The Infinite Sites Model of Genome Evolution (The Supplement)

Jian Ma, Aakrosh Ratan, Brian J. Raney, Bernard B. Suh,
Webb Miller, and David Haussler

Contents

List of Figures

List of Tables

Here we provide details to accompany the main paper. For the purposes of illustrating the algorithm, we will use the same example derivation tree that is used in the main paper (Figure S1).

1 Derivation with missing data

During the process of deriving a descendant genome from an ancestral genome through a sequence of evolutionary operations, missing data can also be introduced. Thus to formally define a derivation tree, we must formally define the notion of derivation with missing data.

A typical edge along a species path in a derivation tree is illustrated in (Figure S2A). We say genome f *derives* genome g in evolutionary distance d via a duplication or rearrangement operation and some subsequent chromosome gains and losses if homologous sites at distance d between f and g can be found that explain g as a result of applying the operation to f followed by the subsequent chromosomal gains and losses (Figure S2B). If there is a genome g' such that f derives g', and g is identical to g' except for missing data, then we say that f *derives* g *with missing data* (Figure S2C). For any branch from parent genome f to child genome g along a species path in an evolutionary tree, we require that f derive g, possibly with missing data. For a speciation node from parent h to children f and g, we require that h derive f and h derive g, possibly with missing data, but with no operations other than gain and loss of chromosomes. This formally defines a derivation tree.

Transitively, we say that f_1 derives f_n in distance d if for $1 \leq i < n$ there are intermediate genomes f_i and distances d_i such that $d_1 + ... + d_{n-1} = d$, f_1 derives f_2 in distance d_1, f_2 derives f_3 in distance d_2, ..., and f_{n-1} derives f_n in distance d_{n-1}.

2 The algorithm

The first step is to convert the continuous problem into a discrete problem by partitioning the present day genomes into maximal segments that do not contain any sites homologous to breakpointse. We will call these maximal segments *atom instances*.

2.1 The dot plot and the decomposition into atom instances

We define the dot plot first for genomes without circular chromosomes. Let G be a set of such genomes and let D be a non-negative function on $G \times G$. Order and orient the chromosomes within each genome in G and then order the genomes in G. Concatenate all the intervals from all the chromosomes in all the genomes in order. This maps G onto an interval I on the real line of length equal to the total length of all chromosomes in all genomes of G. A *dot plot* is a representation of the evolutionary distance function $D(x, y)$ as a function from $I \times I$ to $[0, \infty]$. Via the mapping from G to I, sites x and y in G are now points in I. Each local alignment in the specification of D is represented as a diagonal line segment if it is the '+' orientation and an anti-diagonal line segment if it is in the '-' orientation. In the pictorial representation of a dot plot we use colors to represent the constant distance value $D(x, y)$ along each local alignment, leaving the points where $D(x, y) = \infty$ colorless. We give an example of a dot plot in Figure S3, based on the evolutionary history shown in Figure S1. This dot plot representation is similar to the representation used for genome comparisons in the applied comparative genomics literature (Schwartz *et al.*, 2000; Ovcharenko *et al.*, 2004).

If the dot plot is derived from the leaf genomes of an evolutionary tree, then the endpoints of the local alignments correspond to the breakpoints used by the operations in the tree. Thus, the atom instances in the chromosomes in G can be determined by sweeping a vertical line across the dot plot, terminating one atom instance and beginning the next every time the vertical line encounters the endpoint of a local alignment or a contig boundary (Figure S4).

To map a genome with circular chromosomes to I, we must first randomly choose a point at which to break open each circular chromosome. The mapping is then done as above. However, we do not want to break atom instances at the artificial points we used to open up the chromosomes. Thus, a local alignment that spans the site chosen to break a circular chromosome will actually appear in two parts, one at each end of the linearized chromosome. This is called a *split local alignment*. For the purpose of defining atom instances, a split local alignment is treated as an unbroken segment.

2.2 Atoms, distances, ends, and adjacencies

We may reject the problem instance (G, D) as not having been produced from an evolutionary tree if D is such that the local alignments lack any of the three properties that define an equivalence relation, namely, identity (x is aligned to itself), symmetry (if x is aligned to y then y is aligned to x) or transitivity (if x is aligned to y and y is aligned to z, then x is aligned to z). If these conditions are satisfied, the atom instances in the genome set G can be partitioned into equivalence classes that we call *atoms*, where two atom instances are from the same atom if there is a local alignment between them in the dot plot. We say that such atom instances are *homologous*. The evolutionary distance $D(x, y)$ between a pair of homologous atom instances x and y is the value of D for the local alignment between them. For nonhomologous atom instances $D(x, y) = \infty$.

We label the atoms $\{1, 2, ..., N\}$. If genome f has, say, three atom instances from atom 1, then these are denoted $f[1_1], f[1_2], f[1_3]$, or if the genome is clear from the context, simply $1_1, 1_2, 1_3$.

For an atom instance x, or indeed for any segment of DNA x, $-x$ denotes the reverse complement, e.g., in the finite sites case, if $x = \texttt{ATG}$ then $-x = \texttt{CAT}$. For a segment or circle of double-stranded DNA, the forward stand will be some DNA sequence x and the reverse strand $-x$, but by the symmetry of double-stranded DNA, the molecule is the same whether we denote it by x or by $-x$. Because of this property of double-stranded DNA, what uniquely defines a genome is not a particular order and orientation of the atom instances along its chromosomes, but the *adjacencies* between the ends of its atom instances, including the null atom ends at the ends of contigs, as discussed below. The special case of a circular atom with no ends will be ignored in this treatment. It arises only when a chromosome is never altered by a rearrangement, and can be handled appropriately as a special case.

For example, the two atom instances x and y have four ends. We denote these by marking them with an asterisk, i.e. $*x$, $x*$, $*y$, and $y*$. An adjacency of the end a with the end b is denoted (a, b), and indicates that these ends abut in the genome. For example, the set of adjacencies $\{(x*, *y), (y*, *x)\}$ defines the ring shown in the upper right in Figure 1(A) in the main text, with segment x followed by segment y, or equivalently, its reverse complement, $-y$ followed by $-x$. In a genome of circular chromosomes, every atom instance end must be paired in an adjacency with exactly one other atom instance end, and conversely, every such pairing defines a genome of circular chromosomes made from those atom instances. For two atom instances x and y, there are three such pairings, hence these two atom instances can be organized into three distinct genomes of circular chromosomes.

The atom instance end at the end of a contig will not have an adjacency with any other atom instance end. We say that the atom instance end at the end of a contig has a *null adjacency*, or lack of an adjacency.

Note that it is possible to put special "telomere atoms" at the telomere ends of chromosomes where you are sure that you have reached the telomere end. In this case, if one is doing a reconstruction of primate genome evolution, where there is a chromosome fusion from human-chimp ancestor to human that created human chr2, one can put a special atom, call it "2AT" at all the non-human primate telomere homologs of the region of human 2A at the point where it joins 2B, and similarly a special atom "2BT" at all the primate homologs of the region of human 2B where it joins 2A. These atoms would be missing in the representation of the human chromosome 2. Then, in the reconstruction the infinite sites algorithm would deduce that there was series of events in human evolution as follows: `2A 2AT, 2BT 2B => 2A 2B, 2BT 2AT (reciprocal translocation) => 2A 2B (loss of the tiny chromosome 2BT 2AT)`. This is essentially a Robertsonian translocation.

In general, the model assumes missing data at the ends of chromosomes, but by adding explicit "telomere atoms" you can override this. We believe that this approach addresses the the important distinction between cases 2 and 3 below.

Since a genome is completely determined by the atom instances it contains and their adjacencies, given an initial genome f and a set S of rearrangements of the atom instance adjacencies such that no adjacency is changed more than once (as is always the case in the infinite sites model), the resulting genome g is the same no matter what order is assumed for the rearrangements.

2.3 The discrete representation of the simplest history problem

The partition into atom instances converts the input to the simplest history problem into a discrete structure consisting of a set G of leaf genomes, each of which is defined by a finite set of atom instances and their adjacencies, and a partition of all the atom instances into families of homologous atom instances (atoms), with a pairwise distance relationship D between the atom instances in each family.

We operate henceforth with this representation of G and D. The notions of an evolutionary tree, parent, child, ancestor and descendant relationships between segments of the chromosomes in the tree, evolutionary operations and derivations between the genomes in that tree, and lack of breakpoint reuse naturally carry over from the original continuous representation of genome evolution to this discrete representation in which genomes are represented by atom instances. For the latter, in the analogy with ends of individual atom instances, we may formally define the two ends of an atom k, where $1 \leq k \leq N$, denoting these as $*k$ and $k*$. The atom end $*k$ denotes, collectively, one of the two aligned (i.e. homologous) ends of the set of all atom instances in the atom k, and $k*$ denotes the other end. If x is an instance of atom k then the end $*x$ always refers to the end of x that is homologous to the "universal" end $*k$ for the atom, and similarly for $x*$ and $k*$. If x is an instance of atom k and y is an instance of atom j, then we say the adjacency $(x*, *y)$ has *type* $(k*, *j)$. An evolutionary operation with a breakpoint between $x*$ and $*y$ is said to *break* the adjacency of type $(k*, *j)$. The property of no breakpoint reuse formally means that every type of adjacency between the ends of the N atoms is broken at most once in the evolutionary tree that generates the leaf genomes.

2.4 Atom trees

If the data (G, D) derive from an actual evolutionary tree, then two segments will be instances of the same atom only if they descend from a common ancestral segment, and the pairwise distances between all the instances of an atom will have the property of *additivity*. This means that there is a undirected tree with non-negative length edges and a mapping from the atom instances to the leaves of that tree such that the distance between any pair of instances is the sum of the edge lengths along the path in the tree connecting them. If such a tree exists then it is unique, and there is an efficient algorithm for either finding it or determining that the distances are not additive (Zarekskii, 1965; Waterman *et al.*, 1977). In fact, the usual neighbor joining algorithm used to determine a phylogenetic tree from pairwise distances between the species is suitable for this purpose (Saitou and Nei, 1987; Studier and Keppler, 1988). Since in the infinite sites model the distances are exact, the resulting tree and its distances are exact. We use this algorithm to determine an undirected *atom tree* for each atom (Figure S5A). If there is any atom that does not have an additive set of pairwise distances, we reject the data (G, D) as not having been produced from an evolutionary tree. We denote the undirected atom tree for atom k by T_k.

When the data (G, D) derive from an evolutionary tree, each atom will also have a directed tree, representing the ancestor and descendant relations for its instances (Figure S5B). The directed tree for atom k is denoted T_k.

2.5 Fitch orthologs and paralogs

Suppose the data (G, D) do derive from an actual evolutionary tree. Consider two homologous atom instances, x in genome f and y in genome g, where f and g may be the same genome. Let the genome h be the last common ancestor of the genomes f and g in the evolutionary tree. Let the atom instance z be the last common ancestor of the atom instances x and y in their directed atom tree (see Figure S6). Clearly either z is in the genome h or z is in some ancestor of h. Using the definition of Fitch (Fitch, 1970, 2000), in the former case we say that x and y are *orthologs* and in the latter case we say that x and y are *paralogs*. This parallels the usual definition of these concepts for genes.

The distinction between paralog and ortholog revolves around whether the two sequences originally diverged due to a duplication or due to a speciation. If the last common atom ancestor z is in the last common genome ancestor h then x and y must have diverged by a speciation of h rather than a duplication (Figure S6A). In this case we must have $D(x, y) = d(f, g) = d(h, f) + d(h, g)$, where d denotes the distance between genomes in the actual evolutionary tree from which these data derive. If z is in an ancestor h_1 of h, then in h the ancestor x' of x and the ancestor y' of y must be distinct, and z must have duplicated to produce the divergent atom instances x' and y' (Figure S6B). In this case $D(x, y) = D(x', x) + D(y', y) + D(z, x') + D(z, y') = d(f, g) + 2d(h, h_1) > d(f, g)$. Thus, atom instances x and y are orthologs *iff* $D(x, y) = d(f, g)$ and paralogs *iff* $D(x, y) > d(f, g)$.

In our example in Figure S1 (and also the atom tree in Figure S5), the atom instance $f[3_3]$ is orthologous to the atom instance $g[3_3]$, but paralogous to the atom instance $g[3_4]$. $f[3_3]$ is also paralogous to atom instances $f[3_1]$, $f[3_2]$, and $f[3_4]$. On the other hand, atom instance $e[3]$ is orthologous to all six of these atom instances from genomes f and g.

2.6 Determining the species tree

Since we are assuming that there has not been complete turnover of genome segments between any two leaf genomes, there must always be least one pair of orthologous atom instances in any pair of leaf genomes. For any leaf genomes f and g in G, we set $D(f, g) = \min_{x \in f, y \in g} D(x, y)$, where the minimum is taken over all pairs of atom instances x in f and y in g. By the above results, any pair of orthologous atom instances will have the minimal distance and this distance will be equal to $d(f, g)$, the evolutionary distance between the two leaf genomes in the underlying evolutionary tree. Thus, when there is not complete turnover, $D(f, g) = d(f, g)$. If the pairwise distances between genomes in G defined by $D(f, g)$ are not additive, we reject (G, D) as not being derived from an evolutionary tree without complete turnover. Otherwise we use the neighbor-joining algorithm to determine a unique undirected species tree for the genomes in G (Figure S7). If we have a designated outgroup genome for the set G, then we use it to determine which edge of this undirected tree contains the root for species tree, and from this derive a directed species tree with a precise placement of the root at a position along one of the branches of the undirected species tree. For simplicity, we will assume that such an outgroup genome is available.

2.7 The duplication tree

When the data (G, D) come from an actual evolutionary tree, the undirected atom trees have the additional property that they all derive from a single underlying directed evolutionary tree for the entire genomes. The next step in the simplest history algorithm is to attempt to merge the undirected atom trees into the rooted and directed species tree to form a single master directed tree, representing all speciation and duplication events that have left a record of their effects in the leaf genomes. This process is somewhat similar to the process of reconciliation of an undirected gene tree with an undirected species tree, extensively studied in the literature (Ma *et al.*, 2000; Howe *et al.*, 2002; Chen *et al.*, 2000). We start with the species tree and reconcile the atom trees into it one at a time (see details in Section 4). If any of the reconciliations fail, we reject the input (G, D).

Each reconciliation labels the bifurcating nodes of the atom tree being reconciled as either duplication nodes or speciation nodes, maps the speciation nodes to the corresponding speciation nodes in the species tree, and maps the duplication nodes to inferred duplication nodes along the branches of the species tree (Figure S8A and B). The final reconciled species tree, including these additional duplication nodes, is called the *duplication tree* (Figure S8B). Each node in the duplication tree is a genome. All of the nodes of the species tree are in the duplication tree, and there is an additional chain of nodes along each branch of the species tree that represent intermediate genomes at times when duplications occurred. If there are duplications that occurred before the root of the species tree, then the root of the duplication tree is an ancestor of the species tree root, and these "ancient" duplications are represented on an additional species path leading from the root of the overall duplication tree to the root of the species tree within it.

In the course of this joint reconciliation, atom instances are also added along the branches of each atom tree to represent intermediate forms that are inferred to have existed at duplication branches but do not appear in the original atom tree for the family (Figure S8 (C), e.g. atom instance 4_2 in h). The resulting atom trees are called *augmented atom trees* and denoted $\mathbf{T}_1, \mathbf{T}_2$, etc. The mappings from the nodes and edges of the augmented atom trees to the nodes and the edges of the duplication tree is denoted Φ (Figure S8C).

Note that the root of an augmented atom tree need not always map to the root of the duplication tree. If the atom is first introduced by an insertion event, or if for some other reason all instances of the atom are missing from some parts of the duplication tree, then the last common ancestor in the duplication tree of all the observed instances of the atom may be a node below the root. This occurs for the augmented atom tree T_7 in the derivation tree from Figure S1. This augmented atom tree consists of just the single leaf node $e[7]$.

2.8 The master breakpoint graph and the evolution of adjacencies

To determine the adjacencies between inferred ancestral atom instances, first recall that we define the adjacency $(x*, *y)$ to be of type $(k*, *j)$ if x is an instance of atom k and y is an instance of atom j, for $1 \leq k, j \leq N$. The *master breakpoint graph* for G is a graph with $2N$ nodes, one for each atom end, and an edge between nodes s and t if and only if there is an adjacency of type (s, t) in one of the genomes in G (Figure S9). This is similar to the breakpoint graph defined in the analysis of the evolution of one genome into another by rearrangements (Hannenhalli and Pevzner, 1995). By the assumption of no breakpoint reuse, every type of adjacency is broken at most once in the evolutionary tree. Thus, there can be at most two edges incident upon any node in the master breakpoint graph. This allows us to exploit properties in it similar to those present in the breakpoint graphs for two genomes, even though it represents the evolution of many genomes.

The operations that change adjacencies are the rearrangements and the reverse tandem duplication. The 2-breakpoint rearrangement involves four ends a, b, c, and d (Figure 1 in the main text). Before the operation, the adjacencies are (a, b) and (c, d), and after the operation the adjacencies are either (a, d) and (c, b), or (a, c) and (d, b). In either case the edges representing the four corresponding adjacency types, if all present, form a cycle of length four in the master breakpoint graph with edges that alternate between those adjacency types that were there before the operation and those that are there after the operation. We call this structure a *quartet* (Figure S9).

A 3-breakpoint rearrangement involves six ends, and because by definition each end must obtain a new partner as a result of the operation, the six edges representing the adjacency types between these ends, if all present, also form a cycle in the master breakpoint graph alternating between adjacencies before and after the operation. We call this structure a *sextet* (Figure S11).

Finally, the reverse tandem duplication involves two ends for every breakpoint. Before the operation, these two ends are adjacent. After the operation they are each self-adjacent, i.e. one copy of the end is adjacent to the other copy in both cases. This forms a structure in the master breakpoint graph consisting of an edge between two nodes (representing the adjacency present before the operation), and a self edge for each node, representing the two self-adjacencies that are formed by the operation. We call this a *bow tie* (Figure S11). A reverse tandem duplication creates one bow tie for every breakpoint.

A connected component of the master breakpoint graph will be called a *module*. All modules must be subgraphs of these three types of graphs: quartets, sextets and bow ties. If any module is not a subgraph of one of these graphs, we reject the input (G, D). Because arbitrary subsets of data can be missing, all 11 possible subgraphs of these three graphs can appear as modules (Figure S10).

2.9 Iso-adjacency subtrees and adjacency graphs

Suppose $(k*, i*)$ and $(k*, *j)$ are the two edges in the master breakpoint graph incident upon node $k*$. Then $\mathbf{T}_k$, the augmented atom tree of atom k, can be divided into three parts: the $(k*, i*)$ *iso-adjacency subtree* $\mathbf{T}_{(k*,i*)}$, defined as the smallest undirected subtree including all the leaves of $\mathbf{T}_k$ that exhibit an adjacency of type $(k*, i*)$ (ignoring null adjacencies), the $(k*, *j)$ iso-adjacency subtree $\mathbf{T}_{(k*,*j)}$, defined as the smallest undirected subtree including all the leaves that exhibit the adjacency type $(k*, *j)$, and the remainder of $\mathbf{T}_k$, which we refer to at the *tether* (Figure S12). (There is a different partition of $\mathbf{T}_k$ into iso-adjacency subtrees and a tether for the end type $*k$, not shown.) The tether consists of a *main path* of edges and intermediate nodes connecting the two iso-adjacency subtrees, plus possibly some side branches off this main path leading to leaf nodes that have null adjacencies. If all the leaves of $\mathbf{T}_k$ have the same $k*$ adjacency, then the partition is trivial: there is only one iso-adjacency subtree of $\mathbf{T}_k$ for the end type $k*$ consisting of the entire tree, and the tether is empty. Since an adjacency type can be changed at most once, the iso-adjacency subtrees must be disjoint and thus this partition is always well-defined.

A module's *adjacency graph* depicts the individual adjacency relationships between the atom ends of the augmented atom trees for the nodes in the module (Figure S12). An edge between ends s and t in the module becomes a set of edges in the adjacency graph each representing a particular instance of the (s, t) adjacency type in the leaf genomes (Figure S12.)

We say that a module is *trivial* if it contains only one or two nodes and has no self-loops, else it is *nontrivial.* Nontrivial modules are further divided into those containing self-loops, called *rtd modules* (for "reverse tandem duplication"), and those nontrivial modules not containing self-loops, called *rearrangement modules.* Trivial modules show no evidence of an evolutionary operation having occurred. For a trivial module of one node, the adjacency graph is empty. For a trivial module of two nodes s and t and one edge, all adjacencies are of the same type, and thus the partition of the augmented atom trees associated with the module is also trivial.

Each edge in an augmented atom tree maps to a unique edge in the duplication tree by the mapping Φ. Thus for any nontrivial module, the main path of the tether of each atom tree, if the tether is nonempty, maps to a chain of edges in the duplication tree. Since the switch of adjacencies in an operation that gives rise to a nontrivial module must happen all at once, there must be a non-empty subchain of nodes and edges in the duplication tree that is part of all such chains from any single nontrivial module. For rearrangement modules, we define the *switchpoint range* as intersection of the nonempty tether main path images under the mapping Φ (Figure S13). For an rtd module, the point of duplication for the adjacent atom instances can always be traced to a specific node in the duplication tree, as determined by the distance between them and the mapping Φ. If the module has two self adjacencies, then let $\{n\}$ and $\{m\}$ be the two nodes of the duplication tree associated with these. We define the switchpoint range for the rtd module as the intersection of $\{n\}$ and $\{m\}$. Thus, the switchpoint range will be empty unless both self-adjancencies trace back to the same duplication node.

If every nontrivial module of the master breakpoint graph has a nonempty switchpoint range, and nothing discussed in previous sections has caused us to reject the input, we say that the master breakpoint graph and the duplication tree are *consistent.* If not, the input (G, D) is rejected.

2.10 Scheduling and polarizing the rearrangement operations

Assume the input is consistent. To construct an evolutionary history that explains the leaf genome data, each nontrivial module must be associated with an evolutionary operation that accounts for the changes in adjacency represented in it. That operation must occur somewhere within the switchpoint range for the module. For an rtd module, this operation must be a reverse tandem duplication, the switchpoint range must be a single duplication node, and the operation must occur at that node. We will say that the operation is *scheduled* to occur at that node. All the rtd modules scheduled to occur at a given duplication node will be part of the single reverse tandem duplication operation that happens at that node.

For a rearrangement module, the situation is more complex. Each such module must be associated with a rearrangement operation, and that operation must occur at some point within the switchpoint range of the module. The smallest rearrangement module is a chain of three nodes that we call an *elbow.* Since a single rearrangement operation involves at most six atom ends, there is only one way that two rearrangement modules can be associated with the same rearrangement operation: they must both be elbows and their switchpoint ranges must overlap. In this situation, we say that these modules are *pairable.* Two pairable elbows can be joined into a single *compound rearrangement module* in only one way, as illustrated in Figure S14. The switchpoint range for a compound rearrangement module is the intersection of the switchpoint ranges for its two elbows.

The *pairability graph*, denoted P, is a graph with a node for each rearrangement module, and an edge between two rearrangement modules iff they are pairable. A *maximum pairing* within P is a subgraph of P with the largest number of edges possible such that no node has degree greater than 1. A maximum pairing for any graph can be computed using the algorithm of Edmonds (1965). Each connected component of the maximum pairing of P represents either a single isolated rearrangement module or a compound rearrangement module consisting of a pair of elbows with overlapping switchpoint ranges. We will refer to these components as *post-pairing rearrangement modules*, or simply as "modules" when the meaning is clear from the context.

Each post-pairing rearrangement module is associated with a single rearrangement operation. For a compound rearrangement module, this is the unique 3-breakpoint rearrangement that is compatible with its two elbows. For non-compound rearrangement module, this operation is the unique simplest rearrangement operation that is consistent with the module. In particular, for a module with four nodes it is the unique 2-breakpoint rearrangement consistent with the quartet subgraph represented by that module. For a module with three nodes, a null end is added to complete the quartet, and the operation is the unique 2-breakpoint rearrangement consistent with the quartet. Similarly, for a module with six nodes the operation is the unique 3-breakpoint rearrangement consistent with the sextet subgraph represented by that module, and for a module with five nodes, a null end is added to complete the sextet, and the operation is the unique 3-breakpoint rearrangement consistent with the sextet. The additional adjacencies inferred in this process are called *module-level inferred adjacencies* (Figure S15).

A *schedule* of the rearrangement operations associated with the post-pairing rearrangement modules is an assignment that maps each operation to an edge in the duplication tree that lies within the switchpoint range of its associated post-pairing rearrangement module (Figure S16).

Since the edges of the duplication tree are oriented away from the root, the assignment of a rearrangement operation to an edge of the duplication tree *polarizes* the adjacency types in its post-pairing module into those that are *ancestral* (or *primitive*), i.e. existed before the operation, and those that are *derived*, i.e. existed after the operation. This includes the module-level inferred adjacencies. The edges of the module must alternate between ancestral and derived adjacency types (Figure S16). Often, but not always, the polarization is determined by the observed data at the leaves.

Finally, note that scheduling the rearrangement operations each to a branch of the duplication tree does not provide the order and distances between them we would need to expand the duplication tree to a complete evolutionary tree, with a separate node for each rearrangement. To prepare for the construction of a complete evolutionary tree, we schedule the order and timings of the rearrangement operations along a branch of the duplication tree arbitrarily. All orderings and timings are possible. The choice is immaterial to the goal of obtaining a simplest history, because the resulting genome will not depend on the order or relative timing of these rearrangements.

2.11 The complexity of a simplest history problem

It is clear that if there is an evolutionary tree that derives the data in (G, D) then the duplication tree and the master breakpoint graph must be consistent, there must be a speciation event for each speciation node in the duplication tree, a duplication event for each duplication node in the duplication tree, and at least as many rearrangements as there are post-pairing rearrangement modules. We define the *complexity* of the problem (G, D) as the total number of such inferred evolutionary operations. We will show by inductive construction that, conversely, for any (G, D) such that the master breakpoint graph and the duplication tree are consistent, there exists an evolutionary tree that derives (G, D) using a number of evolutionary operations equal to the complexity. This construction proves that the complexity is equal to the number of operations in the most parsimonious evolutionary history, and provides an algorithm for producing a most parsimonious history.

