
        fprintf(logfp, "none\n");
else
        fprintf(logfp, "%s\n", prms.titleprec);
fprintf(logfp, "Matrix has %d rows and %d columns\n", (int) matrix->n,
 (int) matrix->m);
if (fflush(logfp) == EOF)
        crash("write error on log file %s", LOGFNAM);
return;
```

```
}       /* end writeinf() */

static Params setog(Params rcstruct)
/* return new version of rcstruct, with outgroup name and number set */
{
        Objno i;        /* loop counter */

        if (strcicmp(rcstruct.ognam, D_OGNAM) == 0)     /* use default og */
                rcstruct.og = D_OG;
        else                            /* user has given the name */
        {
                rcstruct.og = UNSET;
                for (i = 0; i < matrix->n; ++i)
                {
                        if (strcicmp(matrix->rowtitle[i], rcstruct.ognam) == 0)
                                rcstruct.og = i;
                }
                if (rcstruct.og == UNSET)       /* name not found */
                        crash("unknown object '%s' named as outgroup in file\n"
                         "%s", rcstruct.ognam, RCFNAM);
        }
        return rcstruct;

}       /* end setog() */

static void logtree1(const Branch *const barray, const Integer run,
 const Boolean isrestart)
/* log initial tree for run run (in barray) to outfp, using isrestart to
 * determine if this is a restart and choosing the output file name
 * accordingly */
{
        char outfnam[LVB_FNAMSIZE];     /* current file name */
        const char ressfx[] = "r";      /* restart file suffix */
        const Branchno root = 0;        /* after randtree() or trundump() */
        FILE *outfp;                    /* output file */

        /* choose filename according to whether this is a restart or not */
        if (isrestart == TRUE)
                sprintf(outfnam, "%s%d%s", TREE1FNAM, (int) run, ressfx);
        else
                sprintf(outfnam, "%s%d", TREE1FNAM, (int) run);

        /* create tree file */
```

```
        outfp = clnopen(outfnam, "w");
        treeprint(outfp, barray, root, NULL);
        clnclose(outfp, outfnam);

}       /* end logtree1() */

static void getsoln(const Params rcstruct, struct finnegan *reparams)
/* get and output solution(s) according to parameters in rcstruct, if reparams
 * is NULL;
 * if reparams is not NULL, use the restart details it contains as well as
 * parameters in rcstruct for a restart */
{
        char fnam[LVB_FNAMSIZE];        /* current file name */
        const char *smod;               /* mode for opening statistics files */
        Lint blen;                      /* shortest length */
        Integer run;                    /* current run number */
        Lint treec;                     /* number of trees found */
        Lint treelength;                /* length of each tree found */
        const Branchno initroot = 0;    /* initial root of all trees */
        Lint accepted;                  /* changes accepted at current temp. */
        Lint failedcnt;                 /* 'failed' temperatures */
        Lint iter;                      /* ordinal number of current tree */
        Lint n;                         /* ordinal number of current temp. */
        Lint proposed;                  /* changes proposed at current temp. */
        Real t;                         /* current temperature */
        FILE *sumfp;                    /* best length file */
        FILE *lenfp;                    /* current length file */
        FILE *resfp;                    /* results file */
        Branch *tree;                   /* initial tree */
        Boolean stopflag = FALSE;       /* stopped by user */
        Boolean isrestart;              /* this is a restart */

        if (reparams != NULL)   /* is a restart */
        {
                smod = "a";
                isrestart = TRUE;
                run = (Integer) reparams->run;
                blen = (Lint) reparams->blen;
                proposed = (Lint) reparams->proposed;
                accepted = (Lint) reparams->accepted;
                failedcnt = (Lint) reparams->failedcnt;
                t = (Real) reparams->t;
                n = (Lint) reparams->n;
                iter = (Lint) reparams->iter;
```

```
        tree = reparams->x;
}
else                    /* not a restart */
{
        smod = "w";
        isrestart = FALSE;
        run = 0;
        blen = (Lint) UNSET;
        proposed = (Lint) UNSET;
        accepted = (Lint) UNSET;
        failedcnt = (Lint) UNSET;
        t = (Real) UNSET;
        n = (Lint) UNSET;
        iter = 0;
        tree = treealloc();
        if (tree == NULL)
                crash("out of memory: cannot allocate for initial "
                 "tree");
}

/* open and entitle statistics file shared by all runs */
sumfp = clnopen(SUMFNAM, smod);
if (isrestart == FALSE)
{
        if (rcstruct.titleprec[0] != '\0')
                fprintf(sumfp, "%s ", rcstruct.titleprec);
        fprintf(sumfp, "Run\tInitLen\tBestLen\tTrees\n");
}
for (; run < rcstruct.runs; ++run)
{
        smessg(isrestart, run);

        /* open statistics output files */
        sprintf(fnam, "%s%d", LENFNAM, (int) run);
        if (rcstruct.verbose > 0)
                lenfp = clnopen(fnam, smod);
        else
                lenfp = NULL;

        if (isrestart == FALSE)         /* is start of run */
        {
                randtree(tree);
                bstclear();
                /* start run's entry in sum file */
```

```
        fprintf(sumfp, "%d\t%ld\t", (int) run,
         (long) getplen(tree, 0));
}
logtree1(tree, run, isrestart);

/* find solution(s) */
if (strcicmp(rcstruct.hnam, SAHNAM) == 0)
        treelength = anneal(tree, initroot, rcstruct.t0,
         rcstruct.t1, rcstruct.maxaccept, rcstruct.maxpropose,
         rcstruct.maxfail, run, lenfp, rcstruct.titleprec,
         blen, proposed, accepted, failedcnt, t, n, iter);
else if (strcicmp(rcstruct.hnam, HCHNAM) == 0)
        treelength = hillclimb(tree, initroot,
         rcstruct.trees, lenfp, rcstruct.titleprec);
else if (strcicmp(rcstruct.hnam, RWHNAM) == 0)
        treelength = randomwalk(tree, initroot,
         rcstruct.trees, lenfp, rcstruct.titleprec);
else
        crash("internal error: unknown method in params");

if (treelength < 0)     /* stopped by user */
{
        treelength = -(treelength);
        stopflag = TRUE;
}

/* close closable statistics files */
sprintf(fnam, "%s%d", LENFNAM, (int) run);
if (lenfp != NULL)
        clnclose(lenfp, fnam);

/* output solution and its details */
sprintf(fnam, "%s%d", RESFNAM, (int) run);
resfp = clnopen(fnam, "w");
treec = bstprint(resfp);
clnclose(resfp, fnam);
fprintf(sumfp, "%ld\t%ld\n", (long) treelength, (long) treec);

/* won't use length summary file until end of next run */
if (fflush(sumfp) == EOF)
        crash("file write error on file %s", SUMFNAM);

if (stopflag == TRUE)
        fprintf(logfp, "Early stop requested in run %d\n",
```

```
                        (int) run);
                fprintf(logfp, "Ending run %d\n", (int) run);
                if (fflush(logfp) != 0)
                        crash("write error on log file");
                if (stopflag == TRUE)
                {
                        clnremove(STOPFNAM);
                        break;
                }

                /* start next run as normal: it is never a restart */
                reparams = NULL;
                isrestart = FALSE;
        }
        clnclose(sumfp, SUMFNAM);
        return;

}       /* end getsoln() */

static void smessg(const Boolean isrestart, const Integer run)
/* print start or continuation of run message as appropriate */
{
        static Boolean normmsg = FALSE;         /* must give normal message */
        extern FILE *logfp;                     /* log file */

        /* give appropriate message */
        if (normmsg == FALSE)
        {
                if (isrestart == TRUE)
                {
                        fprintf(logfp, "Continuing run %d\n",
                         (int) run);
                }
                else
                        fprintf(logfp, "Beginning run %d\n", (int) run);
        }
        else
                fprintf(logfp, "Beginning run %d\n", (int) run);

        if ((fflush(logfp) != 0) || (ferror(logfp) != 0))
                        crash("file error on file %s", LOGFNAM);
        normmsg = TRUE;
        return;
```

```
}       /* end smessg() */
```

C.4 "bstack.c"

```
/* ********** bstack.c - maintain stack of best trees ********** */

#include "lvb.h"

#define CLOSEBR 0100    /* 'close parentheses' */
#define END 00          /* 'terminator (";\n\0")' */
#define LEASTBITS 6     /* number of least significant bits */
#define NUMBER 0300     /* 'number follows' */
#define OPENBR 0200     /* 'open parentheses' */
#define PART1 077       /* least significant bits of number */
#define PART2 (~PART1)  /* most significant bits of number */

/* write byte chr the advance write position pos */
#define sputc(pos, chr) *pos++ = (Byte) (chr), ++written

/* write least sig. 14 bits of i into 2 bytes and adv. write pos. pos by 2 */
#define sputno(pos, i) \
sputc(pos, NUMBER | (i & PART1)), \
sputc(pos, (unsigned) (i & PART2) >> LEASTBITS)

/* unpack integer at read position pos into i & advance pos by 1 */
#define getsno(i, pos) i = *pos & PART1, ++pos, i = i | (*pos << LEASTBITS)

/* get flag bits of byte */
#define flag(chr) (chr & ~PART1)

/* N.B. The 2 flag bits of a byte of a packed tree as read are CLOSEBR, OPENBR,
 * END or NUMBER. NUMBER indicates that a 14-bit object number follows: the
 * rest of the byte contains the LEASTBITS least significant bits, and the next
 * byte contains the most significant bits of the object number. With the other
 * flag bits, the rest of the byte is irrelevant, and nothing should be assumed
 * about the next byte. */

static void dopush(const Branch *const barray, const Branchno root);
static Byte *treepack(Byte *const str, const Branch *const barray,
 const Branchno root);
static Byte *packalloc(void);
static Byte *poppack(void);
```

```
static void trunpack(Branch *const barray, const Byte *const packtree);
static void upsize(void);

static Integer trspc = 0;       /* allocation per tree */
static Byte **stack = NULL;     /* stack of packed best trees */
static Lint next = (Lint) 0;    /* next unused element of stack */
static Lint size = (Lint) 0;    /* total trees held in stack */
static Objno outgrp = UNSET;    /* set to og if rooted output desired */
static Boolean rooted = FALSE;  /* tree should be output rooted */

static Byte *treepack(Byte *const str, const Branch *const barray,
const Branchno root)
/* send tree in barray (of root root) to memory pointed to by str,
 * in packed and 'unrooted' form;
 * return new print position in str */
{
        static Byte *s;                         /* current write position */
        static Boolean doneabsroot = FALSE;     /* have output root of tree */
        Branchno obj;                           /* current object */
        static Integer written = 0;             /* bytes written */

        obj = barray[root].object;
        if (doneabsroot == FALSE)       /* pack whole tree */
        {
                /* start tree */
                s = str;
                sputc(s, OPENBR);
                sputno(s, obj);
                doneabsroot = TRUE;

                treepack(s, barray, barray[root].left);
                treepack(s, barray, barray[root].right);

                /* end tree */
                sputc(s, CLOSEBR);
                sputc(s, END);

                /* check packed tree is expected size (it always will be) */
                if (written != trspc)
                        crash("internal error: expected packed tree of %d "
                         "bytes but have written %d bytes", (int) trspc,
                         (int) written);

                /* clean up for next time */
```

```
                doneabsroot = FALSE;
                written = 0;
        }
        else    /* pack remainder of tree */
        {
                if (obj != UNSET)       /* leaf */
                        sputno(s, obj);
                else
                {
                        sputc(s, OPENBR);       /* start subtree */
                        treepack(s, barray, barray[root].left);
                        treepack(s, barray, barray[root].right);
                        sputc(s, CLOSEBR);      /* end subtree */
                }
        }
        return s;

}       /* end treepack() */

Lint bstcnt(void)
/* return number of trees on best stack */
{
        return next;

}       /* end bstcnt() */

static void trunpack(Branch *const barray, const Byte *const packtree)
/* unpack the tree packtree into the array of branches barray, ensuring
 * barray[0] is the root branch */
{
        const Byte *pos = packtree;     /* current position in packed tree */
        Branchno current = 0;           /* current branch number */
        Branchno nextbr = 0;            /* next unused branch number */
        Branchno obj;                   /* current object number */
        const Branchno root = 0;        /* root of unpacked trees */

        treeclear(barray);

        /* start tree with root at branch 0 */
        if (*pos != OPENBR)
                crash("internal error: first byte of packed tree\n"
                 "is 0%o but should be 0%o", (int) *pos, (int) OPENBR);
        ++pos;
        getsno(obj, pos);
```

```
        barray[current].object = obj;
        ++pos;
        ++nextbr;

        for (; *pos != END; ++pos)
        {
                switch flag(*pos)
                {
                case OPENBR:
                        childadd(barray, current, nextbr);
                        current = nextbr;
                        ++nextbr;
                        break;
                case CLOSEBR:
                        if (current != root)
                                current = barray[current].parent;
                        break;
                case NUMBER:
                        getsno(obj, pos);
                        barray[nextbr].object = obj;
                        childadd(barray, current, nextbr);
                        ++nextbr;
                        break;
                default:        /* impossible condition */
                        crash("internal error: byte 0%o of packed tree has\n"
                         "unknown flag bits", (int) *pos);
                        break;
                }
        }
        if (current != 0)       /* impossible condition */
                crash("internal error: have not returned to root at end of "
                 "unpacking tree");

        return;

}       /* end trunpack() */

void bstinit(Params rcstruct)
/* set things up for later use of best stack, based on matrix and rcstruct */
{
        extern Dataptr matrix;  /* data matrix */

        trspc = matrix->n * 2 - 4;      /* OPENBRs and CLOSEBRs */
        trspc += matrix->n * 2;         /* packed object numbers */
```

```
        ++trspc;                        /* END */

        outgrp = rcstruct.og;
        if (outgrp != UNSET)
                rooted = TRUE;
        else
                rooted = FALSE;
        return;

}       /* end bstinit() */

static Byte *packalloc(void)
/* return pointer to new memory for a packed tree in the best stack */
{

        return (Byte *) alloc((size_t) trspc, "packed tree");

}       /* end packalloc() */

void bstrestore(FILE *const stream, const struct finnegan reparams)
/* add reparams.bst_trees dumped trees to the stack by reading stream;
 * N.B. the best tree stack must have been initialized already by calling
 * bstinit() */
{
        Lint i;                 /* loop counter */
        static Branch *barray = NULL;   /* current restored tree */

        if (barray == NULL)     /* 1st call, allocate memory */
        {
                barray = treealloc();
                if (barray == NULL)
                        crash("out of memory: cannot allocate for unpacked\n"
                         "current tree to restore best tree stack");
        }
        for (i = 0; i < (Lint) reparams.bst_trees; ++i)
        {
                trundump(stream, barray);
                dopush(barray, 0);      /* trundump() ensures root is 0 */
        }
        return;

}       /* end bstrestore() */

Integer bstpush(const Branch *const barray, const Branchno root)
```

```
/* push tree in barray (of root root) on to the best stack only if its
 * topology is not already stored there;
 * return 1 if it is pushed, or 0 if not */
{
        Lint i;                                 /* loop counter */
        static Branch *stacktree = NULL;        /* current tree on stack */

        if (stacktree == NULL)  /* 1st call, allocate memory */
        {
                stacktree = treealloc();
                if (stacktree == NULL)
                        crash("out of memory: cannot allocate for unpacked\n"
                         "current tree in stack");
        }

        /* return before push if not a new topology */
        for (i = 0; i < next; ++i)
        {
                trunpack(stacktree, stack[i]);
                if (treecmp(stacktree, 0, barray, root) == 0)
                        return 0;
        }

        /* topology is new so must be pushed */
        dopush(barray, root);
        return 1;

}       /* end bstpush() */

static void dopush(const Branch *const barray, const Branchno root)
/* push tree in barray (of root root) on to best stack, in packed form */
{
        if (next >= size)     /* want more memory for stack */
                upsize();
        treepack(stack[next], barray, root);
        ++next;
        return;

}       /* end dopush() */

Lint bstprint(FILE *const outfp)
/* print all trees on stack to file outfp, according to value of static rooted
 * either unrooted or rooted with branches collapsed where necessary;
 * return number of trees printed */
```

```
{
        Lint cnt = 0;                   /* tree count */
        Boolean *haslen = NULL;         /* rooted: collapse br. where FALSE */
        Branchno root;                  /* root of current tree */
        Byte *packtree;                 /* current packed tree */
        const Objno d_obj1 = 0;         /* 1st obj. for unrooted trees */
        static Branch *barray = NULL;   /* current unpacked tree */

        if (barray == NULL)     /* 1st call, allocate memory */
        {
                barray = treealloc();
                if (barray == NULL)
                        crash("out of memory: cannot allocate for unpacked\n"
                         "current tree to write best trees");
        }
        while ((packtree = poppack()) != NULL)
        {
                trunpack(barray, packtree);
                if (rooted == TRUE)     /* root on outgroup */
                {
                        root = objreroot(barray, 0, outgrp);
                        haslen = getlmask(barray, root);
                }
                else
                        root = objreroot(barray, 0, d_obj1);
                treeprint(outfp, barray, root, haslen);
                ++cnt;
        }
        if (fflush(outfp) != 0)
                crash("file write error when writing best trees");
        return cnt;

}       /* end bstprint() */

Lint bstdump(FILE *const outfp)
/* dump all trees on stack to file outfp, with root as first branch (number 0);
 * return number of trees dumped */
{
        Lint cnt = 0;                   /* tree count */
        Byte *packtree;                 /* current packed tree */
        static Branch *barray = NULL;   /* current unpacked tree */

        if (barray == NULL)     /* 1st call, allocate memory */
        {
```

```
                barray = treealloc();
                if (barray == NULL)
                        crash("out of memory: cannot allocate for current\n"
                         "unpacked tree for dumping best tree stack");
        }
        while ((packtree = poppack()) != NULL)
        {
                trunpack(barray, packtree);
                treedump(outfp, barray);
                ++cnt;
        }
        if (fflush(outfp) != 0)
                crash("file write error when dumping best trees");
        return cnt;

}       /* end bstdump() */

void bstfree(void)
/* free all memory in stack */
{
        Lint i;         /* loop counter */

        for (i = (Lint) 0; i < size; ++i)
                free(stack[i]);
        free(stack);
        next = (Lint) 0;
        size = (Lint) 0;
        stack = NULL;
        return;

}       /* end bstfree() */

void bstclear(void)
/* clear stack and reduce its allocation to STACKBLOCK trees if it is
 * greater */
{
        Lint i;         /* loop counter */

        /* resize stack if bigger than 1 block */
        if (size > (Lint) STACKBLOCK)
        {
                /* free trees in stack */
                for (i = (Lint) STACKBLOCK; i < size; ++i)
                        free(stack[i]);
```

```
                /* resize stack itself */
                stack = (Byte **) realloc(stack, STACKBLOCK * sizeof(Byte *));
                if (stack == NULL)
                        crash("memory problem: cannot decrease allocation\n"
                         "for best tree stack from %u to %d trees",
                         (unsigned) size, STACKBLOCK);
                size = STACKBLOCK;
        }

        next = (Lint) 0;        /* clear stack */

}       /* end bstclear() */

static Byte *poppack(void)
/* pop compressed tree from stack in compressed form;
* or return NULL if stack is empty */
{
        --next;
        if (next < (Lint) 0)
        {
                next = (Lint) 0;
                return NULL;
        }
        else
                return stack[next];     /* next push will overwrite */

}       /* end poppack() */

static void upsize(void)
/* increase allocation for best tree stack */
{
        Lint i;         /* loop counter */

        size = size + STACKBLOCK;

        /* allocate for stack itself */
        if (stack == NULL)   /* 1st call, stack does not exist */
        {
                stack = (Byte **) alloc((size_t) size * sizeof(Byte *),
                 "initial best tree stack");
                next = 0;
        }
        else
        {
```

```
                if (((unsigned long) size * (unsigned long) sizeof(Byte *))
                 > ((unsigned long) MAX_ALLOC))
                        crash("cannot increase allocation for best tree\n"
                         "stack to %u elements because the allocation limit\n"
                         "of %u bytes would be exceeded", (unsigned) size,
                         (unsigned) MAX_ALLOC);
                stack = (Byte **) realloc(stack,
                 (size_t) size * sizeof(Byte *));
                if (stack == NULL)
                        crash("out of memory: cannot increase allocation for\n"
                         "best tree stack to %u elements", (unsigned) size);
        }

        /* allocate for trees within stack */
        for (i = next; i < size; ++i)
                stack[i] = packalloc();

        return;

}       /* end upsize() */
```

C.5 "cleanup.c"

```
/* ********** cleanup.c - prepare for exit ********** */

#include "lvb.h"

void cleanup(void)
/* log end time, prevent memory leaks, unlock directory */
{
        extern FILE *logfp;             /* log file */
        time_t endtim;                  /* time at end of run */
        Boolean writerr = FALSE;        /* write error on log file */
        extern Dataptr matrix;          /* data matrix */

        /* log end time, if possible */
        endtim = time(NULL);
        if (logfp != NULL)
        {
                if (endtim != -1)
                        fprintf(logfp, "End time: %s", ctime(&endtim));
                else
```

```
                        fputs("End of execution\n", logfp);
        }

        /* log file won't be used again */
        if ((ferror(logfp) != 0) || (fclose(logfp) != 0))
                writerr = TRUE;

        /* prevent leaks */
        bstfree();
        memfree();

        /* if error found on log file, leave lock file to indicate disaster */
        if (writerr == FALSE)
                remove(LOCKFNAM);       /* unlock directory */
        return;

}       /* end cleanup() */
```

C.6 "com.c"

```
/* ********** com.c - user communication functions ********** */

#include "lvb.h"

Boolean pausenow(void)
/* return TRUE if a readable text file meaning 'pause now' exists, or FALSE if
 * not */
{
        FILE *fp;       /* semaphore file */

        if ((fp = fopen(PAUSEFNAM, "r")) != NULL)
        {
                clnclose(fp, PAUSEFNAM);
                return TRUE;
        }
        else
                return FALSE;

}       /* end pausenow() */

Boolean stopnow(void)
/* return TRUE if a readable text file meaning 'stop now' exists, or FALSE if
```

```
 * not */
{
        FILE *fp;       /* semaphore file */

        if ((fp = fopen(STOPFNAM, "r")) != NULL)
        {
                clnclose(fp, STOPFNAM);
                return TRUE;
        }
        else
                return FALSE;

}       /* end stopnow() */
```

C.7 "datops.c"

```
/* ********** datops.c - data matrix functions ********** */

#include "lvb.h"

static void badchar(const Dataptr matrix, const char *const badchars,
 Boolean *const togo, const Boolean verbose);
static void checkmsz(const Lint m, const char *const fnam);
static void checknames(const Dataptr matrix, const char *const fnam);
static void checknsz(const Objno n, const char *const fnam);
static void checktlen(const Integer len, const char *const fnam);
static Charno colcnt(FILE *const stream, const char *const fnam);
static void constchar(const Dataptr matrix, Boolean *const togo,
 const Boolean verbose);
static void cr_ilt(const Objno i, const char *const fnam);
static void cr_mlow(const Charno m, const char *const fnam);
static void cr_nhi(const Objno n, const char *const fnam);
static void cr_nlow(const Objno n, const char *const fnam);
static void cr_nsam(const Dataptr matrix, const char *const fnam,
 const Objno i, const Objno j);
static void cr_obs(const char *const fnam);
static void cr_rne(const char *const fnam, const Objno row, const Charno currm,
 const Charno prevm);
static void cr_sfe(const char *const fnam);
static void cr_tlenhi(const Integer len, const char *const fnam);
static void cr_tnr(const Objno i, const char *const fnam);
static void cr_mhi(const Lint m, const char *const fnam);
```