2.12 Inductive construction

Let h be any internal node in the duplication tree. Assume that the subtrees rooted at the children of h have already been solved, i.e. for each subtree there exists an evolutionary tree of ancestral genomes deriving the genomes at the leaves, possibly with missing data, via the scheduled operations and producing the correct distances between sites. We show how to extend this solution to the entire subtree rooted at h, reconstructing an ancestral genome at h that derives, possibly with missing data, the children below it, and then, by the inductive assumption, the leaves below them, using the duplication operation scheduled to occur at h and the rearrangement operations scheduled to occur along the branches from h to its children (*scheduled rearrangements*).

There are two cases: either h is a speciation node or a duplication node. If h is a duplication node then it has only one child. The single child genome g will contain both copies of some of the atom instances that were duplicated at h; for others, one or both copies will be missing. If h is a speciation node, then it has two children f and g respectively, each derived from a separate copy of h. To simplify the construction in this section, we will treat speciation as a special case of duplication, by adding the chromosomes of f to the genome g, and treating both together as a single child genome g of h. Thus there is only one inductive case to solve.

2.12.1 The sibling graph

We say that a pair of atom instances in the child genome g are *siblings* if they are derived from a single parent atom instance in h via the copy operation at h, as can be determined from the distance between them. The corresponding ends are called *sibling ends*. The *sibling graph* G is a structure that is used to calculate inferred ancestral adjacencies in the parent genome h for the atom instance ends in the child genome g, using the sibling relationship. There is a node in G for every atom instance end in g. For an atom instance that has no sibling in g, an *inferred sibling* is created, and a node for each of its ends is also included in G. There are four types of edges in G: there is an *atom edge* connecting the two ends of each atom instance, there is a *sibling edge* connecting each pair of siblings, there is a *child adjacency edge* connecting every pair of atom instance ends that are adjacent in g, and finally, there are *parent adjacency edges* that are defined in three phases as follows (Figure S17).

Initially the parent adjacency edges are defined to be identical to the child adjacency edges. In the first phase, for each scheduled rearrangement we map the nodes of its module to the atom instance ends of g and for any pair of non-adjacent ends in g that have a preexisting ("old"), inferred adjacency in the module that is scheduled to be broken in the operation, we add a parent adjacency edge in G representing that adjacency. In the second phase we remove all the parent adjacency edges in G corresponding to "new" adjacencies between nodes in g formed by the scheduled rearrangements. Note that after these steps there cannot be any *sibling rivalry* in which there is a parent adjacency edge in G from a node s to a node t, but the siblings s' of s and t' of t are connected by parent adjacency edges to something other than each other. If there was a rivalry, it would imply that there was a rearrangement operation after the duplication at h that changed the adjacency of an atom instance end to a new one without changing the adjacency of its sibling. In such a circumstance, that rearrangement would have been among the scheduled rearrangements, and its new adjacency would have been removed in phase 2. Finally, in the third phase, in every case where there is a parent adjacency edge from a node s to a node t, but no parent adjacency edge from the sibling s' of s to the sibling t' of t, we add a parent adjacency edges corresponding to this *inferred sib adjacency* between s' and t'. This defines the parent adjacency edges of the sibling graph G.

Because in creating the parent adjacency edges new adjacencies are removed wherever old adjacencies are added and there is never any sibling rivalry, each node in the sibling graph has parent adjacency edge degree at most 1. Furthermore, because all possible inferred sib adjacencies have been added, whenever one end in a pair of sibling ends has a parent adjacency edge to another atom instance end, its sibling end has a corresponding adjacency to the sibling of that other atom instance end. Thus, for every pair of sibling ends, either both will be adjacent to another pair of sibling ends, both will be adjacent to nothing (i.e. they will both be contig ends), or both will be adjacent to each other. This implies that there are only five possible types of connected components defined by the combination of the atom edges, sibling edges and the parent adjacency edges: *paired rings*, which are separate isomorphic rings of adjacent sibling pairs, *tandem rings*, which are paired rings that are topologically connected into a single ring containing both siblings from each pair, *cigars*, which are a run of adjacent sibling pairs self-adjacent pairs on either end, and two kinds of connected subgraphs of these three types, namely the *double contig*, which is a run of adjacent sibling pairs with contig ends at either side, and a *hairpin*, which is a run of adjacent sibling pairs with a self-adjacency on one end and a contig end on the other (Figure S18).

2.12.2 The reconstruction of the parent genome from the sibling graph

The parent genome h is obtained from the sibling graph essentially by replacing each sibling pair with a single atom instance, and maintaining only the parent adjacencies. Hairpin ends are treated the same as contig ends, becoming contig ends in h. Paired rings and tandem rings each become a single, unduplicated ring in h. Thus, we simply convert the components of the type shown in Figure S18 into chromosomes of a genome. Call this genome G'. The parent genome h is obtained from G' by removing any chromosomes that are entirely composed of atom instances whose atom trees do not show a duplication at h and that are nonhomologous to any atom instances in outgroups of the subtree below h. These chromosomes will be treated as having been gained (horizontally) after the duplication at h.

It remains to demonstrate that the parent h derives the child g via a duplication followed by the above mentioned chromosome gains and scheduled rearrangement operations, possibly with missing data. Let h' be the genome obtained from h by duplicating the chromosomes as specified in the parent adjacency contigs of the sibling graph, and then adding a single copy of each horizontally gained chromosome. Two isomorphic rings in h' are derived by separate duplication of a ring in h, a tandem ring in h' is derived by a tandem duplication of a ring in h, and a cigar in h' is derived by a reverse tandem duplication of a ring in h, creating the required self adjacencies between the two sibling pairs at the ends. Two isomorphic contigs in h' are derived by duplication of a contig in h, and a hairpin in h' is derived by reverse tandem duplication of a contig in h with the breakpoint at the contig end, creating the required self adjacency at the loop of the hairpin.

It is clear that all the atom instances needed to make the child genome g are present in h', and all the adjacencies between them that are observed in g are present in h', except those that are newly created in the scheduled rearrangements. Because only inferred old adjacencies and inferred sib adjacencies are added to the sibling graph in the construction leading to h', any adjacencies present in h' that would conflict with the new adjacencies must be the old adjacencies (added directly or reflected in the sibling) that existed before the scheduled rearrangement operation. The scheduled rearrangements will break any such old adjacencies and form the required new adjacencies. If all atom instances for the module representing the scheduled rearrangement were present in g, then by construction, all old adjacencies required for the operation to occur will be present in h'. If there is an atom instance from the module that is entirely missing from g, as in the case of an elbow that represents a 2-breakpoint rearrangement operation, or a module consisting of chain of

5 nodes that represents a 3-breakpoint operation, then the operation can occur using the null ends of the module. Thus, applying the scheduled rearrangement operations to h' we obtain a genome g' that is identical to g, apart from some possible missing data in g. Any chromosomes of g' that are entirely missing in g are considered to be chromosomal losses.

The above steps demonstrate one way that h derives g via a duplication operation, some chromosome gains, the scheduled rearrangement operations, some chromosomal losses, and possibly some additional missing data. However, rather than putting all the chromosomal gains before all the rearrangements and all the losses after all the rearrangements, it is more elegant and biologically realistic to associate specific gains and losses with particular rearrangements when possible, forming explicit insertion and deletion operations. In particular, after the order of the rearrangements has been arbitrarily chosen (Section 2.10), we reallocate the chromosome gains and losses as follows. For a chromosome loss, we find the point at which a rearrangement first creates this chromosome that is destined to be lost, and label that rearrangement a deletion, effecting the chromosome loss at that point, rather than waiting until all the other rearrangements have been completed. We do the symmetric procedure for chromosome gains, determining the last point at which the gained chromosome exists as a separate chromosome before a rearrangement combines it with other material, and label that rearrangement as an insertion.

2.13 The reverse evolution algorithm

Based on the above inductive construction, we define the *reverse evolution* algorithm to reconstruct a most parsimonious evolutionary history for a set of observed leaf genomes, after the operations have been scheduled. The basis case for the inductive construction is that case where h is a leaf of the duplication tree, and hence no reconstruction is required. The inductive step guarantees that once the children of a node are reconstructed, then the parent node can be reconstructed. The reverse evolution algorithm thus works backwards from the leaves of the duplication tree toward the root, first reconstructing the children, then the parent, until it reaches the root. The order of the reconstructions is otherwise arbitrary. The outcome of reconstructing the ancestors of the leaf genomes from the example in Figure S1 is shown in Figure S19.

At each step, a specific ancestral genome is reconstructed for the parent node h, and a recipe is given for applying a speciation or duplication operation to h, followed by a set of rearrangements with chromosome gains and losses, and possibly the introduction of some missing data, to obtain the children. The evolutionary tree obtained by the reverse evolution algorithm is thus sufficient to explain all of the observed data at the leaves of the tree. This provides a solution to the simplest history problem.

2.14 Filling in missing data

Although it solves the simplest history problem, the evolutionary tree obtained by reverse evolution does not necessarily contain all of the atom instances and adjacencies that can be inferred to have existed under the evolutionary model adopted here. An example is atom 7 in Figure S19. Parsimony suggests atom 7 would have been present in more of the ancestral genomes, e.g. in h_5, and thus that there is missing data in some of the ancestral genomes in the history depicted in Figure S19. To fill in additional parsimony-inferred atom instances and adjacencies, a second phase called the *fill in* phase is defined.

We say that genome h' is an *extension* of genome h if h' is identical to h except for some missing data in h. If h derives g, then starting instead with the extension h', we can track all the additional material in h' through essentially the same derivation we used to get g from h and obtain an extension g' of g with no missing data (see Section 5). We call this *propagating an extension.* If we like, we can then treat any extra material in g' that is not in g as missing data in g. Hence, it follows that whenever h derives g, any extension h' of h also derives g, possibly with some missing data.

The goal of the fill in phase is to infer and then eliminate as much of the missing data as possible in the derivation tree constructed by reverse evolution. This process begins at the root of the derivation tree and propagates out to the leaves, i.e. in the opposite direction from reverse evolution. At each step, we begin with an extension h' of the parent genome h and propagate this extension to the children of h. When there is more than one choice for how to propagate an extension, we choose one possibility as discussed in Section 5. To begin the process, for the root, we use the trivial extension $h' = h$.

Figure S20 shows the result of fill in phase on the example in Figure S19. Here the linear chromosome of the root genome, $r =$ 0 1 2 3 4 -11 -10 -9 -8 -7 -6 -5, derives two contigs 0 1 2 3 4 5 -10 -9 -8 and -6 11 of the genome h_5 via two rearrangement operations in the derivation induced by reverse evolution. Propagation of the trivial extension $r' = r$ through this derivation replaces these two contigs with the single contig 0 1 2 3 4 5 -10 -9 -8 -7 -6 11. From this we may infer (by parsimony) that an atom instance homologous to the observed atom instance 7 of genome e existed between atom instances 6 and 8 of h_5. Having made this inference, we now recognize the two rearrangements operations as inversions.

In the next steps we propagate this extension from h_5 to h_1 through the two duplications and two reciprocal translocations. After this, in moving from h_1 to h, all missing data filled in at earlier stages is eliminated in chromosome losses, leaving us with just the genome h that was previously derived by reverse evolution. The filling in of the missing atom 7 in the ancestral genomes is immaterial to the derivation of the data in the leaf genomes f and g, because whatever segment was present at the region occupied by atom 7 was deleted on both lineages leading to f and g. Nevertheless, it can be useful sometimes to replace missing data in the ancestral genomes with inferred atom instances, e.g. to properly classify operations as inversions, etc., or to infer that certain genes existed at certain times. It may also be useful to use this fill in procedure to infer missing data from the leaf genomes, e.g. to make a guess as to what might lie in an unsequenced gap of the genome of a living species, or how to order and orient separate contigs.

2.15 Efficiency

Since each local alignment segment endpoint creates at most one additional atom instance boundary in any other local alignment segment, the number of atom instances is at most quadratic in the number of local alignments plus the number of chromosomes in the leaf genomes. All of the constructions and manipulations in the above algorithm can be performed in time polynomial in the number of atom instances. Thus the algorithm takes time that is polynomial in the number of chromosomes and local alignments in the input.

3 Preserving continuity after an operation

Since we are dealing with continuous chromosomes, in every operation, care must be taken to insure that continuity is maintained in the child genome. A cut in a continuous chromosome at breakpoint x leaves one closed free end that contains the point x and one open free end that does not. When k cuts are made in a set of chromosomes, we get k closed free ends and k open free ends. When rejoining these $2k$ free ends in pairs to form a new child genome, each pair of joined ends must consist of one open end and one closed end. All combinations are possible, subject to this rule. In practice, we can imagine that we cut at k breakpoints, cleave off the k breakpoints leaving all $2k$ ends open, rejoin the $2k$ ends pairwise in a new combination any way we like by creating k joins, then we "seal" these joins by reinserting the original breakpoints in any way we like, subject to the "breakpoint assignment constraint" that the breakpoint used in sealing a join between two flanking open-ended segments must have originally been at the end of one of them. It is clear that there is always an assignment satisfying the breakpoint assignment constraint; such assignments can be obtained, e.g., in a simple greedy fashion. In practice, with 2-breakpoint operations, this rule implies, e.g., that an inversion must always invert either a closed or an open segment, and a fusion or circular incision must always fuse the clopen (open on one end and closed on the other) segment created by a single break in a circular chromosome with the similar clopen segment in the other chromosome in a tail-to-head fashion.

4 Reconciliation of the atom trees

We assume that the species tree and all the atom trees are given such that branch lengths in both the species tree and the atom trees reflect the exact evolutionary distances. The species are given for each leaf of each atom tree, but not the species for internal nodes. Any mapping Φ from an atom tree B to a species tree T that roots the atom tree is an *isometric reconciliation* if

(1) Every leaf of B maps to the leaf of the designated species in T.
(2) Each internal node of B maps to a speciation node in T or a point on a branch in T.
(3) The new root r of B maps to a point $\Phi(r)$ on a branch in T such that any other node x in B maps to $\Phi(x)$ below $\Phi(r)$ and $d(\Phi(x), \Phi(r)) = D(x, r)$.

Internal nodes of the atom tree that map to speciation nodes in the species tree are determined to be speciation events by an isometric reconciliation Φ, and nodes that map to points along branches are determined to be duplication events. These determinations in turn define the relationships of orthology and paralogy for the atom instances.

Theorem 1. *Given a set $\mathcal{T}_1, \ldots, \mathcal{T}_N$ of unrooted atom trees with designated leaf species and a species tree T, the isometric reconciliation of $\mathcal{T}_1, \ldots, \mathcal{T}_N$ with T is unique and there is an efficient procedure to either construct it or detect that no such isometric reconciliation exists.*

The reconciliation algorithm we define works from the leaves of an atom tree inwards. When all the nodes in two of the three subtrees branching off from an internal node x have been processed, then x can be processed. Other than this, the order of processing is arbitrary. In the last step, all three subtrees of the last remaining unprocessed node x will already have been processed, and processing x completes the reconciliation. This reconciliation algorithm is less complicated than the traditional methods, e.g. Goodman *et al.* (1979) and Guigo *et al.* (1996). When the species tree is unknown and one wants to reconcile an optimal species tree to minimize the number of duplications using the given topologies of atom trees, the problem is **NP**-hard (Ma *et al.*, 2000). In our case, the true species tree is known and the distances in the atom trees are exact.

Let x be an unmapped internal node of the atom tree B, with branches to nodes u, v and z. Suppose the mappings $\Phi(u)$ and $\Phi(v)$ of the nodes u and v to the species tree T have already been determined. Then the following procedure will map the node x to the species tree, or reject the input B as not reconcilable. As a side effect, if necessary the procedure will root the tree B.

MapAtomTreeNode(x) Let $d_1 = D(x, u)$, $d_2 = D(x, v)$, λ be the last common ancestor of $\Phi(u)$ and $\Phi(v)$ in T (if $\Phi(u) = \Phi(v)$ then $\lambda = \Phi(u) = \Phi(v)$), $d_1' = d(\Phi(u), \lambda)$ and $d_2' = d(\Phi(v), \lambda)$.

(1) If $d_1 + d_2 < d'_1 + d'_2$, reject and exit.
(2) Let $\epsilon = d_1 + d_2 - d'_1 - d'_2$ (≥ 0).
(3) If $d_1 = d'_1 + \epsilon/2$ (and $d_2 = d'_2 + \epsilon/2$) then map x to a point $\Phi(x)$ at distance $\epsilon/2$ above λ in the species tree T.
(4) Else if $\epsilon = 0$ then map x to a point $\Phi(x)$ at distance d_1 from $\Phi(u)$ and d_2 from $\Phi(v)$ on the path connecting $\Phi(u)$ and $\Phi(v)$ in T.
(5) Else (i.e. if $\epsilon > 0$ and $d_1 \neq d'_1 + \epsilon/2$) if the atom tree B is already rooted then reject and exit
(6) Else
  (a) Place the root r of B at distance $d'_1 + \epsilon/2$ from u and $d'_2 + \epsilon/2$ from v on the path that connects u and v in B, and map r to a point $\Phi(r)$ at distance $\epsilon/2$ above λ.
  (b) If $d_1 < d'_1 + \epsilon/2$ then map x to the point $\Phi(x)$ at distance d_1 above $\Phi(u)$, else (i.e. if $d_2 < d'_2 + \epsilon/2$) map x to the point $\Phi(x)$ at distance d_2 above $\Phi(v)$.
(7) If z has already been mapped to the node $\Phi(z)$ in T, then
  (a) If the tree is already rooted, reject and exit unless $\Phi(z)$ lies below $\Phi(x)$ in T or vice-versa and $d(\Phi(x), \Phi(z)) = D(x, z)$.
  (b) Else
    i. Let $d = D(x, z)$, λ be the last common ancestor of $\Phi(x)$ and $\Phi(z)$ in T, $d'_1 = d(\Phi(x), \lambda)$, $d'_2 = D(\Phi(z), \lambda)$ and $\epsilon = d - d'_1 - d'_2$.
    ii. If $\epsilon < 0$, reject and exit.
    iii. Else if $\epsilon = 0$ and $\Phi(x) = \lambda$, reject and exit.
    iv. Else place the root r of B at distance $d'_1 + \epsilon/2$ from x and $d'_2 + \epsilon/2$ from z on the path that connects x and z in B, and map r to a point $\Phi(r)$ at distance $\epsilon/2$ above λ.

We construct the reconciled tree $\mathbb{T}$ for a set of atom trees as follows (see also Figure S8). We initialize $\mathbb{T}$ to be the same as species tree T. Eventually in $\mathbb{T}$, additional points will be labeled along branches based on the mapping from nodes in the atom trees.

ReconcileTrees$(T, \mathbf{T}_{i,\ i=1,2,\ldots,N})$
Input: species tree T, atom trees $\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_N$
Output: reconciled duplication tree $\mathbb{T}$

```
$\mathsf{T} \Leftarrow T$
for all atom trees $\mathsf{T}_i$ such that $1 \leq i \leq N$ do
  $M \Leftarrow \emptyset$ (the set of maximally internal mapped nodes)
  for all leaf nodes $k$ in $\mathsf{T}_i$ do
    $\Phi(k) \Leftarrow g$, where $g$ is the genome to which $k$ belongs.
    add $k$ to $M$ and set $t_k = \{k\}$ (mapped subtree for $k \in M$).
  end for
  while $M$ has more than two tree nodes do
    if $u, v \in M$ and $u$, $v$ and $z$ are connected to an unmapped node $x$ in $\mathsf{T}_i$ then
      MapAtomTreeNode($x$)
      remove $u$, $v$ and (if necessary) $z$ from $M$
      if $M$ is empty then
        set $M$ to the set consisting of just the root node $r$ and set $t_r = B$
      else
        add $x$ to $M$ and set $t_x = t_u \cup t_v \cup \{x\}$
      end if
```

```
        end if
    end while
end for
```

Now we prove Theorem 1.

Proof. We will show that the algorithm ReconcileTrees produces an isometric reconciliation or detects if no such reconciliation is possible.

(A) We prove that if the algorithm accepts the tree then the tree is reconcilable and the reconciliation is unique.

First, we prove that at every step in the process, each of the maximally internal mapped nodes $m \in M$ in the current atom tree $B = \mathsf{T}_i$ is associated with a subtree t_m (constructed in ReconcileTrees) of mapped nodes of B that has the following properties:

(1) If t_m does not contain the root r then $\Phi(m)$ lies above $\Phi(y)$ for all $y \neq m \in t_m$ and $D(m, y) = d(\Phi(m), \Phi(y))$. Else, if t_m contains the root r then $\Phi(r)$ lies above $\Phi(y)$ for all $y \neq r \in t_m$ and $D(r, y) = d(\Phi(r), \Phi(y))$.
(2) All leaves of t_m are mapped to the genome of the species to which they belong

and that Φ is the unique mapping with these properties.

We prove this by structural induction. It is clear that the above claim is true after the initial step, in which each of the leaves of B is mapped. Let m be an unmapped internal node of the atom tree, with branches to nodes u, v and z. Suppose the mappings of u and v to the species tree have already been determined and we have subtrees t_u and t_v that satisfy the above claim.

We first consider the case that both t_u and t_v do not contain root r. If the tree has not been rooted after mapping m, then $\Phi(m)$ will be mapped to a point in T that is above $\Phi(u)$ and $\Phi(v)$. Based on the induction, $\Phi(m)$ lies above $\Phi(y)$ for all $y \in t_u$ and $y \in t_v$. Hence, $\Phi(m)$ lies above $\Phi(y)$ for all $y \in t_m$.
Since we have

$$d(\Phi(m), \Phi(u)) = D(m, u) \text{ and } d(\Phi(m), \Phi(v)) = D(m, v),$$

therefore, for all $y \in t_u$, we have

$$d(\Phi(m), \Phi(y)) = d(\Phi(m), \Phi(u)) + d(\Phi(u), \Phi(y)) = D(m, u) + D(u, y) = D(m, y)$$

Similarly, for all $y \in t_v$, we have

$$d(\Phi(m), \Phi(y)) = d(\Phi(m), \Phi(v)) + d(\Phi(v), \Phi(y)) = D(m, v) + D(v, y) = D(m, y)$$

When the atom tree can be rooted after mapping m using u and v, without loss of generality, suppose r is on the path from m to v. Then $\Phi(r)$ will be mapped to a point in T that is above $\Phi(m)$ and $\Phi(v)$. Based on the induction, $\Phi(r)$ lies above $\Phi(y)$ for all $y \in t_m$. For all $y \in t_u$,

$$d(\Phi(r), \Phi(y)) = d(\Phi(r), \Phi(u)) + d(\Phi(u), \Phi(y)) = D(r, m) + D(m, u) + D(u, y) = D(r, y)$$

For for all $y \in t_v$,

$$d(\Phi(r), \Phi(y)) = d(\Phi(r), \Phi(v)) + d(\Phi(v), \Phi(y)) = D(m, v) - D(m, r) + D(v, y) = D(r, y)$$

We then consider the case that one of t_u and t_v contains root r. Without loss of generality, suppose t_v contains the root r. After the mapping of m, $\Phi(m)$ will be mapped to a point on the path from $\Phi(v)$ to $\Phi(u)$ such that $\Phi(m)$ lies under $\Phi(v)$ and $\Phi(u)$ lies under $\Phi(m)$. Based on the induction, $\Phi(r)$ lies above $\Phi(m)$ and $\Phi(y)$ for all $y \in t_v$ and $y \in t_u$. Hence, $\Phi(r)$ lies above $\Phi(y)$ for all $y \in t_m$. Again, based on MapAtomTreeNode, we have

$$d(\Phi(m), \Phi(u)) = D(m, u) \text{ and } d(\Phi(m), \Phi(v)) = D(m, v)$$

Therefore,

$$d(\Phi(r), \Phi(m)) = d(\Phi(r), \Phi(v)) + d(\Phi(v), \Phi(m)) = D(r, v) + D(v, m) = D(r, m)$$

Also, for all y in the original t_u,

$$d(\Phi(r), \Phi(y)) = d(\Phi(r), \Phi(v)) + d(\Phi(v), \Phi(m)) + d(\Phi(m), \Phi(u)) + d(\Phi(u), \Phi(y))$$

$$= D(r, v) + D(v, m) + D(m, u) + D(u, y) = D(r, y)$$

Finally we consider the case where $\Phi(z)$ has also been determined before we map m. There are two possible situations where (i) either t_m or t_z contain the root and (ii) both t_m and t_z do not contain the root at this point. We can use the similar logic above to show that $\Phi(r)$ lies above $\Phi(y)$ for all $y \in t_m$ and $y \in t_z$, and that for all y in t_m and all y in t_z

$$d(\Phi(r), \Phi(y)) = D(r, y).$$

Since the above properties (1) and (2) hold at every step in the process, they also hold when the processing of B is complete, if the input is not rejected. However, at this point the set M consists of just the root node r and all nodes of B are included in t_r. Thus, every node x in B will be uniquely mapped to a point $\Phi(x)$ in T that lies below $\Phi(r)$ such that $d(x, r) = D(\Phi(x), \Phi(r))$. Hence, B will be uniquely isometrically reconciled with T. It follows that if the algorithm ReconcileTrees terminates without rejecting its input, it produces an isometric reconciliation of all its input atom trees with its input species tree.

(B) We prove that if the algorithm rejects the atom tree then the atom tree is not reconcilable.

We now verify that if the above procedure rejects B, then B is not reconcilable. In the algorithm MapAtomTreeNode, we use node u and v in the atom tree B to map the unmapped node m. ¿From the above proof, we know that subtrees t_u and t_v have been uniquely mapped to the T.

In step (5) and step (7)(a) in MapAtomTreeNode, if one tries to root the atom tree B again when B is already rooted when mapping m, then B is certainly not reconcilable because all the nodes that have already been mapped to T are uniquely mapped based on the proof in (A).

We then consider the case in step (1) in MapAtomTreeNode. Because a successful reconciliation maps the nodes of the atom tree to the nodes and edges of the species tree, there will be no shorter path between two mapped nodes in the atom tree than the distance between the two original nodes in the species tree. d'_1 and d'_2 are the distances from $\Phi(u)$ and $\Phi(v)$ to their last common ancestor λ in T. Hence $d'_1 + d'_2$ will be the distance from $\Phi(u)$ to $\Phi(v)$ in T. Also, $d_1 + d_2$ is the distance of the shortest path from u to v in the atom tree. Therefore, we must have $d_1 + d_2 \geq d'_1 + d'_2$. In step (1), if $d_1 + d_2 < d'_1 + d'_2$, then there is no possible way to map u and v to T that can achieve $d(\Phi(u), \Phi(v)) = D(u, v)$. Therefore B is not reconcilable. Step (b)(ii) is proved similarly.

For step (b)(iii), i.e. $\epsilon = 0$ and $\Phi(m) = \lambda$, since t_u, t_v, and t_z all have been uniquely reconciled based on part (A), the only way to map $\Phi(m)$ is to map to λ such that there is a three way split of the root of the reconciled atom tree with three descendant subtrees, t_u, t_v, and t_z. Hence, B is not reconcilable.

Therefore, in ReconcileTrees, if the procedure MapAtomTreeNode rejects its input then the tree B is not isometrically reconcilable. It follows that if ReconcileTrees rejects its input then it is not isometrically reconcilable. Combined with part (A), this establishes that the ReconcileTrees algorithm is correct.

(C) We prove that the algorithm runs in polynomial time.

For each node in an atom tree, the running time of MapAtomTreeNode depends on the procedure of finding the last common ancestor between two points on branches in T. The crude (non-amortized) bound on the running time for this procedure is $O(n)$, where n is the total number of tree nodes in species tree T. Therefore, the overall computational complexity of ReconcileTrees can be bounded in $O(nm)$, where m represents the total number of tree nodes in all atom trees. □

5 Propagating an extension of a genome through a derivation

Suppose that h derives g and h' is an extension of h. We demonstrate that then h' derives a genome g' with no missing data, such that g' is an extension of g, and hence h' derives g, possibly with some missing data. If h derives g via a single rearrangement operation then we can use the corresponding breakpoints in the extension h' of h to perform the same breaking and reforming of adjacencies, and thereby obtain from h' a genome g' that is clearly an extension of g.

Suppose h derives g by a duplication. Then for any ring x in h, the same ring must appear in any extension h' of h and can be duplicated in the same manner (separate, tandem or reverse tandem), obtaining the corresponding duplicated chromosomes in g'. Any contig x in h will be contained in a chromosome in h' and be duplicated along with that chromosome in forming g'. If x is separately duplicated in the derivation of g from h, then by creating missing data at the ends of the copies of x in g' that result from the duplication of the chromosome in h' that contains x, we obtain equivalent copies of x in g. If x is reverse tandem duplicated at k breakpoints in the derivation of g from h, then we can apply reverse tandem duplication at these same breakpoints when the chromosome in h' containing x is duplicated in the derivation of g', and then invoke missing data at the ends x as above to obtain equivalent copies of x in g.

It follows that if h derives g from any single operation, be it a rearrangement or a duplication, then for any extension h' of h there is an extension g' of g such that h' derives g' from an analogous operation. By induction, this argument can be extended to derivation from any series of operations. Similarly, it can be shown that if h derives f and g through speciation, then any extension of h derives an extension of f and g as well. Using these procedures, any extension of the root genome can be propagated through a derivation tree, creating extended genomes at each node that are connected by derivation edges with no missing data, as illustrated in Figure S20.

There is one important choice in this process. When one or more contigs in h form an extended chromosome in h' that is circular and is not reverse tandem duplicated, then it can either be tandem duplicated or separately duplicated in the derivation of g' from h'. Either way the result is a valid extension g' of g. If the chromosomes in the derivation tree of Figure S1 were circular, then this choice would have occurred twice in the example in Figure S20, once in the derivation of h_4 from h_5, and once in the derivation of h_2 from h_3. Here, if tandem duplication is chosen the intermediate genomes remain small, but if separate duplication is chosen instead, the intermediate genomes h_4, h_2, and h_1 are much larger, because deletion of duplicated material is postponed until h is created. There is in general there is no unique way to fill in missing data, and the choice of how to fill in missing data can affect the size of the inferred ancestral genomes.

6 Exponentially large ancestral genomes

In extreme cases, if judicious choices are not made when rearrangements are scheduled, ancestral genomes can become exponentially large compared to the size of the observed leaf genomes, a kind of "ancestral genome bloat". An example of this is the following. Let n be a positive integer and define atoms $\{1, \dots, (2n)\}$. Let the genome $h = 1\ 2\ \dots\ (2n)$, a contig, and let the genome $g = 1\ 2\ 2\ 3\ 4\ 4\ 5\ 6\ 6 \dots (2n-1)\ (2n)\ (2n)$, a contig, where for each even k, $2 \leq k \leq 2n$, the distance between the two copies of k is k. It is easy to see that h derives g with n duplications and n rearrangements. In fact, given just the leaf genome g, the reverse evolution algorithm will recover the ancestral genome h with just such a derivation, including missing data.

In this example we have considerable liberty in scheduling the rearrangements. If we schedule them to occur as early as possible in the derivation, then the intermediate reconstructed ancestral genomes remain small. However, if we postpone all the rearrangements until after all the duplications have occurred, then we accumulate, after all the n duplications of h, a genome with 2^n chromosomes and $2n \cdot 2^n$ atom instances. It turns out that most of this genome will be represented as missing data in the reverse evolution algorithm, so this step is still very efficient. However, trouble ensues in the fill in phase if the missing data is explicitly restored by filling in all these genome copies. Luckily we can still compute this fill in phase efficiently using pointers to refer to duplicated chromosomes and chromosome segments, rather than copying them explicitly. So this does not contradict the existence of a polynomial time algorithm for the general case, with an arbitrary consistent schedule for the operations, and a complete fill in phase. Nevertheless, the existence of exponentially large ancestral genomes is not biologically realistic, and it would be useful to explore policies or heuristics for scheduling rearrangement operations that avoid ancestral genome bloat as much as possible.

7 Atoms and atom trees for mammalian chromosome X

We partition the six genomes using a method we call Buster. The first stage of Buster is to run the standard UCSC chaining and netting pipeline (Kent *et al.*, 2003) on a pairwise basis between a set of chromosomes from a set of input species. In this case we created alignment chains between the X chromosomes of the six species: human, chimp, rhesus, mouse, rat, dog. These alignments include the "self" chains which are built from aligning a chromosome to itself. For pairwise chains, we only retain alignments where both pieces are on chrX, since we assume the evolution of mammalian X chromosomes is self-contained. From these chains we harvest a set of pairwise gap-less blocks that we then aggregate into multiple species gap-less blocks that we call gap-less atoms. A gap-less atom is defined by a set of one or more contiguous intervals within the set of input chromosomes. Each interval is called an instance of the atom.

To aggregate the pairwise blocks from the chains into atoms we use a greedy method that first creates a set of putative atoms using each species in turn as a reference species. We call these referenced atoms. For each chromosome in the input set, we establish that chromosome as the reference species and lay out the blocks each on it's own row. Once all the blocks are laid out, we chop up the blocks whenever the number or content of the aligning rows changes. The composition of the atoms within a referenced atom set differs depending on the reference species and the order in which the bases within the reference chromosome are traversed because the pairwise alignments are not constrained to be consistent or transitive.

To arrive at the best set of unreferenced atoms in a species independent manner we use the second phase of Buster which is another greedy method that first sorts each of the referenced atom lists by a scoring method that rewards total length (number_of_base_pairs X number_of_instances). We then iteratively grab a candidate atom which is the highest scoring gap-less atom from any of the sorted referenced atom lists. If any of the bases in any of the instances of the candidate atom overlap any base in any instance from any previously chosen atom we discard the atom, otherwise we choose the atom to be in the consensus gap-less atom set. This consensus gap-less atom set necessarily has lower total coverage of the input chromosomes than the pairwise alignments do because some alignments are discarded because they are not consistent with an alignment that was accepted earlier in the atom building process.

Depending on a variety of parameters including the score filtering on the chain alignments, we end up with something on the order of one to ten million gap-less atoms in the consensus set from the six species X chromosomes. Many of these gap-less alignments are separated by only a base or two in one or more of the species that has been inserted or deleted since the common ancestor. To create a set of atoms that's more amenable to a human understanding and the limits of our computational algorithms, we aggregate gap-less atoms into smaller sets of gapped atoms where we assume a common ancestry to regions discovered in a common order and orientation, even if the regions are separated by small stretches of un-aligning bases which we infer to be an insertion or deletion in one of the species.

We can partition a set of chromosomes into any number of gapped atoms by tuning the size of the minimum atom and joining atoms that are made contiguous by the 'tuning out' of the atoms that are below the minimum size. In the Buster algorithm's third phase, we use an annealing process that tunes out a certain percentage of low scoring atoms from each generation of prospective atoms and then generates the next generation of atoms by aggregating atoms that are contiguous when not considering those tuned out atoms.

The atom trees are inferred using a heuristic combination of distance information, atom adjacencies, the minimum evolution principle, and duplication/loss parsimony. Methods are similar to those in related work (Chen *et al.*, 2000; Durand *et al.*, 2006; Bansal *et al.*, 2007; Rasmussen and Kellis, 2007). The goal of this approach is to have a fast method which will readily scale up to genome-level analysis, inferring trees correctly under the infinite sites model and gracefully handling violations of that model. The species tree is assumed to be known and utilized in several respects to constrain the atom tree - one key aspect being calculating the total number of duplications and losses via reconciliation.

For each atom, a distance matrix is first calculated using the Jukes-Cantor model from a Buster multiple alignment. Additionally the flanking adjacencies (defined to be the nearest 5' and 3' atoms) for each instance of an atom are determined. Similar flanking adjacencies define a shared context for atom instances that is indicative of shared history and can be thought of as a crude measure of synteny and orthology. Thus, instances of an atom in the same context are grouped together and a tree associated with each context is created via Neighbor-Joining using constraints from the species tree. This results in a set of trees which we refer to as *context trees*.

Figure S22A shows a hypothetical multiple genome alignment. The atom of interest is `B`, which has multiple duplicated copies in several of the species (indicated by `B'` and `B''`). The three boxes represent the three different contexts that are relevant to atom `B`. The left box represents the `AC` context, the middle box the `DE` context and the right box the `FG` context. Figure S22B shows the tree that is inferred for each context.

The set of context trees can be thought of as subtrees of the final tree that we wish to infer. These trees are sorted by the number of leaves and greedily assembled in that order. At each iteration, all possible combinations of merged trees are checked for their total branch length and duplication/loss count. The best scoring tree is saved and merged with the remaining trees. The process of merging continues until a single tree remains, which is the atom tree. Figure S22C shows the atom tree that is constructed from combining the three trees from Figure S22B. The subtrees that represent the original context trees are illustrated by differing line dashes. The connectivity of the context trees in the atom tree is determined by both duplication/loss parsimony and minimizing the total tree branch length. The fact that `human.B'` sister leaf is `human.B` and not `human.B''` is not arbitrary but would come about due to the utilization of distance information, since the duplication cost for either choice would be the same.

Finally, before the global reconciliation step (Section 4), the atom trees are then globally adjusted with some scaling and modification of the branch lengths of individual atom trees to adjust for rate variation across atoms in an attempt to make the branch lengths comparable to those in the species tree.

8 Reconstruction of mammalian ancestral chromosome X

We devised a progressive reconstruction algorithm that is consistent with the standard infinite sites algorithm to be applied in practice to handle data with breakpoint reuse and approximate distances. The progressive algorithm is summarized below. We assume that we have identified atoms, reconciled the duplication tree, and built the augmented atom trees, all of which are the input data for the reconstruction algorithm.

(1) Build the master breakpoint graph and identify connected components in the graph. Since the data may contain breakpoint reuse, the connected components are not always a subgraph of one of the cases in Figure S11. We classify all the components of the master breakpoint graph into two categories: (i) *good component*: the component is one of the subgraphs in Figure S11 and its switchpoint range is not empty; (ii) *bad component*: the component is an arbitrary graph other than subgraphs in Figure S11 or the component is a subgraph in Figure S11 but its switchpoint range is empty.

(2) Start from the leaf genomes and in a post-order fashion, reconstruct every intermediate node h in the reconciled duplication tree progressively as follows.

   (a) Build the sibling graph based on the one or two (depending on whether h is a duplication node or a speciation node) children of h that have already been reconstructed. Define the *local breakpoint graph* by collapsing each pair of sibling atom ends into a single node. This node represents the corresponding parent atom end in h. The edges incident upon the parent atom end are those derived from the child adjacency edges. These represent the adjacencies for the corresponding sibling ends in h's children. Thus, there are at most two adjacency edges incident upon any node in the local breakpoint graph.

   (b) As done for the master breakpoint graph, define a nontrivial component of the local breakpoint graph to be a *local bad component* if it is not a subgraph of a quartet, sextet or bowtie, else call it a *local good component*. (All local components will switch on this edge, so they have nonempty switchpoint range by definition.) Break each local bad component of the local breakpoint graph into local good components by adding engineered atoms as described in Figure S21.

   (c) For each connected component C in the local breakpoint graph, if C is a subgraph of a good component C' in the master breakpoint graph, infer the missing adjacencies to complete the component as in Figure S15.

   (d) For each connected component in the local breakpoint graph, determine the ancestral and derived adjacencies in the connected component. The inference is based on parsimonious adjacency changes using information from both the descendants and the outgroups.

Choose randomly if both choices are consistent. Note that if the input data is infinite-sites compatible, then the determination of ancestral and derived adjacencies in this step is always unambiguous as long as there is outgroup information for at least one of the adjacencies in the connected component.

   (e) Construct a new sibling graph including the engineered atoms, using the procedure from Section 2.12.1.
   (f) Reconstruct the parent genome in g based on sibling graph by undoing the operations corresponding to the components using the procedure from Section 2.12.2.

(3) Start from the root of the reconciled duplication tree, redo the operations assigned during the reverse evolution and fill in the missing data for each intermediate and leaf genome using the procedure from Section 2.14.

(4) Remove all the engineered atoms.

It is easily verified that if the original data are consistent with the infinite sites model, then this procedure produces a solution with the optimal number of operations, just as the procedure given in Sections 2.13 and 2.14. It is an extension of that method that produces a heuristic solution instead of rejecting the input when it fails to satisfy that model.

When reconstructing an ancestral nuclear chromosome from an animal, we also want to avoid fragmentation into multiple contigs and circular (sub)chromosomes. If the reverse evolution algorithm creates an ancestral genome that contains circular (sub)chromosomes, then we look for operations in the step that created this genome where there was ambiguous outgroup support in making the decision as to which adjacencies were primitive and which derived. Then we switch the primitive and derived edges in those operations in an attempt to decrease the number of circular chromosomes. Finally, we remove adjacencies to break the remaining circular chromosomes. If at this point the reconstructed ancestral chromosome consists of multiple linear contigs, we join these together heuristically in an order that best reflects the order in the leaf genomes.

To demonstrate the above procedure, we reconstructed the evolutionary history of the placental mammalian chromosome X starting from a decomposition of the human, chimp, rhesus, mouse, rat and dog chromosome X sequences into 1,917 atoms. This decomposition, constructed as described above, was expected to exhibit considerable breakpoint reuse due to the large number of smaller atoms that were "tuned out". The 1,917 atoms were reconciled into a duplication tree with 110 duplication nodes in addition to the 5 speciation nodes. The reconstruction used 15 engineered atoms, broke circular chromosomes 51 times, and merged 50 linear contigs at the root genome into one linear reconstructed ancestral chromosome X.

The master breakpoint graph consists of 568 connected components, and only 274 of these are good components (Figure S23). Among all the nodes in the master breakpoint graph, 576 nodes have degree more than 2, indicating explicit breakpoint reuse. Intermediate local breakpoint graphs at every speciation node are shown in Figure S24. These show that although the master breakpoint graph has a lot of bad components, except for the rodent ancestor and Boreoeutherian ancestor, the local breakpoint graphs have no bad components. This indicates that even with numerous violations to the theory at the global evolutionary history level, the theory may still apply very well at the local level.

In addition to the figure of the evolution of the mouse genome in the main text, Figures S28, S29, S30, S31, S32 show the evolution from the ancestors to other five leaf genomes. Figure S33 further illustrates a zoom-in view of part of the human chromosome X history, which contains an inversion happened after Catarrhini common ancestor.

9 Evaluation of reconstruction accuracy

As discussed in the text, we developed a simulation program to generate synthetic data to evaluate our algorithms. Since our purpose is to evaluate the infinite sites reconstruction algorithm, we did not model actual DNA in the simulation. The simulator starts with a hypothetical "ancestor" genome that evolves into the genomes of the extant species through duplication, rearrangement and speciation operations as described above. While more sophisticated simulations are possible, the simulations we performed followed a very simple procedure. An initial excess of potential breakpoints was defined, and when an operation was applied, breakpoints were chosen uniformly at random from the set of used or unused breakpoints on that chromosome depending on whether we want to get pure infinite sites compatible data or data with breakpoint reuse.

In general we adjusted the rates of the operations to match those estimated from the evolution of the placental mammalian genomes in the phylogenetic tree `((((human,chimp),rhesus),(mouse,rat)),dog)`, but in specific experiments we modified some rates to more extreme values to assay the effects on the reconstruction. The root genome was assigned 4,000-5,000 segments. After simulated evolution was performed, we formed atoms by collapsing segments where order and orientation was conserved across all species. The simulator's parameters were adjusted to guarantee that the extant species had approximately 2,000 atoms after collapsing. We chose simulation parameters so that approximately 5-10% of the atom instances have observed paralogs in the extant species created by duplications, roughly consistent with what we observed in the real data from the six species above, so that the net change in the number of atoms due to insertion, duplication and deletion was consistent with what we observed in the different lineages (achieved using an overall deletion/insertion ratio of 3), and so that the apparent level of breakpoint reuse was similar to that observed in the real data. Data that show how realistic our simulated data is as a model of the evolution of chromosome X in the above six species are given in Table S1.

We compared the infinite sites algorithm with the DUPCAR reconstruction program Ma *et al.* (2008), a method we developed previously that was purely based on parsimonious inference of ancestral atoms and adjacencies without explicitly modeling operations. For each data set, we compare the predicted ancestral genomes with the true ancestral genomes generated in the simulations. We calculate the error rates, S_{atom} and S_{adj}, of atom instances and adjacencies respectively.

$$S_{atom} = \frac{|R \cup P| - |R \cap P|}{|R \cup P|} \times 100\%$$

where R is the set of atom instances in the true ancestral genome, P is the set of atom instances in the predicted genome, and $|X|$ denotes the size of the set X. A predicted atom instance in an ancestral genome is defined to be the same as an atom instance in the true ancestral genome if its distance to all leaf atom instances is the same in the prediction and in the true history. S_{atom} represents the percentage of atom instances where the true genome and the predicted genome disagree. S_{adj} is defined analogously using adjacencies between atom instances that are present in both the predicted and the true ancestral genomes. (See below for further details on these definitions).

We first compared the two methods by varying the overall breakpoint reuse ratio in the evolutionary tree, which we denote by r. This quantity is a generalization of the breakpoint reuse ratio defined for rearrangements between two species (Sankoff and Trinh, 2005). It is defined as

$$r = \frac{2x + 3y}{a + m - n}$$

where x is the number of 2-breakpoint operations in the whole evolutionary history, y is the number of 3-breakpoint operations, a is the total number of atoms, m is the number of uses of contig ends as breakpoints, and n is the number of contigs in the root genome. For our atomization of chromosome X in the six species above and the heuristic solution to the simplest history problem obtained from it by the infinite sites algorithm, the breakpoint reuse ratio is 1.39. The range of breakpoint reuse ratio between 1 and 1.5 thus represents a reasonable test of the ability of our heuristic methods to cope with breakpoint reuse.

The results of varying the breakpoint reuse ratio in the simulations are shown in Figure S26A (see also Table S2). For these experiments the predicted number of operations varied from 96% of the actual number for $r = 1$ to 43% for $r = 1.4$ and 25% for $r = 1.5$ (Table S2). We then varied the deletion vs. insertion rate ratio but kept the breakpoint reuse ratio at 1 (Figure S26, panel B, and Table S3).

For any particular ancestor, we can separately count the atom instances that are in the true ancestor but not in the predicted ancestor. We call these *false negative atom instances*, and denote their number by FN_{atom}. We can also count the atom instances that are in the predicted ancestor but not in the true ancestor (*false positive atom instances*, FP_{atom}), and atom instances that are in both the true and the predicted ancestor (*common atom instances*, C_{atom}). The prediction algorithms we examine here use exact distances between homologous leaf atom instances to reconstruct an atom tree that is necessarily a subtree of the true tree of atom instances generated in the simulation, differing only because of atom instance losses. Therefore they do not have any false positive atom instances.

For the atom instances that are in both the predicted and actual histories, we can define *false negative adjacencies* (FN_{adj}), *false positive adjacencies* (FP_{adj}) and *common adjancecies* (C_{adj}) for any given ancestor genome in an analogous way. Thus, in our analysis we only count adjacencies in FN_{adj} and FP_{adj} if both endpoint atom instances of an adjacency are in C_{atom}.

For each ancestral genome, we further categorize the atom instances into three cases:

- *unambiguous*: an atom instance can be unambiguously determined to exist in a particular ancestral genome by parsimony based on the presence of and distances between homologs in the leaf genomes.
- *ambiguous*: the existence of the atom instance is ambiguous given the homologies in the leaf genomes, again using parsimony.
- *novel*: there are no homologs of the atom instance in any of the leaf genomes.

Similarly, we also classify adjacencies into three categories:

- *unambiguous*: the adjacency can be unambiguously determined by parsimony using the presence of homologous adjacencies in the leaf genomes.
- *ambiguous*: the existence of the adjacency is uncertain based on the homologies in the leaf genomes.
- *novel*: the adjacency does not appear in any of the leaf genomes.

An example of distinguishing different kinds of atom instances and adjacencies in a particular ancestor can be found in Figure S25.

Based on the classification of different types of ancestral atom instances and ancestral adjacencies, we separate FN_{atom} into (1) FN^{u}_{atom} for unambiguous false negative atom instances; (2) FN^{a}_{atom} for ambiguous false negative atom instances; (3) FN^{n}_{atom} for novel false negative atom instances. C_{atom}, C_{adj}, FP_{atom}, FN_{adj}, FP_{adj} can be similarly partitioned.

As above, we use a normalized measurement of the symmetric difference, denoted S, to quantify the extent of the disagreement between predicted and true ancestral genomes. This measure is computed separately for each type of atom instance and adjacency. For instance, for ambiguous false negative atom instances, S^a_{atom} is defined as

$$S^a_{atom} = \frac{FN^a_{atom} + FP^a_{atom}}{C^a_{atom} + FN^a_{atom} + FP^a_{atom}}.$$

We also compute a combined S value, as reported in the main paper, to capture the overall performance of a reconstruction method, i.e.

$$S_{atom} = \frac{FN_{atom} + FP_{atom}}{C_{atom} + FN_{atom} + FP_{atom}}.$$

Tables with the numbers that are illustrated in Figure 7 in the main paper (also Figure S26) can be found in Table S2 and Table S3. In addition, the detailed S values for unambiguous, ambiguous, and novel atom instances and adjacencies predicted by two different methods (infinite sites algorithm and DUPCAR (Ma *et al.*, 2008)) in Boreoeutherian and Euarchontoglires ancestor can also be found in Table S4, S5, S6, and S7. Figure S27 shows additional comparisons. Detailed S values are shown in Table S8, S9, S10, S11, and S12, along with analysis of turnover and operation counts.

10 Weighted Parsimony

We can extend the infinite sites algorithm to solve a weighted parsimony problem in which 2-breakpoint rearrangements, 3-breakpoint rearrangements, duplications (with arbitrary numbers of bivalent breaks) and speciations can have different costs and the goal is to find an evolutionary history with minimal total cost for the operations. First, it can be shown that since we have exact distances there can be no tradeoffs between speciations and other operations, or duplications and other operations, as the reverse evolution procedure used by the infinite sites algorithm already produces the minimum number required for each of these regardless of what other operations are employed. The only tradeoff we can make is between 2-breakpoint and 3-breakpoint rearrangements. If 3-breakpoint rearrangements cost less than twice what 2-breakpoint rearrangements cost, then the infinite sites method as we have defined it will produce a minimum cost solution. This is because the only possible tradeoff between 2-breakpoint rearrangements and 3-breakpoint rearrangements occurs with the decision of how often to combine elbows. Two elbows would otherwise become two 2-breakpoint rearrangements. If a single 3-breakpoint rearrangement is cheaper than two 2-breakpoint rearrangements, we will want to combine as many pairs of elbows as possible, which is what the reverse evolution method does.

On the other hand, if a single 3-breakpoint rearrangement is more expensive than two 2-breakpoint rearrangements, then we will never want to combine elbows. It is easy to modify the reverse evolution method not to combine elbows, but rather always to make a single 2-breakpoint operation for each elbow, by skipping the Edmonds matching step. Other than this, the algorithm remains the same. This modified infinite sites algorithm solves the weighted parsimony problem for cases when a single 3-breakpoint rearrangement is as expensive or more expensive than two 2-breakpoint rearrangements. We use this algorithm in our analysis of the evolution of chromosome X.

References

Bansal, M. S., Burleigh, J. G., Eulenstein, O., and Wehe, A., 2007. Heuristics for the gene-duplication problem: A $\Theta(n)$ speed-up for the local search. In *RECOMB*, pages 238–252.

Chen, K., Durand, D., and Farach-Colton, M., 2000. NOTUNG: a program for dating gene duplications and optimizing gene family trees. *J Comput Biol*, **7**(3-4):429–447.

Durand, D., Halldorsson, B. V., and Vernot, B., 2006. A hybrid micro-macroevolutionary approach to gene tree reconstruction. *J Comput Biol*, **13**(2):320–335.

Edmonds, J., 1965. Paths, trees, and flowers. *Canad J Math*, **17**:449–467.

Fitch, W. M., 1970. Distinguishing homologous from analogous proteins. *Syst Zool*, **19**(2):99–113.

Fitch, W. M., 2000. Homology a personal view on some of the problems. *Trends Genet*, **16**(5):227–231.

Goodman, M., Czelusniak, J., Moore, G. W., Romero-Herrera, A. E., and Matsuda, G., 1979. Fitting the gene lineage into its species lineage: a parsimony strategy illustrated by cladograms constructed from globin sequences. *Syst Zool*, **28**(2):132–163.