```
static void cr_ntnl(const char *const fnam);
static Charno cutcols(Dataptr matrix, const Boolean *const tocut);
static void cutmsg(const Boolean *const togo, const Charno m,
 const char *const msg);
static void equivchar(Dataptr matrix, char **equivarr);
static void logcut(const Boolean *const cut, const Charno m);
static Dataptr matalloc(const Objno n);
static void relabs(Dataptr matrix, const char matchchar);
static Objno rowcnt(FILE *stream, const char *const fnam);
static char *rowread(FILE *const stream, const char *const fnam,
 const Charno m);
static FILE *skipspace(FILE *const stream, const char *const fnam);
static void testneof(const int c, const char *const fnam);
static void testnforbid(const int c, const char *const fnam);
static void testnueof(const int c, const char *const fnam);
static void testnnz(const int c, const char *const fnam);
static Boolean *tgarralloc(const Integer m);
static char *titleread(FILE *stream, const char *const fnam);

Lint getminlen(const Dataptr matrix)
/* return minimum length of any tree based on matrix */
{
        Lint minlen = 0;        /* return value */
        char *statev;           /* list of states in current character */
        Charno k;               /* loop counter */

        for (k = 0; k < matrix->m; ++k)
        {
                statev = getstatev(matrix, k);
                minlen += (Lint) strlen(statev) - (Lint) 1;
        }
        return minlen;

}       /* end getminlen() */

Dataptr matrin(const char *const fnam)
/* return a new matrix read from file called fnam,
 * or crash verbosely on error;
 * push all new memory on to the general memstack, except that for row strings
 * and the array of row strings (since these may be freed by rowfree) */
{
        char *rowstr;           /* current row string */
        char *titlestr;         /* current title string */
        FILE *stream;           /* input file */
```

```
        Objno i;                /* loop counter */
        Charno m = UNSET;       /* columns in input matrix */
        Charno currentm;        /* columns in current row of matrix */
        Objno n;                /* rows in input matrix */
        Dataptr mat;            /* new data matrix */

        stream = clnopen(fnam, "r");
        n = rowcnt(stream, fnam);
        checknsz(n, fnam);

        /* read rows and row titles into memory */
        for (i = 0; i < n; ++i)
        {
                titlestr = titleread(stream, fnam);
                if (titlestr == NULL)   /* no row follows title on same line */
                        cr_tnr(i, fnam);
                currentm = colcnt(stream, fnam);
                if (currentm != m)
                {
                        if (m == UNSET)         /* alright, not got m yet */
                        {
                                m = currentm;
                                mat = matalloc(n);
                        }
                        else                    /* unequal row lengths */
                                cr_rne(fnam, i, currentm, m);
                }
                rowstr = rowread(stream, fnam, m);

                /* save strings in matrix data structure */
                mat->row[i] = rowstr;
                mat->rowtitle[i] = titlestr;
        }

        clnclose(stream, fnam);

        /* record dimensions */
        mat->m = m;
        mat->n = n;

        checknames(mat, fnam);
        return mat;

}       /* end matrin() */
```

```
void rowfree(Dataptr matrix)
/* free memory used for row strings and array of row strings in matrix,
 * and make the array of row title strings NULL;
 * or, if the array of row title strings is already NULL, do nothing */
{
        Objno i;        /* loop counter */

        if (matrix->row != NULL)
        {
                for(i = 0; i < matrix->n; ++i)
                        free(matrix->row[i]);
                free(matrix->row);
                matrix->row = NULL;
        }
        return;

}       /* end rowfree() */

static void checkmsz(const Lint m, const char *const fnam)
/* check m is in the interval [MIN_M, MAX_M] inclusive */
{
        if (m < (Lint) MIN_M)
                cr_mlow(m, fnam);
        else if (m > (Lint) MAX_M)
                cr_mhi(m, fnam);
        return;

}       /* end checkmsz() */

static void checknames(const Dataptr matrix, const char *const fnam)
/* check all row titles in matrix are different and not equal to UNSETNAM,
 * considered case-insensitively;
 * crash verbosely if they are not */
{
        Objno i;        /* loop counter */
        Objno j;        /* loop counter */

        for (i = 0; i < matrix->n; ++i)
        {
                if (strcicmp(matrix->rowtitle[i], UNSETNAM) == 0)
                        cr_ilt(i, fnam);
                for (j = 0; j < matrix->n; ++j)
                {
```

```
                        if ((i != j) && (strcicmp(matrix->rowtitle[i],
                         matrix->rowtitle[j]) == 0))
                                cr_nsam(matrix, fnam, i, j);
                }
        }

}       /* end checknames() */

static void checknsz(const Objno n, const char *const fnam)
/* check n is in the interval [MIN_N, MAX_N] */
{
        if (n < (Objno) MIN_N)
                cr_nlow(n, fnam);
        else if (n > (Objno) MAX_N)
                cr_nhi(n, fnam);
        return;

}       /* end checknsz() */

static void checktlen(const Integer len, const char *const fnam)
/* check row title length len in file of name fnam does not exceed the limit
 * MAX_TITLELEN */
{
                if (len > MAX_TITLELEN)
                        cr_tlenhi(len, fnam);
                return;

 }      /* end checktlen() */

static Charno colcnt(FILE *const stream, const char *const fnam)
/* count columns of row of matrix proper in stream str (of name fnam);
 * N.B. before return, the read position for stream is restored to its value
 * on calling colcnt */
{
        int inchar = 0;         /* current character of input */
        Lint m = 0;             /* column count */
        fpos_t inipos;          /* file position on function call */

        /* save file position at function call */
        testnnz(fgetpos(stream, &inipos), fnam);

        /* count columns */
        while (inchar != '\n')
        {
```

```
                inchar = getc(stream);
                testneof(inchar, fnam);
                if (!(isspace(inchar)))
                        ++m;
        }

        testnnz(fsetpos(stream, &inipos), fnam);        /* restore file pos. */
        checkmsz(m, fnam);
        return (Charno) m;

}       /* end colcnt() */

static void cr_tnr(const Objno i, const char *const fnam)
/* crash because 'row' i of matrix in the file called fnam does not have both
 * a title and a row proper */
{
        crash("row %d of matrix in file %s either lacks a title, or has a\n"
         "title only. Each row must be preceded by a title on the same line,\n"
         "separated from it by white space.",
          (int) i + 1, fnam);

}       /* end cr_tnr() */

static void cr_ilt(const Objno i, const char *const fnam)
/* crash because row i of matrix has the illegal title UNSETNAM */
{
        crash("row %d of matrix in file %s has the illegal title '%s'.\n"
         "Never use this as a title for any row.",
          (int) i + 1, fnam, UNSETNAM);

}       /* end cr_ilt() */

static void cr_mhi(const Lint m, const char *const fnam)
/* crash because the number of columns in the matrix file called fnam exceeds
 * the upper limit MAX_M */
{
        crash("matrix in file %s has %u columns, which exceeds LVB's upper\n"
         "limit of %d columns", fnam, (unsigned) m, MAX_M);

 }      /* end cr_mhi() */

static void cr_mlow(const Charno m, const char *const fnam)
/* crash because the number of columns m in the matrix in the file called
 * fnam is less than the lower limit MIN_M */
```

```
{
        crash("matrix in file %s has %d columns, which is less than\n"
         "LVB's lower limit of %d columns.", fnam, (int) m, MIN_M);

}       /* end cr_mlow() */

static void cr_nhi(const Objno n, const char *const fnam)
/* crash because the number of rows n in the matrix in the file called fnam
 * exceeds the upper limit MAX_N */
{
        crash("matrix in file %s has %d rows, which exceeds\n"
         "LVB's upper limit of %d rows.", fnam, (int) n, MAX_N);

}       /* end cr_nhi() */

static void cr_nlow(const Objno n, const char *const fnam)
/* crash because the number of rows n in the matrix in the file called fnam is
 * less than the lower limit MIN_N */
{
        crash("matrix in file %s has %d rows, which is less than\n"
         "LVB's lower limit of %d rows.", fnam, (int) n, MIN_N);

}       /* end cr_nlow() */

static void cr_nsam(const Dataptr matrix, const char *const fnam,
 const Objno i, const Objno j)
/* crash because rows i and j of matrix in file named fnam have same title */
{
        crash("row %d and row %d of matrix in file %s have '%s'\n"
         "and '%s' as their titles. Use a different name for every row.",
          (int) i + 1, (int) j + 1, fnam, matrix->rowtitle[i],
          matrix->rowtitle[j]);

}       /* end cr_nsam() */

static void cr_ntnl(const char *const fnam)
/* crash because file of name fnam does not end in '\n' */
{
        crash("final line in file %s is not terminated. The final line must\n"
         "end in the usual way before end of file.", fnam);

}       /* end cr_ntnl() */

static void cr_obs(const char *const fnam)
```

```
/* crash because of obscure error reading matrix in file of name fnam */
{
        crash("error reading matrix in file %s.", fnam);

}       /* end cr_obs() */

static void cr_rne(const char *const fnam, const Objno row, const Charno currm,
 const Charno prevm)
/* crash because row row in file called fnam has currm columns, but previous
 * rows have prevm columns */
{
        crash("matrix in file %s has unequal row lengths: row %d has\n"
         "%d columns but previous rows have %d columns.", fnam, (int) row + 1,
         (int) currm, (int) prevm);

}       /* end cr_rne() */

static void cr_sfe(const char *const fnam)
/* crash because of unexpected file error on input stream */
{
        crash("file error on stream for matrix file %s.", fnam);

}       /* end cr_sfe() */

static void cr_tlenhi(const Integer len, const char *const fnam)
/* crash because length of row title, len, exceeds upper limit MAX_TITLELEN */
{
        crash("row title too long (%d machine characters) in file %s:\n"
         "limit is %d machine characters", (int) len, fnam, MAX_TITLELEN);

}       /* end cr_tlenhi() */

static Dataptr matalloc(const Objno n)
/* return pointer to new matrix, with n pointers to rows and row titles;
 * the strings in these arrays, and the array of state counts, are not
 * allocated for and are initialized to NULL;
 * new memory for matrix struct itself and array of pointers is pushed on to
 * the general memstack */
{
        Objno i;        /* loop counter */
        Dataptr mat;    /* new data matrix */
        char **l_row;   /* new array of rows */

        /* matrix struct itself */
```

```
        mat = (Dataptr) alloc(sizeof(struct data), "matrix structure");
        mempush(mat);

        /* array for row title strings */
        mat->rowtitle = (char **) alloc((size_t) (n + 1) * sizeof(char *),
         "pointers to row title strings");
        mempush(mat->rowtitle);

        /* array for row strings (local variable so matrix->row NULL until can be freed) */
        l_row = (char **) alloc((size_t) (n + 1) * sizeof(char *),
         "pointers to row strings");

        /* initialize unallocated pointers to NULL */
        for (i = 0; i <= n; ++i)
        {
                mat->rowtitle[i] = NULL;
                l_row[i] = NULL;
        }

        /* initialize scalars to zero */
        mat->m = 0;
        mat->n = 0;

        mat->row = l_row;       /* now can be freed e.g. by rowfree() from cleanup() */

        return mat;

}       /* end matalloc() */

static Objno rowcnt(FILE *stream, const char *const fnam)
/* count rows of matrix in file stream (of name fnam);
 * crash verbosely if last character in file is not '\n';
 * N.B. before return, the read position for stream will be restored to its
 * value on calling rowcnt() */
{
        int inchar;                     /* current character of input */
        int oldchar = EOF;              /* previous character of input */
        fpos_t start;                   /* initial position */
        Objno n = 0;                    /* row count */
        Boolean fulline = FALSE;        /* line contains some non-wht space */

        testnnz(fgetpos(stream, &start), fnam);         /* save file pos. */
        stream = skipspace(stream, fnam);
        while ((inchar = fgetc(stream)) != EOF)
```

```
        {
                if (!(isspace(inchar)))
                        fulline = TRUE;         /* non-empty line */
                if (inchar == '\n')
                {
                        if (fulline == TRUE)
                                ++n;
                        fulline = FALSE;        /* for next line */
                }
                oldchar = inchar;
        }
        if (ferror(stream))     /* 'EOF' above indicated error */
                cr_sfe(fnam);
        if (oldchar != '\n')
                cr_ntnl(fnam);

        testnnz(fsetpos(stream, &start), fnam);         /* rewind */
        return n;

}       /* end rowcnt() */

static char *rowread(FILE *const stream, const char *const fnam,
 const Charno m)
/* read and return m-column row from stream (of filename fnam),
 * converting all cells read to upper case;
 * leave stream so next read gets character after row-terminating newline;
 * N.B. the new string is not pushed on to the general memstack */
{
        char *rowstr;           /* row */
        int inchar;             /* current character of input */
        Charno cells = 0;       /* cells read */

        rowstr = salloc(m, "matrix row strings");

        /* get row text */
        for (;;)
        {
                inchar = getc(stream);
                testnueof(inchar, fnam);
                testnforbid(inchar, fnam);
                if (cells > m)  /* should not happen */
                        cr_obs(fnam);
                if (!(isspace(inchar))) /* is matrix cell */
                {
```

```
                        rowstr[cells] = (char) inchar;
                        ++cells;
                }
                else if (inchar == '\n')        /* row terminator */
                {
                        if (cells != m)         /* should not happen */
                                cr_obs(fnam);
                        else                    /* have read whole row */
                                break;
                }
        }

        rowstr[m] = '\0';       /* terminate string */
        supper(rowstr);         /* make upper case */

        return rowstr;

}       /* end rowread() */

static FILE *skipspace(FILE *const stream, const char *const fnam)
/* move to next non-white space character of file stream (of name fnam);
 * return stream, or crash verbosely if EOF encountered */
{
        int inchar = ' ';       /* current character of input */

        /* read up to and including first non-white space byte */
        while (isspace(inchar))
        {
                inchar = getc(stream);
                testnueof(inchar, fnam);
        }

        /* push back the first non-white space byte so it will be read next */
        testneof(ungetc(inchar, stream), fnam);

        return stream;

}       /* end skipspace() */

static void testneof(const int c, const char *const fnam)
/* crash with message about file stream error in file called fnam if c is
 * EOF */
{
        if (c == EOF)
```

```
                cr_sfe(fnam);

}       /* end testneof() */

static void testnforbid(const int c, const char *const fnam)
/* crash verbosely if c is a forbidden character, i.e. '\0' or one of the
 * characters in the string FORBIDDEN_CHRS */
{
        if (c == '\0')
                crash("forbidden machine character in file %s: null "
                 "character", fnam);
        if (strchr(FORBIDDEN_CHRS, c) != NULL)

                crash("forbidden machine character in file %s. The following\n"
                 "are not allowed:\n"
                 "%s\n"
                 "and the null character.", fnam, FORBIDDEN_CHRS);

}       /* end testnforbid() */

static void testnueof(const int c, const char *const fnam)
/* crash with message about unexpected end of file in file called fnam if c is
 EOF */
{
        if (c == EOF)
                crash("unexpected end of file when reading file %s.", fnam);

}       /* end testnueof() */

static void testnnz(const int c, const char *const fnam)
/* crash with message about error on file called fnam if c is not zero */
{
        if (c != 0)
                cr_sfe(fnam);

}       /* end testnnz() */

static char *titleread(FILE *stream, const char *const fnam)
/* read a row title from file stream (of name fnam), and return a new string
 * copy of it, or NULL if it is not followed by a matrix row;
 * push the new string on to the general memstack;
 * advance position in stream to start of associated row */
{
        char *title;            /* title string */
```

```
        int inchar;             /* current character of input */
        Integer len = 0;        /* length of title string */
        fpos_t start;           /* start of relevant input */

        /* find and save start of row title */
        stream = skipspace(stream, fnam);
        testnnz(fgetpos(stream, &start), fnam);

        /* get length of title */
        inchar = 'A';   /* arbitrary non-white space */
        while (!(isspace(inchar)))
        {
                inchar = getc(stream);
                testnueof(inchar, fnam);
                testnforbid(inchar, fnam);
                ++len;
        }
        checktlen(len, fnam);

        /* read title into new memory */
        title = salloc(len, "matrix row title strings");
        mempush(title);
        testnnz(fsetpos(stream, &start), fnam); /* rewind to start of title */
        if (fscanf(stream, "%s", title) != 1)
                cr_sfe(fnam);

        /* move to start of associated row */
        for (inchar = ' '; isspace(inchar); inchar = getc(stream))
        {
                if (inchar == '\n')     /* end of line reached before row */
                        return NULL;
        }
        testnueof(inchar, fnam);        /* end of file reached before row */
        testneof(ungetc(inchar, stream), fnam);         /* 1st byte of row */

        return title;

}       /* end titleread() */

static void relabs(Dataptr matrix, const char matchchar)
/* replace all instances of matchchar in matrix with the character state
 * of the first row for the column where it occurs */
{
        Objno i;                /* loop counter */
```

```
        Charno k;               /* loop counter */

        for (i = 1; i < matrix->n; ++i)
        {
                for (k = 0; k < matrix->m; ++k)
                {
                        if (matrix->row[i][k] == matchchar)
                                matrix->row[i][k] = matrix->row[0][k];
                }
        }
        return;

}       /* end relabs() */

static void equivchar(Dataptr matrix, char **equivarr)
/* replace lists of character states in equivarr each with the first
 * member of the list, where they are found in matrix's row strings */
{
        Integer list;           /* current list */
        Integer listcnt;        /* number of lists */
        Objno i;                /* loop counter */
        Charno k;               /* loop counter */

        /* count lists */
        for (listcnt = 0; equivarr[listcnt][0] != '\0'; ++listcnt)
                ;

        for (list = 0; list < listcnt; ++list)
        {
                for (i = 0; i < matrix->n; ++i)
                {
                        for (k = 0; k < matrix->m; ++k)
                        {
                                if (strchr(equivarr[list],
                                 (int) matrix->row[i][k]) != NULL)
                                        matrix->row[i][k] = equivarr[list][0];
                        }
                }
        }
        return;

}       /* end equivchar */

static void badchar(const Dataptr matrix, const char *const badchars,
```

```
 Boolean *const togo, const Boolean verbose)
/* make sure matrix->m-element array togo is TRUE where rows contain
 * one of the characters in string badchars;
 * log message about which columns will be ignored if verbose is TRUE */
{
        char *msg;              /* message about columns to be ignored */
        Integer badno;          /* index of current element of badchars */
        Objno i;                /* loop counter */
        Charno k;               /* loop counter */
        Integer badcnt;         /* length of badchars */
        Boolean *tocutl;        /* TRUE where original column must go */

        badcnt = (Integer) strlen(badchars);
        tocutl = tgarralloc(matrix->m);         /* local array of cols to go */

        /* identify columns to cut */
        for (badno = 0; badno < badcnt; ++badno)        /* for ea. 'bad' ch. */
        {
                for (i = 0; i < matrix->n; ++i)         /* for each row */
                {
                        for (k = 0; k < matrix->m; ++k)         /* ea. col. */
                        {
                                if (matrix->row[i][k] == badchars[badno])
                                {
                                        togo[k] = TRUE;         /* caller's */
                                        tocutl[k] = TRUE;       /* local */
                                }
                        }
                }
        }

        if (verbose == TRUE)
        {
                msg = (char *) salloc(50 + badcnt,
                 "message about ignored columns");
                strcpy(msg, "Ignoring columns containing any of: ");
                strcat(msg, badchars);
                cutmsg(tocutl, matrix->m, msg);
                free(msg);      /* local 'automatic' memory */
        }
        free(tocutl);           /* 'automatic' memory */
        return;

}       /* end badchar() */
```

```
static void constchar(const Dataptr matrix, Boolean *const togo,
 const Boolean verbose)
/* make sure matrix->m-element array togo is TRUE where matrix column
 * contains only one character state;
 * log details of new columns to ignore if verbose is TRUE */
{
        Charno k;               /* loop counter */
        Objno i;                /* loop counter */
        Boolean *isconst;       /* TRUE where column is constant */

        /* local array of columns to go */
        isconst = tgarralloc(matrix->m);
        for (k = 0; k < matrix->m; ++k)
                isconst[k] = TRUE;

        /* discover variable columns */
        for (i = 1; i < matrix->n; ++i)
        {
                for (k = 0; k < matrix->m; ++k)
                {
                        if (matrix->row[i][k] != matrix->row[0][k])
                                isconst[k] = FALSE;
                }
        }

        /* update togo, for caller */
        for (k = 0; k < matrix->m; ++k)
        {
                if (isconst[k] == TRUE)
                        togo[k] = TRUE;
        }

        if (verbose == TRUE)
                cutmsg(isconst, matrix->m, "Ignoring constant columns");

        free(isconst);  /* 'automatic' memory */
        return;

}       /* end constchar() */

static void cutmsg(const Boolean *const togo, const Charno m,
 const char *const msg)
/* log message that columns for which the m-element array togo is TRUE will be
```

```
 * ignored */
{
        Charno cutcnt = 0;      /* number of columns being ignored */
        Charno k;               /* loop counter */
        extern FILE *logfp;     /* log file */

        fprintf(logfp, "%s\n", msg);
        for (k = 0; k < m; ++k)
        {
                if (togo[k] == TRUE)
                        ++cutcnt;
        }
        if (cutcnt == 0)
                fprintf(logfp, "... none found.\n");
        else
                logcut(togo, m);
        return;

}       /* end cutmsg() */

static Boolean *tgarralloc(const Charno m)
/* return pointer to new m-element array of columns to ingore, with each
 * element initialized to FALSE;
 * does not push the new memory onto the general memstack */
{
        Boolean *togo;  /* new array */
        Charno k;       /* loop counter */

        togo = (Boolean *) alloc((size_t) m * sizeof(Boolean),
         "array of columns to ignore");

        /* initialize all elements to FALSE ('don't ignore') */
        for (k = 0; k < m; ++k)
                togo[k] = FALSE;

        return togo;

}       /* end tgarralloc() */

void matchange(Dataptr matrix, const Params rcstruct, const Boolean verbose)
/* change and remove columns in matrix, partly in response to rcstruct,
 * verbosely or not according to value of verbose */
{
        Boolean *togo;          /* TRUE where column must go */
```

```
        Charno colsgone;        /* columns cut */
        extern FILE *logfp;     /* log file */

        togo = tgarralloc(matrix->m);                   /* columns to ignore */
        equivchar(matrix, rcstruct.equivalent);         /* equivalent states */

        /* relative states to absolute */
        if (rcstruct.matchchar != '\0')
                relabs(matrix, rcstruct.matchchar);

        /* 'bad' states */
        if (strlen(rcstruct.badstates) != (size_t) 0)
                badchar(matrix, rcstruct.badstates, togo, verbose);

        constchar(matrix, togo, verbose);       /* compuslory cut */

        /* N.B. a function to mark autapomorphic characters for cutting
         * could be called at this point. The effect would be more noticable
         * with unrealistically small test matrices than with real data */

        /* cut the cols as indicated, and crash verbosely if too few remain */
        colsgone = cutcols(matrix, togo);       /* make changes to matrix */
        if (matrix->m < MIN_M)
                crash("after columns are ignored, matrix has %d columns,\n"
                 "which is less than LVB's lower limit of %d columns.",
                  (int) matrix->m, MIN_M);
        else
        {
                if (verbose == TRUE)
                        fprintf(logfp, "A total of %d columns will be "
                         "ignored\n", (int) colsgone);
        }

        free(togo);     /* 'automatic' memory */

}       /* end matchange() */

static Charno cutcols(Dataptr matrix, const Boolean *const tocut)
/* remove columns in matrix for which the corresponding element of
matrix->m-element array tocut is TRUE, and update matrix->m;
return the number of columns cut */
{
        char **newrow;                  /* rows of reduced matrix */
        Objno i;                        /* loop counter */
```

```
Charno newm;                    /* new number of columns */
const Charno oldm = matrix->m;  /* old number of columns */
Charno k;                       /* loop counter */
Charno newk;                    /* current column of reduced matrix */

/* count columns that will not be cut */
newm = 0;
for (k = 0; k < matrix->m; ++k)
{
        if (tocut[k] == FALSE)
                ++newm;
}