Guigo, R., Muchnik, I., and Smith, T., 1996. Reconstruction of ancient molecular phylogeny. *Mol Phylogenet Evol*, **6**(2):189–213.

Hannenhalli, S. and Pevzner, P. A., 1995. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *STOC*, pages 178–189.

Howe, K., Bateman, A., and Durbin, R., 2002. QuickTree: building huge Neighbour-Joining trees of protein sequences. *Bioinformatics*, **18**(11):1546–1547.

Kent, W. J., Baertsch, R., Hinrichs, A., Miller, W., and Haussler, D., 2003. Evolution's cauldron: Duplication, deletion, and rearrangement in the mouse and human genomes. *PNAS*, **100**(20):11484–11489.

Ma, B., Li, M., and Zhang, L., 2000. From gene trees to species trees. *SIAM J Comput*, **30**(3):729–752.

Ma, J., Ratan, A., Raney, B. J., Suh, B. B., Zhang, L., Miller, W., and Haussler, D., 2008. DUPCAR: Reconstructing contiguous ancestral regions with duplications. *J Comput Biol*, . (in press).

Ovcharenko, I., Loots, G. G., Hardison, R. C., Miller, W., and Stubbs, L., 2004. zPicture: dynamic alignment and visualization tool for analyzing conservation profiles. *Genome Res*, **14**(3):472–477.

Rasmussen, M. D. and Kellis, M., 2007. Accurate gene-tree reconstruction by learning gene- and species-specific substitution rates across multiple complete genomes. *Genome Res*, **17**(12):1932–1942.

Saitou, N. and Nei, M., 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol*, **4**(4):406–425.

Sankoff, D. and Trinh, P., 2005. Chromosomal Breakpoint Reuse in Genome Sequence Rearrangement. *J Comput Biol*, **12**(6):812–821.

Schwartz, S., Zhang, Z., Frazer, K. A., Smit, A., Riemer, C., Bouck, J., Gibbs, R., Hardison, R., and Miller, W., 2000. PipMaker–a web server for aligning two genomic DNA sequences. *Genome Res*, **10**(4):577–586.

Studier, J. and Keppler, K., 1988. A note on the neighbor-joining algorithm of saitou and nei. *Mol Biol Evol*, **5**:729–731.

Waterman, M. S., Smith, T. F., Singh, M., and Beyer, W. A., 1977. Additive evolutionary trees. *J Theor Biol*, **64**(2):199–213.

Yancopoulos, S., Attie, O., and Friedberg, R., 2005. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, **21**(16):3340–3346.

Zarekskii, K., 1965. Construction of a tree on the basis of a set of distances between its leaves. *Upekki Math Nauk*, **20**:90–92.

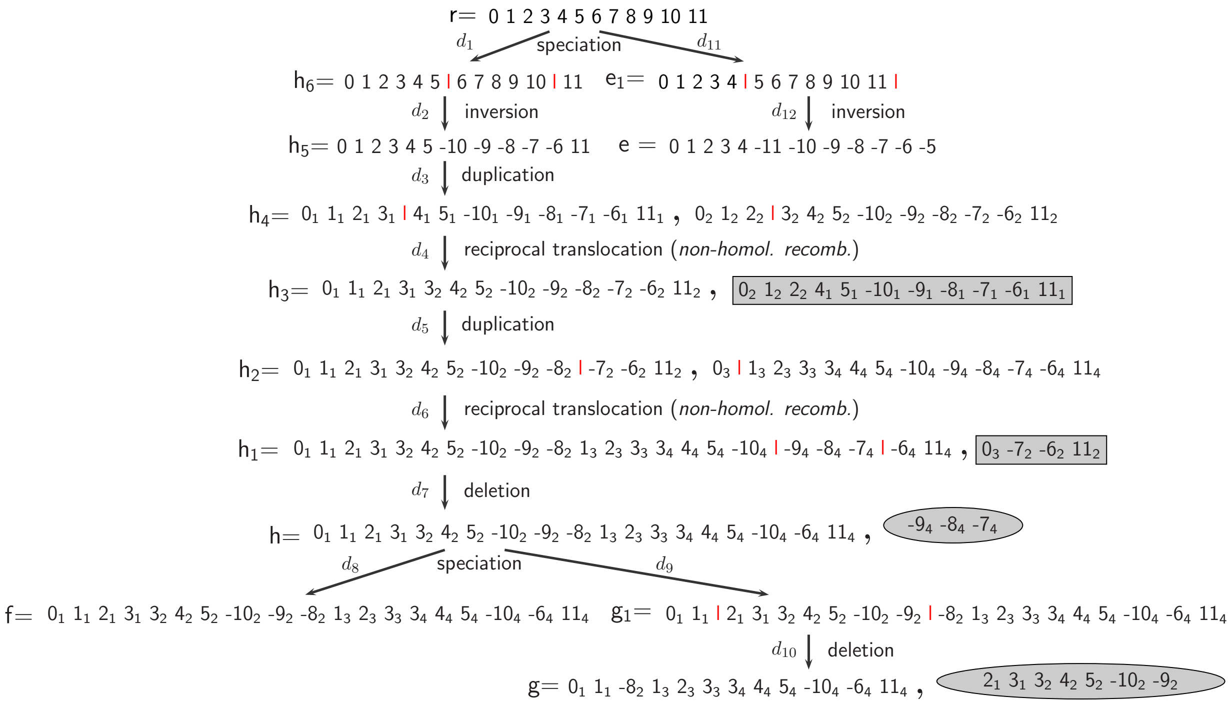

$r = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11$
d_1
speciation
d_{11}
$h_6 = 0\ 1\ 2\ 3\ 4\ 5 \mid 6\ 7\ 8\ 9\ 10 \mid 11$
$e_1 = 0\ 1\ 2\ 3\ 4 \mid 5\ 6\ 7\ 8\ 9\ 10\ 11 \mid$
d_2
inversion
d_{12}
inversion
$h_5 = 0\ 1\ 2\ 3\ 4\ 5\ -10\ -9\ -8\ -7\ -6\ 11$
$e = 0\ 1\ 2\ 3\ 4\ -11\ -10\ -9\ -8\ -7\ -6\ -5$
d_3
duplication
$h_4 = 0_1\ 1_1\ 2_1\ 3_1 \mid 4_1\ 5_1\ -10_1\ -9_1\ -8_1\ -7_1\ -6_1\ 11_1,\ 0_2\ 1_2\ 2_2 \mid 3_2\ 4_2\ 5_2\ -10_2\ -9_2\ -8_2\ -7_2\ -6_2\ 11_2$
d_4
reciprocal translocation (*non-homol. recomb.*)
$h_3 = 0_1\ 1_1\ 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2\ -8_2\ -7_2\ -6_2\ 11_2,$
$0_2\ 1_2\ 2_2\ 4_1\ 5_1\ -10_1\ -9_1\ -8_1\ -7_1\ -6_1\ 11_1$
d_5
duplication
$h_2 = 0_1\ 1_1\ 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2\ -8_2 \mid -7_2\ -6_2\ 11_2,\ 0_3 \mid 1_3\ 2_3\ 3_3\ 3_4\ 4_4\ 5_4\ -10_4\ -9_4\ -8_4\ -7_4\ -6_4\ 11_4$
d_6
reciprocal translocation (*non-homol. recomb.*)
$h_1 = 0_1\ 1_1\ 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2\ -8_2\ 1_3\ 2_3\ 3_3\ 3_4\ 4_4\ 5_4\ -10_4 \mid -9_4\ -8_4\ -7_4 \mid -6_4\ 11_4,$
$0_3\ -7_2\ -6_2\ 11_2$
d_7
deletion
$h = 0_1\ 1_1\ 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2\ -8_2\ 1_3\ 2_3\ 3_3\ 3_4\ 4_4\ 5_4\ -10_4\ -6_4\ 11_4,$
$-9_4\ -8_4\ -7_4$
d_8
speciation
d_9
$f = 0_1\ 1_1\ 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2\ -8_2\ 1_3\ 2_3\ 3_3\ 3_4\ 4_4\ 5_4\ -10_4\ -6_4\ 11_4$
$g_1 = 0_1\ 1_1 \mid 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2 \mid -8_2\ 1_3\ 2_3\ 3_3\ 3_4\ 4_4\ 5_4\ -10_4\ -6_4\ 11_4$
d_{10}
deletion
$g = 0_1\ 1_1\ -8_2\ 1_3\ 2_3\ 3_3\ 3_4\ 4_4\ 5_4\ -10_4\ -6_4\ 11_4,$
$2_1\ 3_1\ 3_2\ 4_2\ 5_2\ -10_2\ -9_2$

Figure S1: This example derivation tree will be used to illustrate the algorithm. It is the same as the example used in the main text. Here explicit distances are denoted on the branches.

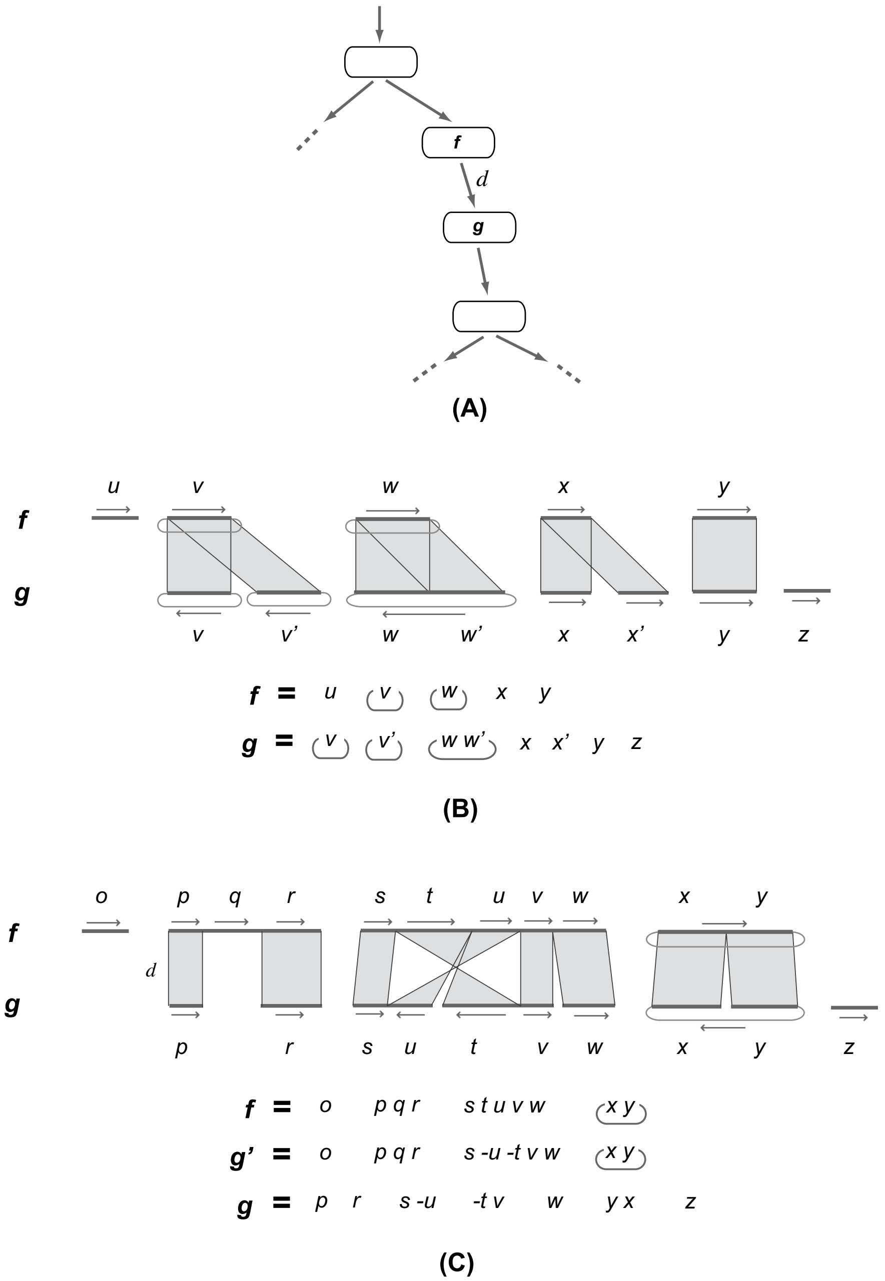

f
d
g
(A)
u
v
w
x
y
f
g
v
v'
w
w'
x
x'
y
z
f = u (v) (w) x y
g = (v) (v') (w w') x x' y z
(B)
o
p
q
r
s
t
u
v
w
x
y
f
d
g
p
r
s
u
t
v
w
x
y
z
f = o p q r s t u v w (x y)
g' = o p q r s -u -t v w (x y)
g = p r s -u -t v w y x z
(C)

Figure S2: (A) Along a species path, the child genome g results from the application of a rearrangement or duplication operation to the parent genome f followed by some chromosome gains and losses. The evolutionary distance between homologous sites in f and g is d. (B) Detail showing a case where f derives g by duplication followed by gain of chromosome z, loss of one copy of chromosome y, and loss of both copies of chromosome u. Homologous sites are shown by the grey shading, direction of the DNA segments by the arrows, and rings are indicated by circular connection of the ends. Chromosomes v and x are duplicated separately, while chromosome w is duplicated in tandem. (C) A case where f derives g with missing data. Initially, f derives a genome g' by a rearrangement (an inversion of segment tu) followed by loss of chromosome o and gain of chromosome z. In the last step of the derivation g' becomes g through the introduction of missing data consisting of a missing segment q and missing adjacencies between the inverted segments t and u, between v and w and between x and y. The latter turns a ring xy (same as the ring yx) into the contig yx.

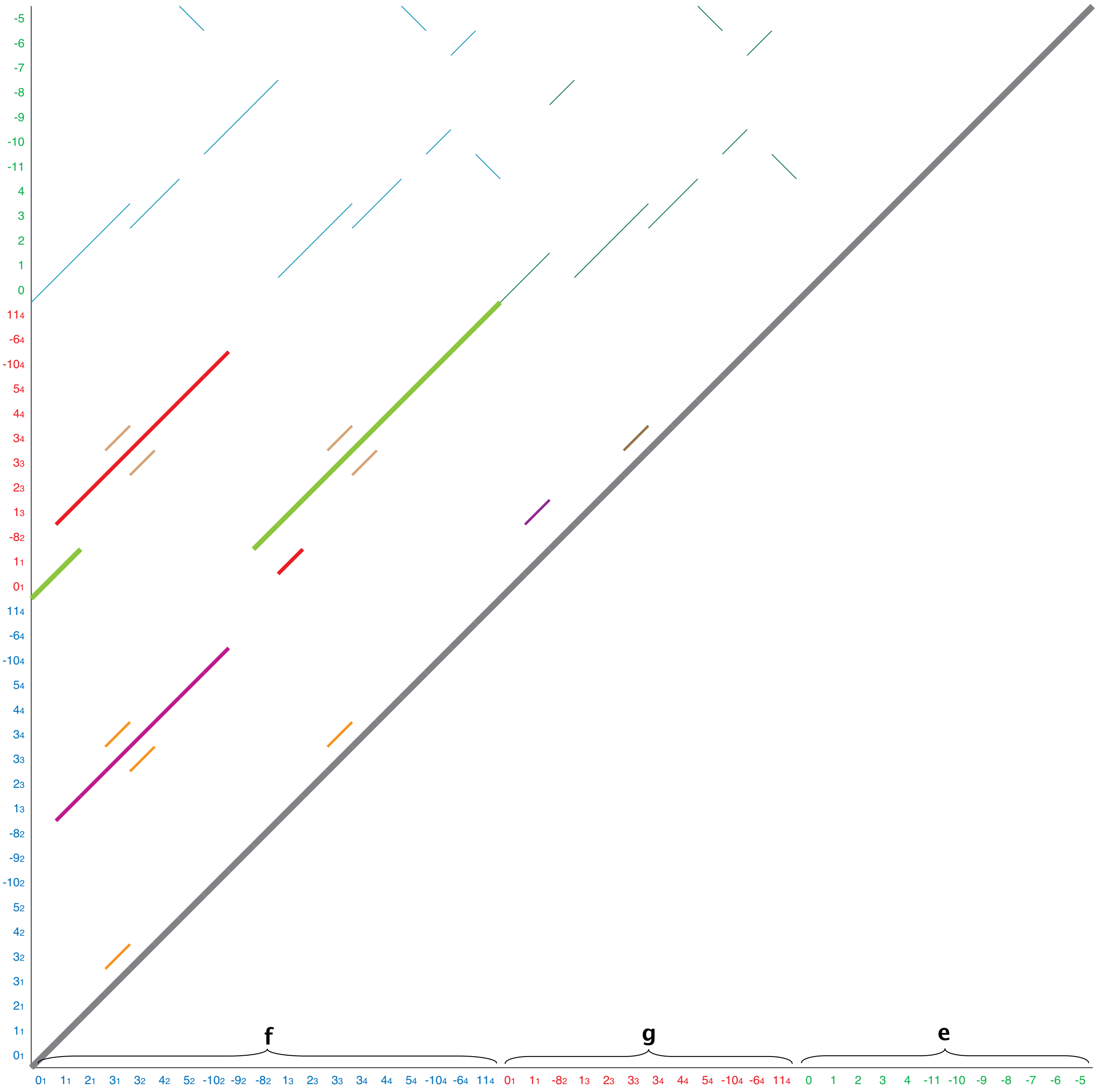


Figure S3: The dot plot for the evolutionary distance function D for the genome sequence $G = \{f, g, e\}$ from Figure 1 in main text. Only the upper triangular region is shown; the other is symmetric. The different distances present in the local alignments in the dot plot are represented by different line widths and colors, with wider lines representing shorter distances. The thick black diagonal represents identical sites at distance 0. For example, the purple line on lower left side represents the local alignment between two segments in f, and the corresponding sites all have distance $2(d_5 + d_6 + d_7 + d_8)$. The six thin dark green lines on upper right side represent local alignments between segments of g and e, and the corresponding sites all have distance $d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 + d_9 + d_{10} + d_{11} + d_{12}$.

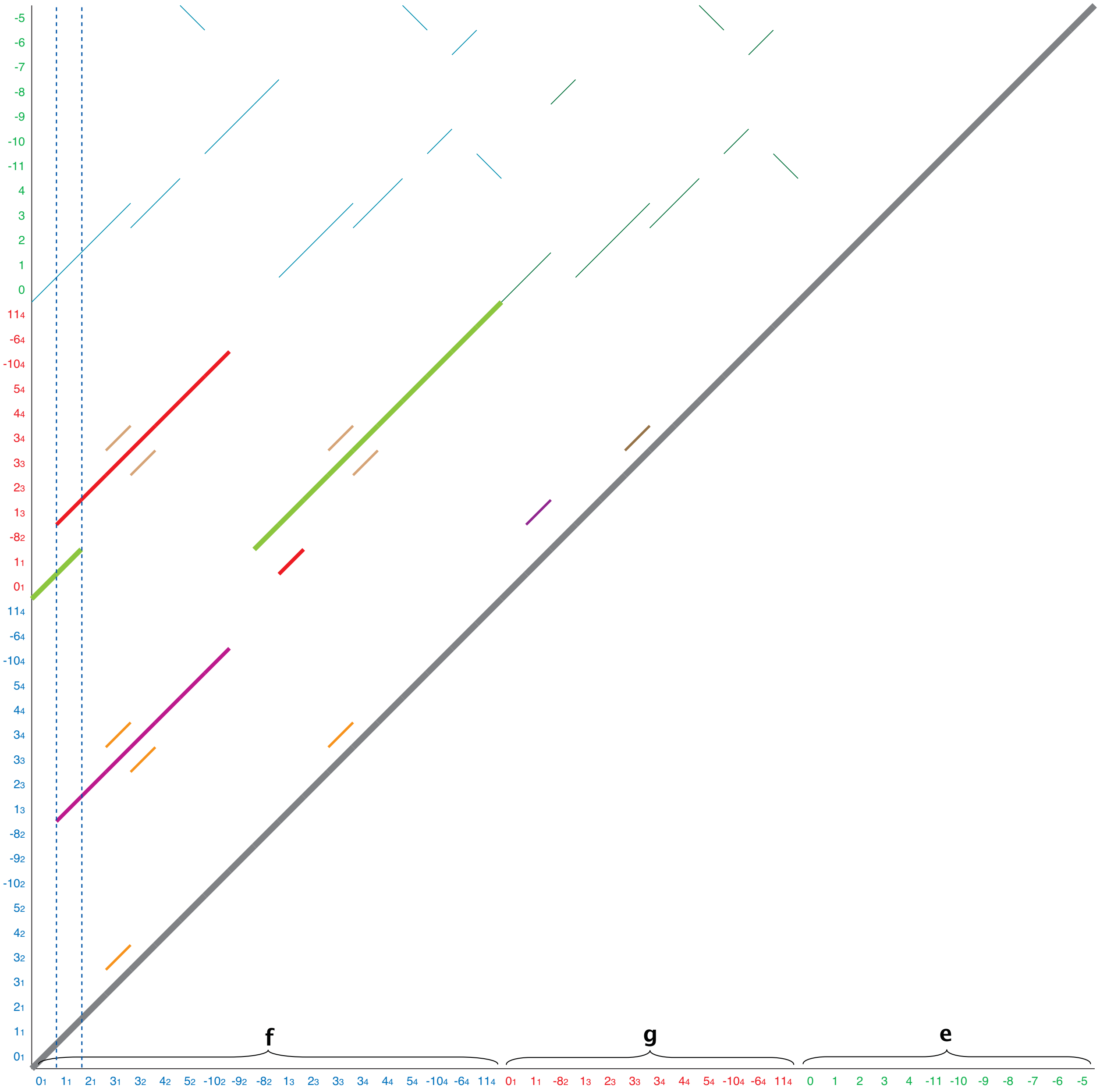


Figure S4: Illustration shows how atoms are determined from the dot plot. Atom instance $f[0]$ is the initial region of f up to the first vertical blue dotted line, and is homologous to atom instance $g[0_1]$ and $e[0]$ as indicated by the segments parallel to the main diagonal. Atom instance $f[1_1]$ is the region from the first vertical blue line to the second vertical blue line, and is homologous to atom instances $f[1_3]$, $g[1_1]$, $g[1_3]$, and $e[1]$, as indicated. This process continues as we sweep a vertical line across the dot plot.

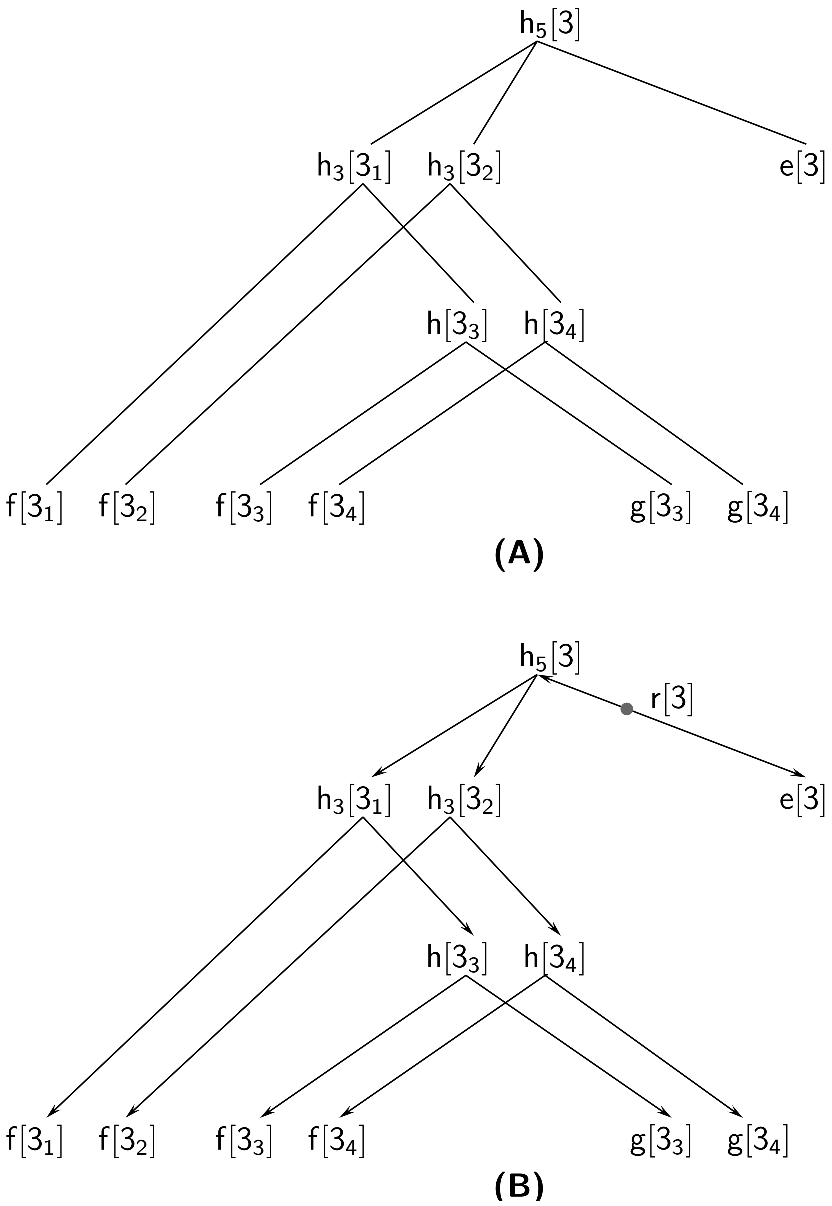


Figure S5: (A) The undirected atom tree T_3 for instances of atom 3, given the leaf genomes, for the derivation shown in Figure S1. Note that the atom instance $r[3]$ from the root genome r is not present as a node in the undirected atom tree, because it is not at a branching node of this tree. (B) The directed tree T_k for atom 3. The root atom instance $r[3]$ is shown between $h_5[3]$ and $e[3]$.