/* make no change if keeping all columns */
if (newm == matrix->m)
        return 0;

/* memory for new matrix row array */
newrow = (char **) alloc((size_t) (matrix->n + 1) * sizeof(char *),
 "pointers to new row strings");
for (i = 0; i < matrix->n; ++i)
        newrow[i] = salloc(newm, "new row strings");
newrow[matrix->n] = NULL;

newk = 0;
for (k = 0; k < matrix->m; ++k) /* for every column */
{
        if (tocut[k] == FALSE)  /* keep this column */
        {
                for (i = 0; i < matrix->n; ++i) /* fill for ea. row */
                        newrow[i][newk] = matrix->row[i][k];
                ++newk; /* fill next row next time */
        }
}

/* trap impossible condition */
if (newk != newm)
        crash("internal error: expected new matrix to have %d\n"
         "columns but it apparently has %d\n", (int) newm, (int) newk);

/* terminate new row strings */
for (i = 0; i < matrix->n; ++i)
        newrow[i][newk] = '\0';
```

```
        /* update matrix structure */
        rowfree(matrix);
        matrix->row = newrow;
        matrix->m = newm;

        return (oldm - newm);

}       /* end cutcols() */

char *getstatev(const Dataptr matrix, const Charno k)
/* return pointer to string containing 1 instance of each character state in
 * column k of matrix, or crash verbosely if more than MAXSTATES states are
 * found;
 * N.B. string is static and will be overwritten by later calls, and is pushed
 * on to the general memstack */
{
        static char *statev = NULL;     /* array of states */
        Integer statec;                 /* number of states */
        Objno i;                        /* loop counter */

        if (statev == NULL)     /* 1st call, allocate memory */
        {
                statev = salloc(MAXSTATES, "array of states");
                mempush(statev);
        }

        /* clear record of states */
        statev[0] = '\0';
        statec = 0;

        /* update record of states for column k */
        for (i = 0; i < matrix->n; ++i)
        {
                if (strchr(statev, (int) matrix->row[i][k]) == NULL)
                /* new state */
                {
                        statev[statec++] = matrix->row[i][k];
                        if (statec > MAXSTATES)
                                crash("number of character states in column\n"
                                 "%d of matrix exceeds the maximum of %d.\n"
                                 "Use fewer states, or recompile LVB with a\n"
                                 "larger unsigned integeral type for Uint in "
                                 "lvb.h", (int) k + 1, MAXSTATES);
                        statev[statec] = '\0';  /* for strchr() */
```

```
                }
        }

        return statev;

 }      /* end getstatev() */

static void logcut(const Boolean *const cut, const Charno m)
/* log message saying columns for which m-element array cut is TRUE are being
 * cut */
{
        Charno k;                       /* loop counter */
        Integer noperln = 0;            /* no. of numbers on current line */
        const Integer max_noperln = 8;  /* max. numbers written per line */
        Boolean newline = FALSE;        /* last number followed by '\n' */
        extern FILE *logfp;             /* log file */

        fputs("... will ignore column numbers:\n", logfp);

        /* give formatted list of columns to go */
        for (k = 0; k < m; ++k)
        {
                if (cut[k] == TRUE)
                {
                        fprintf(logfp, "%d", (int) k + 1);
                        ++noperln;
                        if (noperln == max_noperln)     /* end line */
                        {
                                noperln = 0;
                                putc('\n', logfp);
                                newline = TRUE;
                        }
                        else    /* just put some space on this line */
                        {
                                putc('\t', logfp);
                                newline = FALSE;
                        }
                }
        }
        if (newline == FALSE)
                putc('\n', logfp);

        if (fflush(logfp) != 0)
                crash("write error on file %s", LOGFNAM);
```

```
        return;

}       /* end logcut() */
```

C.8 "epause.c"

```
/* ********** epause.c - pause and restart execution ********** */

#include "lvb.h"

static void preamble(FILE *const savefp);
static void restcrash(void);

#define ENDAMBLE '!'    /* end-of-preamble mark */

void epause(const Branch *const x, const Integer run, const Lint blen,
 const Lint proposed, const Lint accepted, const Lint failedcnt, const Real t,
 const Lint n, const Lint iter)
/* write save file for pausing of simulated annealing, log a message and exit
 * normally after freeing the matrix row array and rows;
 * parameters are as in anneal() */
{
        FILE *savefp;           /* save file name */
        extern FILE *logfp;     /* log file */
        extern Dataptr matrix;  /* data matrix */

        clnremove(PAUSEFNAM);   /* get rid of pause semaphore for next time */
        savefp = clnopen(SAVEFNAM, "w");
        preamble(savefp);
        fprintf(logfp, "Pause requested in run %d, restart LVB to continue\n",
         (int) run);
        fprintf(savefp, "bst_trees %ld\n", (long) bstcnt());
        fprintf(savefp, "run %ld\n", (long) run);
        fprintf(savefp, "blen %ld\n", (long) blen);
        fprintf(savefp, "proposed %ld\n", (long) proposed);
        fprintf(savefp, "accepted %ld\n", (long) accepted);
        fprintf(savefp, "failedcnt %ld\n", (long) failedcnt);
        fprintf(savefp, "t %g\n", (double) t);
        fprintf(savefp, "n %ld\n", (long) n);
        fprintf(savefp, "iter %ld\n", (long) iter);
        fprintf(savefp, "seed %ld\n", (long) randpint((Integer) MIN_MAX_SEED));
```

```
        /* current tree */
        fprintf(savefp, "current_tree\n");
        treedump(savefp, x);

        /* best trees */
        fprintf(savefp, "best_trees\n");
        bstdump(savefp);

        clnclose(savefp, SAVEFNAM);
        rowfree(matrix);        /* otherwise row array and contents leaked */
        exit(0);

}       /* end epause() */

struct finnegan restart(void)
/* return scalars, current tree and best tree left in save file;
 * fill bstack with the best trees in the save file */
{
        int inchar;                     /* current character input */
        FILE *savefp;                   /* save file */
        struct finnegan reparams;       /* saved values */

        savefp = clnopen(SAVEFNAM, "r");

        /* skip preamble */
        while ((inchar = getc(savefp)) != ENDAMBLE)
        {
                if (inchar == EOF)
                        restcrash();
        }

        /* general scalars */
        if (fscanf(savefp, " bst_trees %ld", &reparams.bst_trees) != 1)
                restcrash();
        if (fscanf(savefp, " run %ld", &reparams.run) != 1)
                restcrash();
        if (fscanf(savefp, " blen %ld", &reparams.blen) != 1)
                restcrash();
        if (fscanf(savefp, " proposed %ld", &reparams.proposed) != 1)
                restcrash();
        if (fscanf(savefp, " accepted %ld", &reparams.accepted) != 1)
                restcrash();
        if (fscanf(savefp, " failedcnt %ld", &reparams.failedcnt) != 1)
                restcrash();
```

```
        if (fscanf(savefp, " t %lg", &reparams.t) != 1)
                restcrash();
        if (fscanf(savefp, " n %ld", &reparams.n) != 1)
                restcrash();
        if (fscanf(savefp, " iter %ld", &reparams.iter) != 1)
                restcrash();
        if (fscanf(savefp, " seed %ld", &reparams.seed) != 1)
                restcrash();

        /* skip string "current_tree" */
        if (fscanf(savefp, "%*s") != 0)
                restcrash();

        reparams.x = treealloc();
        if (reparams.x == NULL)
                crash("out of memory: cannot allocate for current tree on\n"
                 "restart");
        trundump(savefp, reparams.x);

        /* skip string "best_trees" */
        if (fscanf(savefp, "%*s") != 0)
                restcrash();

        bstrestore(savefp, reparams);
        clnclose(savefp, SAVEFNAM);
        clnremove(SAVEFNAM);
        return reparams;

}       /* end restart() */

static void preamble(FILE *const savefp)
/* write preamble, ending in ENDAMBLE then '\n', to save file savefp */
{
        fprintf(savefp, "PAUSED EXECUTION SAVE FILE FOR %s %s %s\n",
         PROGNAM, VERSION, SUBVERSION);
        fprintf(savefp, "Leave this file, unaltered, where you found it,\n");
        fprintf(savefp, "and LVB will continue where it left off next time\n");
        fprintf(savefp, "it is run in the same directory. If you would\n");
        fprintf(savefp, "prefer LVB to begin again from scratch, delete\n");
        fprintf(savefp, "this file and that will happen instead.\n");
        fprintf(savefp, "%c\n", ENDAMBLE);
        if (fflush(savefp) == EOF)
                crash("file write error on writing save file %s", SAVEFNAM);
        return;
```

```
}       /* end preamble() */

static void restcrash(void)
/* crash due to error in save file when scanning during restart */
{
        crash("error on reading save file %s", SAVEFNAM);

}       /* end restcrash() */
```

C.9 "err.c"

```
/* ********** err.c - errors and warnings ********** */

#include "lvb.h"

#ifdef IAMDANIEL        /* write full but poss. out of date message on crash */
static void logserr(void);
#endif  /* ifdef IAMDANIEL */

void crash(const char *const fmt, ...)
/* log dire warning followed by vfprintf-acceptable message supplied, and if
 * IAMDANIEL is defined also log system error message if any;
 * exit abnormally */
{
        const char *const warning = "FATAL ERROR";      /* dire warning */
        extern FILE *logfp;                             /* output file */
        va_list args;                                   /* arguments */

        va_start(args, fmt);
        fprintf(logfp, "%s: ", warning);
        vfprintf(logfp, fmt, args);
        va_end(args);
        fputc('\n', logfp);

#ifdef IAMDANIEL        /* log sys. error message that may not be up to date */
        logserr();
#endif  /* ifdef IAMDANIEL */

        exit(1);

}       /* end crash() */
```

```
void scream(const char *const fmt, ...)
/* log a dire warning, partly composed of vfprintf-acceptable user-supplied
 * message */
{
        const char *const warning = "ERROR";
        extern FILE *logfp;     /* output file */
        va_list args;           /* supplied message */

        va_start(args, fmt);
        fprintf(logfp, "%s: ", warning);
        vfprintf(logfp, fmt, args);
        va_end(args);
        fputc('\n', logfp);

        /* flush log file so the warning is immediately visible */
        if (fflush(logfp) == EOF)
                crash("write error on log file");       /* msg may not work! */

}       /* end scream() */

#ifdef IAMDANIEL
static void logserr(void)
/* log system error that may be out of date */
{
        extern FILE *logfp;     /* output file */
        extern int errno;       /* error */

        if (errno != 0)
                fprintf(logfp, "System error '%s'\n", strerror(errno));
        else
                fprintf(logfp, "Error 0\n");
        return;

}       /* end logserr() */
#endif  /* ifdef IAMDANIEL */
```

C.10 "fops.c"

```
/* ********** fops.c - file operations ********** */

#include "lvb.h"
```

```
FILE *clnopen(const char *const nam, const char *const mod)
/* open file of name nam with mode mod, and return it;
 * or crash verbosely on error */
{
        FILE *fp;       /* file */

        fp = fopen(nam, mod);
        if (fp == NULL)
        {
                if (strcmp(mod, "w") == 0)
                        crash("cannot create file %s", nam);
                else if (strcmp(mod, "r") == 0)
                        crash("cannot open file %s for reading", nam);
                else if (strcmp(mod, "a") == 0)
                        crash("cannot open file %s for appending to", nam);
                else    /* rare mode */
                        crash("cannot open file %s with mode \"%s\"",
                         nam, mod);
        }
        return fp;

}       /* end clnopen() */

void clnclose(FILE *const fp, const char *const fnam)
/* check file fp, associated with file named fnam, has no errors and close it;
 * crash verbosely on error;
 * or, if fp is NULL, do nothing */
{
        if (fp != NULL)
        {
                if (ferror(fp) != 0)
                        crash("file error on file %s", fnam);
                if (fclose(fp) != 0)
                        crash("cannot close file %s", fnam);
        }
        return;

}       /* end clnclose() */

void clnremove(const char *const fnam)
/* remove file whose name is in fnam, and print strong if error */
{
        if (remove(fnam) != 0)
```

```
                scream("cannot delete file %s", fnam);
        return;

}       /* end clnremove */

char *f2str(FILE *const stream)
/* return string copy of file pointed to by stream,
 * terminated with newline then '\0';
 * the file is treated as text, so newline is always mapped to '\n' */
{
        char *input;                                    /* input string */
        unsigned long inbytes;                          /* bytes for string */
        unsigned long off;                              /* position in str. */
        unsigned long offmax = 0UL;                     /* length of string */
        const unsigned long maxom = ULONG_MAX - 2;      /* maximum offmax */

        while (getc(stream) != EOF)
        {
                ++offmax;
                if (offmax >= maxom)    /* crash while value has meaning */
                        crash("input is too long");
        }
        if (ferror(stream) != 0)
                crash("file error on reading file");
        inbytes = offmax + (unsigned long) 2;   /* '\n', '\0' */

        /* check memory required does not exceed maximum for allocation */
        if (inbytes > (unsigned long) MAX_ALLOC)
                crash("cannot allocate for input: allocation limit of\n"
                 "%u bytes exceeded", (unsigned) MAX_ALLOC);

        input = (char *) alloc((size_t) inbytes, "input");

        /* get string */
        rewind(stream);
        for (off = 0; off < offmax; ++off)
                input[off] = (char) getc(stream);
        if (ferror(stream) != 0)
                crash("file error on reading file");
        input[off] = '\n';
        input[off + 1] = '\0';

        return(input);
```

```
}       /* end f2str() */
```

C.11 "getparam.c"

```
/* ********** getparam.c - get and set configurable parameters ********** */

#include "lvb.h"

static void check(const Params *const prms);
static void checkchars(FILE *const stream);
static void defaults(Params *const prms);
static void equivarr(char **equiv, char *liststr);
static Params *prmalloc(void);
static void pupdate(Params *const prms);

static const char *const ignore = "\t\n ";      /* valid white spc in input */

Params *getparam(void)
/* return pointer to user-supplied run-time configuration parameters, if any;
 * otherwise, use defaults */
{
        Params *prms;           /* configurable parameters */

        prms = prmalloc();
        defaults(prms);
        pupdate(prms);
        check(prms);
        return prms;

}       /* end getparam() */

static Params *prmalloc(void)
/* return newly-allocated pointer to a Params structure and all its elements;
 * new memory is pushed on to the general memstack */
{
        Params *prms;   /* new structure */
        Integer i;      /* loop counter */

        prms = alloc(sizeof(Params), "configurable parameters");
        mempush(prms);
        prms->hnam = salloc(MAX_HNAMLEN, "heuristic method name");
        mempush(prms->hnam);
```

```
        /* equivalent lists are terminated by "" then NULL for added safety */
        prms->equivalent = alloc((size_t)
         (2 + MAX_EQUIVALENT) * sizeof(char *), "equivalent lists");
        mempush(prms->equivalent);
        for (i = 0; i <= MAX_EQUIVALENT; ++i)
        {
                prms->equivalent[i] = salloc(MAX_EQUIVALENT,
                 "equivalent strings");
                mempush(prms->equivalent[i]);
        }
        prms->equivalent[MAX_EQUIVALENT + 1] = NULL;

        prms->titleprec = salloc(MAX_TITLEPRECLEN, "titleprec string");
        mempush(prms->titleprec);
        return prms;

}       /* end prmalloc() */

static void defaults(Params *const prms)
/* set seed in *prms to unacceptable value, and other parameters to defaults */
{
        strcpy(prms->hnam, D_HNAM);
        prms->badstates = D_BADSTATES;
        prms->matchchar = D_MATCHCHAR;
        prms->equivalent[0][0] = D_EQUIVALENT;
        prms->og = D_OG;
        prms->ognam = D_OGNAM;
        prms->runs = D_RUNS;
        prms->seed = -1;                /* 'impossible' value */
        prms->t0 = D_T0;
        prms->t1 = D_T1;
        prms->titleprec[0] = D_TITLEPREC;
        prms->maxaccept = D_MAXACCEPT;
        prms->maxpropose = D_MAXPROPOSE;
        prms->maxfail = D_MAXFAIL;
        prms->trees = D_TREES;
        prms->verbose = D_VERBOSE;
        return;

}       /* end defaults() */

static void checkchars(FILE *const stream)
/* check '\0' and white space not in static string ignore do not occur in
```

```
 * stream;
 * N.B. stream is rewound before return */
{
        int inchar;     /* current character from stream */

        while ((inchar = getc(stream)) != EOF)
        {
                if ((inchar == '\0') || (isspace(inchar)
                 && (strchr(ignore, inchar) == NULL)))
                        crash("illegal machine character (value %d) in file "
                         "%s", inchar, RCFNAM);
        }

        /* check file error since erros are cleared by rewind */
        if (ferror(stream) != 0)
                crash("file error reading file %s", RCFNAM);

        rewind(stream);
        return;

}       /* end checkchars() */

static void pupdate(Params *const prms)
/* update *prms according to user-supplied values */
{
        char *instr;    /* input */
        char *line;     /* current line of input */
        char *name;     /* variable's name in input */
        char *invalue;  /* mashable copy of value */
        char *value;    /* variable's value in input */
        Integer len;    /* string length */
        long lval;      /* actual integral value */
        Real fval;      /* actual real value */
        FILE *rcfp;     /* configuration file */

        rcfp = fopen(RCFNAM, "r");      /* can't use clnopen() here */
        if (rcfp == NULL)
                instr = NULL;
        else
        {
                checkchars(rcfp);
                instr = f2str(rcfp);
        }
        line = instr;
```

```
/* parse input line by line, if it exists */
while ((instr != NULL) && ((line = nextnonwspc(line)) != NULL))
{
        name = line;                    /* first token on line */
        line = strchr(line, '\n');      /* for next iteration */
        if (name[0] == COMMENT)         /* ignore line */
                continue;

        /* make tokens terminated strings */
        len = (Integer) strcspn(name, ignore);  /* length of name */
        name[len] = '\0';
        invalue = nextnonwspc(name + len + 1);
        if ((invalue >= line) || (invalue == NULL))     /* missing */
                crash("syntax error in configuration file %s: each\n"
                 "variable named must be followed by a value", RCFNAM);
        len = (Integer) strcspn(invalue, ignore);       /* val. len. */
        invalue[len] = '\0';

        /* make local copy of token since '\0' may be put inside it */
        value = salloc(len, "value from configuration file");
        strcpy(value, invalue);

        /* read value */
        if (strcicmp(name, "seed") == 0)        /* followed by int. */
        {
                lval = strtol(value, NULL, 10);
                if ((lval < 0L) || (lval > (long) MAX_SEED))
                        crash("illegal value for variable 'seed' in\n"
                         "file %s: must be an integer from 0 to %d",
                         RCFNAM, MAX_SEED);
                else
                        prms->seed = (int) lval;
        }
        else if (strcicmp(name, "runs") == 0)   /* followed by int. */
        {
                lval = strtol(value, NULL, 10);
                if ((lval < 0L) || (lval > (long) (MAX_INTEGER - 1)))
                        crash("illegal value for variable 'runs' in\n"
                         "file %s: must be an integer from 1 to %d",
                         RCFNAM, (int) MAX_INTEGER - 1);
                else
                        prms->runs = (Integer) lval;
        }
```

```
else if (strcicmp(name, "verbose") == 0)        /* int. */
{
        lval = strtol(value, NULL, 10);
        if ((lval == 0L) || (lval == 1L))
                prms->verbose = (Integer) lval;
        else
                crash("illegal value for variable 'verbose'\n"
                 "in file %s: must be either 0 or 1", RCFNAM);
}
else if (strcicmp(name, "trees") == 0)  /* f. by long int. */
{
        lval = strtol(value, NULL, 10);
        if ((lval < 0L) || (lval > ((long) MAX_LINT - 1L)))
                crash("illegal value for variable 'trees' in\n"
                 "file %s", RCFNAM);
        else
                prms->trees = (Lint) lval;
}
else if (strcicmp(name, "maxaccept") == 0)      /* long int. */
{
        lval = strtol(value, NULL, 10);
        if ((lval <= 0L) || (lval > ((long) MAX_LINT - 1L)))
                crash("illegal value for variable\n"
                 "'maxaccept' in file %s", RCFNAM);
        else
                prms->maxaccept = (Lint) lval;
}
else if (strcicmp(name, "maxpropose") == 0)     /* long int. */
{
        lval = strtol(value, NULL, 10);
        if ((lval <= 0L) || (lval > ((long) MAX_LINT - 1L)))
                crash("illegal value for variable\n"
                 "'maxpropose' in file %s", RCFNAM);
        else
                prms->maxpropose = (Lint) lval;
}
else if (strcicmp(name, "maxfail") == 0)        /* long int. */
{
        lval = strtol(value, NULL, 10);
        if ((lval < 0L) || (lval > ((long) MAX_LINT - 1L)))
                crash("illegal value for variable 'maxfail'\n"
                 "in file %s", RCFNAM);
        else
                prms->maxfail = (Lint) lval;
```

```
}
else if (strcicmp(name, "method") == 0)         /*  string */
{
        if ((strlen(value) > (size_t) MAX_HNAMLEN)
         || ((strcicmp(prms->hnam, SAHNAM) != 0)
         && (strcicmp(prms->hnam, HCHNAM) != 0)
         && (strcicmp(prms->hnam, RWHNAM) != 0)))
                crash("illegal method '%s' in file %s",
                 value, RCFNAM);
        else
                strcpy(prms->hnam, value);
}
else if (strcicmp(name, "outgroup") == 0)       /* string */
{
        prms->ognam = (char *) salloc((Integer) strlen(value),
         "outgroup name");
        mempush(prms->ognam);
        strcpy(prms->ognam, value);
}
else if (strcicmp(name, "ignorechars") == 0)    /* string */
{
        if (strcicmp(value, UNSETNAM) == 0)
                prms->badstates = "";
        else
        {
                prms->badstates = salloc(
                 (Integer) strlen(value),
                 "list of 'ignore' states");
                mempush(prms->badstates);
                strcpy(prms->badstates, value);
                supper(prms->badstates);
        }
}
else if (strcicmp(name, "titleprec") == 0)      /* string */
{
        if (strcicmp(value, UNSETNAM) == 0)
                prms->titleprec[0] = '\0';
        else    /* N.B. leave case as found */
        {
                if (strlen(value) > (size_t) MAX_TITLEPRECLEN)
                        crash("invalid value for titleprec\n"
                         "in file %s: use a string of up to "
                         "%d machine characters, or\n"
                         "'titleprec none' for none", RCFNAM,
```

```
                        MAX_TITLEPRECLEN);
                else
                        strcpy(prms->titleprec, value);
        }
}
else if (strcicmp(name, "equivalent") == 0)     /* str. list */
        equivarr(prms->equivalent, value);
else if (strcicmp(name, "matchchar") == 0)      /* character */
{
        if (strlen(value) != (size_t) 1)
                crash("invalid value for matchchar in file\n"
                 "%s: should be a single machine character,\n"
                 "or use 'matchchar none' for none", RCFNAM);
        prms->matchchar = (char) toupper((int) value[0]);
}
else    /* followed by floating point number */
{
        fval = (Real) strtod(value, NULL);
        if (errno == ERANGE)
                crash("value for variable '%s' in file %s is\n"
                 "out of range", name, RCFNAM);
        if (strcicmp(name, "t0") == 0)
        {
                prms->t0 = fval;
                if ((prms->t0 <= 0.0)
                 || (prms->t0 > 1.0))
                        crash("illegal value %g for variable "
                         "'t0' in file %s: please use a\n"
                         "number greater than 0.0 and less "
                         "than or equal to 1.0",
                         (double) prms->t0, RCFNAM);
        }
        else if (strcicmp(name, "t1") == 0)
        {
                prms->t1 = fval;
                if ((prms->t1 <= 0.0)
                 || (prms->t1 > 1.0))
                        crash("illegal value %g for variable "
                         "'t1' in file %s: please use a\n"
                         "number greater than 0.0 and less "
                         "than or equal to 1.0",
                         (double) prms->t1, RCFNAM);
        }
        else    /* unknown or inappropriate token */
```

```
                        crash("unknown variable '%s' in file %s", name,
                         RCFNAM);
                }
                line = invalue + len + 1;       /* go past end of value */
                free(value);                    /* alloc'd at start of loop */

                /* crash on extra value(s) on line */
                if ((nextnonwspc(name + len + 1) != NULL)
                 && (nextnonwspc(name + len + 1)
                 < strchr(name + len + 1, '\n')))
                        crash("syntax error in configuration file %s:\n"
                         "each variable named must be followed by exactly one "
                         "value", RCFNAM);
        }

        /* free 'automatic' memory */
        if (instr != NULL)
                free(instr);

        /* set default seed if none specified by user */
        if (prms->seed == -1)
                prms->seed = (int) time(NULL);
        if ((prms->seed < 0) || (prms->seed > MAX_SEED))        /* can't use */
                crash("cannot get random number seed from system clock:\n"
                 "please specify a value from 0 to %d for variable seed in\n"
                 "configuration file %s", MAX_SEED, RCFNAM);

        if (rcfp != NULL)
                clnclose(rcfp, RCFNAM);
        return;

}       /* end pupdate() */

static void equivarr(char **equiv, char *liststr)
/* fill equiv with lists of states named as equivalent to each other in
 * liststr (in which lists are separated by commas, without white space),
 * converted to upper case;
 * terminate equiv with the string "";
 * N.B. liststr may be mashed by addition of '\0's */
{
        char *currlist;         /* current list string */
        Integer currnum = 0;    /* current list number */

        if (strcicmp(liststr, "none") == 0)
```

```
        {
                equiv[0][0] = '\0';
                return;
        }

        while ((currlist = strtok(liststr, ",")) != NULL)
        {
                if (strlen(currlist) > MAX_EQUIVALENT)
                        crash("value of equivalent too long in file %s,\n"
                         "no equivalent list may contain more than %d\n"
                         "machine characters (biological character states)",
                          RCFNAM, MAX_EQUIVALENT);
                if (currnum >= MAX_EQUIVALENT)
                        crash("too many lists for value of equivalent in\n"
                         "file %s, there may be no more then %d lists",
                         RCFNAM, MAX_EQUIVALENT);
                strcpy(equiv[currnum], currlist);
                supper(equiv[currnum]);
                ++currnum;
                liststr = NULL; /* get second and subsequent lists */
        }
        equiv[currnum][0] = '\0';
        return;

}       /* end equivarr() */

static void check(const Params *const prms)
/* check parameters in prms are mutually sensible, and crash verbosely
 * if not */
{
        if (prms->t1 > prms->t0)
                crash("t1 must be less than or equal to t0 (currently t0 is\n"
                "%g and t1 is %g)", (double) prms->t0, (double) prms->t1);

}       /* end check() */
```

C.12 "makebin.c"

```
/* ********** makebin.c - make integer-encoded matrix ********** */

#include "lvb.h"
```

```
static Uint **bmatalloc(const Dataptr matrix);

Uint **makebin(const Dataptr matrix)
/* return an integer-encoded mapping of the matrix proper, where each state in
 * each column (biological character) is represented by a different power of 2;
 * this can be used to calculate state sets using bitwise operations OR
 * (intersection) and AND (union);
 * the matrix returned is transposed with respect to the original, since it
 * will be accessed by biological characters, so it should only be accessed by
 * the macro binmat() (defined in lvb.h) to avoid confusion;
 * new memory for the matrix is pushed on to the general memstack */
{
        char *statev;                   /* state array for current character */
        Uint **ltbinmat;                /* integer-encoded mapping of matrix */
        Objno i;                        /* loop counter */
        Charno k;                       /* loop counter */
        ptrdiff_t power;                /* power of 2 for current cell */

        ltbinmat = bmatalloc(matrix);
        for (k = 0; k < matrix->m; ++k)
        {
                statev = getstatev(matrix, k);
                for (i = 0; i < matrix->n; ++i)
                {
                        power = strchr(statev, (int) matrix->row[i][k])
                         - statev;
                        if (power < 0)  /* impossible condition */
                                crash("internal error: cannot create encoded\n"
                                 "version of character %d", (int) k + 1);
                        /* N.B. don't rotate by 0: it is not wholly explicit
                         * in K. & R. ed. 2 that rotating by 0 is allowed */
                        else if (power > 0)
                                ltbinmat[k][i] = 1U << power;
                        else
                                ltbinmat[k][i] = 1U;
                }
        }

        return ltbinmat;

}       /* end makebin() */

static Uint **bmatalloc(const Dataptr matrix)
/* return memory for a new integer-encoded, transposed version of matrix->row;
```

```
 * push new memory on to the general memstack;
 * N.B. the new matrix rows are contiguous if possible, otherwise allocated
 * individually */
{
        Charno k;                       /* loop counter */
        Uint **bmat;                    /* new encoded matrix */
        unsigned long matspace;         /* bytes of contig. memory for cells */

        /* allocate for first dimension, allowing for terminating NULL */
        bmat = (Uint **) alloc((size_t) (matrix->m + 1) * sizeof(Uint *),
         "encoded matrix");
        mempush(bmat);

        /* allocate for rows all at once if possible, otherwise individually */
        matspace = (unsigned long) matrix->m * (unsigned long) matrix->n
         * (unsigned long) sizeof(Uint);
        if ((matspace > (unsigned long) MAX_ALLOC)      /* can't allocate */
         || ((bmat[0] =
         (Uint *) malloc((size_t) matspace)) == NULL))  /* failed allocation */
        {
                for (k = 0; k < matrix->m; ++k)
                {
                        bmat[k] = (Uint *) alloc((size_t) matrix->n,
                         "encoded matrix row");
                        mempush(bmat[k]);
                }
        }
        else    /* have allocated for rows contigously above */
        {
                mempush(bmat[0]);
                for (k = 1; k < matrix->m; ++k)
                        /* N.B. can't assume Integer can hold m * n */
                        bmat[k] = bmat[0] + (size_t) matrix->n * (size_t) k;
        }

        bmat[matrix->m] = NULL;         /* however the rows were allocated */
        return bmat;

}       /* end bmatalloc() */
```

C.13 "mem.c"

```
/* ********** mem.c - maintain a memory stack to prevent leaks ********** */

#include "lvb.h"

static void memchange(const Lint pos, void *const newptr);
static void *mempop(void);
static void memsize(void);

static void **memstack = NULL;  /* pointers to memory to free before exit */
static Lint memstsize = 0;      /* no. of elements allocated for in memstack */
static Lint memstnext = 0;      /* next unused element in memstack */

/* N.B. Once pointers have been pushed on to the memstack, they can only be
 * popped then freed one after the other until the stack is empty, by calling
 * memfree(). */

Lint mempush(void *const mem)
/* push pointer to memory on to stack to be freed at exit;
return pointer's index in stack array */
{
        if (memstnext >= memstsize)     /* want more memory for stack */
                memsize();
        memstack[memstnext++] = mem;
        return (memstnext - 1);

}       /* end mempush() */

static void memchange(const Lint pos, void *const newptr)
/* change entry in memstack whose offset from stack start is pos, to newptr */
{
        memstack[pos] = newptr;
        return;

}       /* end memchange() */

void memfree(void)
/* free all pointers in memstack in reverse order of their registration with
 * mempush */
{
        void *tofree;   /* memory to free */

        while ((tofree = mempop()) != NULL)
                free(tofree);
        memstack = NULL;        /* freed, since was as bottom of memstack */
```

```
        return;

}       /* end memfree() */

static void *mempop(void)
/* pop pointer to memory from stack;
or return NULL if stack is empty */
{
        --memstnext;
        if (memstnext < 0)
        {
                memstnext = 0;
                return NULL;
        }
        else
                return memstack[memstnext];     /* next push will overwite */

}       /* end mempop() */

static void memsize(void)
/* increase allocation for memstack by STACKBLOCK elements */
{
        memstsize += (Lint) STACKBLOCK;         /* new stack size */
        if (memstack == NULL)   /* 1st call, stack does not exist */
        {
                memstack = (void **) alloc((size_t) memstsize * sizeof(void *),
                 "general memstack");
                mempush(memstack);      /* to free stack itself, last of all */
                memstnext = 1;
        }
        else
        {
                memstack = (void **)
                 realloc(memstack, (size_t) memstsize * sizeof(void *));
                if (memstack == NULL)
                        crash("out of memory: cannot increase allocation for\n"
                         "general memstack to %d elements", (int) memstsize);
                memchange((Lint) 0, memstack);  /* in case stack has moved */
        }
        return;

}       /* end memsize() */
```

C.14 "mops.c"

```
/* ********** mops.c - basic memory operation ********** */

#include "lvb.h"

void *alloc(const size_t bytes, const char *const msg)
/* return pointer to bytes bytes of new memory, or on failure crash with
 * message "FATAL ERROR: out of memory: cannot allocate for ", followed by msg;
 * or, if bytes exceeds MAX_ALLOC, crash verbosely without attempting
 * allocation */
{
        void *p;        /* new memory */

        if ((unsigned long) bytes > (unsigned long) MAX_ALLOC)
                crash("can't allocate memory for %s:\n"
                 " requested block too large", msg);
        p = malloc(bytes);
        if (p == NULL)
                crash("out of memory: cannot allocate for %s", msg);
        return p;

}       /* end alloc() */
```

C.15 "myuni.c"

```
/* ********** myuni.c - random number generation ********** */

/* exactly as suppplied by EPCC, except printf() then exit() replaced with
 * crash() (and braces made redundant removed), float replaced with Uni_float,
 * declaration for rstart added, "lvb.h" included, global variables and
 * rstart() made static and comment after declaration of uni_u shortened to fit
 * 80 columns after the addition of the word 'static' */

/*
 *      C version of Marsaglia's UNI random number generator
 *      More or less transliterated from the Fortran -- with 1 bug fix
 *      Hence horrible style
 *
 *      Features:
 *              ANSI C
```

```
 *              not callable from Fortran (yet)
 */

#include "lvb.h"

static void rstart(int i, int j, int k, int l);

/*
 *      Global variables for rstart & uni
 */

static Uni_float uni_u[98];     /* Was U(97) in Fortran version */
static Uni_float uni_c, uni_cd, uni_cm;
static int uni_ui, uni_uj;

Uni_float uni(void)
{
        Uni_float luni;                 /* local variable for uni */

        luni = uni_u[uni_ui] - uni_u[uni_uj];
        if (luni < 0.0)
                luni += 1.0;
        uni_u[uni_ui] = luni;
        if (--uni_ui == 0)
                uni_ui = 97;
        if (--uni_uj == 0)
                uni_uj = 97;
        if ((uni_c -= uni_cd) < 0.0)
                uni_c += uni_cm;
        if ((luni -= uni_c) < 0.0)
                luni += 1.0;
        return (Uni_float) luni;
}

static void rstart(int i, int j, int k, int l)
{
        int ii, jj, m;
        Uni_float s, t;

        for (ii = 1; ii <= 97; ii++) {
                s = 0.0;
                t = 0.5;
                for (jj = 1; jj <= 24; jj++) {
                        m = ((i*j % 179) * k) % 179;
```

```
                        i = j;
                        j = k;
                        k = m;
                        l = (53*l+1) % 169;
                        if (l*m % 64 >= 32)
                                s += t;
                        t *= 0.5;
                }
                uni_u[ii] = s;
        }
        uni_c  = 362436.0   / 16777216.0;
        uni_cd = 7654321.0  / 16777216.0;
        uni_cm = 16777213.0 / 16777216.0;
        uni_ui = 97;    /*  There is a bug in the original Fortran version */
        uni_uj = 33;    /*  of UNI -- i and j should be SAVEd in UNI()     */
}


/* ~rinit: this takes a single integer in the range
                0 <= ijkl <= 900 000 000
        and produces the four smaller integers needed for rstart. It is
 *      based on the ideas contained in the RMARIN subroutine in
 *              F. James, "A Review of Pseudorandom Number Generators",
 *                      Comp. Phys. Commun. Oct 1990, p.340
 *      To reduce the modifications to the existing code, rinit now
 *      takes the role of a preprocessor for rstart.
 *
 *      This is useful for the parallel version of the code as James
 *      states that any integer ijkl will produce a statistically
 *      independent sequence of random numbers.
 *
 *     Very funny. If that statement was worth anything he would have provided
 *     a proof to go with it. spb 12/12/90
 */

void rinit(int ijkl)
{
        int i, j, k, l, ij, kl;

        /* check ijkl is within range */
        if( (ijkl < 0) || (ijkl > 900000000) )
                crash("rinit: ijkl = %d -- out of range\n\n", ijkl);

/*        printf("rinit: seed_ijkl = %d\n", ijkl); */
```

```
        /* decompose the long integer into the the equivalent four
         * integers for rstart. This should be a 1-1 mapping
         *      ijkl <--> (i, j, k, l)
         * though not quite all of the possible sets of (i, j, k, l)
         * can be produced.
         */

        ij = ijkl/30082;
        kl = ijkl - (30082 * ij);

        i = ((ij/177) % 177) + 2;
        j = (ij % 177) + 2;
        k = ((kl/169) % 178) + 1;
        l = kl % 169;

        if( (i <= 0) || (i > 178) )
              crash("rinit: i = %d -- out of range\n\n", i);

        if( (j <= 0) || (j > 178) )
              crash("rinit: j = %d -- out of range\n\n", j);

        if( (k <= 0) || (k > 178) )
              crash("rinit: k = %d -- out of range\n\n", k);

        if( (l < 0) || (l > 168) )
              crash("rinit: l = %d -- out of range\n\n", l);

        if (i == 1 && j == 1 && k == 1)
              crash("rinit: 1 1 1 not allowed for 1st 3 seeds\n\n");

/*        printf("rinit: initialising RNG via rstart(%d, %d, %d, %d)\n",
                              i, j, k, l); */

        rstart(i, j, k, l);

}
```

C.16 "parsim.c"

```
/* ********** parsim.c - tree evaluation functions ********** */
```

```
#include "lvb.h"

static void cr_otdne(const Integer els);
static void makessets(Boolean *const haslen, const Branchno lower,
 const Branchno upper);
static Lint ssupdate (void);
static void statalloc(void);
static void uppass(const Branchno root);
static void upupdate(const Branchno parent, const Branchno child);

static Uint *stateset = NULL;           /* branches' state sets */
static const Branch *barray;            /* tree */
static Branchno *todoarr = NULL;        /* branches of unknown stateset */
static Branchno ext_todo;               /* elements in todoarr when full */

Boolean *getlmask(const Branch *const tree, const Branchno root)
/* return pointer to static array whose elements are TRUE if the
 * corresponding branch may have length > 0 and FALSE otherwise, according to
 * the accelerated transformation model of character change;
 * N.B. caller should not free returned array, which will be overwritten by
 * subsequent calls and will be freed transparently on exit */
{
        Integer off = 0;                /* array index */
        Objno obj;                      /* current object */
        Branchno i;                     /* loop counter */
        Charno k;                       /* current character number */
        static Boolean *haslen = NULL;  /* TRUE if evidence for branch */
        extern Uint **tbinmat;          /* transposed, encoded matrix */
        extern Branchno nbranches;      /* branches in tree */
        extern Dataptr matrix;          /* data matrix */

        /* set up static objects */
        barray = tree;
        if (!stateset)  /* first call, allocate memory */
                statalloc();
        if (haslen == NULL)     /* first call, allocate memory */
        {
                haslen = (Boolean *)
                 alloc((size_t) nbranches * sizeof(Boolean), "length mask");
                mempush(haslen);
        }

        /* clear length mask */
        for (i = 0; i < nbranches; ++i)
```

```
                haslen[i] = FALSE;

        /* get l. mask for all branches except root (which has UNSET parent) */
        for (k = 0; k < matrix->m; ++k)          /* for each character */
        {
                off = 0;
                /* set state sets for non-root leaves */
                for (i = 0; i < nbranches; ++i)
                {

                        if ((obj = barray[i].object) != UNSET)  /* leaf */
                                stateset[i] = binmat(obj, k);
                        else
                        {
                                stateset[i] = 0U;
                                todoarr[off++] = i;
                        }
                }

                /* get root's state set from children */
                stateset[root] = 0U;
                todoarr[off++] = root;

                /* check impossible condition */
                if (off != (Integer) ext_todo)
                        cr_otdne(off);

                /* get MPR state sets for character k */
                ssupdate();             /* downpass */
                uppass(root);

                /* update haslen according to state sets for this character */
                makessets(haslen, 0, root - 1);
                makessets(haslen, root + 1, nbranches - 1);
        }

        haslen[root] = TRUE;    /* since tree is rooted */
        return haslen;

}       /* end getlmask() */

static void makessets(Boolean *const haslen, const Branchno lower,
 const Branchno upper)
/* ensure element numbers lower to upper inclusive in haslen are TRUE where
```

```
 * static stateset indicates the branch of that number may have length */
{
        Branchno i;     /* loop counter */

        for (i = lower; i <= upper; ++i)
        {
                if (stateset[i] != stateset[barray[i].parent])
                        haslen[i] = TRUE;
        }
        return;

}       /* end makessets() */

Lint getplen(const Branch *const tree, const Branchno root)
/* return total changes in all characters for tree in barray (of root root);
 * N.B. all characters in one function) to guarantee moderate speed, but it is
 * well worth inlining ssupdate() as well */
{
        Integer off;                    /* array index */
        Objno obj;                      /* current object */
        Charno k;                       /* current character number */
        register Branchno i;            /* loop counter (heavily used) */
        Lint changes = 0;               /* number of changes in character k */
        extern Uint **tbinmat;          /* transposed encoded matrix */
        extern Branchno nbranches;      /* branches in tree */
        extern Dataptr matrix;          /* data matrix */

        /* set up static objects */
        barray = tree;
        if (!stateset)  /* first call, allocate memory */
                statalloc();

        /* calculate total tree length */
        for (k = 0; k < matrix->m; ++k)
        {
                off = 0;
                /* set state sets for non-root leaves */
                for (i = 0; i < nbranches; ++i)
                {
                        if ((obj = barray[i].object) != UNSET)  /* leaf */
                                stateset[i] = binmat(obj, k);
                        else
                        {
                                stateset[i] = 0U;
```

```
                        todoarr[off++] = i;
                }
        }

        /* get root's state set from children */
        stateset[root] = 0U;
        todoarr[off++] = root;

        /* check impossible condition */
        if (off != (Integer) ext_todo)
                cr_otdne(off);

        changes += ssupdate();  /* get unions used to update s. sets */

        /* add one step if root's character isn't in its state set */
        changes +=
         (Lint) !(stateset[root] & binmat(barray[root].object, k));
}

        return changes;

}       /* end getplen() */

static void statalloc(void)
/* set ext_todo, and allocate for static arrays stateset and todoarr */
{
        extern Branchno nbranches;      /* branches per tree */
        extern Dataptr matrix;          /* data matrix */

        ext_todo = nbranches - matrix->n + 1;   /* interior branches + root */
        stateset = (Uint *) alloc((size_t) nbranches * sizeof(Uint),
         "state sets");
        mempush(stateset);
        todoarr = (Branchno *) alloc((size_t) ext_todo * sizeof(Branchno),
         "'to do' array");
        mempush(todoarr);
        return;

}       /* end statalloc() */

static Lint ssupdate (void)
/* use a down pass to update empty state sets in stateset according to the
 * non-empty ones and the shape of the tree in barray;
 * return the number of union operations necessary to do this;
```

```
 * N.B. when compiling for speed, this function should be inlined into
 * getplen() */
{
        Lint unions = 0;                /* no. of union operations */
        Uint leftss;                    /* left child's state set */
        Uint rightss;                   /* right child's state set */
        Branchno l_todo = ext_todo;     /* modifiable copy of 'to do' count */
        /* N.B. current is used 2 to 5 times per inner loop, and i 3 to 4
         * times. No other automatic variables are definitely or possibly used
         * so much, so I have declared them register. I haven't done this for
         * more variables because I don't want to limit assignment of external
         * variables to registers by the compiler */
        register Branchno current;      /* current branch */
        register Branchno i;            /* loop counter */

        while (l_todo)
        {
                for (i = 0; i < l_todo; ++i)
                {
                        current = todoarr[i];
                        if ((rightss = stateset[barray[current].right])
                         && (leftss = stateset[barray[current].left]))
                        {
                                if (!(stateset[current] = leftss & rightss))
                                /* no intersection, get union */
                                {
                                        stateset[current] = leftss | rightss;
                                        ++unions;
                                }
                                todoarr[i] = todoarr[--l_todo];
                        }
                }
        }
        return unions;

}       /* end ssupdate() */

static void uppass(const Branchno root)
/* use an up pass to update state sets in stateset according to the shape of
 * the tree in barray (of root root);
 * use after ssupdate() to leave accelerated transformation state sets in
 * static array stateset */
{
        if (barray[root].left != UNSET)         /* not leaf */
```

```
        {
                upupdate(root, barray[root].left);
                upupdate(root, barray[root].right);
                uppass(barray[root].left);
                uppass(barray[root].right);
        }

        return;

}       /* end uppass */

static void upupdate(const Branchno parent, const Branchno child)
/* in an up pass, update child's entry in stateset according to the parent's */
{
        Uint newset;    /* child's new state set */

        newset = stateset[child] & stateset[parent];
        if (!(newset))          /* no intersection, get union */
                stateset[child] = stateset[child] | stateset[parent];
        else                    /* use intersection */
                stateset[child] = newset;
        return;

}       /* end upupdate */

static void cr_otdne(const Integer els)
/* crash because of internal error: todoarr has had els elements set, which
 * differs from the number expected, ext_todo */
{
        crash("internal error: todoarr has had %d elements set, but %d\n"
         "were expected", (int) els, (int) ext_todo);

}       /* end cr_otdne() */
```

C.17 "randpint.c"

```
/* ********** randpint.c ********** */

/* get random non-negative integer */

#include "lvb.h"
```

```
Integer randpint(const Integer upper)
/* return an integer greater than or equal to 0 and less than or equal to
 * upper;
 * N.B. assumes the random number generator has been initialized by calling
 * rinit()*/
{
        Real frand;     /* random real */
        Real fupper;    /* upper limit */
        Integer rand;   /* return value */

        fupper = (Real) upper;
        frand = (Real) uni();
        frand = frand * fupper;                 /* scale to right range */
        rand = (Integer) (frand + 0.5);         /* round to nearest integer */

        /* guard against arithmetic inaccuracy */
        if (rand < 0)
                rand = 0;
        else if (rand > upper)
                rand = upper;

        return rand;

}       /* end randpint() */
```

C.18 "signal.c"

```
/* ********** signal.c - portable ANSI signal trapping ********** */

#include "lvb.h"

static void tr_fpe(int sig);
static void tr_hup(int sig);
static void tr_int(int sig);
static void tr_term(int sig);

void settraps(void)
/* set non-returning traps for SIGFPE, SIGHUP, SIGINT and SIGTERM if they are
 * defined */
{
#ifdef SIGFPE
        signal(SIGFPE, tr_fpe);
```

```
#endif  /* ifdef SIGFPE */
#ifdef SIGHUP
        signal(SIGHUP, tr_hup);
#endif  /* ifdef SIGHUP */
#ifdef SIGINT
        signal(SIGINT, tr_int);
#endif  /* ifdef SIGINT */
#ifdef SIGTERM
        signal (SIGTERM, tr_term);
#endif  /* ifdef SIGTERM */
        return;

}       /* end settraps() */

static void tr_fpe(int sig)
/* trap arithmetic error */
{
        crash("arithmetic error signal SIGFPE (%d) received", sig);

}       /* end tr_sigfpe() */

static void tr_hup(int sig)
/* trap hangup */
{
        crash("hangup signal SIGHUP (%d) received", sig);