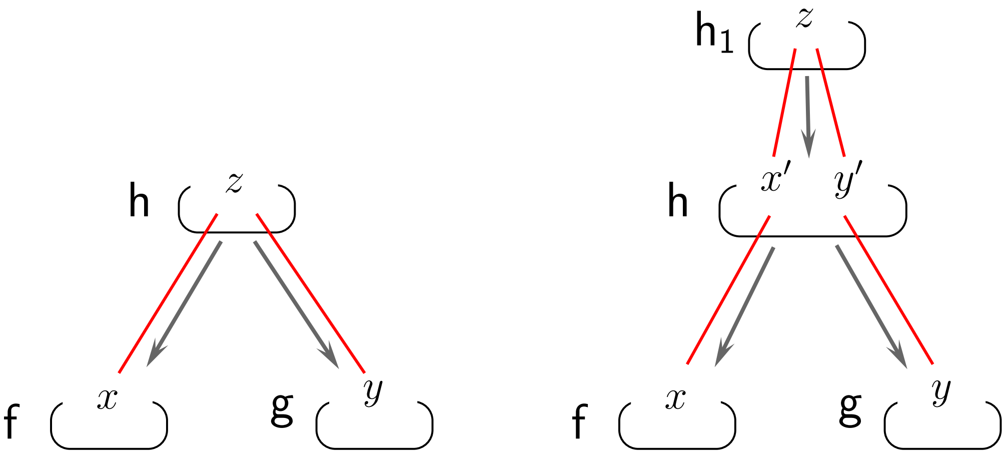


Figure S6: Fitch orthologs and paralogs. Red lines are part of the atom tree, while grey lines come from the species tree. (A) The last common ancestor of x and y is z, which corresponds to a speciation event at genome h, so x is orthologous to y. (B) The last common ancestor z of x and y is duplicated to produce the separate ancestors x' of x and y' of y in h, so x is paralogous to y.

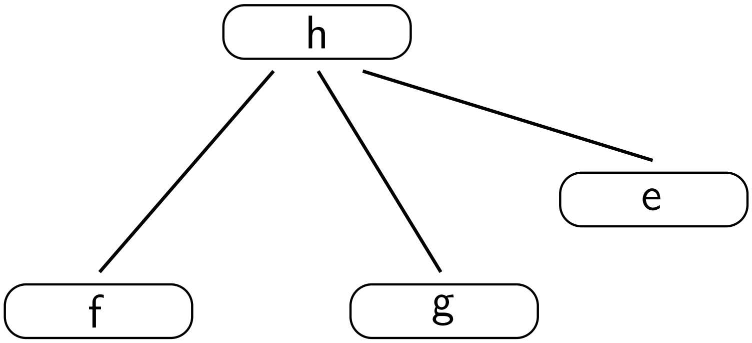


Figure S7: The undirected species tree given the evolutionary tree from Figure S1.

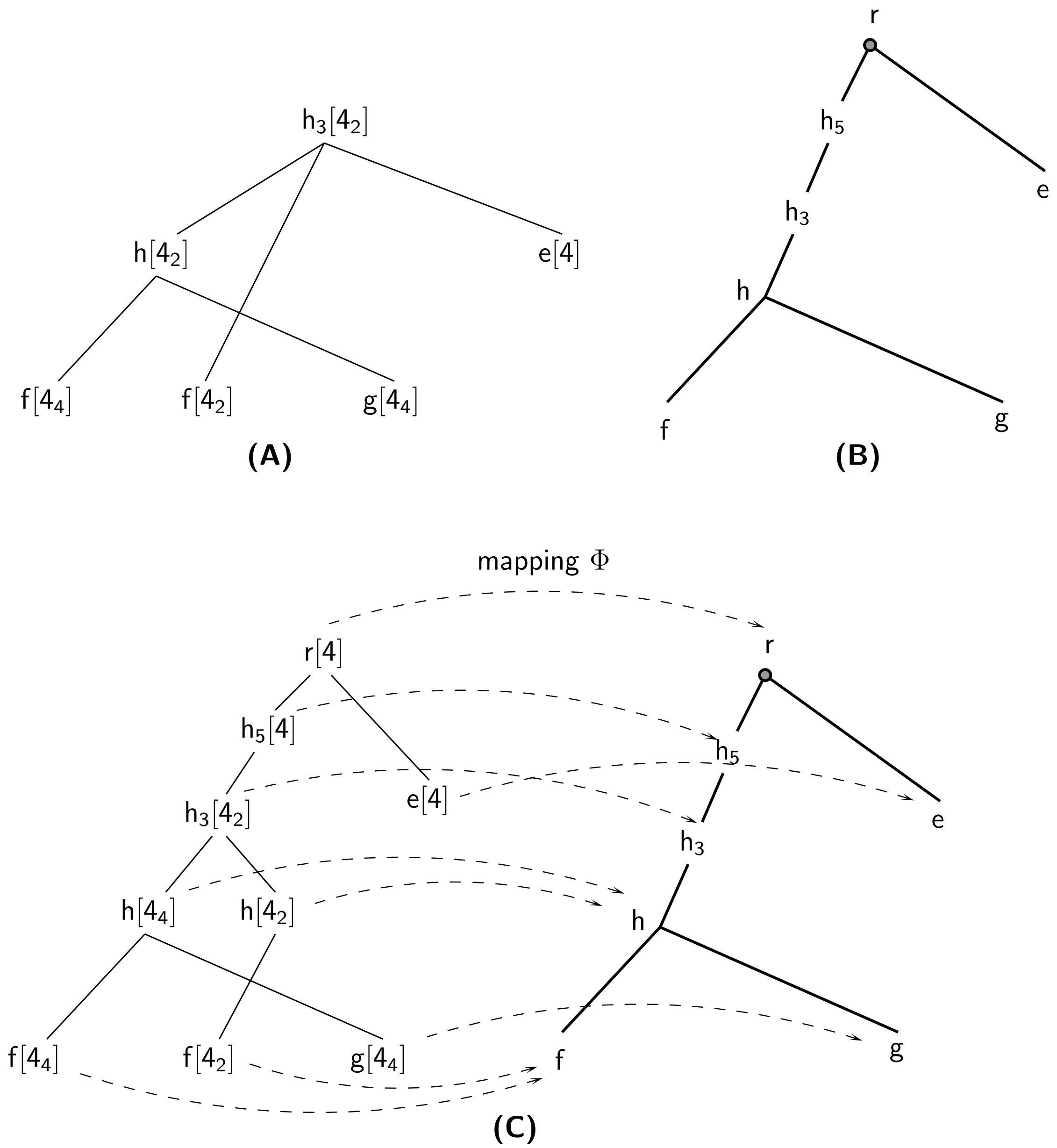


Figure S8: (A) The undirected atom tree T_4 for atom 4. (B) The duplication tree for the evolutionary history from Figure S1. The edges between genomes in the duplication tree are (implicitly) oriented downward to indicate the direction of ancestry between the genomes. If the outgroup genome e were not present, then the undirected species tree would have included only f and g, but the duplication tree would still have included genomes h_5, h_3, and h, with h_5 the root of the duplication tree, h representing the inferred root of the directed species tree, and h_5 and h_3 representing nodes for ancient duplications. (C) The reconciliation of the undirected atom tree T_4 with the duplication tree defines the directed, augmented atom tree $\mathbf{T}_4$ and the mapping Φ (dotted arrows) of its nodes to the nodes of the duplication tree. Note that in addition to the root $r[4]$, two other nodes $h_5[4]$ and $h[4_2]$ that were not nodes of the undirected atom tree T_4 are added to the directed, augmented tree. These are inferred to exist during reconciliation.

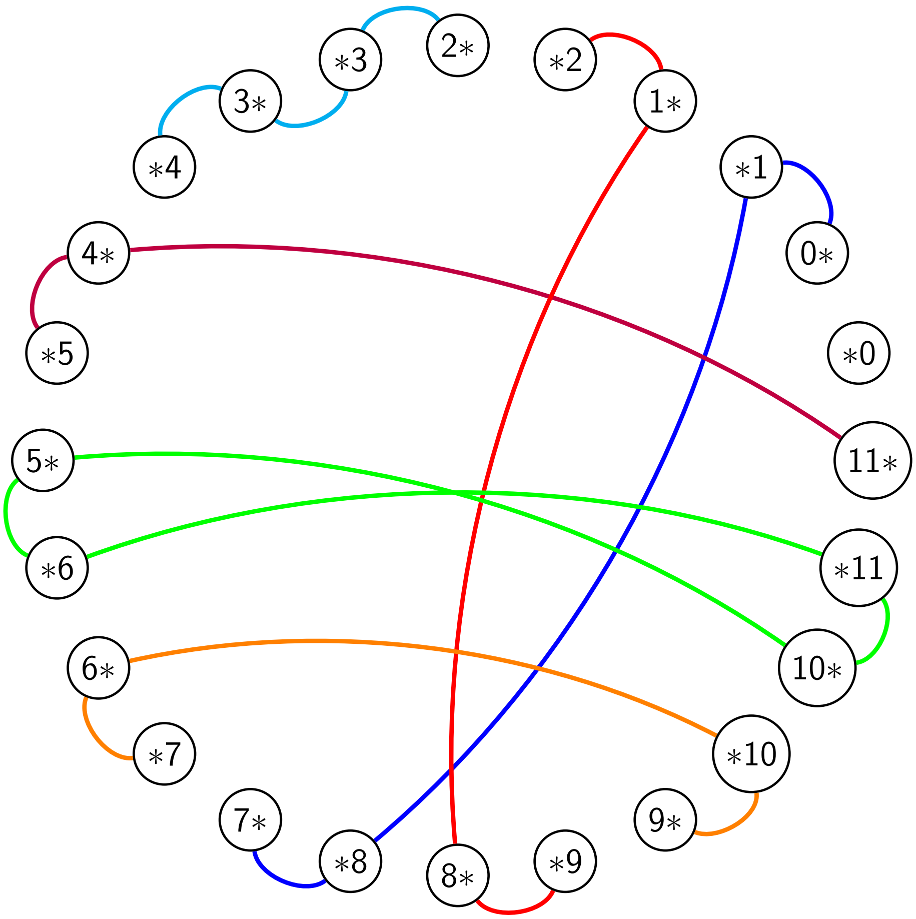


Figure S9: The master breakpoint graph for the evolutionary history from Figure S1. Only quartets and their subgraphs appear as connected components, because the only operations that change adjacencies in this example are 2-breakpoint rearrangements. For example, the green edges form a quartet that represents the inversion operation of genome e_1, changing the chromosome by breaking adjacencies of types $(5*, *6)$ and $(10*, *11)$ and creating new adjacencies of types $(*6, *11)$ and $(5*, 10*)$ to form the genome e. The red edges form a subgraph of a quartet with adjacencies of types $(8*, *9)$, $(1*, *2)$ and $(1*, 8*)$, representing the deletion that broke adjacencies $(g_1[8]*, *g_1[9])$ and $(g_1[1]*, *g_1[2])$ and created the new adjacency $(g[1]*, g[8]*)$. Not observed is the other new adjacency $(*g[9], *g[2])$ that was created when the deleted segment was circularized. A quartet with a missing edge like this is called a *snake*.

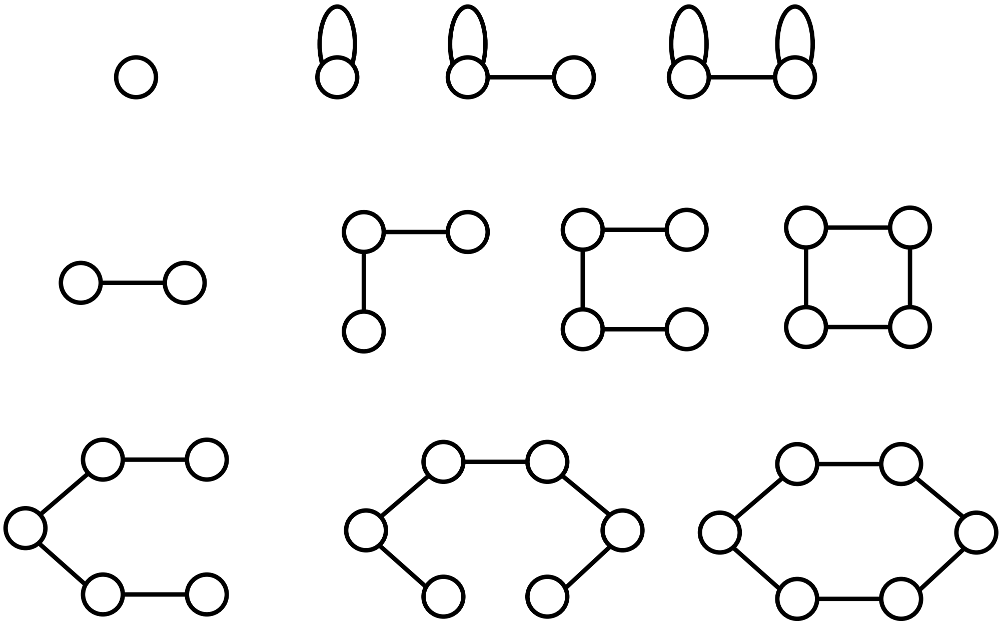
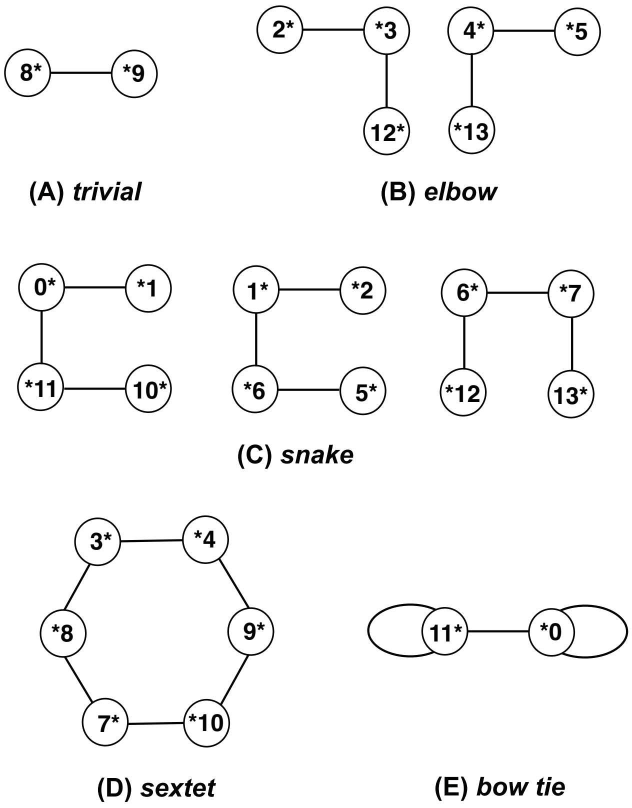


Figure S10: All possible connected subgraphs of a bowtie, quartet, or sextet in the master breakpoint graph.



(A) *trivial*

(B) *elbow*

(C) *snake*

(D) *sextet*

(E) *bow tie*

Figure S11: The master breakpoint graph for a problem with two leaf genomes f=(0 1 2 3 4 5 6 7 8 9 10 11) and g=(0 1 6 12 3 8 9 4 13 7 10 11 -11 -0). These chromosomes are circular, i.e. $f[11]$ is adjacent to $f[0]$ and $g[-0]$ is adjacent to $g[0]$. Pairs of duplicated atom instances in genome g are all at some fixed distance d and homologous atom instances between f and g are at some distance $d' > d$. All eight modules in the master breakpoint graph of f and g are shown above. This master breakpoint graph provides examples of elbow components, a trivial component, a bow tie, and a sextet.

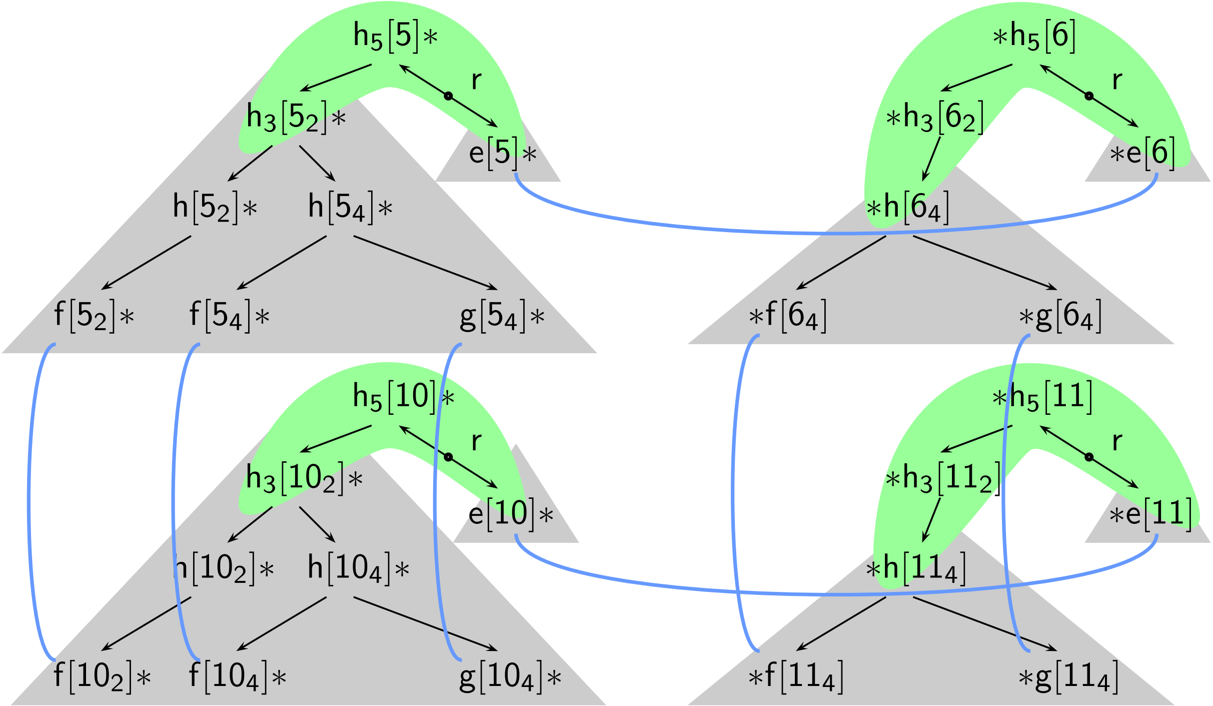


Figure S12: The partition of $\mathbf{T}_5$ into two iso-adjacency subtrees for the end $5*$ (gray triangles) connected by a tether (shaded in green) is depicted in the upper left. The adjacencies of type $(5*, 10*)$ (blue vertical lines) define the iso-adjacency subtree $\mathbf{T}_{(5*,10*)}$, depicted by the large gray triangle, and the adjacency type $(5*, *6)$ (blue horizontal line) defines the iso-adjacency subtree $\mathbf{T}_{(5*,*6)}$, depicted by the small gray triangle and including only the node $g[5_4]*$. The tether consists of the nodes $h_5[5]*$, the root r, and the edges connecting these nodes to each other and to the two iso-adjacency subtrees. Here the tether consists only of a mian path without side branches. The partitions of the other atoms involved in the module that contains $5*$, the purple quartet in Figure S9, defined by the adjacencies that are changed in this module, are depicted in the other quadrants of the figure. The adjacencies $(5*, *6)$ and $(10*, *11)$ (blue horizontal lines) represent one possible configuration for the module, and $(5*, 10*)$ and $(*6, *11)$ (blue vertical lines) represent the other. The entire structure depicted is the adjacency graph of the module.

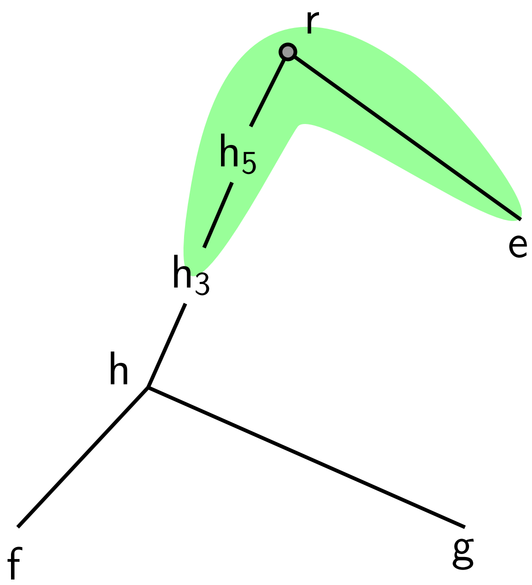


Figure S13: The switchpoint range for the module in Figure S12. The switchpoint range, which is shaded in green, in the intersection of tethers shown in Figure S12

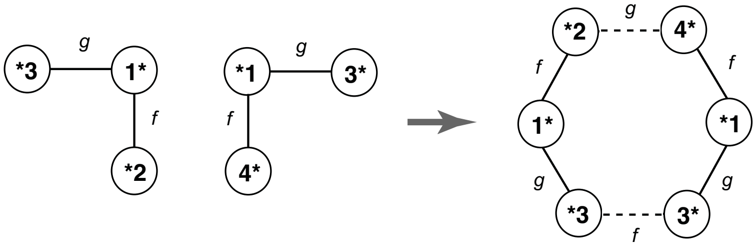


Figure S14: The master breakpoint graph for a problem with two leaf genomes f = 1 2 4 and g = 1 3. Both f and g are circular chromosomes. Since the edges must alternate between adjacencies in g and adjacencies in f, there is only one way to join the elbows into a compound rearrangement module. A similar set of pairable elbows appears in part (B) of the example from Figure S11. Thus, even though that example has 7 nontrivial modules, the genomes f and g can be derived from a common ancestor with only 6 operations: two 3-breakpoint rearrangements, three 2-breakpoint rearrangements, and one single breakpoint reverse tandem duplication.

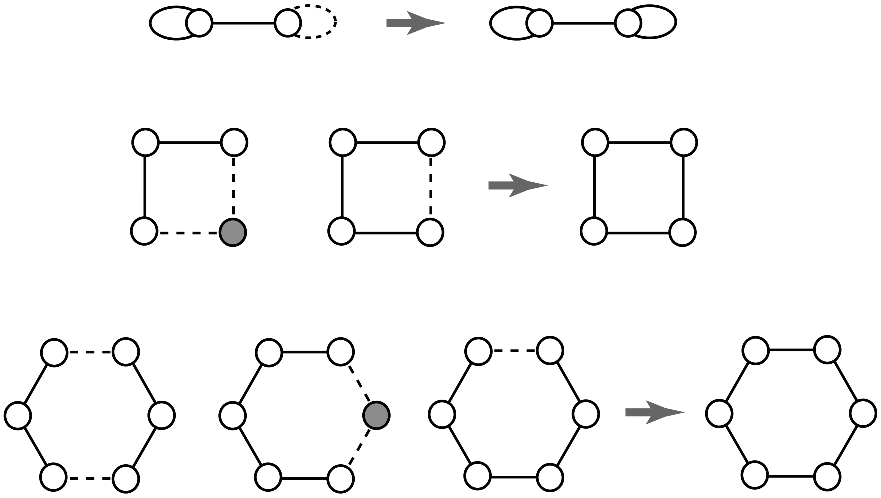


Figure S15: Module-level inferred adjacencies complete modules to be a bow tie, a quartet, or a sextet.

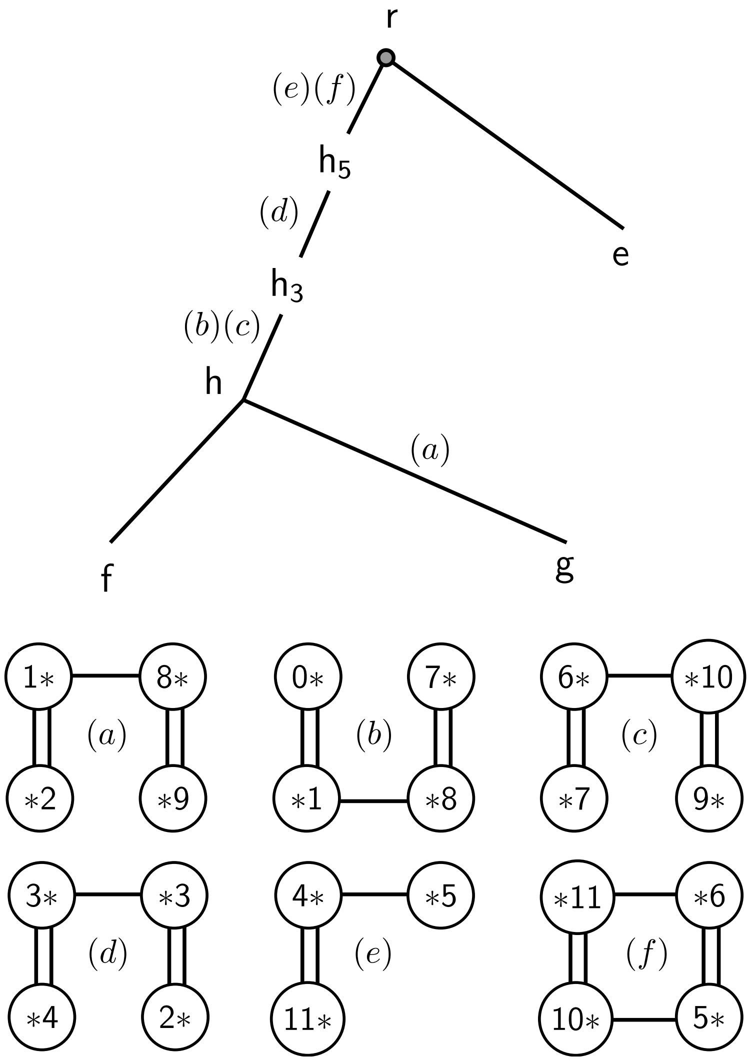
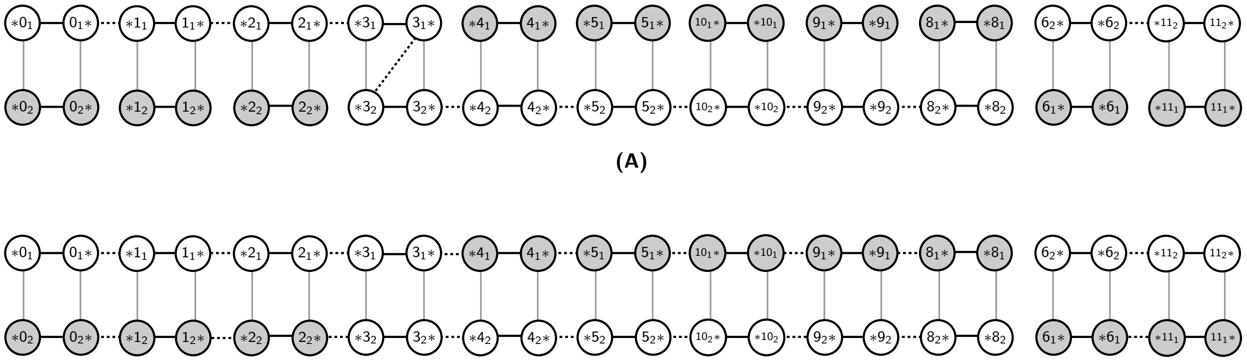


Figure S16: Possible scheduling for the rearrangement operations to edges in the duplication tree in the example of Figure S1. For example, we schedule rearrangement (e) and (f) on one side of the root leading toward h_5. Modules (e) to (f) are polarized accordingly. The edge $=$ denotes a primitive adjacency and $-$ denotes a derived adjacency.