}       /* end tr_hup() */

static void tr_int(int sig)
/* trap interrupt */
{
        crash("interrupt signal SIGINT (%d) received", sig);

}       /* end tr_int() */

static void tr_term(int sig)
/* trap termination request */
{
        crash("termination request signal SIGTERM (%d) received", sig);

}       /* end tr_term() */
```

C.19 "solve.c"

```
/* ********** solve.c - solving functions ********** */

#include "lvb.h"

static FILE *lengthfp;          /* length file */
static Branch *x = NULL;        /* current configuration */
static Branch *xdash = NULL;    /* proposed new configuration */

static void checkedom(void);
static void configalloc(void);
static void lenlog(const char *const fmt, ...);

Lint anneal(const Branch *const inittree, Branchno root, const Real t0,
 const Real t1, const Lint maxaccept, const Lint maxpropose,
 const Lint maxfail, const Integer run, FILE *const lenfp,
 const char *const titleprec, Lint lenbest, Lint proposed, Lint accepted,
 Lint failedcnt, Real t, Lint n, Lint iter)
/* seek parsimonious tree from initial tree in inittree (of root root)
 * with initial temperature t0, and subsequent temperatures obtained by
 * multiplying the current temperature by
 * by (t1 / t0) ** n * t0 (where n is the ordinal number of this temperature,
 * originally 0) after at least maxaccept changes have been
 * accepted or maxpropose changes have been proposed, whichever is
 * sooner;
 * return after maxfail consecutive temperatures have led to no new accepted
 * solution;
 * run is the number of the current run (expected to be initially 0), lenfp is
 * for output of current tree length and associated details or NULL for no
 * such output, titleprec is to precede the line of title text in output
 * statistics;
 * with a restart, other parameters contain values of anneal() automatic
 * variables of the same name saved at the time of pausing,
 * otherwise they should all have value (Real) UNSET or (Lint) UNSET */
{
        Real deltah;            /* change in energy (1 - C.I.) */
        Integer deltalen;       /* change in length with new tree */
        Lint len;               /* length of current tree */
        Lint lendash;           /* length of proposed new tree */
        Lint lenmin;            /* minimum length for any tree */
        Real r_lenmin;          /* minimum length for any tree */
        Real pacc;              /* prob. of accepting new config. */
        Branchno rootdash;      /* root of new configuration */
```

```
Boolean newtree;        /* accepted a new configuration */
Boolean dect;           /* should decrease temperature */
Boolean probaccd;       /* have accepted based on Pacc */
extern Dataptr matrix;  /* data matrix */

/* initializations irrespective of whether restart or not */
lengthfp = lenfp;       /* for lenlog() */
configalloc();          /* allocate for x and xdash if necessary */
treecopy(x, inittree);  /* current configuration */
len = getplen(x, root);
dect = FALSE;           /* made TRUE as necessary at end of loop */

if (proposed == (Lint) UNSET)   /* not a restart */
{
        lenbest = len;
        accepted = 0;
        proposed = 0;
        failedcnt = 0;
        t = t0;
        n = 0;
        iter = 0;
        bstpush(x, root);       /* initial tree initially best tree */
        if (titleprec[0] != '\0')
                lenlog("%s ", titleprec);
        lenlog("Tree\tBestLen\tTempNo\tTemp\tCurrLen\tPropLen\t"
         "ProbAcc\n");
}

lenmin = getminlen(matrix);
r_lenmin = (Real) lenmin;

while (t >= (Real) 0)   /* inf. loop might go < 0 at accuracy lim.? */
{
        newtree = FALSE;
        probaccd = FALSE;
        rootdash = mutate(xdash, x, root);      /* propose a change */
        lendash = getplen(xdash, rootdash);
        deltalen = lendash - len;
        deltah = (r_lenmin / (Real) len) - (r_lenmin / (Real) lendash);
        if (deltalen <= 0)      /* accept the change */
        {
                if (lendash <= lenbest)         /* store tree if new */
                {
                        if (lendash < lenbest)  /* very best so far */
```

```
                        bstclear();     /* discard old bests */
                if (bstpush(xdash, rootdash) == 1)
                        newtree = TRUE;         /* new */
        }

        /* log current stats */
        lenlog("%ld\t%ld\t%ld\t%g\t%ld\t%ld\t1\n",
         (long) iter, (long) lenbest, (long) n, (double) t,
         (long) len, (long) lendash);

        /* update current tree and its stats */
        len = lendash;
        treeswap(&x, &root, &xdash, &rootdash);

        if (lendash < lenbest)          /* very best so far */
                lenbest = lendash;
}
else                    /* poss. accept change for the worse */
{
        pacc = exp((double) (-deltah/t));
        checkedom();
        if ((Real) uni() < pacc)        /* do accept change */
        {
                probaccd = TRUE;
                treeswap(&x, &root, &xdash, &rootdash);
        }
        lenlog("%ld\t%ld\t%ld\t%g\t%ld\t%ld\t%g\n",
         (long) iter, (long) lenbest, (long) n, (double) t,
         (long) len, (long) lendash, (double) pacc);
        if (probaccd == TRUE)
                len = lendash;
}
++proposed;
if (newtree == TRUE)
        ++accepted;

/* decide whether to reduce temperature */
if (accepted >= maxaccept)              /* enough new trees */
{
        failedcnt = 0;  /* this temperature a 'success' */
        dect = TRUE;
}
else if (proposed >= maxpropose)        /* enough proposals */
{
```

```
                        ++failedcnt;
                        if (failedcnt >= maxfail)       /* system frozen */
                                break;                  /* end of cooling */
                        else                            /* decrease temp. */
                                dect = TRUE;
                }

                if (dect == TRUE)
                {
                        ++n;    /* originally n is 0 */
                        t = (Real) pow((double) (t1/t0), (double) n) * t0;
                        checkedom();
                        proposed = 0;
                        accepted = 0;
                        dect = FALSE;
                }
                ++iter;
                if (!(iter % CHECK_INTERVAL))
                {
                        if (stopnow() == TRUE)          /* stopped by user */
                                return -(len);
                        if (pausenow() == TRUE)         /* paused by user */
                        {
                                epause(x, run, lenbest, proposed, accepted,
                                 failedcnt, t, n, iter);
                        }
                }
        }

        return lenbest;

}       /* end anneal() */

static void configalloc(void)
/* allocate for static x and xdash if they are NULL */
{
        if (x == NULL)
                x = treealloc();
        if (xdash == NULL)
                xdash = treealloc();
        if ((x == NULL) || (xdash == NULL))
                crash("cannot allocate for changing configurations");
        return;
```

```
}       /* end configalloc() */

Lint hillclimb(const Branch *const inittree, Branchno root,
 const Lint maxtrees, FILE *const lenfp, const char *const titleprec)
 /* find 'shortest' tree(s) using hill climbing from initial tree in inittree
  * (of root root) to look at maxtrees trees, and return the minumum length;
  * lenfp is for output of current tree length and associated details or
  * NULL for no such output;
  * if stopped prematurely, returns 0 - length */
{
        Branchno rootdash;              /* root of new config. */
        Lint deltalen;                  /* change in length */
        Lint len;                       /* length of current tree */
        Lint lenbest;                   /* length of best tree */
        Lint lendash;                   /* length of new tree */
        Lint iter;                      /* iter. of mut./eval. loop */

        lengthfp = lenfp;       /* for lenlog() */
        configalloc();          /* allocate for x and xdash if necessary */
        treecopy(x, inittree);
        bstpush(x, root);       /* initial tree initially best tree */
        len = getplen(x, root);
        lenbest = len;
        if (titleprec[0] != '\0')
                lenlog("%s ", titleprec);
        lenlog("Tree\tBestLen\tCurrLen\tPropLen\n");    /* entitle columns */

        /* do maximum number of iterations */
        for (iter = 0; iter < maxtrees; ++iter)
        {
                rootdash = mutate(xdash, x, root);      /* propose change */
                lendash = getplen(xdash, rootdash);
                lenlog("%ld\t%ld\t%ld\t%ld\n", (long) iter, (long) lenbest,
                 (long) len, (long) lendash);
                deltalen = lendash - len;
                if (deltalen <= 0)      /* accept change */
                {
                        treeswap(&x, &root, &xdash, &rootdash);
                        len = lendash;
                        if (deltalen < 0)       /* very best so far */
                        {
                                lenbest = len;
                                bstclear();     /* discard old bests */
                        }
```

```
                        bstpush(x, root);
                }
                if (!(iter % CHECK_INTERVAL))
                {
                        if (stopnow() == TRUE)  /* stopped by user */
                                return -(len);
                }
        }

        return lenbest;

}       /* end hillclimb() */

Lint randomwalk(const Branch *const inittree, Branchno root,
 const Lint maxtrees, FILE *const lenfp, const char *const titleprec)
/* find 'shortest' tree(s) using random walk (i.e. always accept update)
 * from initial tree in inittree (of root root) to look at maxtrees trees,
 * and return the minimum length;
 * lenfp is for output of current tree length and associated details or
 * NULL for no such output;
 * if stopped prematurely, returns 0 - length */
{
        Branchno rootdash;              /* root of new config. */
        Lint len;                       /* length of current tree */
        Lint lenbest;                   /* length of best tree */
        Lint lendash;                   /* length of new tree */
        Lint iter;                      /* iter. of mut./eval. loop */

        lengthfp = lenfp;               /* for lenlog() */
        configalloc();
        treecopy(x, inittree);
        bstpush(x, root);       /* initial tree initially best tree */
        len = getplen(x, root);
        lenbest = len;
        if (titleprec[0] != '\0')
                lenlog("%s ", titleprec);
        lenlog("Tree\tBestLen\tCurrLen\tProplen\n");    /* entitle columns */

        /* do requested number of iterations */
        for (iter = 0; iter < maxtrees; ++iter)
        {
                rootdash = mutate(xdash, x, root);      /* propose change */
                lendash = getplen(xdash, rootdash);
                lenlog("%ld\t%ld\t%ld%\tld\n", (long) iter, (long) lenbest,
```

```
                (long) len, (long) lendash);
                treeswap(&x, &root, &xdash, &rootdash);         /* accept */
                len = lendash;
                if (len <= lenbest)  /* keep the tree, if new */
                {
                        if (len < lenbest)      /* best so far */
                        {
                                lenbest = len;
                                bstclear();     /* discard old bests */
                        }
                        bstpush(x, root);
                }
                if (!(iter % CHECK_INTERVAL))
                {
                        if (stopnow() == TRUE)
                                return -(lenbest);
                }
        }

        return lenbest;

}       /* end randomwalk() */

static void checkedom(void)
/* crash if a domain error has occurred */
{
        extern int errno;       /* error */

        if (errno == EDOM)
                crash("domain error in solution function");

}       /* end checkedom() */

static void lenlog(const char *const fmt, ...)
/* write a message to the static length file pointer lengthfp, if it is not
 * NULL */
{
        va_list args;   /* arguments */

        if (lengthfp != NULL)
        {
                va_start(args, fmt);
                vfprintf(lengthfp, fmt, args);
                va_end(args);
```

```
        }
        return;

}       /* end lenlog() */
```

C.20 "sops.c"

```
/* ********** sops.c - string operations ********** */

#include "lvb.h"

enum { SAME, DIFFERENT };       /* strings considered same or different */

Integer strcicmp(const char *const s1, const char *const s2)
/* compare strings s1 and s2, whose length may be different,
 * in a case-insensitive way;
 * return 0 if they are the same, non-zero if different */
{
        Integer i;      /* loop counter */
        Integer len1;   /* length of s1 */
        Integer len2;   /* length of s2 */

        len1 = (Integer) strlen(s1);
        len2 = (Integer) strlen(s2);

        if (len1 != len2)       /* can't be identical */
                return (Integer) DIFFERENT;

        for (i = 0; i < len1; ++i)
        {
                if (toupper((int) s1[i]) != toupper((int) s2[i]))
                        return (Integer) DIFFERENT;
        }
        return (Integer) SAME;

}       /* end strcicmp() */

char *nextnonwspc(const char *string)
/* return pointer to next non-white space character in string, or NULL if
 * none */
{
        while (isspace(*string))
```

```
                ++string;
        if (*string)
                return (char *) string;
        else
                return NULL;

}       /* end nextnonwspc() */

char *salloc(const Integer len, const char *const msg)
/* return space for string len letters long (i.e. len + 1 bytes);
 * on failure, crash with message "out of memory: cannot allocate for "
 * followed by msg */
{
        char *s;        /* new space */

        s = (char *) alloc((size_t) (len + 1), msg);
        return s;

}       /* end salloc() */

char *supper(char *const s)
/* make all letters in string s upper case;
 * return s */
{
        char *elementptr = s;

        while (*elementptr)
        {
                *elementptr = (char) toupper((int) *elementptr);
                ++elementptr;
        }
        return s;

}       /* end supper() */
```

C.21 "trops.c"

```
/* ********** trops.c - tree operations ********** */

#include "lvb.h"

#define CLADESEP ",\n"  /* clade separator for trees */
```

```
#define DUMPROW "%d\t%d\t%d\t%d\t%d\n"  /* row format for dumped trees */

/* title for dumped tree columns */
#define DUMPTITLE "Branch:\tParent:\tLeft:\tRight:\tObject:\n"

typedef struct  /* object set derived from a cladogram */
{
        Objno *set;     /* arrays of object sets */
        Objno cnt;      /* sizes of object sets */
}       Objset;

static Branchno arbreroot(Branch *const tree, const Branchno oldroot);
static void clear(Branch *const barray, const Branchno brnch);
static Branchno compress(Branch *const tree, const Branchno root,
 const Branchno brnch);
static void cr_bpnc(const Branch *const barray, const Branchno branch);
static void cr_chaf(const Branch *const barray, const Branchno destination,
 const Branchno newchild);
static void cr_cmpe(Branch *const barray, const Branchno branch);
static void cr_nbo(const Branch *const barray, const Objno obj);
static void cr_tbo(const Branch *const barray, const Objno obj);
static void cr_uxe(FILE *const stream, const char *const msg);
static Branchno cut(Branch *const tree, const Branchno brnch);
static void fillsets(Objset *const sstruct, const Branch *const tree,
 const Branchno root);
static void getobjs(const Branch *const barray, const Branchno root,
 Objno *const objarr, Objno *const cnt);
static Branchno getsister(const Branch *const barray, const Branchno branch);
static void lnk(Branch *const tree, const Branchno root,
 const Branchno destination, const Branchno brnch);
static Integer makesets(const Branch *const tree_1, const Branchno root_1,
 const Branch *const tree_2, const Branchno root_2);
static int objnocmp(const void *o1, const void *o2);
static int osetcmp(const void * oset1, const void *oset2);
static void r_print(FILE *const stream, const Branch *const barray,
 const Branchno root, const Boolean *const haslen);
static void randleaf(Branch *const barray, const Boolean *const leafmask,
 const Objno objs);
static void realgetobjs(const Branch *const barray, const Branchno root,
 Objno *const objarr, Objno *const cnt);
static Branchno reroot(Branch *const barray, const Branchno oldroot,
 const Branchno newroot);
static Boolean *randtopology(Branch *const barray, const Objno nobjs);
static Integer setstcmp(Objset *const oset_1, Objset *const oset_2,
```

```
 const Integer nels);
static void sort(Objset *const oset, const Integer nels);
static Objset *ssarralloc(const Integer nsets, const Integer setsize);
static void ur_print(FILE *const stream, const Branch *const barray,
 const Branchno root);

Objset *sset_1;         /* object sets for tree 1 in comparison */
Objset *sset_2;         /* object sets for tree 2 in comparison */

static void clear(Branch *const barray, const Branchno brnch)
/* initialize all scalars in branch brnch to UNSET */
{
        barray[brnch].object = UNSET;
        barray[brnch].left = UNSET;
        barray[brnch].right = UNSET;
        barray[brnch].parent = UNSET;
        return;

}       /* end clear() */

void treeclear(Branch *const barray)
/* clear all branches in array barray */
{
        extern Branchno nbranches;      /* branches in barray */
        Branchno i;                     /* loop counter */

        for (i = 0; i < nbranches; ++i)
                clear(barray, i);
        return;

}       /* end treeclear() */

Branchno brcnt(void)
/* return number of branches in any binary tree structure */
{
        extern Dataptr matrix;  /* data matrix */

        return (Branchno) (matrix->n * 2 - 3);

}       /* end brcnt() */

Branchno mutate(Branch *const desttree, const Branch *const sourcetree,
 Branchno root)
/* make a copy of the tree sourcetree (of root root) in desttree,
```

```
 * with a random change in topology, the change being caused by taking a random
 * leaf branch and putting it in a random place different from where it was;
 *  return the number of the (possibly changed) root branch in the new tree */
{
        Branchno leafbranch;            /* branch to move */
        Branchno destination;           /* destination of branch to move */
        Branchno free;                  /* branch freed temporarily in move */
        Branchno parent;                /* parent of branch to move */
        Branchno sister;                /* sister of branch to move */
        Branch *tree;                   /* destination tree */
        extern Branchno nbranches;      /* branches in tree */

        /* for ease of reading, set up desttree with an alias, tree */
        tree = desttree;
        treecopy(tree, sourcetree);

        /* get a random leaf */
        do
        {
                leafbranch = (Branchno) randpint(nbranches - 1);
        } while (tree[leafbranch].object == UNSET);

        /* re-root tree if random leaf is the root, since can't move root */
        if (leafbranch == root)
                root = arbreroot(tree, root);

        parent = tree[leafbranch].parent;
        if (parent == UNSET)
                crash("internal error in function mutate(): parent of random\n"
                 "leaf (%d) is UNSET", (int) leafbranch);
        sister = getsister(tree, leafbranch);
        if (sister == UNSET)
                crash("internal error in function mutate(): sister of random\n"
                 "leaf (%d) is UNSET", (int) leafbranch);

        /* get a destination that is not source or its parent or sister */
        do
        {
                destination = (Branchno) randpint(nbranches - 1);
        } while ((destination == leafbranch) || (destination == parent)
         || (destination == sister));

        parent = cut(tree, leafbranch);         /* excise source branch */
        free = compress(tree, root, parent);    /* make tree intact */
```

```
        lnk(tree, root, destination, free);     /* make space for new child */
        childadd(tree, destination, leafbranch);
        return root;

}       /* end mutate() */

static Branchno arbreroot(Branch *const tree, const Branchno oldroot)
/* change tree's root arbitrarily, to a leaf other than oldroot;
 * return the number of the new root */
{
        Branchno newroot = 0;           /* new root */

        /* find a leaf that is not the current root */
        while ((tree[newroot].object == UNSET) || (newroot == oldroot))
                ++newroot;

        reroot(tree, oldroot, newroot);
        return newroot;

}       /* end arbreroot() */

static Branchno reroot(Branch *const barray, const Branchno oldroot,
 const Branchno newroot)
/* change the root of the tree in barray from oldroot to newroot, which must
 * not be the same;
 * return oldroot */
{
        Branchno current;                       /* current branch */
        Branchno parnt;                         /* parent of current branch */
        Branchno sister = UNSET;                /* sister of current branch */
        Branchno previous;                      /* previous branch */
        static Branchno *oldparent = NULL;      /* el. i is old parent of i */
        extern Branchno nbranches;              /* branches in tree */

        /* check new root is a leaf but not the current root */
        if (barray[newroot].object == UNSET)
                crash("internal error in function reroot(): proposed new\n"
                 "root (branch %d) is interior\n", (int) newroot);
        if ((newroot == oldroot))
                crash("internal error in function reroot(): old root and\n"
                 "proposed new root are the same (branch %d)\n",
                 (int) oldroot);

        /* create record of parents as they are now */
```

```
if (oldparent == NULL)  /* 1st call: allocate memory */
{
        oldparent = (Branchno *)
         alloc((size_t) nbranches * sizeof(Branchno),
         "old parent array");
        mempush(oldparent);
}
for (current = 0; current < nbranches; ++current)
        oldparent[current] = barray[current].parent;

current = newroot;
previous = UNSET;
while (current != oldroot)
{
        if (current == UNSET)
                crash("internal error in function reroot(): current\n"
                 "branch is UNSET");
        parnt = oldparent[current];             /* original parent */
        if (current == barray[parnt].left)
                sister = barray[parnt].right;
        else if (current == barray[parnt].right)
                sister = barray[parnt].left;
        else    /* error in tree structure */
                crash("internal error in function reroot(): current\n"
                 "branch %d has old parent %d, but old parent does\n"
                  "not have it as a child", (int) current,
                  (int) parnt);
        barray[current].parent = previous;      /* now chld of prev. */

        /* make former parent the new left child, and former sister the
         new right child of the current branch */
        barray[current].left = parnt;
        barray[current].right = sister;
        barray[parnt].parent = current;
        barray[sister].parent = current;

        /* move towards orig. root, i.e. to orig. parnt of curr. br. */
        previous = current;
        current = parnt;
}

/* former root is now a normal leaf, without descendants */
barray[oldroot].left = UNSET;
barray[oldroot].right = UNSET;
```

```
        return oldroot;

}       /* end reroot() */

Branchno objreroot(Branch *const barray, const Branchno oldroot,
 const Objno newrobj)
/* change the root of the tree in barray from oldroot to the branch associated
 * with object newrobj;
 * return the number of the new root */
{
        Branchno i;                     /* loop counter */
        Branchno newroot = UNSET;       /* new root */
        extern Branchno nbranches;      /* number of branches in tree */
        Boolean foundroot = FALSE;      /* have found new root object */

        for (i = 0; i < nbranches; ++i)
        {
                if (barray[i].object == newrobj)
                {
                        if (foundroot == TRUE)
                                cr_tbo(barray, newrobj);
                        newroot = i;
                        foundroot = TRUE;
                }
        }
        if (newroot == UNSET)
                cr_nbo(barray, newrobj);
        else if (newroot != oldroot)
                reroot(barray, oldroot, newroot);
        return newroot;

}       /* end objreroot() */

static void cr_nbo(const Branch *const barray, const Objno obj)
/* crash because no branches of tree in barray have proposed root object obj
 * as their object */
{
        crash("internal error: in tree array %p, no branch is associated\n"
         "with proposed root object %d", (void *) barray, (int) obj);

}       /* end cr_nbo() */

static void cr_tbo(const Branch *const barray, const Objno obj)
```

```
/* crash because 2 branches of tree in barray have object obj */
{
        crash("internal error in tree array %p, 2 branch records claim\n"
         "object %d", (void *) barray, (int) obj);

}       /* end cr_tbo() */

static Branchno getsister(const Branch *const barray, const Branchno branch)
/* return number of sister of branch branch in tree in barray, or UNSET if
 * branch has none */
{
        Branchno parnt;         /* parent of current branch */

        parnt = barray[branch].parent;
        if (parnt == UNSET)
                return UNSET;
        if (branch == barray[parnt].left)
                return barray[parnt].right;
        else if (branch == barray[parnt].right)
                return barray[parnt].left;
        else    /* error in tree structure */
        {
                cr_bpnc(barray, branch);
                return 0;       /* NEVER reached but it shuts up compilers */
        }

}       /* end getsister() */

Branchno getroot(const Branch *const barray)
/* return index of root branch in tree in barray */
{
        Branchno i;                     /* loop counter */
        extern Branchno nbranches;      /* branches in tree */

        for (i = 0; i < nbranches; ++i)
        {
                if ((barray[i].parent == UNSET) && (barray[i].object != UNSET))
                        return i;
        }
        crash("internal error in tree array %p of %d branches:\n"
         "there is no root branch", (void *) barray, (int) nbranches);
        return 0;       /* NEVER reached but it shuts up compilers */
}       /* end getroot() */
```

```
Branchno childadd(Branch *const tree, const Branchno destination,
 const Branchno newchild)
/* replace unset child of destination with newchild, and return destination */
{
        if (tree[destination].right == UNSET)
                tree[destination].right = newchild;
        else if (tree[destination].left == UNSET)
                tree[destination].left = newchild;
        else    /* error: destination already has 2 children */
                cr_chaf(tree, destination, newchild);
        tree[newchild].parent = destination;
        return destination;

}       /* end childadd() */

static void cr_chaf(const Branch *const barray, const Branchno destination,
 const Branchno newchild)
/* crash because we want to add branch newchild to the children of branch
 * destination in tree in barray, but it already has 2 so there is no room */
{
        crash("internal error in tree array %p: cannot make branch %d a\n"
         "child of branch %d since this already has 2 children (left is\n"
         "branch %d, right is branch %d)", (void *) barray, (int) newchild,
         (int) destination, (int) barray[destination].left,
         (int) barray[destination].right);

}       /* end cr_chaf() */

static void lnk(Branch *const tree, const Branchno root,
 const Branchno destination, const Branchno brnch)
/* link brnch into the tree tree (of root root) by adding it between
 * destination and destination's children */
{
        Branchno lft;   /* left child of destination */
        Branchno rght;  /* right child of destination */

        lft = tree[destination].left;
        rght = tree[destination].right;

        tree[brnch].left = lft;
        tree[brnch].right = rght;
        tree[brnch].parent = destination;

        /* make brnch the only child of dest.; left is chosen arbitrarily */
```

```
        tree[destination].left = brnch;
        tree[destination].right = UNSET;

        /* give destination's children brnch as their parent, if they exist */
        if (lft != UNSET)
                tree[lft].parent = brnch;
        if(rght != UNSET)
                tree[rght].parent = brnch;

        /* move dest's object, if any, to brnch, unless destination is root */
        if (destination == root)
                tree[brnch].object = UNSET;
        else
        {
                tree[brnch].object = tree[destination].object;
                tree[destination].object = UNSET;
        }

        return;

}       /* end lnk () */

static Branchno compress(Branch *const tree, const Branchno root,
 const Branchno brnch)
/* collapse the branch leading from brnch, assuming there is only one such;
 * return the number of the freed branch */
{
        Branchno child = UNSET;         /* live child of brnch */
        Branchno lft;                   /* left child of child */
        Branchno rght;                  /* right child of child */

        /* find live child, or crash if there are 2 */
        if (tree[brnch].left != UNSET)
                child = tree[brnch].left;
        else if (tree[brnch].right != UNSET)
                child = tree[brnch].right;
        else    /* error: branch has two children */
                cr_cmpe(tree, brnch);

        lft = tree[child].left;
        rght = tree[child].right;
        tree[brnch].left = lft;
        tree[brnch].right = rght;
        if (lft != UNSET)
```

```
                tree[lft].parent = brnch;
        if (rght != UNSET)
                tree[rght].parent = brnch;

        /* give obj. of freed child, unless root (would UNSET root's object) */
        if (brnch != root)
                tree[brnch].object = tree[child].object;

        clear(tree, child);
        return child;

}       /* end compress() */

static void cr_cmpe(Branch *const barray, const Branchno branch)
/* crash because branch branch in array barray has 2 children, but 1 was
 * expected [by compress()] */
{
        crash("internal error in tree array %p: branch %d has 2 children\n"
         "when 1 was expected\n", (void *) barray, (int) branch);

}       /* end cr_cmpe() */

static Branchno cut(Branch *const tree, const Branchno brnch)
/* excise branch brnch from tree;
return former parent of excised branch */
{
        Branchno parnt;         /* parent of branch to cut */

        parnt = tree[brnch].parent;
        if (tree[parnt].left == brnch)
                tree[parnt].left = UNSET;
        else if (tree[parnt].right == brnch)
                tree[parnt].right = UNSET;
        else    /* error in tree structure: branch and parent not connected */
                cr_bpnc(tree, brnch);

        tree[brnch].parent = UNSET;
        return parnt;

}       /* end cut() */

static void cr_bpnc(const Branch *const barray, const Branchno branch)
/* crash because branch branch in tree in barray is not connected to its
 * parent, i.e. it is not a child of the branch it claims as parent, according
```

```
 * to that 'parent's' record of its own children */
{
        const Branchno parnt = barray[branch].parent;   /* parent of branch */

        crash("internal error in tree array %p: branch record %d says it has\n"
         "parent %d, but branch record %d has left child %d and right child "
         "%d", (void *) barray, (int) branch, (int) parnt, (int) parnt,
         (int) barray[parnt].left, (int) barray[parnt].right);

}       /* end cr_bpnc() */

void treecopy(Branch *const dest, const Branch *const source)
/* copy tree from source to dest;
dest must have enough space */
{
        extern Branchno nbranches;      /* branches in tree */

        memcpy(dest, source, (size_t) nbranches * sizeof(Branch));
        return;

}       /* end treecopy() */

void randtree(Branch *const barray)
/* fill barray with a random tree, where barray[0] is the root */
{
        Boolean *leafmask;      /* TRUE where branch in array is a leaf */
        extern Dataptr matrix;  /* data matrix */

        treeclear(barray);
        leafmask = randtopology(barray, matrix->n);
        randleaf(barray, leafmask, matrix->n);
        return;

}       /* end randtree() */

Branch *treealloc(void)
/* return array of extern nbranches branches with scalars all UNSET, or NULL if
 * impossible;
 * N.B. pushes array on to the general memstack */
{
        Branch *barray;                 /* tree */
        extern Branchno nbranches;      /* branches per tree */

        barray = (Branch *) malloc((size_t) nbranches * sizeof(Branch));
```

```
        if (barray != NULL)
        {
                mempush(barray);
                treeclear(barray);
        }
        return barray;

}       /* end treealloc() */

static Boolean *randtopology(Branch *const barray, const Objno nobjs)
/* fill barray with tree of random topology, where barray[0] is root;
 * return static array where element i is TRUE if barray[i] is a leaf,
 * or, FALSE if it is not;
 * this array will be overwritten in subsequent calls and is pushed on to the
 * general memstack */
{
        Branchno i;                     /* loop counter */
        Branchno leaves = 0;            /* number of leaves */
        Branchno nextfree = 0;          /* next unused el. of barray */
        Branchno togrow;                /* random candidate for sprouting */
        static Boolean *isleaf = NULL;  /* return value */
        extern Branchno nbranches;      /* branches in tree */

        if (isleaf == NULL)     /* 1st call, allocate memory */
        {
                isleaf = (Boolean *)
                 alloc((size_t) nbranches * sizeof(Boolean), "leaf mask");
                mempush(isleaf);
        }

        /* clear the leaf mask */
        for (i = 0; i < nbranches; ++i)
                isleaf[i] = FALSE;

        /* start with initial tree of 3 leaves */
        barray[0].parent = UNSET;
        isleaf[nextfree++] = TRUE;
        barray[0].left = nextfree;
        barray[nextfree].parent = 0;
        isleaf[nextfree++] = TRUE;
        barray[0].right = nextfree;
        barray[nextfree].parent = 0;
        isleaf[nextfree++] = TRUE;
        leaves = 3;
```

```
        /* sprout! */
        while(leaves < nobjs)
        {
                do      /* select a random leaf other than the root */
                {
                        togrow = 1 + (Branchno) randpint(nextfree - 2);
                } while (isleaf[togrow] == FALSE);
                /* left child */
                barray[togrow].left = nextfree;
                barray[nextfree].parent = togrow;
                isleaf[nextfree++] = TRUE;
                /* right child */
                barray[togrow].right = nextfree;
                barray[nextfree].parent = togrow;
                isleaf[nextfree++] = TRUE;
                /* other updates */
                isleaf[togrow] = FALSE;
                ++leaves;
        }

        return isleaf;

}       /* end randtopology() */

static void randleaf(Branch *const barray, const Boolean *const leafmask,
 const Objno objs)
/* randomly assign objects numbered 0 to objs - 1 to leaves of tree in barray;
leaves in barray must be indicated by corresponding TRUEs in leafmask */
{
        Objno assigned = 0;             /* for safety: should == objs at end */
        Objno candidate;                /* random object */
        Branchno i;                     /* loop counter */
        static Boolean *used = NULL;    /* el. i TRUE if obj. i has leaf */
        extern Branchno nbranches;      /* number of branches in tree */

        if (used == NULL)               /* 1st call, allocate memory */
        {
                used = (Boolean *) alloc((size_t) objs * sizeof(Boolean),
                 "used object mask");
                mempush(used);
        }

        /* clear 'used' array */
```

```
        for (i = 0; i < objs; ++i)
                used[i] = FALSE;

        /* assign an object to every leaf */
        for (i = 0; i < nbranches; ++i)
        {
                if (leafmask[i] == TRUE)        /* leaf, requires object */
                {
                        do      /* get a new object number */
                        {
                                candidate = randpint(objs - 1);
                        } while(used[candidate] == TRUE);
                        /* assign object to leaf */
                        barray[i].object = candidate;
                        used[candidate] = TRUE;
                        ++assigned;
                }
        }

        /* for safety, check on number of objects assigned */
        if (assigned != objs)
                crash("internal error in function randleaf(): %d objects\n"
                 "exist, but have assigned %d objects to branches", (int) objs,
                 (int) assigned);

        return;

}       /* end randleaf() */

void treeswap(Branch **const tree1, Branchno *const root1,
 Branch **const tree2, Branchno *const root2)
/* swap trees pointed to by tree1 and tree2;
 * also swap records of their roots, as pointed to by root1 and root2 */
{
        Branchno tmproot;       /* temporary value-holder for swapping roots */
        Branch *tmptree;        /* temporary value-holder for swapping trees */

        /* swap pointers to branch arrays */
        tmptree = *tree1;
        *tree1 = *tree2;
        *tree2 = tmptree;

        /* swap roots */
        tmproot = *root1;
```

```
        *root1 = *root2;
        *root2 = tmproot;

        return;

}       /* end treeswap() */

void treedump(FILE *const stream, const Branch *const tree)
/* send tree as table of integers to file pointed to by stream */
{
        Branchno i;                     /* loop counter */
        Objno obj;                      /* obj. assoc'd with current branch */
        extern Branchno nbranches;      /* branches in tree */

        fprintf(stream, DUMPTITLE);
        for (i = 0; i < nbranches; ++i)
        {
                obj = tree[i].object;
                fprintf(stream, DUMPROW, (int) i, (int) tree[i].parent,
                 (int) tree[i].left, (int) tree[i].right, (int) obj);
        }
        putc('\n', stream);
        if(ferror(stream) != 0)
                cr_uxe(stream, "dumping tree");
        return;

}       /* end treedump() */

static void cr_uxe(FILE *const stream, const char *const msg)
/* crash because of problem on file stream, with message consisting of
 * "FATAL ERROR: ", then "file error " if the error indicator for stream is set
 * or "unexpected end of file " if not, then "when ", followed by msg */
{
        if (ferror(stream))
                crash("file error when %s", msg);
        else
                crash("unexpected end of file when %s", msg);

}       /* end cr_uxe */

void trundump(FILE *const stream, Branch *const barray)
/* read the dumped tree at the current position in stream into barray,
 * rerooting where necessary to ensure barray[0] is the root */
{
```

```
        const char act[] = "reading "
         "dumped tree file";            /* activity str. for crash messages */
        int i = 0;                      /* current branch */
        int inchar;                     /* character of input */
        int par;                        /* parent of current branch */
        int lft;                        /* left child of current branch */
        int rght;                       /* right child of current branch */
        int obj;                        /* obj. associated with curr. branch */
        Branchno oldroot;               /* root of dumped tree */
        extern Branchno nbranches;      /* branches per tree */

        /* move past column titles */
        while ((inchar = getc(stream)) != '0')  /* go to first branch number */
        {
                if (inchar == EOF)
                        cr_uxe(stream, act);
        }
        if (ungetc('0', stream) != '0')
                crash("file error when %s", act);

        /* get tree */
        while (i < (int) (nbranches - 1))
        {
                if (fscanf(stream, DUMPROW, &i, &par, &lft, &rght, &obj) != 5)
                        crash("error %s", act);
                barray[i].parent = (Branchno) par;
                barray[i].left = (Branchno) lft;
                barray[i].right = (Branchno) rght;
                barray[i].object = (Objno) obj;
        }
        if (ferror(stream) != 0)
                cr_uxe(stream, act);

        /* re-root tree if necessary */
        oldroot = getroot(barray);
        if (oldroot != 0)
                reroot(barray, oldroot, 0);

        return;

}       /* end trundump() */

void treeprint (FILE *const stream, const Branch *const barray,
 const Branchno root, const Boolean *const haslen)
```

```
/* print tree in barray (of root root) in bracketed text form to stream stream;
 * if haslen is NULL, the tree is output unrooted;
 * otherwise, haslen is an array FALSE where branches are to be collapsed and
 * TRUE elsewhere */
{
        if (haslen == NULL)
                ur_print(stream, barray, root);
        else
                r_print(stream, barray, root, haslen);
        return;

}       /* end treeprint() */

static void ur_print(FILE *const stream, const Branch *const barray,
 const Branchno root)
/* send tree in barray, of root root, to file pointed to by stream in
 * unrooted form */
{
        Branchno obj;                           /* current object */
        static Boolean doneabsroot = FALSE;     /* have output root of tree */
        static Boolean usecomma;                /* do output clade separator */
        extern Dataptr matrix;                  /* data matrix */

        obj = barray[root].object;

        if (doneabsroot == FALSE)       /* print whole tree */
        {
                /* start tree */
                fprintf(stream, "(%s", matrix->rowtitle[obj]);
                usecomma = TRUE;
                doneabsroot = TRUE;

                ur_print(stream, barray, barray[root].left);
                ur_print(stream, barray, barray[root].right);

                /* end tree */
                fputs(");\n", stream);
                if (ferror(stream))
                        crash("file error when writing unrooted tree");

                /* clean up for next call */
                usecomma = FALSE;
                doneabsroot = FALSE;
        }
```

```
        else    /* print remainder of tree */
        {
                if (usecomma == TRUE)
                        fputs(CLADESEP, stream);
                if (barray[root].object != UNSET)       /* leaf */
                {
                        fprintf(stream, "%s", matrix->rowtitle[obj]);
                        usecomma = TRUE;
                }
                else
                {
                        putc('(', stream);
                        usecomma = FALSE;
                        ur_print(stream, barray, barray[root].left);
                        ur_print(stream, barray, barray[root].right);
                        fputc(')', stream);
                        usecomma = TRUE;
                }
        }
        return;

}       /* end ur_print() */

static void r_print(FILE *const stream, const Branch *const barray,
 const Branchno root, const Boolean *const haslen)
/* send tree in barray, of root root, to file pointed to by stream in
 * rooted form;
 * collapse ingroup branches where the corresponding element of haslen is
 * FALSE */
{
        Branchno obj;                           /* current object */
        static Boolean doneabsroot = FALSE;     /* have output root of tree */
        static Boolean usecomma;                /* should output a comma */
        extern Dataptr matrix;                  /* data matrix */

        obj = barray[root].object;

        if (doneabsroot == FALSE)       /* print whole tree */
        {
                /* start tree */
                fprintf(stream, "(%s%s(", matrix->rowtitle[obj], CLADESEP);
                usecomma = FALSE;
                doneabsroot = TRUE;
```

```
                treeprint(stream, barray, barray[root].left, haslen);
                treeprint(stream, barray, barray[root].right, haslen);

                /* end tree */
                fputs("));\n", stream);
                if (ferror(stream))
                        crash("file error when writing rooted tree");

                /* clean up for next call */
                usecomma = FALSE;
                doneabsroot = FALSE;
        }
        else    /* print remainder of tree */
        {
                if (usecomma == TRUE)
                        fputs(CLADESEP, stream);
                if (barray[root].object != UNSET)       /* leaf */
                {
                        fprintf(stream, "%s", matrix->rowtitle[obj]);
                        usecomma = TRUE;
                }
                else
                {
                        if (haslen[root])       /* start new clade */
                                putc('(', stream);
                        usecomma = FALSE;       /* did ',' after prev. name */
                        treeprint(stream, barray, barray[root].left, haslen);
                        treeprint(stream, barray, barray[root].right, haslen);
                        if (haslen[root])       /* terminate clade */
                        {
                                fputc(')', stream);
                                usecomma = TRUE;
                        }
                }
        }
        return;

}       /* end r_print() */

Integer treecmp(const Branch *const tree_1, const Branchno root_1,
 const Branch *const tree_2, Branchno root_2)
/* return 0 if the topology of tree_1 (of root root_1) is the same as that
 * of tree_2 (of root root_2), or non-zero if different */
{
```

```
        static Branch *copy_2 = NULL;   /* possibly re-rooted tree 2 */
        Integer nsets;                  /* elements per set array */

        /* set up local copy of tree 2 with same root as tree 1 */
        if (copy_2 == NULL)     /* first call, allocate memory */
        {
                copy_2 = treealloc();
                if (copy_2 == NULL)
                        crash("out of memory: cannot allocate for copy of\n"
                         "tree for comparison");
        }
        treecopy(copy_2, tree_2);
        root_2 = objreroot(copy_2, root_2, tree_1[root_1].object);

        nsets = makesets(tree_1, root_1, copy_2, root_2);
        return setstcmp(sset_1, sset_2, nsets);

}       /* end treecmp() */

static Integer setstcmp(Objset *const oset_1, Objset *const oset_2,
 const Integer nels)
/* return 0 if the same sets of objects are in oset_1 and oset_2,
 * and non-zero otherwise */
{
        Integer i;      /* loop counter */

        /* sort the set arrays and their constituent sets */
        sort(oset_1, nels);
        sort(oset_2, nels);

        /* compare the set arrays */
        for (i = 0; i < nels; ++i)
        {
                if (oset_1[i].cnt != oset_2[i].cnt)
                        return 1;
                else if (memcmp(oset_1[i].set, oset_2[i].set,
                 (size_t) oset_1[i].cnt * sizeof(Objno)) != 0)
                        return 1;
        }

        return 0;

}       /* end setstcmp() */
```

```
static void sort(Objset *const oset, const Integer nels)
/* sort the nels object sets in oset so that each is in order, and sort oset so
 * that the sets themselves are in order of size and content */
{
        Integer i;      /* loop counter */

        /* first sort each set member list */
        for (i = 0; i < nels; ++i)
                qsort(oset[i].set, (size_t) oset[i].cnt, sizeof(Objno),
                 objnocmp);

        /* now sort the arrays of sets by size and content */
        for (i = 0; i < nels; ++i)
                qsort(oset, (size_t) nels, sizeof(Objset), osetcmp);

        return;

 }      /* end sort() */

static int osetcmp(const void *oset1, const void *oset2)
/* comparison function for object sets (type Objset):
 * return negative if *oset1 is a smaller set of objects than *oset2 or is the
 * same size but with a list of elements that compares lower;
 * return positive if *ostet1 is bigger or the same size but with a list of
 * elements that compares higher;
 * return 0 if they are the same;
 * N.B. the object numbers must be in numerical order within the sets */
{
        Integer i;                                      /* loop counter */
        const Objset loset_1 = *((Objset *) oset1);     /* simpler alias */
        const Objset loset_2 = *((Objset *) oset2);     /* simpler alias */

        /* sets of different size differ */
        if (loset_1.cnt != loset_2.cnt)
                return (int) (loset_1.cnt - loset_2.cnt);

        /* then see if sets' contents differ */
        for (i = 0; i < loset_1.cnt; ++i)
        {
                if (loset_1.set[i] != loset_2.set[i])
                        return (int) (loset_1.set[i] - loset_2.set[i]);
        }

        return 0;       /* really they are the same */
```

```
}       /* end osetcmp() */

static Integer makesets(const Branch *const tree_1, const Branchno root_1,
 const Branch *const tree_2, const Branchno root_2)
/* fill static sset_1 and static sset_2 with arrays of object sets for
 * tree_1 and tree_2 (of root_1 and root_2 respectively), and return the
 * extent of each array;
 * the trees must have the same object in the root branch;
 * arrays will be overwritten on subsequent calls */
{
        extern Dataptr matrix;                          /* data matrix */
        const Integer nsets = (Integer) matrix->n - 3;  /* sets per tree */
        const Integer mssz = (Integer) matrix->n - 2;   /* max. objs per set */

        if (sset_1 == NULL)     /* first call, allocate memory */
        {
                sset_1 = ssarralloc(nsets, mssz);
                sset_2 = ssarralloc(nsets, mssz);
        }
        fillsets(sset_1, tree_1, root_1);
        fillsets(sset_2, tree_2, root_2);
        return nsets;   /* set by ssarralloc() */

}       /* end makesets() */

static Objset *ssarralloc(const Integer nsets, const Integer setsize)
/* return a new object set array with space for nsets object sets of setsize
 * objects each;
 * new memory is pushed on to the general memstack */
{
        Objset *nobjset;                                /* new set array */
        Integer i;                                      /* loop counter */

        nobjset = (Objset *)
         alloc((size_t) nsets * sizeof(Objset), "object set struct array");
        mempush(nobjset);
        for (i = 0; i < nsets; ++i)
        {
                nobjset[i].set = (Objno *) alloc((size_t)
                 setsize * sizeof(Objno), "object set object arrays");
                mempush(nobjset[i].set);
                nobjset[i].cnt = UNSET;
        }
```

```
        return nobjset;

}       /* end ssarralloc() */

static void fillsets(Objset *const sstruct, const Branch *const tree,
 const Branchno root)
/* fill object sets in sstruct with all sets of objects in tree tree, descended
 * from but not including root and not including sets of one object */
{
        static Integer i = UNSET;               /* current set being filled */

        if (i == UNSET)         /* not a recursive call */
        {
                i = 0;

                /* avoid generating sets for true root and leaves */
                if (tree[tree[root].left].object == UNSET)      /* interior */
                        fillsets(sstruct, tree, tree[root].left);
                if (tree[tree[root].right].object == UNSET)     /* interior */
                fillsets(sstruct, tree, tree[root].right);

                i = UNSET;      /* clean up for next non-recursive call */
                return;
        }
        if (tree[root].left != UNSET)   /* not leaf */
        {
                getobjs(tree, root, sstruct[i].set, &sstruct[i].cnt);
                ++i;
                fillsets(sstruct, tree, tree[root].left);
                fillsets(sstruct, tree, tree[root].right);
                return;
        }

}       /* end fillsets */

static void getobjs(const Branch *const barray, const Branchno root,
 Objno *const objarr, Objno *const cnt)
/* fill objarr (which must be large enough) with numbers of all objects
 * in the tree in barray in the clade starting at branch root;
 * fill the number pointed to by cnt with the number of objects found
 * (i.e. the number of elements written to objarr) */
{
        *cnt = 0;
        realgetobjs(barray, root, objarr, cnt);
```

```
}       /* end getobjs() */

static void realgetobjs(const Branch *const barray, const Branchno root,
 Objno *const objarr, Objno *const cnt)
/* fill objarr (which must be large enough) with numbers of all objects
 * in the tree in barray in the clade starting at branch root;
 * fill the number pointed to by cnt, which must initially be zero, with
 * the number of objects found (i.e. the number of elements written to objarr);
 * this function should not be called from anywhere except getobjs(), which is
 * a safer interface */
{
        if (barray[root].object != UNSET)
        {
                objarr[*cnt] = barray[root].object;
                ++(*cnt);
        }
        else
        {
                if (barray[root].left != UNSET)
                        realgetobjs(barray, barray[root].left, objarr, cnt);
                if (barray[root].right != UNSET)
                        realgetobjs(barray, barray[root].right, objarr, cnt);
        }
        return;

}       /* end realgetobjs() */

static int objnocmp(const void *o1, const void *o2)
/* comparison function for comparing object numbers:
 * return negative if *o1 < *o2, zero if *o1 == *o2,
 * or positive if *o1 > *o2 */
{

        return((int) (*((Objno *) o1) - *((Objno *) o2)));