(A)

(B)

Figure S17: Here we depict the sibling graph constructed when inferring the parent genome h_5 from the child genome h_3 after the second duplication in the history shown in Figure S19. There is only one operation scheduled, the operation (d) in Figure S16. (A) The sibling graph with atom edges shown in black, sibling edges shown in grey, and child adjacency edges as dotted lines. Parent adjacency edges are not shown here. Inferred siblings are shaded in grey, e.g. node 0_2. (B) The sibling graph as in (A), but with the dotted lines representing the parent adjacency edges, obtained by performing phases 1-3. In this case nothing happens in phase 1, as there are no "old" adjacencies inferred to have existed in the parent, which are not already present in the child. In the second phase we remove the "new" edge $(3*, *3)$ that is added by the operation (d). Finally we add edges for all inferred sib adjacencies. This creates the dotted edges as shown.

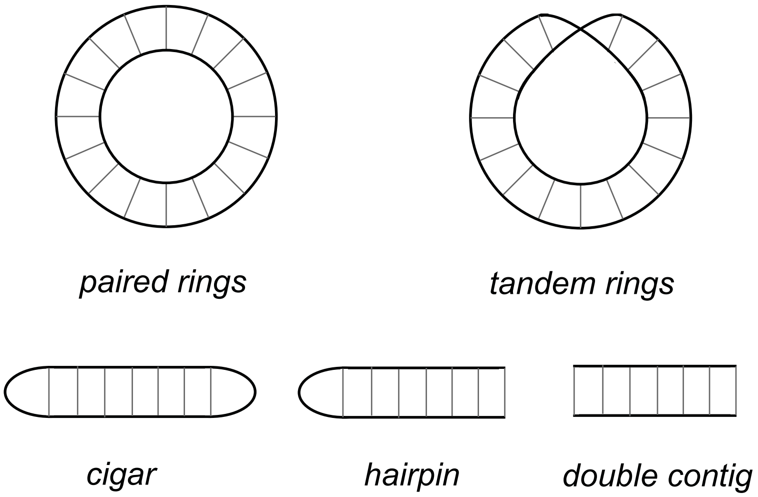


Figure S18: Abstract illustrations of the five possible types of connected components in the sibling graph formed by the atom, sibling and parent adjacency edges. For example, in the example in Figure S17 there are two double contigs.

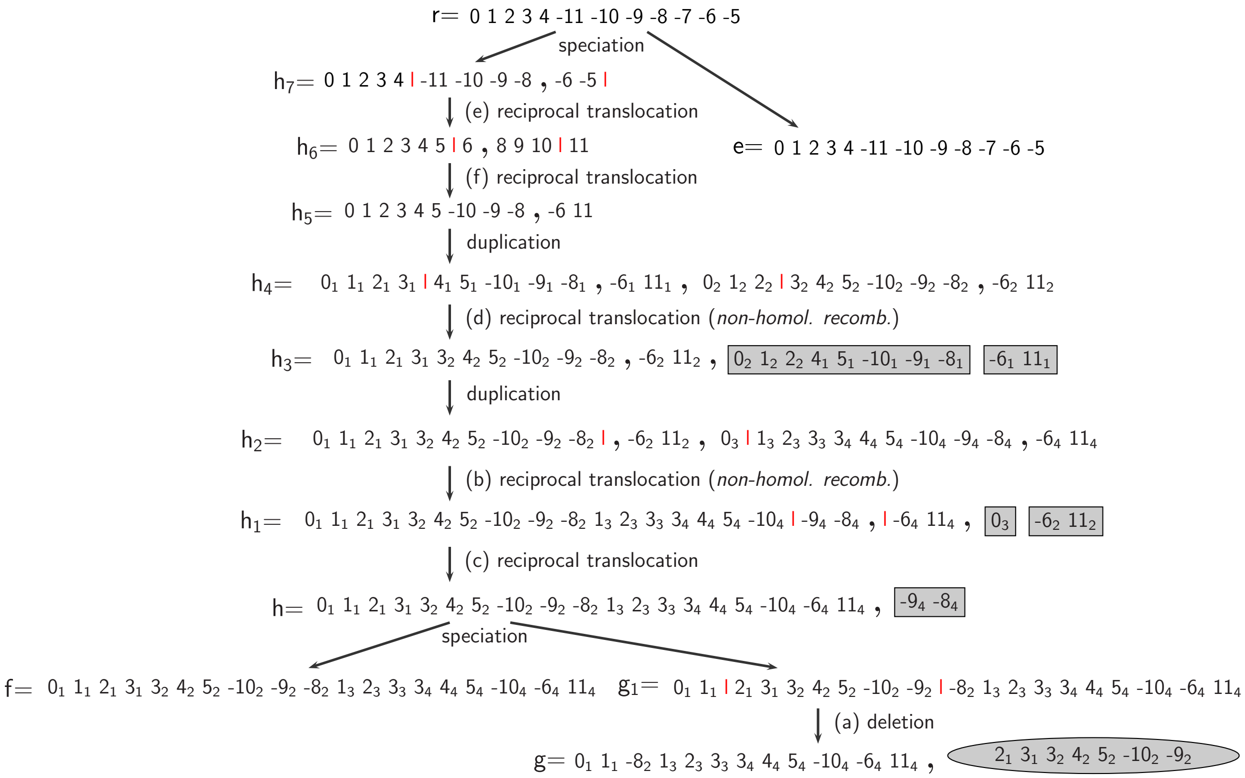

r= 0 1 2 3 4 -11 -10 -9 -8 -7 -6 -5
speciation
h7= 0 1 2 3 4 | -11 -10 -9 -8 , -6 -5 |
(e) reciprocal translocation
e= 0 1 2 3 4 -11 -10 -9 -8 -7 -6 -5
h6= 0 1 2 3 4 5 | 6 , 8 9 10 | 11
(f) reciprocal translocation
h5= 0 1 2 3 4 5 -10 -9 -8 , -6 11
duplication
h4= 01 11 21 31 | 41 51 -101 -91 -81 , -61 111 , 02 12 22 | 32 42 52 -102 -92 -82 , -62 112
(d) reciprocal translocation (non-homol. recomb.)
h3= 01 11 21 31 32 42 52 -102 -92 -82 , -62 112 , 02 12 22 41 51 -101 -91 -81 -61 111
duplication
h2= 01 11 21 31 32 42 52 -102 -92 -82 | , -62 112 , 03 | 13 23 33 34 44 54 -104 -94 -84 , -64 114
(b) reciprocal translocation (non-homol. recomb.)
h1= 01 11 21 31 32 42 52 -102 -92 -82 13 23 33 34 44 54 -104 | -94 -84 , | -64 114 , 03 -62 112
(c) reciprocal translocation
h= 01 11 21 31 32 42 52 -102 -92 -82 13 23 33 34 44 54 -104 -64 114 , -94 -84
speciation
f= 01 11 21 31 32 42 52 -102 -92 -82 13 23 33 34 44 54 -104 -64 114
g1= 01 11 | 21 31 32 42 52 -102 -92 | -82 13 23 33 34 44 54 -104 -64 114
(a) deletion
g= 01 11 -82 13 23 33 34 44 54 -104 -64 114 , 21 31 32 42 52 -102 -92

Figure S19: The solution of the example in Figure S1 based on the schedule in Figure S16. Deleted chromosomes are shown as shaded. Note that this solution is different from the original derivation because operation (e) is scheduled on the branch leading from r into the first duplication event h_5 rather than the branch leading to leaf genome e. The absence of atom 7 in some genomes also makes some operations appear different.

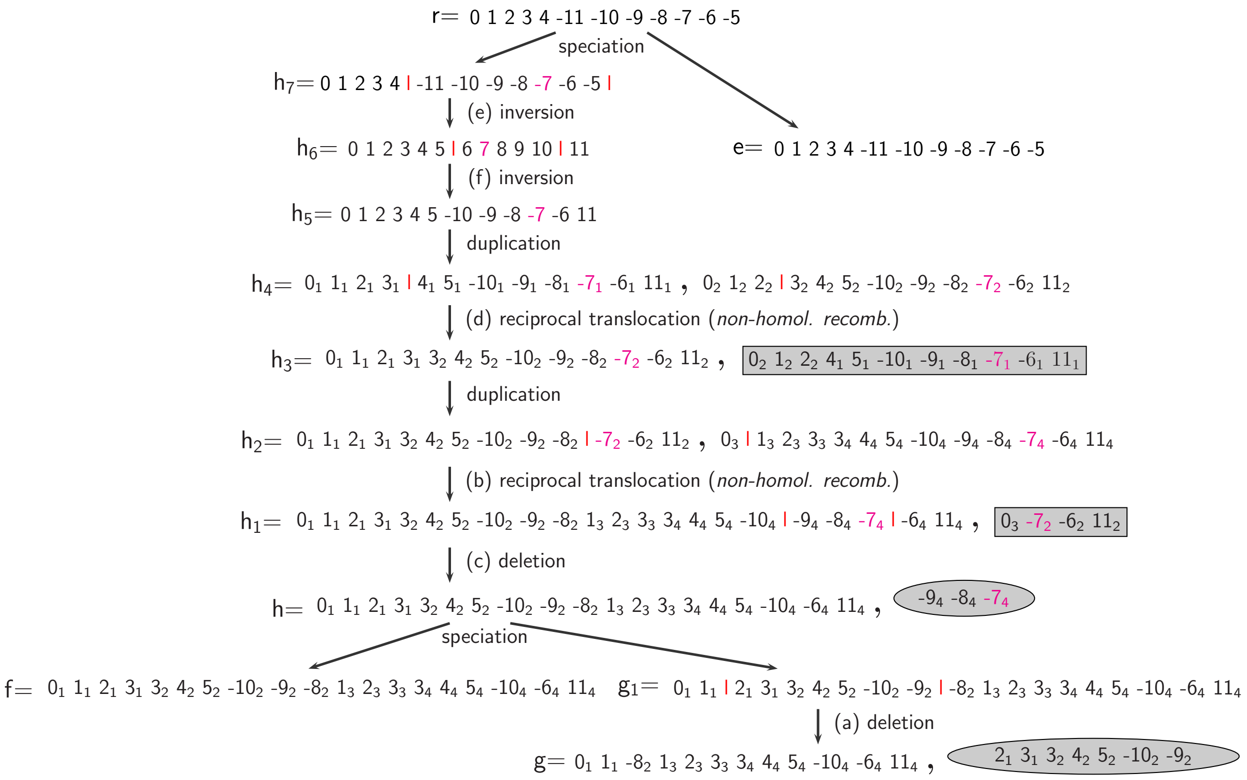

r= 0 1 2 3 4 -11 -10 -9 -8 -7 -6 -5
speciation
h_7= 0 1 2 3 4 | -11 -10 -9 -8 -7 -6 -5 |
(e) inversion
h_6= 0 1 2 3 4 5 | 6 7 8 9 10 | 11
e= 0 1 2 3 4 -11 -10 -9 -8 -7 -6 -5
(f) inversion
h_5= 0 1 2 3 4 5 -10 -9 -8 -7 -6 11
duplication
h_4= 0_1 1_1 2_1 3_1 | 4_1 5_1 -10_1 -9_1 -8_1 -7_1 -6_1 11_1 , 0_2 1_2 2_2 | 3_2 4_2 5_2 -10_2 -9_2 -8_2 -7_2 -6_2 11_2
(d) reciprocal translocation (*non-homol. recomb.*)
h_3= 0_1 1_1 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2 -8_2 -7_2 -6_2 11_2 , 0_2 1_2 2_2 4_1 5_1 -10_1 -9_1 -8_1 -7_1 -6_1 11_1
duplication
h_2= 0_1 1_1 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2 -8_2 | -7_2 -6_2 11_2 , 0_3 | 1_3 2_3 3_3 3_4 4_4 5_4 -10_4 -9_4 -8_4 -7_4 -6_4 11_4
(b) reciprocal translocation (*non-homol. recomb.*)
h_1= 0_1 1_1 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2 -8_2 1_3 2_3 3_3 3_4 4_4 5_4 -10_4 | -9_4 -8_4 -7_4 | -6_4 11_4 , 0_3 -7_2 -6_2 11_2
(c) deletion
h= 0_1 1_1 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2 -8_2 1_3 2_3 3_3 3_4 4_4 5_4 -10_4 -6_4 11_4 , -9_4 -8_4 -7_4
speciation
f= 0_1 1_1 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2 -8_2 1_3 2_3 3_3 3_4 4_4 5_4 -10_4 -6_4 11_4
g_1= 0_1 1_1 | 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2 | -8_2 1_3 2_3 3_3 3_4 4_4 5_4 -10_4 -6_4 11_4
(a) deletion
g= 0_1 1_1 -8_2 1_3 2_3 3_3 3_4 4_4 5_4 -10_4 -6_4 11_4 , 2_1 3_1 3_2 4_2 5_2 -10_2 -9_2

Figure S20: The result of fill in phase on the example in Figure S19. The filled in atom instances are shown in magenta. All are instances of atom 7.

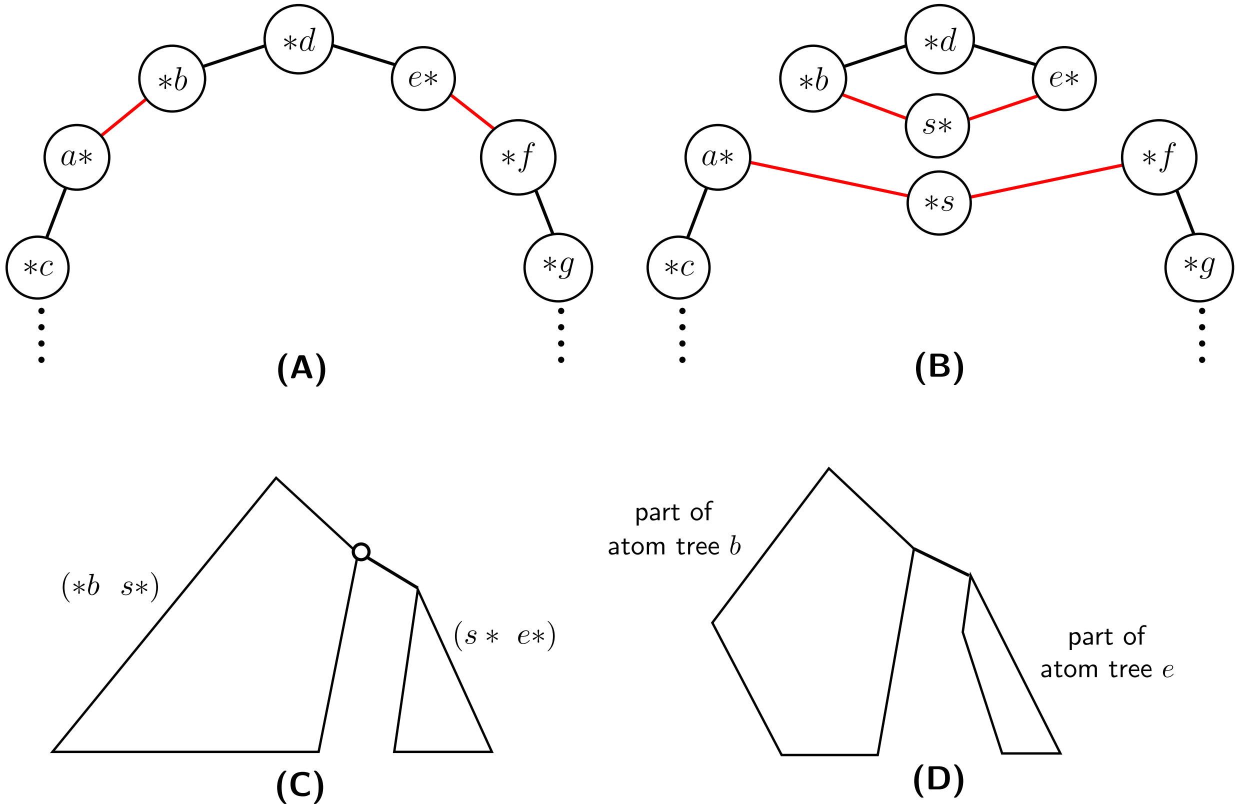


Figure S21: (A) A local bad connected component in the local breakpoint graph. Because the local breakpoint graph has degree at most 2, a local bad component is either a long chain or a long cycle. (B) We use engineered atom s to modify the bad component in (A) by breaking edge $(a*, *b)$ and $(e*, *f)$. This increases by 1 the number b of breakpoints represented in the local breakpoint graph, and it also increases the number c of cycles in this graph by 1. Therefore it leaves the rearrangement complexity $b-c$ for 2-breakpoint rearrangements unchanged (Yancopoulos *et al.*, 2005). The process of inserting engineered atoms will continue if the rest of the component containing node $*s$ is still a bad component. (C) This figure shows the situation when $(*b, s*)$ is an ancestral adjacency and $(s*, e*)$ is a derived adjacency in the quartet obtained from the procedure in (B). The small circle indicates the current node under consideration in the reconciled duplication tree. (D) Let $\mathsf{T}_{e*,derived}$ be the subtree of the augmented atom tree T_e representing nodes below the edge where the operation representing this quartet is scheduled to occur, and similarly, let $\mathsf{T}_{*b,ancestral}$ be the subtree of T_b outside of the nodes below the edge where the operation is scheduled to occur. The engineered atom s is inserted so that $s*$ is adjacent to $e*$ (and hence $*s$ is adjacent to $*f$) for all atom instances in $\mathsf{T}_{e*,derived}$ (where the adjacency used to be $(e*, *f)$) and $s*$ is adjacent to $*b$ (and hence $*s$ is adjacent to $a*$) for all atom instances in $\mathsf{T}_{*b,ancestral}$ (where the adjacency used to be $(a*, *b)$). This insures that the new adjacency configuration is consistent in all the reconstructed and leaf genomes. The augmented atom tree for new engineered atom s is constructed by combining $\mathsf{T}_{*b,ancestral}$ and $\mathsf{T}_{e*,derived}$. The shapes of $\mathsf{T}_{*b,ancestral}$ and $\mathsf{T}_{e*,derived}$ are different from the shapes of the left subtree and right subtree shown in (C) is because b and/or e both could be missing in parts of the trees.

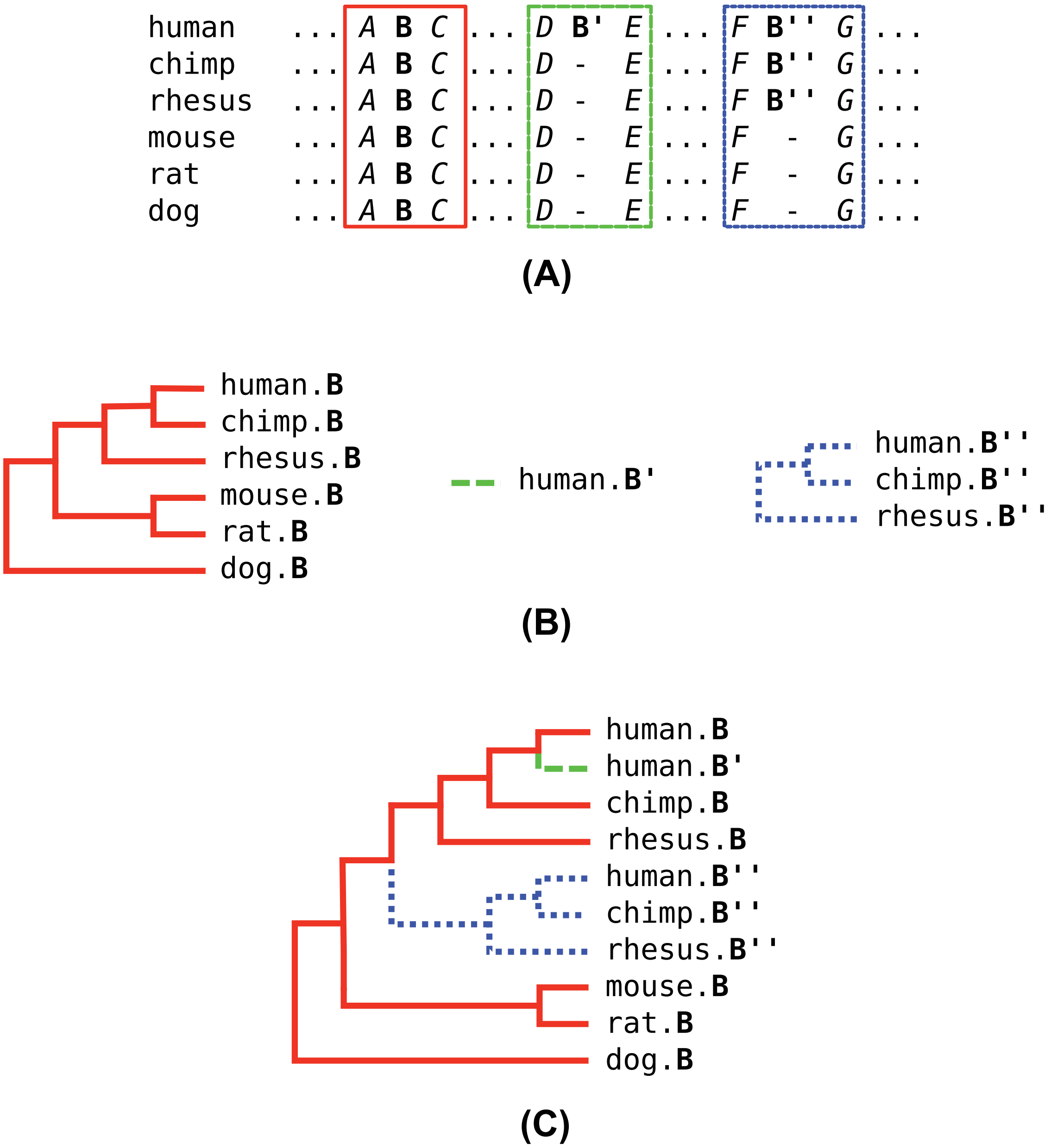

human ...A B C...D B' E...F B'' G...
chimp ...A B C...D - E...F B'' G...
rhesus ...A B C...D - E...F B'' G...
mouse ...A B C...D - E...F - G...
rat ...A B C...D - E...F - G...
dog ...A B C...D - E...F - G...
(A)
human.B
chimp.B
rhesus.B
mouse.B
rat.B
dog.B
human.B'
human.B''
chimp.B''
rhesus.B''
(B)
human.B
human.B'
chimp.B
rhesus.B
human.B''
chimp.B''
rhesus.B''
mouse.B
rat.B
dog.B
(C)

Figure S22: (A) Hypothetical multiple genome alignment with emphasis on atom B. (B) The three trees representing the AC, DE, and FG contexts, respectively. (C) The atom tree for atom B assembled together from the three context trees.