}       /* end objnocmp() */
```

Fitch, W. M. 1970. Distinguishing homologous from analogous proteins. 'Systematic Zoology' 19(2): 99-113.

Fitch, W. M. 1971. Toward defining the course of evolution: minimum change for a specific tree topology. 'Systematic Zoology' 20(4): 406-416.

Kernighan, B. W. & Ritchie, D. M. 1988. The C programming language, 2nd edition (Englewood Cliffs, New Jersey: Prentice Hall)

Kirkpatrick, S., Gelatt, C. D. & Vecci, M. P. 1983. Optimization by simulated annealing. 'Science' 220(4598): 671-680

Maddison, W. P & Maddison, D. R. 1992. 'MacClade: analysis of phylogeny and character evolution', version 3 (Sunderland, Massachusetts: Sinauer)

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. 1953. Equation of state calculations by fast computing machines. 'Journal of Chemical Physics' 21(6): 1087-1092

Swofford, D. L. 1993. 'PAUP: phylogenetic analysis using parsimony, version 3.1' (Champaign, Illinois: Illinois Natural History Survey)

Appendix E

LVB 1.0A

Almost as soon as LVB 1.0 had been released, during the summer of 1997, some minor problems were detected by the author. This led to release of LVB 1.0A on 18 August 1997. LVB 1.0A 18 August 1997 is the version of LVB used for all experiments in this thesis.

An insert describing the changes made has been distributed with printed LVB manuals [31] since then. The insert is now available as plain (ASCII) text on the LVB Web page (Section G.1). This ASCII version is given below.

BUG FIXES IN LVB VERSION 1.0A 18 AUGUST 1997

DOCUMENT HISTORY

Revision date: 24 March 1999

Supersedes: previous document entitled Bug fixes in LVB version 1.0A 18 August 1997.

Significant changes in the current version of this document: Information on transient problems compiling the Windows 95 version of LVB has been removed. The author's postal address has changed slightly. The author's e-mail address has been replaced by the address of the LVB Web page. A section entitled Document history has been added. The three introductory paragraphs have been entitled Introduction, and subsequent sections have been re-numbered.

Note: No changes have been made to the source code of the program, as generally distributed, since LVB 1.0A was released on 18 August 1997. No changes have been made to the body of the printed manual LVB 1.0: reconstructing evolution with parsimony and simulated annealing since it was first published.

1. INTRODUCTION

Five bugs in LVB 1.0 have been fixed in LVB 1.0A. The bugs were minor and would have been noticed by very few users.

The user interface and internal interface remain the same as in LVB 1.0, so the same manual (Barker, D. 1997. LVB 1.0: reconstructing evolution with parsimony and simulated annealing) covers both versions.

The changes to the source code of LVB 1.0 made in LVB 1.0A are listed below. This is for information only. All versions of LVB 1.0A already include these changes.

2. RESTARTING AFTER A PAUSE SOMETIMES FAILED IN LVB 1.0

2.1. Summary.

On some restarts, LVB 1.0 would log an internal error message and exit early. This was because the restart function made an assumption about the structure of the saved tree that was incorrect in some cases. This has been fixed in LVB 1.0A.

2.2. Complete detail.

The bug was in function trundump(). This section of file trops.c in LVB 1.0:

```
        if (oldroot != 0)
                reroot(barray, oldroot, 0);

        return;
```

has been replaced in LVB 1.0A by this:

```
        if (oldroot != 0)
        /* BEGIN BUG FIX in LVB 1.0: allow original branch 0 to be internal */
        {
                if (barray[0].object != UNSET)  /* 0 is a leaf, can reroot() */
                        reroot(barray, oldroot, 0);
                else                            /* 0 intern., can't reroot() */
                {
                        Branch tmp;     /* temporary branch for swapping */
                        Branchno br;    /* loop counter */
                        for (br = 0; br < nbranches; ++br)
                        {
                                if (barray[br].left == 0)
                                        barray[br].left = oldroot;
                                if (barray[br].right == 0)
                                        barray[br].right = oldroot;
                                if (barray[br].parent == 0)
                                        barray[br].parent = oldroot;
                                else if (barray[br].parent == oldroot)
                                        barray[br].parent = 0;
                        }
                        tmp = barray[0];
                        barray[0] = barray[oldroot];
                        barray[oldroot] = tmp;
                }
        }
        /* END BUG FIX */

        return;
```

3. COMPILATION OF LVB FAILED ON SOME SYSTEMS

3.1. Summary.

This was because of an unnecessary assumption in the code that is not always correct. This has been fixed in LVB 1.0A.

3.2. Complete detail.

The bug was in function checkedom(). This section of file solve.c in LVB 1.0:

```
{
        extern int errno;       /* error */

        if (errno == EDOM)
```

has been replaced in LVB 1.0A by this:

```
{
        /* BEGIN BUG FIX in LVB 1.0A: allow any integral expr. for errno */
/*      extern int errno;       */  /* error */
        /* END BUG FIX */

        if (errno == EDOM)
```

4. matchchar none IN THE FILE params WAS NOT ALLOWED BY LVB 1.0

4.1. Summary.

This was due to an error in the params parser. This has been fixed in LVB 1.0A.

4.2. Complete detail.

The bug was in function getparam(). This section of file getparam.c in LVB 1.0:

```
        if (strlen(value) != (size_t) 1)
                crash("invalid value for matchchar in file\n"
                 "%s: should be a single machine character,\n"
                 "or use 'matchchar none' for none", RCFNAM);
        prms->matchchar = (char) toupper((int) value[0]);
```

has been replaced in LVB 1.0A by this:

```
        if (strlen(value) != (size_t) 1)
```

```
        /* BEGIN BUG FIX in LVB 1.0A: allow 'matchchar none' */
        {
                if (strcicmp(value, UNSETNAM) == 0)
                        prms->matchchar = '\0';
                else
                        crash("invalid value for matchchar\n"
                         "in file %s: should be a single "
                         "machine character, or use\n"
                         "'matchchar none' for none", RCFNAM);
        }
        else
        /* END BUG FIX */
                prms->matchchar = (char)
                 toupper((int) value[0]);
```

5. LVB 1.0 FAILED AT AN EARLY STAGE UNDER WINDOWS

5.1. Summary.

LVB 1.0 attempted to close the standard input, because it is unused. This failed and caused LVB to exit with an error message under Windows. This problem has been solved in LVB 1.0A.

5.2. Complete detail.

The problem was in function main(). This section of file main.c in LVB 1.0:

```
        }
        clnclose(stdin, "stdin");  /* no standard input */

        settraps();
        rcstruct = *(getparam());

        /* entitle log file */
        fprintf(logfp, "This is %s version %s %s\n", PROGNAM, VERSION,
         SUBVERSION);
        /* global files: part 2 */
        clnclose(stderr, "stderr");
        clnclose(stdout, "stdout");

        matrix = matrin(MATFNAM);  /* get data matrix */
```

has been replaced in LVB 1.0A by this:

```
        }
        /* BEGIN BUG FIX in LVB 1.0A: to work under Windows (part 1 of 2) */
/*      clnclose(stdin, "stdin");       */      /* no standard input */
        /* END BUG FIX */

        settraps();
        rcstruct = *(getparam())


        /* entitle log file */
        fprintf(logfp, "This is %s version %s %s\n", PROGNAM, VERSION,
         SUBVERSION);

        /* global files: part 2 */
        /* BEGIN BUG FIX in LVB 1.0A: to work under Windows (part 2 of 2) */
/*      clnclose(stderr, "stderr");     */
/*      clnclose(stdout, "stdout");     */
        /* END BUG FIX */

        matrix = matrin(MATFNAM);       /* get data matrix */
```

6. PROTECTION AGAINST DIVISION BY ZERO IS ENHANCED IN LVB 1.0A

6.1. Summary.

In LVB 1.0, during simulated annealing there may have been a possibility of division by zero (though so far as I know this never occurred in practice).

6.2. Complete detail.

This section of file solve.c in LVB 1.0:

```
        r_lenmin = (Real) lenmin;

        while (t >= (Real) 0) /* inf. loop might go < 0 at accuracy lim.? */
        {
```

has been replaced in version 1.0A by:

```
        r_lenmin = (Real) lenmin;

        /* BEGIN BUG FIX in LVB 1.0A: enhance protection against div. by 0.0 */
```

```
        while (t > (Real) 0)    /* inf. loop might go < 0 at accuracy lim.? */
        /* END BUG FIX */
        {
```

AUTHOR

Further information of these bugs may be obtained from the author. Please also contact me if you have discovered a bug in LVB 1.0A.

Daniel Barker
Biocomputing Research Unit
Institute of Cell and Molecular Biology
University of Edinburgh
King's Buildings
Mayfield Road
Edinburgh
EH9 3JR
United Kingdom

Web page http://www.icmb.ed.ac.uk/sokal.html

Appendix F

LVB 1.0A Release Notes

The release notes for the pre-compiled versions of LVB which may be downloaded from the LVB Web page (Section G.1) are given below. They refer to LVB 1.0A (Appendix E).

These notes were necessary to record details of compilation, but also because these systems differ from the Unix systems on which LVB was developed. For example, it is not possible to use the computer for any other purpose whilst LVB runs under MacOS.

F.1 PowerPC MacOS

LVB 1.0A 18 AUGUST 1997 POWERPC MACOS VERSION RELEASE NOTES

DISCLAIMER

LVB has been checked for viruses and is apparently not infected. However, those worried about viruses or the details of compilation should compile the source code themselves using an ANSI (ISO) C compiler.

SYSTEM REQUIREMENTS

Minimum system: Apple or compatible PowerPC computer, running MacOS System 7, with 10 MB free RAM.

Recommended system: Apple or compatible PowerPC computer, running MacOS System 7, with 16 MB free RAM.

Scope: This release of LVB is for PowerPC-based computers only. It will not work on earlier Apple hardware, even if it is running MacOS System 7.

SUPPORT, COPIES OF PROGRAM AND MANUAL, REGISTRATION

Support, the program, the manual and user registration are available free from:

http://www.icmb.ed.ac.uk/sokal.html

or by e-mailing the author as sokal@holyrood.ed.ac.uk.

The manual is:

Barker D. 1997. LVB 1.0: reconstructing evolution with parsimony and simulated annealing (Edinburgh: Daniel Barker). ISBN 0-9531042-0-6

The manual is available free from the Internet page.

USING POWERPC MACOS LVB 1.0A

Brief instructions on how to use the PowerPC MacOS version of LVB 1.0A are given below. FOLLOW THESE INSTRUCTIONS CAREFULLY, at least the first time you run LVB. LVB does not work like any other MacOS application.

Put the program file, lvb, in an empty folder.

Add a plain text file containing the data matrix to the folder. This file must be called data. A simple way to create this file is to export tab- or space-delimited text from a spreadsheet. Files can usually be saved as text by choosing 'Save As...' from the 'File' menu of the spreadsheet, then adjusting the type of the file. Each row should contain an object name in the first column, then the character states for that object in one or more subsequent columns. There must be no space within the object name. Instead, use an underscore, as in Homo_sapiens. (For a full description of the format of the file, see Section 2.1 of the LVB manual.)

Add a text file called params, containing instructions to configure the behaviour of LVB. Files can usually be saved as text by choosing 'Save As...' from the 'File' menu of a word processor, then adjusting the type of the file. The contents of params should be as described in Chapter 3 of the LVB manual.

Quit all applications (other than Finder).

Double-click the lvb icon.

LVB then starts. It is not obvious that this happens. The clues are that the Finder window becomes un-highlighted, and the computer becomes unresponsive. But it has not crashed. Eventually LVB will terminate. Output files appear in its folder, and the machine returns to normal. Move or rename the output files immediately. Otherwise, they will be overwritten and lost if LVB is run again.

Since running LVB dominates the computer, preventing all other use, communicating with LVB through the filestore is not possible in this version. So pausing LVB and stopping LVB early without loss of results (Sections 5.2 and 5.3 of the manual) are not possible, and you cannot follow progress by looking at output files whilst LVB runs. But you can force LVB to quit, by pressing ESC whilst holding down the COMMAND and OPTION keys. The COMMAND key is found immediately to the left of the

space bar. The OPTION key is called ALT on some keyboards. All trees found will be lost if you force LVB to quit.

To begin with, try configuring LVB to do a very superficial analysis. Try a params file consisting of the following:

```
method anneal
t0 0.9
t1 0.3
maxaccept 10
maxpropose 100
maxfail 1
equivalent none
matchchar none
ignorechars none
outgroup none
runs 1
titleprec none
verbose 1
% CHANGE THE VALUE OF SEED BEFORE EVERY RUN
seed 1789
```

This specifies simulated annealing with a single, extremely brief cooling cycle. It will not give good results, but it will finish quickly. In this version of LVB, you MUST specify the value of seed, which should be different every time you run LVB (Section 3.6 of the manual).

If you do not have a data matrix ready, you could try a data file consisting of the following:

```
1       AAACT11101
2       GTTGG11010
3       AATCG00111
4       GATCT01000
```

The initial tree will be saved in a file called ini0. The results tree or trees will be saved in a file called res0. Progress is logged to a file called log. A brief summary of results is saved to a file called sum. All of these are plain text files. To view them, start a word processor, choose 'Open...' from the 'File' menu, and find the file name.

In LVB tree files, each tree ends with a semicolon. The bracketed text

trees of LVB are hard to interpret, unless very small. Since they conform to the Newick standard, you can make files for printing or display using other programs, for example the PHYLIP program drawtree. This formats unrooted trees. LVB can also output rooted trees (Section 3.4 of the LVB manual), which can be made into files for printing or display using the PHYLIP program drawgram. To produce a consensus if there is more than one results tree, you can use the PHYLIP program consense (but see the warning in Section 4.1 of the LVB manual). See Appendix B of the LVB manual to ensure drawgram, drawtree and consense from PHYLIP 3.572c work smoothly with LVB.

LARGE DATA MATRICES

Those analyzing extremely large matrices should contact the author. Sometimes the memory available to LVB should be increased.

HOW LVB WAS COMPILED FOR POWERPC MACOS

This version of LVB 1.0A was compiled under MacOS System 7.5.5 using Codewarrior Professional Release 1 (Metrowerks Corporation, 1997) as an ANSI C Console PPC target with default settings, except that the preferred heap size was set to 16384 KB, the minimum heap size was set to 10240 KB, the stack size was set to 1024 KB, Auto-Inline and Enums Always Int were set, all C/C++ Warnings were requested, Use FNADD & FMSUBB and Emit Traceback Tables were switched off, Global Optimization at Level 2 (common subexpression elimination, copy propagation, dead code elimination and global register allocation) and Peephole Optimization were switched on, Generate SYM File was switched off, and the file ansi_prefix.mac.h was modified so as not to include MacHeaders (#define MSL_USE_PRECOMPILED_HEADERS 1 was replaced with #define MSL_USE_PRECOMPILED_HEADERS 0).

RELATED SOFTWARE

The PHYLIP package, useful to present trees found by LVB, is available free from:

http://evolution.genetics.washington.edu/phylip.html

ACKNOWLEDGEMENTS

The self-extracting archive was created using Compact Pro 1.52 by Bill Goodman. I thank Metrowerks, Apple and readers of the comp.std.c and comp.lang.c discussion groups for their prompt advice on compilation.

AUTHOR

Daniel Barker
Institute of Cell and Molecular Biology
University of Edinburgh
Daniel Rutherford Building
King's Buildings
Mayfield Road
Edinburgh
EH9 3JH
United Kingdom

e-mail: sokal@holyrood.ed.ac.uk

F.2 Windows 95, Windows 98 and Windows NT

LVB 1.0A 18 AUGUST 1997 WIN32 (WINDOWS 95, WINDOWS 98 AND WINDOWS NT) VERSION RELEASE NOTES

DISCLAIMER

Great care has been taken to ensure LVB is not infected with viruses. However, those worried about viruses or the details of compilation should compile the source code themselves using an ANSI (ISO) C (C89) compiler.

SYSTEM REQUIREMENTS

Minimum system: Intel Pentium-based computer or compatible, running Microsoft Windows 95, Microsoft Windows 98 or Microsoft Windows NT 4, with 10 MB free RAM.

Recommended system: Intel Pentium-based computer or compatible, running Microsoft Windows NT 4, with 16 MB free RAM.

Scope: This release is suitable for use with all Pentium and Pentium-compatible processors. It is not for use with earlier processors, such as the Intel 80486. The operating system must be no earlier than Windows 95 or Windows NT 4.

SUPPORT, COPIES OF PROGRAM AND MANUAL, REGISTRATION

Support, the program, the manual and user registration are available free from:

http://www.icmb.ed.ac.uk/sokal.html

The manual is:

Barker, D. 1997. LVB 1.0: reconstructing evolution with parsimony and simulated annealing (Edinburgh: Daniel

Barker). ISBN 0-9531042-0-6

The manual is available free from the Internet page.

USING WIN32 LVB 1.0A

Brief instructions on how to use the Win32 version of LVB 1.0A are given below. FOLLOW THESE INSTRUCTIONS CAREFULLY, at least the first time you run LVB.

Put the program file, LVB.EXE, in an empty folder (directory).

Put a plain text file containing the data matrix in this folder. This file must be called DATA. A simple way to create this file is to export tab- or space-delimited text from a spreadsheet. Spreadsheet files can usually be saved as text by choosing 'Save As...' from the 'File' menu of the spreadsheet, then adjusting the type of the file. Each row should contain an object name in the first column, then the character states for that object in one or more subsequent columns. There must be no space within the object name. Instead, use an underscore, as in Homo_sapiens. (For a full description of the format of the file, see Section 2.1 of the LVB manual.)

Add a text file called PARAMS, containing instructions to configure the behaviour of LVB. This could be written using a word processor. Word processor files can usually be saved as text by choosing 'Save As...' from the 'File' menu, then adjusting the type of the file. The contents of PARAMS should be as described in Chapter 3 of the LVB manual.

Double-click the LVB.EXE icon. (LVB may also be launched from the MS-DOS prompt under Windows 95, Windows 98 or Windows NT. The method is not given here.)

When LVB starts, a plain window appears. When LVB terminates, the message 'Press Enter to continue' appears in the window. The window may then be closed by bringing it to the foreground and pressing ENTER.

Output files appear in the folder where LVB is run. When LVB has finished, move or rename the output files immediately. Otherwise, they will be overwritten and lost if LVB is run again in the same folder.

With this version of LVB, you cannot follow progress by looking at output files whilst LVB runs.

You can force LVB to quit by bringing its window to the foreground and pressing C while holding down the CTRL key. All trees found will be lost if you force LVB to quit.

To begin with, try configuring LVB to do a very superficial analysis. Try a PARAMS file consisting of the following:

```
method anneal
t0 0.9
t1 0.3
maxaccept 10
maxpropose 100
maxfail 1
equivalent none
matchchar none
ignorechars none
outgroup none
runs 1
titleprec none
verbose 1
% CHANGE THE VALUE OF SEED BEFORE EVERY RUN
seed 1789
```

This specifies simulated annealing with a single, extremely brief cooling cycle. It will not give good results, but it will finish quickly. In this version of LVB, you MUST specify the value of seed, which should be different every time you run LVB (Section 3.6 of the manual).

If you do not have a data matrix ready, you could try a data file consisting of the following:

```
1       AAACT11101
```

```
2       GTTGG11010
3       AATCG00111
4       GATCT01000
```

The initial tree will be saved in a file called INI0. The results tree or trees will be saved in a file called RES0. Progress is logged to a file called LOG. A brief summary of results is saved to a file called SUM. All of these output files are plain text files. To view them, start a word processor, choose 'Open...' from the 'File' menu, and ensure files of all types are being shown.

In LVB tree files, each tree ends with a semicolon. The bracketed text trees of LVB are hard to interpret, unless very small. Since they conform to the Newick standard, you can make files for printing or display using other programs, for example TreeView or the PHYLIP programs DRAWTREE (for unrooted trees, which LVB produces by default) and DRAWGRAM (for rooted trees; see Section 3.4 of the LVB manual). Do not use TreeView with trees that have more than 500 objects (terminals). See Appendix B of the LVB manual to ensure DRAWGRAM and DRAWTREE from PHYLIP 3.572c work smoothly with LVB.

To produce a consensus if there is more than one results tree, you can use the PHYLIP program CONSENSE (but see the warning in Section 4.1 of the LVB manual). See Appendix B of the LVB manual to ensure CONSENSE from PHYLIP 3.572c works smoothly with LVB.

HOW LVB WAS COMPILED FOR WIN32

This version of LVB 1.0A was compiled under MacOS System B1-8.0 using CodeWarrior Professional Release 4 (Metrowerks Corporation, 1998), as a Win32_x86 C Console App with default settings except that the file name was set to 'LVB.EXE', The reserved heap size was set to 10240k, Auto-Inline, ANSI Strict and Enums Always Int were set, Global Optimization was set to Level 3 (global register allocation, dead code elimination, branch optimizations, arithmetic optimizations, expression simplification, common subexpression elimination, copy and expression propagation, peephole optimization, dead

store elimination, live range splitting, loop-invariant code motion, strength reduction, loop transformations, loop unrolling, vectorization, lifetime based register allocation and instruction scheduling), and User ID was set to 1.00.

RELATED SOFTWARE

Two packages useful to present trees found by LVB are available free on the Internet.

The PHYLIP package is available from:

http://evolution.genetics.washington.edu/phylip.html

The TreeView package is available from:

http://taxonomy.zoology.gla.ac.uk/rod/treeview.html

ACKNOWLEDGEMENT

I would like to thank Jamie Norden for helping to test LVB 1.0A for Win32.

AUTHOR

Daniel Barker
Biocomputing Research Unit
Institute of Cell and Molecular Biology
University of Edinburgh
King's Buildings
Mayfield Road
Edinburgh
EH9 3JR
United Kingdom

http://www.icmb.ed.ac.uk/sokal.html

F.3 Windows 3.1 and Windows 3.11

LVB 1.0A 18 AUGUST 1997 WINDOWS 3.1 VERSION RELEASE NOTES

DOCUMENT HISTORY

Revision date: 24 June 1999

Supersedes: previous document entitled 'LVB 1.0A 18 August 1997 Windows 3.1 version release notes'.

Significant changes in the current version of this document: Information has been added to Section 'How LVB was compiled for Windows 3.1'. 'Scope' in Section 'System requirements' is now more detailed, and 'DMPI' is now spelled correctly. Section 'Document history' has been added.

Note: No changes have been made to the source code of the program LVB, as generally distributed, since LVB 1.0A was released on 18 August 1997.

DISCLAIMER

Great care has been taken to ensure LVB is not infected with viruses. However, those worried about viruses or the details of compilation should compile the source code themselves using an ANSI (ISO) C (C89) compiler.

SYSTEM REQUIREMENTS

Minimum system: Intel Pentium-based computer or compatible, running Microsoft Windows 3.1 or Microsoft Windows 3.11, with 10 MB free RAM.

Recommended system: Intel Pentium-based computer or compatible, running Microsoft Windows 3.1, with 16 MB free RAM.

Scope: This release is suitable for use with all Pentium and Pentium-compatible processors. (It might work on the

Intel 80486DX, too, but this has not been tested.) It is not for use with the Intel 80486SX, Intel 80386 or earlier processors. This is a program for protected-mode DOS. It will not work under DOS in the absence of Windows 3.1, Windows 3.11 or some other DPMI server. Under Windows 95, Windows 98 or Windows NT, use the Win32 version of LVB 1.0A instead.

SUPPORT, COPIES OF PROGRAM AND MANUAL, REGISTRATION

Support, the program, the manual and user registration are available free from:

http://www.icmb.ed.ac.uk/sokal.html

The manual is:

Barker, D. 1997. LVB 1.0: reconstructing evolution with parsimony and simulated annealing (Edinburgh: Daniel Barker). ISBN 0-9531042-0-6

The manual is available free from the Internet page.

USING WINDOWS 3.1 LVB 1.0A

Brief instructions on how to use the Windows 3.1 version of LVB 1.0A are given below. FOLLOW THESE INSTRUCTIONS CAREFULLY, at least the first time you run LVB.

Put the PIF file, LVB.PIF, in the directory C:\WINDOWS.

Put the program file, LVB.EXE, in an empty directory.

Put a plain text file containing the data matrix in this directory. This file must be called DATA. A simple way to create this file is to export tab- or space-delimited text from a spreadsheet. Spreadsheet files can usually be saved as text by choosing 'Save As...' from the 'File' menu of the spreadsheet, then adjusting the type of the file. Each row should contain an object name in the first column, then the character states for that object in one or more subsequent columns. There must be no space within the object

name. Instead, use an underscore, as in Homo_sapiens. (For a full description of the format of the file, see Section 2.1 of the LVB manual.)

Add a text file called PARAMS, containing instructions to configure the behaviour of LVB. This could be written using a word processor. Word processor files can usually be saved as text by choosing 'Save As...' from the 'File' menu, then adjusting the type of the file. The contents of PARAMS should be as described in Chapter 3 of the LVB manual.

In File Manager, double-click the LVB.EXE icon. (LVB may also be launched from the MS-DOS prompt under Windows 3.1 or Windows 3.11. Do not launch it from a DOS prompt when Windows is NOT running, however.) This will launch LVB in a full DOS screen. To switch to another application whilst LVB runs, hold down ALT and tap TAB a few times to show the running tasks. Release ALT when the task you wish to bring to the foreground is listed. When you have finished with other application, use a similar method to bring the LVB screen to the foreground again.

Output files appear in the directory where LVB is run. When LVB has finished, move or rename the output files immediately. Otherwise, they will be overwritten and lost if LVB is run again in the same directory.

With this version of LVB, you cannot follow progress by looking at output files whilst LVB runs.

You can force LVB to quit by bringing its screen to the foreground and pressing C while holding down the CTRL key. All trees found will be lost if you force LVB to quit.

To begin with, try configuring LVB to do a very superficial analysis. Try a PARAMS file consisting of the following:

```
method anneal
t0 0.9
t1 0.3
maxaccept 10
maxpropose 100
maxfail 1
equivalent none
```

```
matchchar none
ignorechars none
outgroup none
runs 1
titleprec none
verbose 1
% CHANGE THE VALUE OF SEED BEFORE EVERY RUN
seed 1789
```

This specifies simulated annealing with a single, extremely brief cooling cycle. It will not give good results, but it will finish quickly. In this version of LVB, you MUST specify the value of seed, which should be different every time you run LVB (Section 3.6 of the manual).

If you do not have a data matrix ready, you could try a data file consisting of the following:

```
1       AAACT11101
2       GTTGG11010
3       AATCG00111
4       GATCT01000
```

The initial tree will be saved in a file called INI0. The results tree or trees will be saved in a file called RES0. Progress is logged to a file called LOG. A brief summary of results is saved to a file called SUM. All of these output files are plain text files. To view them, start a word processor, choose 'Open...' from the 'File' menu, and ensure files of all types are being shown.

In LVB tree files, each tree ends with a semicolon. The bracketed text trees of LVB are hard to interpret, unless very small. Since they conform to the Newick standard, you can make files for printing or display using other programs, for example TreeView or the PHYLIP programs DRAWTREE (for unrooted trees, which LVB produces by default) and DRAWGRAM (for rooted trees; see Section 3.4 of the LVB manual). Do not use TreeView with trees that have more than 500 objects (terminals). See Appendix B of the LVB manual to ensure DRAWGRAM and DRAWTREE from PHYLIP 3.572c work smoothly with LVB.

To produce a consensus if there is more than one results

tree, you can use the PHYLIP program CONSENSE (but see the warning in Section 4.1 of the LVB manual). See Appendix B of the LVB manual to ensure CONSENSE from PHYLIP 3.572c works smoothly with LVB.

HOW LVB WAS COMPILED FOR WINDOWS 3.1

This version of LVB 1.0A was compiled under Windows 98 using the DJGPP Development Kit and Runtime (DJDEV) version 2.02 and the DJGPP distribution of GNU C (GCC) version 2.8.1, with compiler options -ansi -O3 -fno-short-enums.

DJGPP is available free through the 'zip picker' at:

http://www.delorie.com/djgpp/

Due to use of the DJGPP standard library, this product (LVB 1.0A for Windows 3.1) includes software developed by the University of California, Berkeley and its contributors.

RELATED SOFTWARE

Two packages useful to present trees found by LVB are available free on the Internet.

The PHYLIP package is available from:

http://evolution.genetics.washington.edu/phylip.html

The TreeView package is available from:

http://taxonomy.zoology.gla.ac.uk/rod/treeview.html

AUTHOR

Daniel Barker
Biocomputing Research Unit
Institute of Cell and Molecular Biology
University of Edinburgh
King's Buildings
Mayfield Road

Edinburgh

EH9 3JR

United Kingdom

http://www.icmb.ed.ac.uk/sokal.html

F.4 OS/2 3.0 and above

LVB 1.0A 18 August 1997 OS/2 (3.0 and above) VERSION
RELEASE NOTES

DISCLAIMER

Great care has been taken to ensure LVB is not infected with viruses. However, those worried about viruses or the details of compilation should compile the source code themselves using an ANSI (ISO) C (C89) compiler.

SYSTEM REQUIREMENTS

Minimum system: Intel Pentium-based computer or compatible, running IBM OS/2 3.0 or above (version 2.x might work, too, but has not been tested), with 10 MB free RAM. More RAM is (always) recommended.

Scope: This release is suitable for use with all Pentium and Pentium-compatible processors. It is not for use with earlier processors, such as the Intel 80486.

SUPPORT, COPIES OF PROGRAM AND MANUAL, REGISTRATION

Support, the program, the manual and user registration are available free from:

http://www.icmb.ed.ac.uk/sokal.html

The manual is:

Barker, D. 1997. LVB 1.0: reconstructing evolution with parsimony and simulated annealing (Edinburgh: Daniel Barker). ISBN 0-9531042-0-6

The manual is available free from the Internet page.

USING OS/2 LVB 1.0A

Brief instructions on how to use the OS/2 version of LVB 1.0A are given below. FOLLOW THESE INSTRUCTIONS CAREFULLY, at least the first time you run LVB.

Put the program file, LVB.EXE, in an empty directory.

Put a plain text file containing the data matrix in this folder. This file must be called DATA. A simple way to create this file is to export tab- or space-delimited text from a spreadsheet. Spreadsheet files can usually be saved as text by choosing 'Save As...' from the 'File' menu of the spreadsheet, then adjusting the type of the file. Each row should contain an object name in the first column, then the character states for that object in one or more subsequent columns. There must be no space within the object name. Instead, use an underscore, as in Homo_sapiens. (For a full description of the format of the file, see Section 2.1 of the LVB manual.)

Add a text file called PARAMS, containing instructions to configure the behaviour of LVB. This could be written using a word processor. Word processor files can usually be saved as text by choosing 'Save As...' from the 'File' menu, then adjusting the type of the file. The contents of PARAMS should be as described in Chapter 3 of the LVB manual.

When using PARAMS or DATA files from other people make sure they are DOS / Windows / OS/2 style text files. LVB currently cannot handle e.g. Unix style text files, it requires lines ending with CR/LF's. If you're not sure what kind of file you're using you can 'blindly' convert them using the OS/2 text editors E.EXE or EPM.EXE by loading the files and then saving them once.

Run LVB.EXE without parameters from the OS/2 command prompt (CMD.EXE, 4OS2.EXE or whatever you prefer). You may also open the hard disk drive object from the WPS, open the folder LVB.EXE and the other files reside in and then double click LVB.EXE.

During execution LVB will create several files in its directory. When LVB has finished, move or rename these files

immediately. Otherwise, they will be overwritten and lost if LVB is run again in the same folder.

With this version of LVB, you cannot follow progress by looking at output files whilst LVB runs.

You can force LVB to quit by bringing its window to the foreground and pressing C while holding down the CTRL key. All trees found will be lost if you force LVB to quit.

To begin with, try configuring LVB to do a very superficial analysis. Try a PARAMS file consisting of the following:

```
method anneal
t0 0.9
t1 0.3
maxaccept 10
maxpropose 100
maxfail 1
equivalent none
matchchar none
ignorechars none
outgroup none
runs 1
titleprec none
verbose 1
% CHANGE THE VALUE OF SEED BEFORE EVERY RUN
seed 1789
```

This specifies simulated annealing with a single, extremely brief cooling cycle. It will not give good results, but it will finish quickly. In this version of LVB, you MUST specify the value of seed, which should be different every time you run LVB (Section 3.6 of the manual).

If you do not have a data matrix ready, you could try a data file consisting of the following:

```
1       AAACT11101
2       GTTGG11010
3       AATCG00111
4       GATCT01000
```

The initial tree will be saved in a file called INI0. The

results tree or trees will be saved in a file called RES0. Progress is logged to a file called LOG. A brief summary of results is saved to a file called SUM. All of these output files are plain text files. To view them, start a text editor (EPM.EXE suits this purpose well), choose 'Open...' from the 'File' menu, and ensure files of all types are being shown. You can of course alternatively just double click the file object you're interested in in the open WPS folder.

In LVB tree files, each tree ends with a semicolon. The bracketed text trees of LVB are hard to interpret, unless very small. Since they conform to the Newick standard, you can make files for printing or display using other programs, for example TreeView or the PHYLIP programs DRAWTREE (for unrooted trees, which LVB produces by default) and DRAWGRAM (for rooted trees; see Section 3.4 of the LVB manual). Do not use TreeView with trees that have more than 500 objects (terminals). See Appendix B of the LVB manual to ensure DRAWGRAM and DRAWTREE from PHYLIP 3.572c work smoothly with LVB.

To produce a consensus if there is more than one results tree, you can use the PHYLIP program CONSENSE (but see the warning in Section 4.1 of the LVB manual). See Appendix B of the LVB manual to ensure CONSENSE from PHYLIP 3.572c works smoothly with LVB.

HOW LVB WAS COMPILED FOR OS/2

This version of LVB 1.0A was compiled under OS/2 4.0 using IBM's VisualAge C++ Version 3.0 as a windowable OS/2 console app with default settings, optimization enabled and debugging disabled.

RELATED SOFTWARE

Two packages useful to present trees found by LVB are available free on the Internet.

The PHYLIP package is available from:

http://evolution.genetics.washington.edu/phylip.html

The TreeView package is available from:

http://taxonomy.zoology.gla.ac.uk/rod/treeview.html

ACKNOWLEDGEMENT

The OS/2 version was compiled by Martin "Herbert" Dietze <herbert@paulina.shnet.org>, who also wrote the OS/2-specific parts of these release notes and helped test LVB under OS/2. Without his work, the OS/2 distribution of LVB 1.0A would not exist.

The ZIP archive was created using INFO-ZIP 2.1.

AUTHOR

Daniel Barker
Biocomputing Research Unit
Institute of Cell and Molecular Biology
University of Edinburgh
King's Buildings
Mayfield Road
Edinburgh
EH9 3JR
United Kingdom

http://www.icmb.ed.ac.uk/sokal.html

Appendix G

Web Pages

This appendix contains the URLs for Web pages mentioned in the text of the thesis. Information and addresses are correct at the time of writing.

G.1 LVB

`http://www.icmb.ed.ac.uk/sokal.html`

The manual (Appendix D) may be downloaded. The program may be downloaded as source code, with a `makefile` and `man` page for Unix, and as executables for PowerPC MacOS, Pentium (or compatible) Win32 (Windows 95, Windows 98 or Windows NT), Pentium (or compatible) Windows 3.1 or Windows 3.11 and Pentium (or compatible) OS/2 3.0 (or later). Related software, for example, experimental new versions of LVB (Section 6.4), are also made available for download.

There are some answers to frequently asked questions, and links to a few sites of related interest.

The site has received over 3 400 visits.

G.2 PHYLIP

`http://evolution.genetics.washington.edu/phylip.html`

This is a very complete site devoted to the PHYLIP package of phylogeny programs, with a vast number of links to related programs and sites.

PHYLIP's programs `drawgram`, `drawtree` and `consense` are useful for post-processing LVB tree files.

G.3 Molphy

`ftp://sunmh.ism.ac.jp/pub/molphy/`

The Molphy package may be downloaded here. `njdist` and `protml` were used in Nakayama's analysis with LVB (Section 6.4).

G.4 Biological Software at the Institut Pasteur

`http://bioweb.pasteur.fr/intro-uk.html`

This is a free Web-based service for running biological computer programs, including LVB (`http://bioweb.pasteur.fr/seqanal/interfaces/lvb.html`). Interfaces were generated using Pise (`http://www.pasteur.fr/~letondal/Pise/`).

G.5 Incomplete Nucleic Acid Sequences

`http://www.chem.qmw.ac.uk/iupac/misc/naseq.html`

Recommended symbols for representing ambiguous nucleotides are given here.

G.6 Edinburgh Parallel Computing Centre

`http://www.epcc.ed.ac.uk/`

The Web pages of Edinburgh's centre for high performance computing.

G.7 Free Software Foundation

`http://www.gnu.org/`

Many useful computer programs are distributed free under the GNU project of the Free Software Foundation.

G.8 Top500 Supercomputer at Mannheim University and Netlib

`http://www.netlib.org/benchmark/top500.html`

A list of the 500 most powerful computers in the world, updated every six months.

G.9 SETI@home

`http://www.setiathome.ssl.berkeley.edu/`

The Search for Extraterrestrial Intelligence at Home, a computational experiment distributed over the Internet. Volunteers perform part of a large analysis on their own computer.

Bibliography

[1] K. R. Sporne. A new approach to the problem of the primitive flower. *New Phytologist*, 48(2):259–276, 1949.

[2] K. R. Sporne. The ovule as an indicator of evolutionary status in angiosperms. *New Phytologist*, 68:555–566, 1969.

[3] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1859. Chapter 4.

[4] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1988. Section 6.5.

[5] A. V. Aho and J. D. Ullman. *Principles of Compiler Design*. Addison-Wesley, Reading, Massachusetts, 1977. Chapter 1.

[6] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38(2):1409–1438, 1958.

[7] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. W. H. Freeman, San Francisco, 1973.

[8] W. M. Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Zoology*, 20(4):406–416, 1971.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecci. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[10] M. Lundy. Applications of the annealing algorithm to combinatorial problems in statistics. *Biometrika*, 72(1):191–198, 1985.

[11] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Massachusetts, corrected reprint of the 1st edition, 1987.

[12] L. L. Cavalli-Sforza and A. W. F. Edwards. Estimation procedures for evolutionary branching processes. *Bulletin of the International Statistical Institute*, 41(2):803–808, 1965.

[13] K. R. Sporne. *The Morphology of Angiosperms*. Hutchinson, London, 1974. Chapter 8.

[14] E. Mayr. *The Growth of Biological Thought: Diversity, Evolution, and Inheritance*. Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1982.

[15] R. D. M. Page and E. C. Holmes. *Molecular Evolution: A Phylogenetic Approach*. Blackwell Science, Oxford, 1998.

[16] D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis. Phylogenetic inference. In D. M. Hillis, C. Moritz, and B. K. Mable, editors, *Molecular Systematics*, pages 407–514. Sinauer Associates, Sunderland, Massachusetts, 2nd edition, 1996.

[17] I. J. Kitching, P. L. Forey, C. J. Humphries, and D. M. Williams. *Cladistics: The Theory and Practice of Parsimony Analysis*. Oxford University Press, Oxford, 2nd edition, 1998. Chapter 3.

[18] M. Ridley. *Evolution and Classification: The Reformation of Cladism*. Longman, London and New York, 1986.

[19] S. J. Gould. *Ontogeny and Phylogeny.* Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1977.

[20] G. Nelson and N. Platnick. *Systematics and Biogeography: Cladistics and Vicariance.* Columbia University Press, New York, 1981.

[21] J. Felsenstein. The number of evolutionary trees. *Systematic Zoology*, 27(1):27–33, 1978.

[22] T. G. Lewis, H. El-Rewini, and I.-K. Kim. *Introduction to Parallel Computing.* Prentice-Hall International, 1992. Chapter 2.

[23] D. H. Lehmer. Combinatorial problems with digital computers. In *Proceedings of the Fourth Canadian Mathematical Congress: Banff, 1957*, pages 160–173. University of Toronto Press, Toronto, 1959.

[24] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Reading, Massachusetts, corrected reprint of the 1st edition, 1987. Section 10.3.

[25] M. D. Hendy and D. Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, 59:277–290, 1982.

[26] E. L. Lawler and D. E. Wood. Branch-and-bound methods: a survey. *Operations Research*, 14(4):699–719, 1966.

[27] D. E. Knuth. *Estimating the efficiency of backtrack programs.* Department of Computer Science, Stanford University, 1974. STAN-CS-74-442.

[28] J. Gruska. *Quantum Computing.* McGraw-Hill, London, 1999.

[29] D. E. Knuth. *Mathematics and Computer Science: Coping with Finiteness.* Department of Computer Science, Stanford University, Stanford, 1976. STAN-CS-76-541.

[30] D. L. Swofford. *PAUP: Phylogenetic Analysis Using Parsimony, Version 3.1.* Illinois Natural History Survey, Champaign, Illinois, 1993.

[31] D. Barker. *LVB 1.0: Reconstructing Evolution with Parsimony and Simulated Annealing.* Daniel Barker, Edinburgh, 1997.

[32] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6:791–812, 1958.

[33] R. J. M. Vaessens. A local search template (extended abstract). In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature, 2: Proceedings of the Second Conference on Parallel Problem Solving from Nature, Brussels, Belgium, 28–30 September, 1992*, pages 65–74. North-Holland, Amsterdam, 1992.

[34] L. P. Lefkovitch. *Optimal Set Covering for Biological Classification.* Research Branch, Agriculture Canada, Ottawa, 1993.

[35] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[36] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

[37] E. H. L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley, Chichester, 1989.

[38] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* The University of Michigan Press, Ann Arbor, 1975.

[39] L. Davis, editor. *Genetic Algorithms and Simulated Annealing*. Pitman, London, 1987.

[40] D. E. Goldberg. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4(4):445–460, 1990.

[41] Quadstone Ltd, Edinburgh. *The Reproductive Plan Language RPL2: Documentation for Version 1.1A, Issue 1*, 1995.

[42] P. O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–283, 1998.

[43] K. A. de Jong. Are genetic algorithms function optimizers? In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature, 2: Proceedings of the Second Conference on Parallel Problem Solving from Nature, Brussels, Belgium, 28–30 September, 1992*, pages 3–13. North-Holland, Amsterdam, 1992.

[44] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1988.

[45] A. Kelly and I. Pohl. *A Book on C: Programming in C*. Benjamin Cummings, Redwood City, California, 3rd edition, 1995.

[46] D. R. Maddison, D. L. Swofford, and W. P. Maddison. NEXUS: an extensible file format for systematic information. *Systematic Biology*, 46(4):590–621, 1997.

[47] D. Gilly. *UNIX in a Nutshell: a Desktop Quick Reference for System V Release 4 and Solaris 2.0*. O'Reilly & Associates, Cambridge, corrected reprint of the 2nd edition, 1998.

[48] J. P. Heckman. *Linux in a Nutshell.* O'Reilly & Associates, Sebastopol, California, 1997.

[49] A. G. Kluge and J. S. Farris. Taxic homology = overall similarity. *Cladistics*, 15:205–212, 1999.

[50] W. P. Maddison and D. R. Maddison. *MacClade: Analysis of Phylogeny and Character Evolution Version 3.* Sinauer Associates, Sunderland, Massachusetts, 1992. Chapter 5.

[51] G. Marsaglia, A. Zaman, and W. W. Tsang. Toward a universal random number generator. *Statistics and Probability Letters*, 8:35–39, 1990.

[52] F. James. A review of pseudorandom number generators. *Computer Physics Communications*, 60(3):329–344, 1990.

[53] M. W. Chase, D. E. Soltis, R. G. Olmstead, D. Morganand D. H. Les, B. D. Mishler, M. R. Duvall, R. A. Price, H. G. Hills, Y.-L. Qiu, K. A. Kron, J. H. Rettig, E. Conti, J. D. Palmer, J. R. Manhart, K. J. Sytsma, H. J. Michaels, W. J. Kress, K. G. Karol, W. D. Clark, M. Hedrén, B. S. Gaut, R. K. Jansen, K.-J. Kim, C. F. Wimpee, J. F. Smith, G. R. Furnier, S. H. Strauss, Q.-Y. Xiang, G. M. Plunkett, P. S. Soltis, S. M. Swensen, S. E. Williams, P. A. Gadek, C. J. Quinn, L. E. Eguiarte, E. Golenberg, G. H. Learn, S. W. Graham, S. C. H. Barrett, S. Dayanandan, and V. A. Albert. Phylogenetics of seed plants: an analysis of nucleotide sequences from the plastid gene *rbc*L. *Annals of the Missouri Botanical Garden*, 80(3):528–580, 1993.

[54] A. Cronquist. *An integrated system of classification of flowering plants.* Columbia University Press, New York, 1981.

[55] D. J. Mabberley. *The Plant-Book: A Portable Dictionary of the Higher Plants.* Cambridge University Press, Cambridge, corrected reprint of the 1st edition, 1990.

[56] R. R. Sokal and F. J. Rohlf. *Biometry: the Principles and Practice of Statistics in Biological Research.* W. H. Freeman, New York, 3rd edition, 1995.

[57] K. A. Rice, M. J. Donoghue, and R. G. Olmstead. Analyzing large data sets: *rbc*L 500 revisited. *Systematic Biology*, 46(3):554–563, 1997.

[58] D. E. Soltis, P. S. Soltis, M. E. Mort, M. W. Chase, V. Savolainen, S. B. Hoot, and C. M. Morton. Inferring complex phylogenies using parsimony: an empirical approach using three large DNA data sets for angiosperms. *Systematic Biology*, 47(1):32–42, 1998.

[59] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin/Cummings, Redwood City, California, 1994.

[60] K. Dowd and C. R. Severance. *High Performance Computing.* O'Reilly & Associates, Sebastopol, California, 2nd edition, 1998. Chapter 3.

[61] High Performance Fortran Forum. High Performance Fortran language specification (Part I). *Fortran Forum*, 12(4):1–86, 1993.

[62] High Performance Fortran Forum. High Performance Fortran language specification (Part II). *Fortran Forum*, 13(2):87–150, 1994.

[63] High Performance Fortran Forum. High Performance Fortran language specification (Part III). *Fortran Forum*, 13(3):22–55, 1994.

[64] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard.* University of Tennessee, Knoxville, Tennessee, 1995.

[65] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis: models and estimation procedures. *Evolution*, 21(3):550–570, 1967.

[66] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[67] Z. Yang. PAML: a program package for phylogenetic analysis by maximum likelihood. *Computer Applications in the Biosciences*, 13(5):555–556, 1997.

[68] H. Kawasaki and R. H. Kretsinger. Calcium-binding proteins 1: EF-hands. *Protein Profile*, 2(4):305–490, 1995.

[69] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[70] B. W. Kernighan and R. Pike. *The UNIX Programming Environment.* Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[71] G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[72] D. Evans. *LCLint User's Guide, Version 2.4.* Software Devices and Systems Group, MIT Laboratory for Computer Science, Cambridge, Massachusetts, 1998.

[73] Ministry of Defence. *Defence Standard 00-55, Issue 2. Requirements for Safety Related Software in Defence Equipment.* Ministry of Defence, Glasgow, 1997.

[74] L. Hatton. *Safer C: Developing Software for High-integrity and Safety-critical Systems.* McGraw-Hill, London, 1995.

[75] M. Metcalf and J. Reid. *Fortran 90/95 Explained.* Oxford University Press, Oxford, 2nd edition, 1999.

[76] N. H. Cohen. *Ada as a Second Language.* McGraw-Hill, New York, 2nd edition, 1996.

[77] K. Dowd and C. R. Severance. *High Performance Computing.* O'Reilly & Associates, Sebastopol, California, 2nd edition, 1998. Chapter 13.