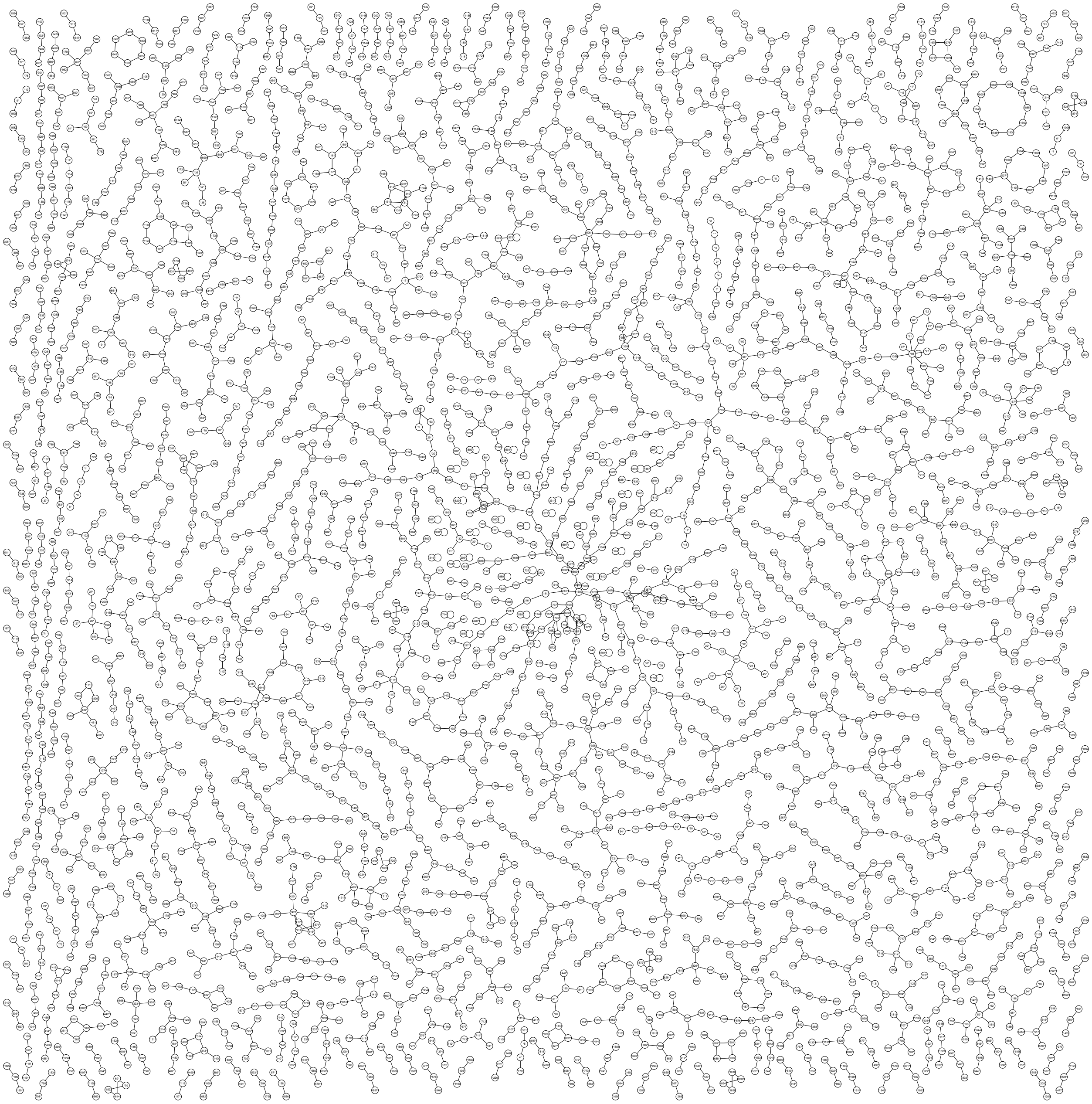
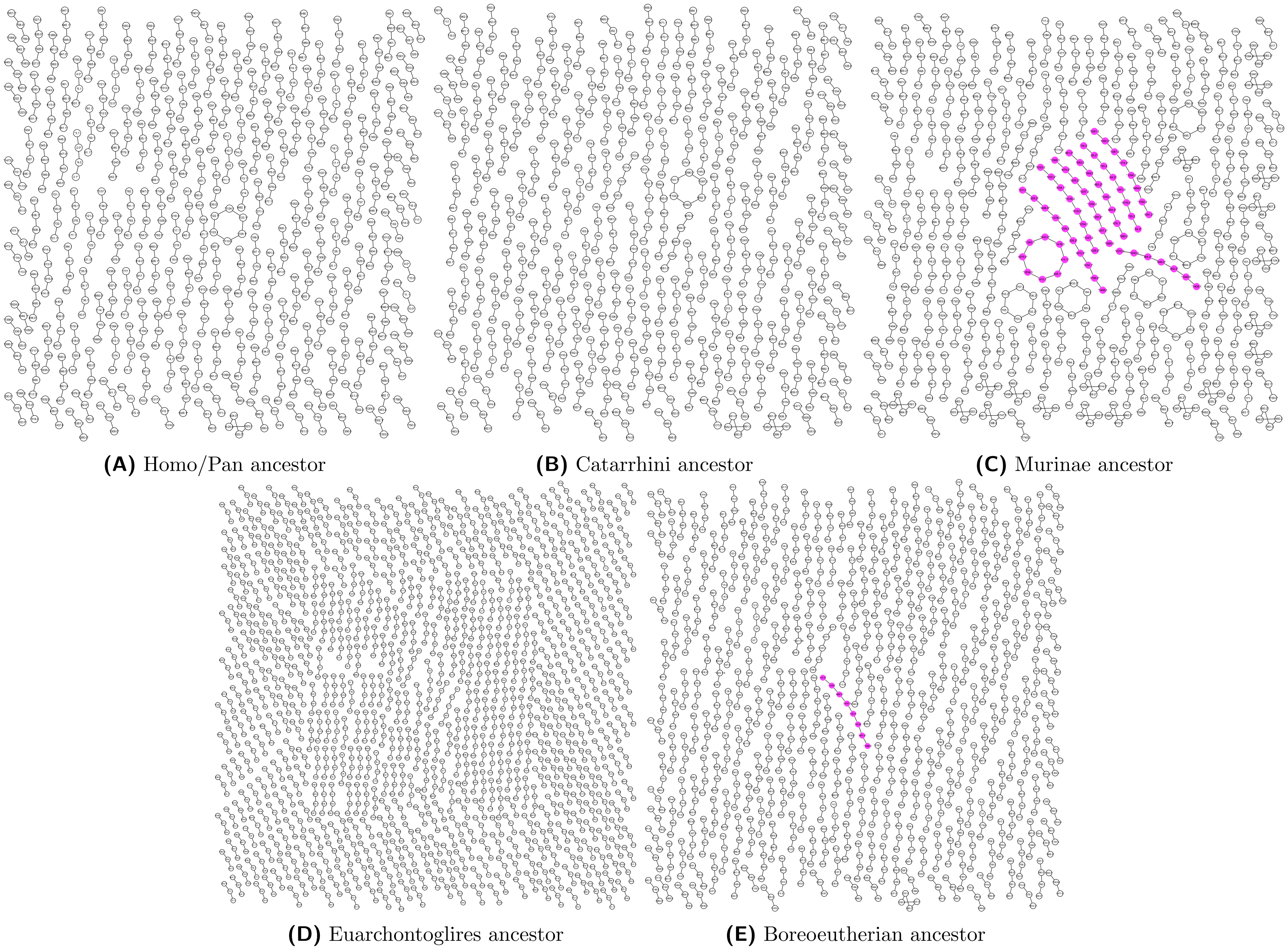


Figure S23: The master breakpoint graph of chrX based on 1,917 atoms from the human, chimp, macaque, mouse, rat, and dog genomes.



(A) Homo/Pan ancestor **(B)** Catarrhini ancestor **(C)** Murinae ancestor

(D) Euarchontoglires ancestor **(E)** Boreoeutherian ancestor

Figure S24: Local breakpoint graphs in speciation nodes before adding engineered atoms. The bad components are highlighted in light purple. These appear only in the Murinae ancestor and Boreoeutherian ancestor.

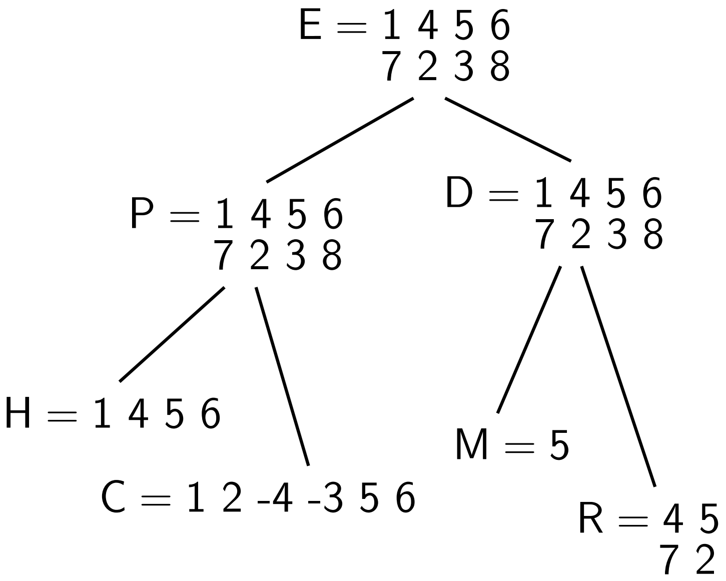


Figure S25: An example of how we define unambiguous, ambiguous, and novel atom instances and adjacencies in a particular ancestor. Here we show a true evolutionary history from root genome E to four leaf genomes. All the chromosomes in this example are contigs. In ancestral genome P, atom instance 1 is unambiguously present because its homologs occur in all the descendant leaf genomes of P. Similarly, 4, 5, and 6 are all unambiguous atom instances in P. Atom instance 2 is also unambiguous because it is in C and also in the outgroup R. Based on parsimony we could infer atom instance 2 is in P. Atom instance 3 is ambiguous in P because it is only in C and hence either could have been inserted in the C lineage after P or have been present before and in P but lost in the H lineage. Similarly, atom instance 7 is also ambiguous in P because it's only in outgroup R. Atom instance 8 is a novel atom instance in P because it's in none of the leaves. In terms of adjacencies in P, $(4*, *5)$ and $(5*, *6)$ are unambiguous, $(1*, *4)$ and $(7*, *2)$ are ambiguous, and $(3*, *8)$ and $(2*, *3)$ are novel.

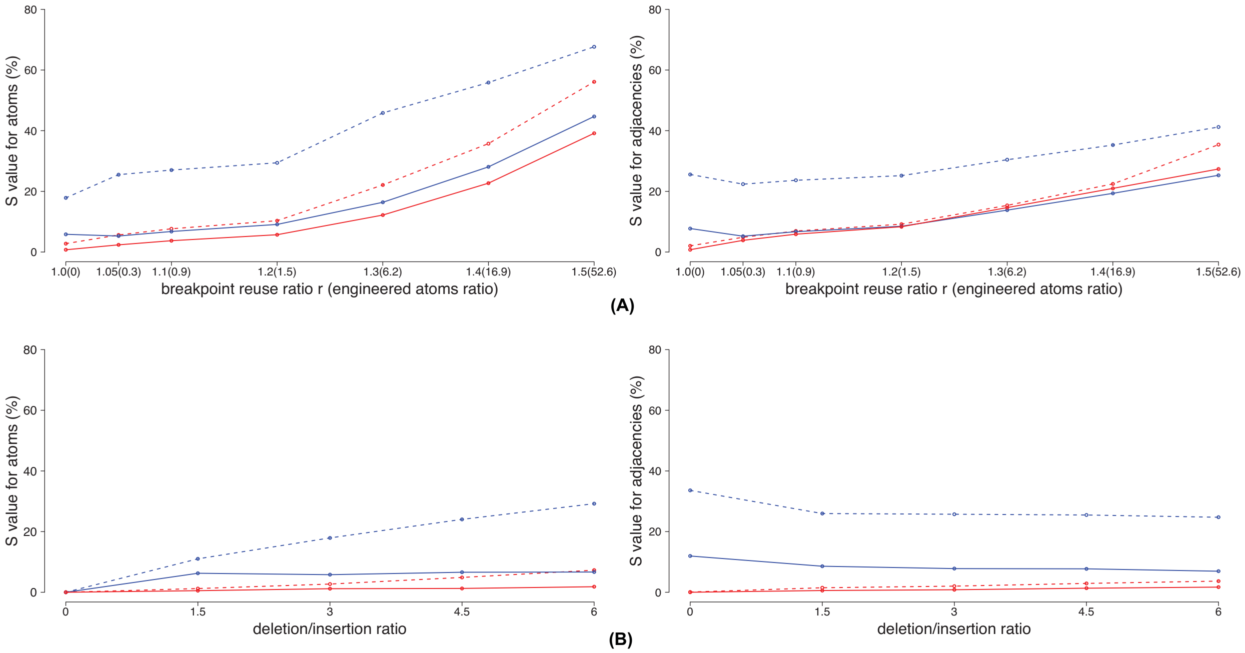


Figure S26: Comparisons between the infinite sites algorithm and DUPCAR. Each data point is the average of 100 simulations. Blue lines represent reconstruction of the genome of the Boreoeutherian common ancestor, for which no outgroup is available in this dataset, while the red lines represent the Euarchontoglires ancestor (i.e. the primate-rodent common ancestor). Straight lines are the results from infinite sites algorithm and dotted lines are the results from DUPCAR. The S value for atoms (left figures) is the normalized measure of symmetric difference between the predicted and true atom sets S_{atom} defined in the text, and the S value for adjacencies (right figures) is the analogous quantity for the adjacencies, considering only atoms that are present in both the true and predicted ancestral genomes. (A) The upper figures show the comparison when the breakpoint reuse ratio varies. We keep deletion/insertion ratio at 3 for these tests. The number in the parenthesis is ((# of engineered atoms used)/(# of atoms))×10^3 (see Section 8). These low numbers indicate that very few engineered atoms are needed by the heuristic extension of the infinite sites algorithm to cope with the breakpoint reuse. (B) The bottom panel of figures show the comparison when the deletion vs. insertion ratio varies and the breakpoint reuse ratio is 1.

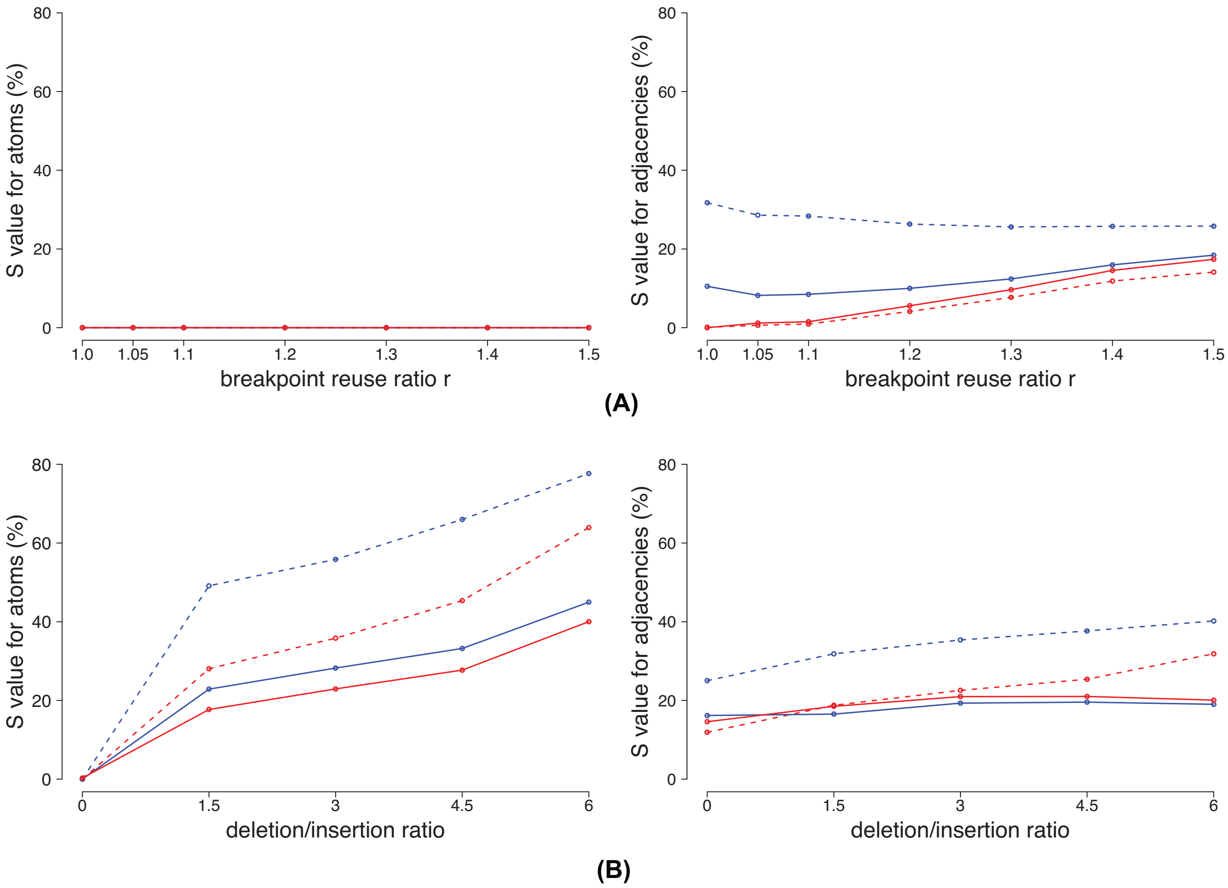


Figure S27: Comparison between the infinite sites algorithm and DUPCAR. Blue lines represent reconstruction of the genome of the Boreoeutherian common ancestor, for which no outgroup is available in this dataset, while the red lines represent the Euarchontoglires ancestor (i.e. the primate-rodent common ancestor). Straight lines are the results from infinite sites algorithm and dotted lines are the results from DUPCAR. The S value for atoms (left figures) is the normalized measure of symmetric difference between the predicted and true atom sets S_{atom} defined in the text, and the S value for adjacencies (right figures) is the analogous quantity for the adjacencies, considering only atoms that are present in both the true and predicted ancestral genomes. (A) The upper figures show the comparison when the breakpoint reuse ratio varies but there are no insertions and deletions. (B) The bottom panel of figures show the comparison when the deletion vs. insertion ratio varies and the breakpoint reuse ratio is 1.4.

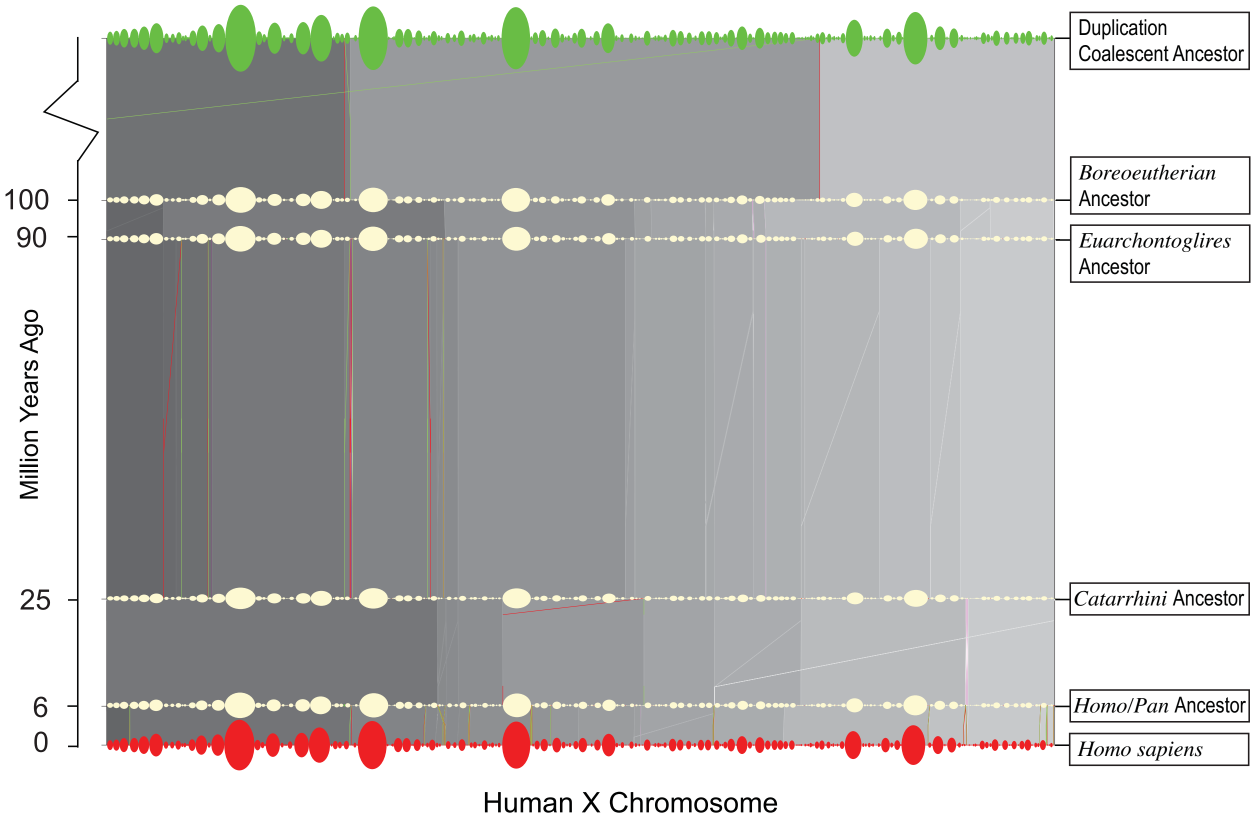


Figure S28: The evolutionary history of human chromosome X.

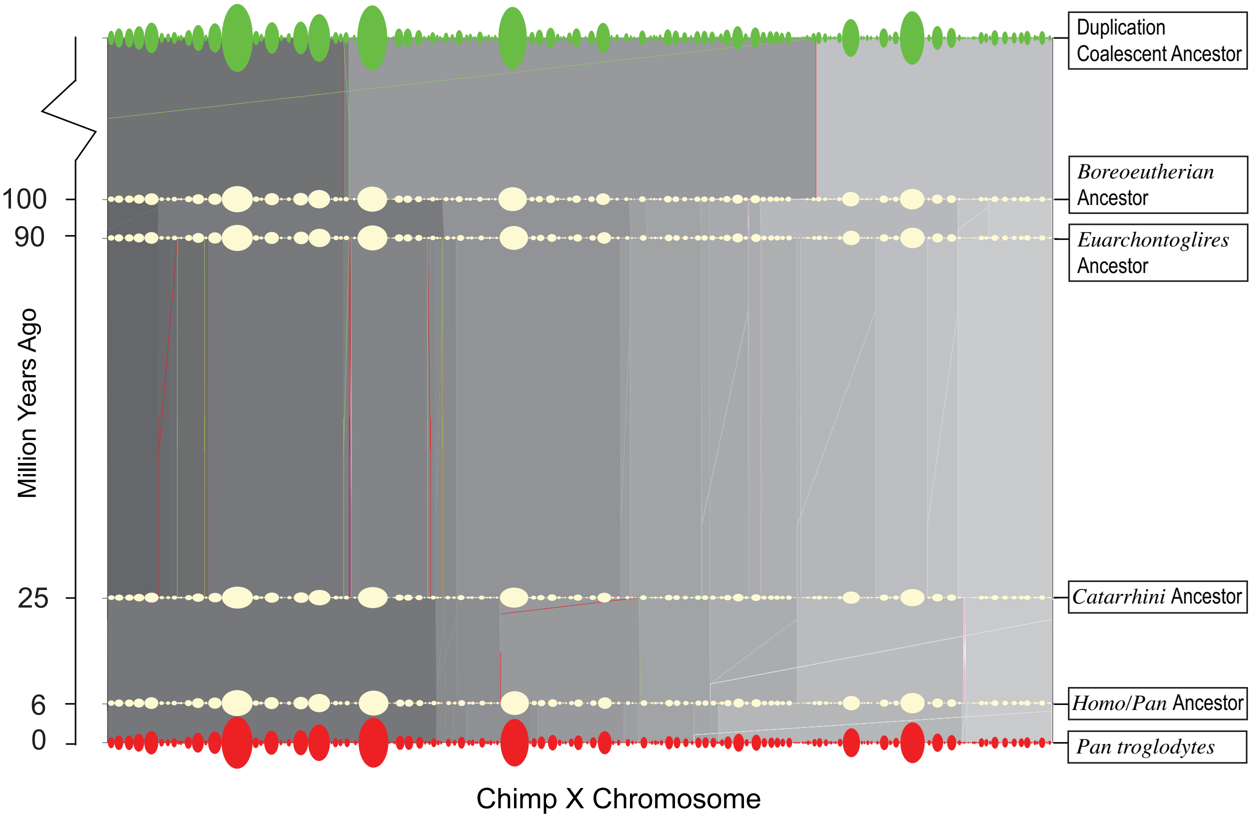


Figure S29: The evolutionary history of chimp chromosome X.

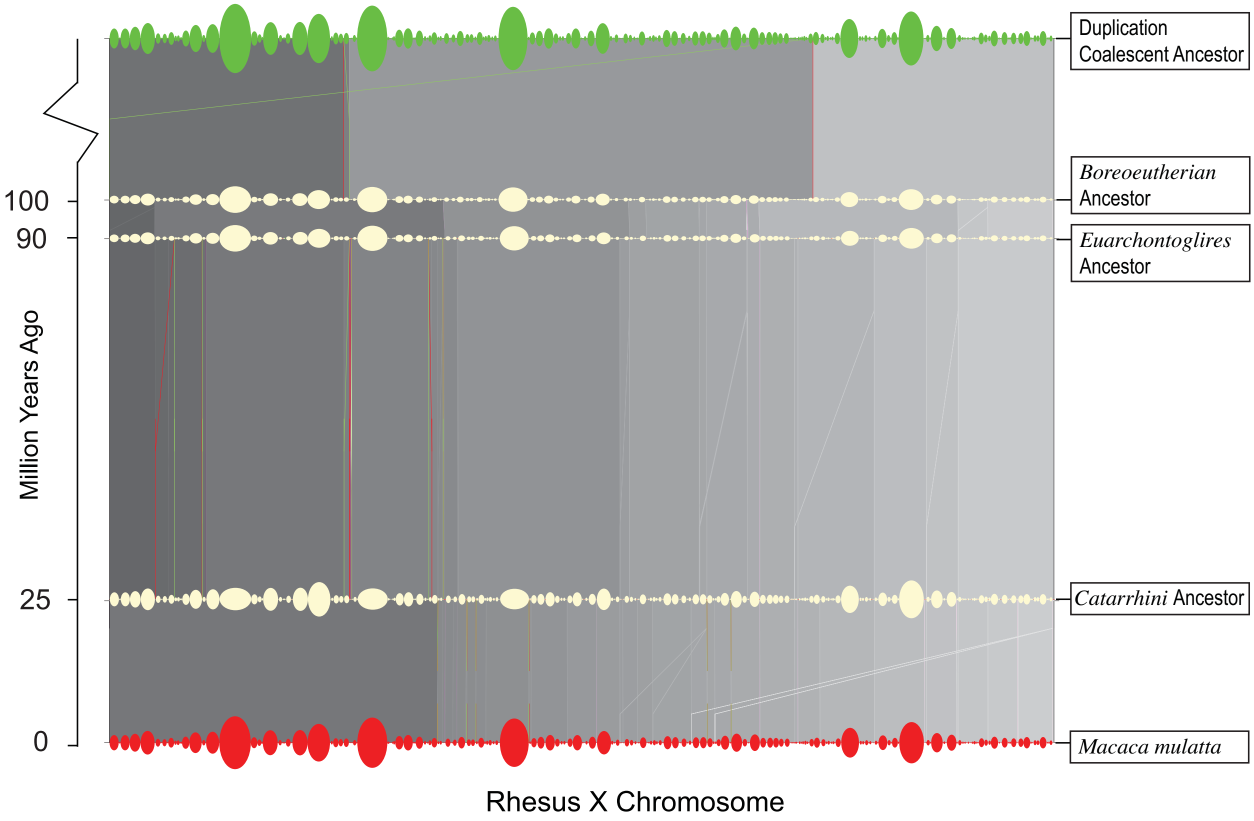


Figure S30: The evolutionary history of rhesus chromosome X.

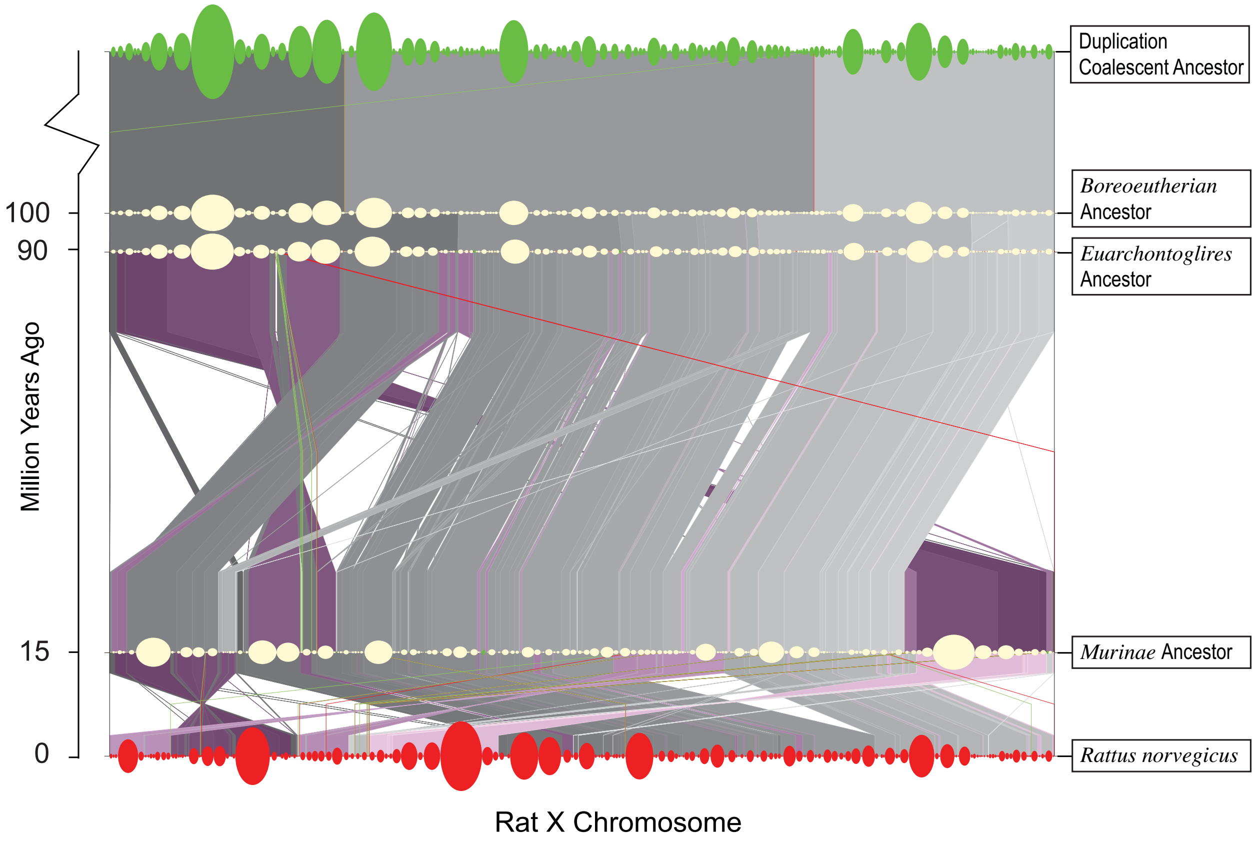


Figure S31: The evolutionary history of rat chromosome X.

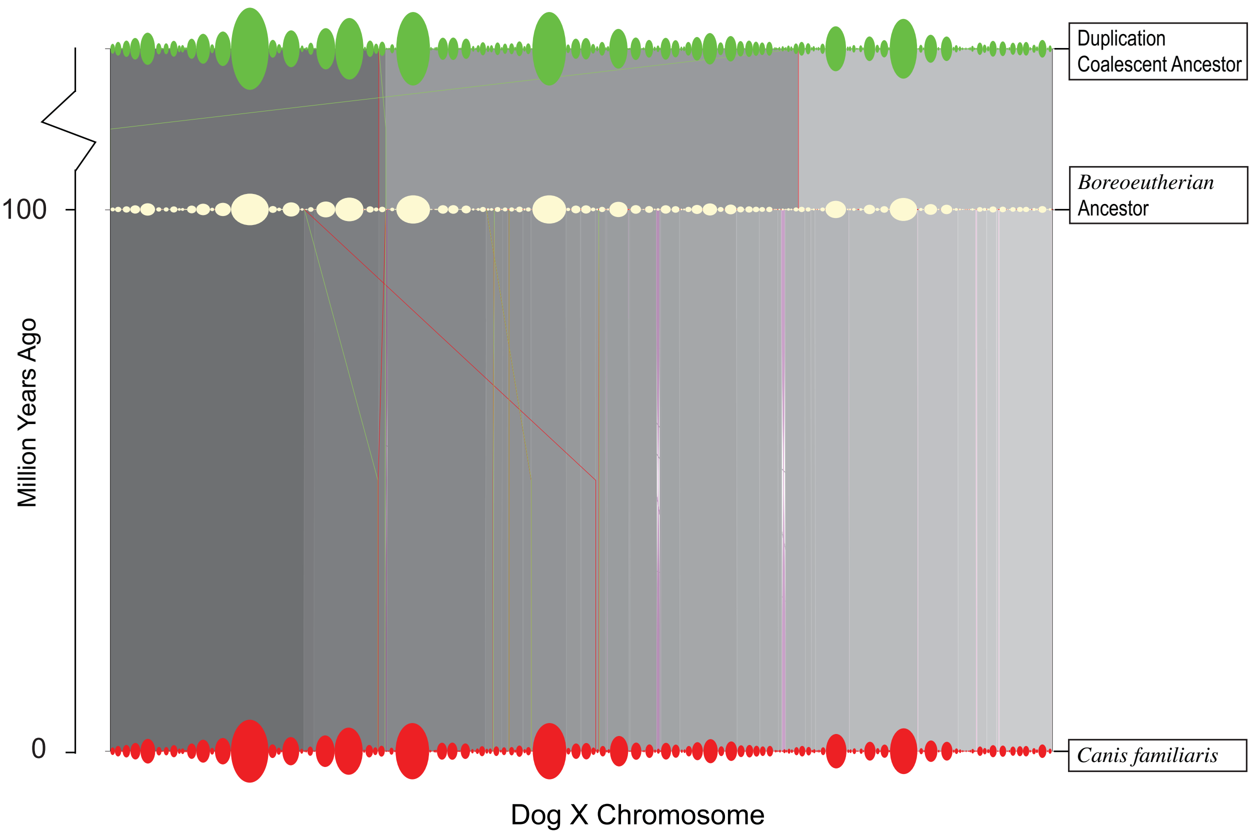


Figure S32: The evolutionary history of dog chromosome X.

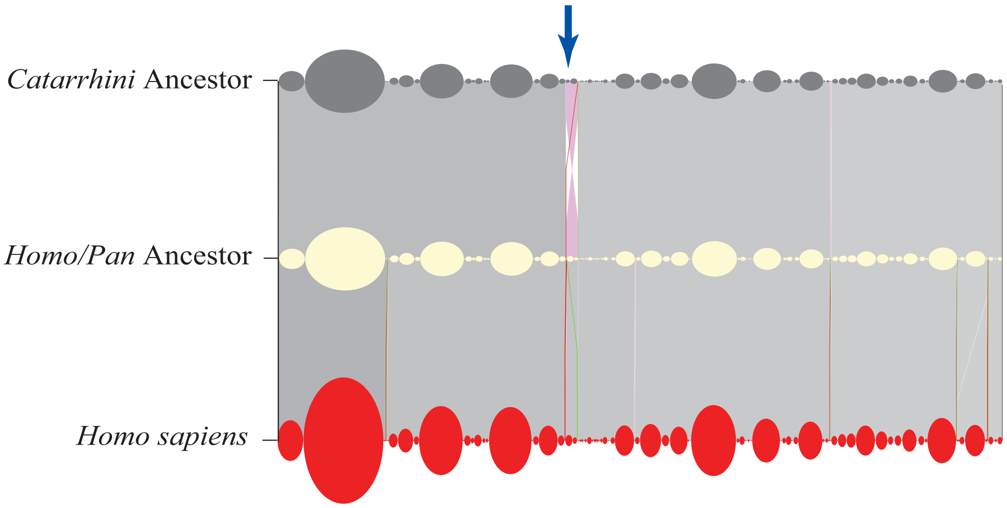


Figure S33: This graph shows the last twenty-four megabases of human chromosome X and the same region in two ancestral chromosomes: the Catarrhini ancestor and the Homo/Pan ancestor. The purple tinted region (pointed by the blue arrow) in the center represents an inversion that took place on the branch to the Homo/Pan ancestor after the split from the rhesus lineage. The corresponding region in human genome is `hg18.chrX:139,946K-140,388K`.

	simulated data	real data
reuse ratio r	1.38	1.38
atoms	2008.10	1917
engineered atoms	33.97	15
reused adj [%]	19.55	15.02
reused adj (once) [%]	13.74	11.53
reused adj (twice) [%]	4.18	3.00
reused adj (3 times) [%]	1.18	0.39
reused adj (4 times) [%]	0.31	0.13
operations	1667.55	1660
2-bp operations	1470.94	1462
3-bp operations	196.61	198
deletions	695.58	747
insertions	232.39	289
atoms in human	1483.47	1343
atoms in chimp	1442.12	1229
atoms in macaque	1457.33	1211
atoms in mouse	1050.76	896
atoms in rat	1046.58	784
atoms in dog	974.41	727

Table S1: Comparison between the simulated data and what we observe from real chrX data based on six species when we try to generate synthetic data with breakpoint reuse ratio r approximately 1.4. Here and in subsequent tables, each data point for the simulated results is the average of 100 simulations.

	S_{atom}				S_{adj}										
	dupcar		infinite		dupcar		infinite			HD			HM		
r	HD	HM	HD	HM	HD	HM	HD	HM	#ops	S_{to}	S_{loss}	S_{gain}	S_{to}	S_{loss}	S_{gain}
1	17.85	2.77	5.85	0.74	25.58	2.04	7.75	0.77	0.97	19.53	0.32	19.32	13.60	0.03	13.58
1.05	25.51	5.63	5.28	2.36	22.37	4.86	5.22	3.85	0.88	13.06	1.23	12.11	8.34	0.32	8.08
1.1	27.02	7.67	6.77	3.73	23.65	6.93	6.70	5.88	0.84	13.79	1.91	12.32	9.04	0.81	8.37
1.2	29.41	10.33	9.10	5.70	25.18	9.19	8.50	8.33	0.78	17.45	2.99	15.29	12.51	1.70	11.16
1.3	45.85	22.10	16.39	12.18	30.42	15.38	13.81	14.64	0.62	25.54	6.98	21.13	20.51	5.58	16.59
1.4	55.87	35.74	28.08	22.70	35.27	22.46	19.35	20.98	0.43	32.01	15.73	22.13	27.80	14.38	17.84
1.5	67.66	56.11	44.69	39.16	41.22	35.40	25.30	27.37	0.25	46.97	30.21	31.17	41.87	28.70	24.12

Table S2: Detailed performance statistics when the breakpoint reuse ratio varies. Column (#ops) is the ratio of (# of operations from infinite sites algorithm)/(# of operations in the true history). In the table, columns 11-13 are the turn-over S values for Boreoeutherian ancestor (HD) and columns 14-16 for Euarchontoglires ancestor (HM). Here $S_{to}, S_{loss}, S_{gain}$ define the turnover of the synthetic data such that: $S_{loss} = ((loss)/(loss + common)) \times 100\%$, $S_{gain} = ((gain)/(gain + common)) \times 100\%$, $S_{to} = ((loss + gain)/(loss + gain + common)) \times 100\%$, i.e. the overall turnover. *common* are all the atom instances shared by the ancestor and its descendants. *loss* include all the ancestral atom instances that have been completely lost due to deletions. *gain* include atom instances born from insertions and duplications after the designated ancestor.

	S_{atom}				S_{adj}											
	dupcar		infinite		dupcar		infinite			HD			HM			
del/ins	HD	HM	HD	HM	HD	HM	HD	HM	#ops	S_{to}	S_{loss}	S_{gain}	S_{to}	S_{loss}	S_{gain}	
0	0	0	0	0	33.59	0.08	11.95	0	1	11.68	0	11.68	8.67	0	8.67	
1.5	11.03	1.23	6.27	0.52	25.95	1.49	8.59	0.57	0.98	24.24	0.13	24.16	16.88	0.01	16.87	
3	17.92	2.71	5.79	1.17	25.73	2.02	7.80	0.83	0.97	19.46	0.34	19.24	13.56	0.05	13.52	
4.5	24.03	4.89	6.60	1.26	25.48	2.92	7.73	1.34	0.95	18.50	0.65	18.06	12.76	0.12	12.67	
6	29.21	7.29	6.68	1.82	24.73	3.68	6.94	1.70	0.93	17.47	1.07	16.73	12.22	0.21	12.05	

Table S3: Detailed performance statistics when the reuse ratio is 1 and the deletion/insertion ratio varies. See Table S2 for definitions of S_{to}, S_{loss}, and S_{gain}. When reuse ratio is 1 and there are no deletions, the infinite sites algorithm always predicts the correct set of operations and correctly reconstructs the Euarchantoglires ancestor using the dog as an outgroup, i.e. #ops=1, $S_{adj} = 0$ and $S_{atom} = 0$. For the Boreoeutherian ancestor, $S_{atom} = 0$, but S_{adj} is not always 0 because there is ambiguity whether some rearrangements happened on the branch leading to dog or on the branch to the Euarchontoglires ancestor.

	S^u_{atom}								S^a_{atom}								S^n_{atom}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
r	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
1	82.15	0	97.23	0	79.08	0	96.75	0	17.53	100	2.74	100	20.61	26.87	3.21	21.38	0.32	100	0.03	100	0.31	100	0.03	100
1.05	74.49	0	94.37	0	73.09	0	93.92	0	24.28	100	5.31	100	25.70	15.82	5.76	34.92	1.23	100	0.32	100	1.21	100	0.31	100
1.1	72.98	0	92.33	0	71.59	0	91.88	0	25.11	100	6.87	100	26.53	18.45	7.33	40.13	1.91	100	0.80	100	1.88	100	0.79	100
1.2	70.59	0	89.67	0	69.19	0	89.10	0	26.42	100	8.65	100	27.87	22.12	9.23	43.65	2.99	100	1.68	100	2.93	100	1.67	100
1.3	54.15	0	77.90	0	52.28	0	76.45	0	38.87	100	16.60	100	40.98	23.55	18.15	37.37	6.98	100	5.50	100	6.74	100	5.40	100
1.4	44.13	0	64.26	0	42.63	0	62.80	0	40.14	100	21.47	100	42.17	30.55	23.25	37.62	15.73	100	14.27	100	15.20	100	13.95	100
1.5	32.34	0	43.89	0	30.48	0	42.49	0	37.45	100	27.45	100	41.03	39.51	29.76	38.35	30.21	100	28.66	100	28.48	100	27.75	100

Table S4: Prediction accuracies for unambiguous, ambiguous, and novel atom instances when the breakpoint reuse ratio varies and deletion/insertion ratio = 3. The first number in each column associated with the result for each ancestor using different methods shows the percentage of atoms fall into that category (unambiguous, ambiguous, or novel) out of all the atoms. Neither algorithm makes a mistake predicting the presence of unambiguous atom instances, and both fail to predict novel atom instances. However, the infinite sites algorithm is able to predict the presence of many ambiguous atom instances correctly.

	S^u_{adj}								S^a_{adj}								S^n_{adj}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
r	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
1	71.00	0.11	95.35	0.01	62.75	0	91.74	0	29.00	87.92	4.65	43.52	36.78	19.94	8.19	8.93	0	NA	0	NA	0.47	88.21	0.07	45.60
1.05	75.54	0.93	93.04	0.59	58.26	0.92	87.17	0.69	24.41	88.58	6.89	61.60	40.99	9.63	12.29	22.14	0.06	100	0.07	100	0.75	98.08	0.54	96.68
1.1	74.74	1.56	91.25	1.23	57.25	1.55	84.55	1.34	25.19	88.97	8.62	65.79	41.79	11.65	14.57	26.75	0.07	100	0.12	100	0.96	98.25	0.88	97.48
1.2	73.60	2.31	89.56	2.14	55.76	2.30	81.71	2.26	26.32	88.92	10.26	69.06	43.11	14.15	17.01	30.78	0.08	100	0.18	100	1.14	98.45	1.28	97.46
1.3	68.04	2.31	81.26	2.32	41.43	2.29	66.04	2.50	31.82	90.25	18.24	71.18	56.19	18.73	31.12	32.87	0.14	100	0.49	100	2.39	98.05	2.84	97.23
1.4	65.15	4.94	75.08	5.17	37.36	4.90	55.73	5.26	34.71	91.96	24.27	73.83	59.86	24.69	40.38	35.18	0.13	100	0.65	100	2.78	98.93	3.88	98.83
1.5	64.59	12.25	66.83	12.30	33.99	11.85	40.72	11.93	35.37	94.13	32.71	81.59	63.42	29.56	55.54	33.87	0.05	100	0.46	100	2.59	99.29	3.74	99.31

Table S5: Prediction accuracies for unambiguous, ambiguous, and novel adjacencies when the breakpoint reuse ratio varies and deletion/insertion ratio = 3. "NA" indicates that the denominator was 0. The first number in each column associated with the result for each ancestor using different methods shows the percentage of adjacencies fall into that category (unambiguous, ambiguous, or novel) out of all the adjacencies. The infinite sites algorithm is able to predict some of the novel adjacencies.

	S^u_{atom}								S^a_{atom}								S^n_{atom}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
del/ins	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
0	100	0	100	0	100	0	100	0	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA
1.5	88.97	0	98.77	0	84.63	0	98.35	0	10.90	100	1.22	100	15.25	40.26	1.63	29.26	0.13	100	0.01	100	0.12	100	0.01	100
3	82.08	0	97.29	0	79.08	0	96.42	0	17.58	100	2.66	100	20.59	26.54	3.53	22.17	0.34	100	0.05	100	0.33	100	0.05	100
4.5	75.97	0	95.11	0	73.07	0	94.42	0	23.38	100	4.77	100	26.30	22.70	5.46	20.85	0.65	100	0.12	100	0.63	100	0.12	100
6	70.79	0	92.71	0	68.54	0	91.79	0	28.14	100	7.08	100	30.42	18.54	8.00	18.00	1.07	100	0.21	100	1.04	100	0.21	100

Table S6: Prediction accuracies for unambiguous, ambiguous, and novel atom instances when the deletion/insertion ratio varies and $r = 1$. "NA" indicates that the denominator was 0. The first number in each column associated with the result for each ancestor using different methods shows the percentage of atoms fall into that category (unambiguous, ambiguous, or novel) out of all the atoms.

	S^u_{adj}								S^a_{adj}								S^n_{adj}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
del/ins	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
0	65.90	0.48	99.93	0.08	74.24	0	99.93	0	34.10	97.59	0.07	0	25.76	46.36	0.07	0	0	NA	0	NA	0	NA	0	NA
1.5	69.80	0.13	96.41	0.02	67.52	0	94.94	0	30.20	85.63	3.59	40.44	32.21	25.85	5.03	10.94	0	NA	0	NA	0.27	91.63	0.04	46.71
3	70.91	0.12	95.38	0.01	62.70	0	91.88	0	29.09	88.14	4.62	43.24	36.86	20.10	8.04	9.82	0	NA	0	NA	0.44	87.65	0.09	40.34
4.5	71.22	0.09	94.04	0.01	57.73	0	87.86	0	28.78	88.28	5.96	48.66	41.66	17.29	11.97	10.43	0	NA	0	NA	0.61	85.19	0.16	53.27
6	72.31	0.06	93.23	0.00	54.35	0	84.21	0	27.69	89.17	6.77	54.11	45.00	14.19	15.58	10.05	0	NA	0	NA	0.66	84.77	0.21	63.24

Table S7: Prediction accuracies values for unambiguous, ambiguous, and novel adjacencies when the deletion/insertion ratio varies and $r = 1$. "NA" indicates that the denominator was 0. The first number in each column associated with the result for each ancestor using different methods shows the percentage of adjacencies fall into that category (unambiguous, ambiguous, or novel) out of all the adjacencies.

	S_{atom}				S_{adj}										
	dupcar		infinite		dupcar		infinite			HD			HM		
del/ins	HD	HM	HD	HM	HD	HM	HD	HM	#ops	S_{to}	S_{loss}	S_{gain}	S_{to}	S_{loss}	S_{gain}
0	0	0	0	0	25.03	11.91	16.18	14.60	0.83	10.91	0	10.91	7.93	0	7.93
1.5	49.13	28.06	22.87	17.70	31.85	18.77	16.52	18.50	0.54	33.09	9.56	27.99	26.75	8.67	21.27
3	55.84	35.82	28.23	22.93	35.37	22.56	19.31	20.98	0.43	32.08	15.97	22.02	27.83	14.66	17.61
4.5	65.99	45.38	33.21	27.71	37.66	25.39	19.57	21.01	0.34	32.75	22.00	17.00	29.38	20.04	14.19
6	77.66	63.92	44.97	40.02	40.19	31.86	19.02	20.07	0.20	41.00	34.72	13.98	38.06	32.30	12.06

Table S8: Detailed turnover statistics when the reuse ratio is 1.4 and the deletion/insertion ratio varies. Let *loss* denote the number of the ancestral atom instances that have been completely lost due to deletions since the designated ancestor. Let *gain* denote the number of atom instances born from insertions and duplications after the designated ancestor. Let *common* denote the number of atom instances that are unchanged (neither lost nor gained) since the designated ancestor. We define $S_{loss} = ((loss)/(loss+common)) \times 100\%$, $S_{gain} = ((gain)/(gain+common)) \times 100\%$, and $S_{to} = ((loss+gain)/(loss+gain+common)) \times 100\%$, i.e. the total turnover. These represent the fractions of atoms lost, gained and lost or gained, respectively. They are shown in the last 6 columns in the table (columns 11-16). Columns 11-13 are the turnover values for the Boreoeutherian ancestor (HD) and columns 14-16 for the Euarchontoglires ancestor (HM). Column (#ops) is the ratio of (# of operations from infinite sites algorithm)/(# of operations in the true history). It is evident that this correlates inversely with turnover, i.e. the more turnover there is, the fewer the operations in the simplest history relative to the true history. The other columns repeat data from previous tables, for ease of comparison. These show how accuracy of reconstruction degrades with increasing turnover.

	S_{atom}				S_{adj}										
	dupcar		infinite		dupcar		infinite			HD			HM		
r	HD	HM	HD	HM	HD	HM	HD	HM	#ops	S_{to}	S_{loss}	S_{gain}	S_{to}	S_{loss}	S_{gain}
1	0	0	0	0	31.74	0.10	10.51	0.00	1.00	14.25	0	14.25	10.65	0	10.65
1.05	0	0	0	0	28.60	0.62	8.21	1.18	0.99	9.44	0	9.44	6.91	0	6.91
1.1	0	0	0	0	28.36	0.93	8.48	1.51	0.98	9.20	0	9.20	6.60	0	6.60
1.2	0	0	0	0	26.33	4.13	10.00	5.57	0.92	10.05	0	10.05	7.17	0	7.17
1.3	0	0	0	0	25.61	7.70	12.40	9.66	0.87	10.25	0	10.25	7.42	0	7.42
1.4	0	0	0	0	25.76	11.84	15.96	14.56	0.84	10.46	0	10.46	7.45	0	7.45
1.5	0	0	0	0	25.81	14.12	18.43	17.38	0.82	10.77	0	10.77	7.63	0	7.63

Table S9: Detailed turnover statistics when the breakpoint reuse ratio varies and there is no insertion and deletion. See Table S8 for definitions of S_{to}, S_{loss}, and S_{gain} Without insertion or deletion, the only kind of turnover possible is gain due to duplication.

	S^u_{adj}								S^a_{adj}								S^n_{adj}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
r	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
1	67.37	0.37	99.89	0.10	75.67	0	99.89	0	32.63	96.51	0.11	0.50	24.33	43.10	0.11	0	0	NA	0	NA	0	NA	0	NA
1.05	70.39	0.33	98.40	0.11	78.56	0.17	97.99	0.30	29.16	95.72	1.57	29.01	20.92	35.68	1.98	41.06	0.45	100	0.02	100	0.52	98.83	0.03	97.14
1.1	70.59	0.36	97.85	0.14	78.56	0.21	97.41	0.36	28.86	95.45	2.11	34.95	20.80	36.70	2.54	42.65	0.55	100	0.03	100	0.64	98.37	0.05	97.70
1.2	72.36	0.79	94.08	0.66	78.85	0.72	92.95	1.35	26.40	92.83	5.64	56.89	19.67	40.26	6.58	58.34	1.24	100	0.28	100	1.48	99.18	0.48	97.16
1.3	73.24	1.18	90.62	1.27	78.35	1.14	89.02	2.28	25.14	91.96	8.58	66.84	19.44	47.74	9.62	65.35	1.61	100	0.80	100	2.20	98.58	1.37	97.57
1.4	73.20	1.53	87.07	2.09	76.77	1.75	84.85	3.48	24.59	91.17	11.23	74.01	19.71	56.38	12.31	71.45	2.21	100	1.70	100	3.52	99.09	2.84	98.53
1.5	73.16	1.60	85.06	2.44	75.60	1.97	82.44	4.15	24.18	90.86	12.62	77.05	19.80	62.46	13.55	73.77	2.66	100	2.31	100	4.60	99.25	4.01	98.83

Table S10: Prediction accuracies for unambiguous, ambiguous, and novel adjacencies when the breakpoint reuse ratio varies and there is no insertion/deletion. "NA" indicates that the denominator was 0. The first number in each column associated with the result for each ancestor using different methods shows the percentage of adjacencies fall into that category (unambiguous, ambiguous, or novel) out of all the adjacencies.

	S^{u}_{atom}								S^{a}_{atom}								S^{n}_{atom}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
del/ins	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
0	100	0	100	0	100	0	100	0	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA
1.5	50.87	0	71.94	0	47.70	0	69.37	0	39.57	100	19.48	100	43.34	32.08	22.36	42.20	9.56	100	8.57	100	8.97	100	8.27	100
3	44.16	0	64.18	0	42.65	0	62.74	0	39.87	100	21.27	100	41.93	30.58	23.04	37.85	15.97	100	14.55	100	15.42	100	14.22	100
4.5	34.01	0	54.62	0	33.35	0	53.78	0	44.00	100	25.47	100	45.08	25.82	26.61	30.42	22.00	100	19.92	100	21.57	100	19.61	100
6	22.34	0	36.08	0	22.10	0	35.73	0	42.94	100	31.66	100	43.55	24.38	32.32	24.97	34.72	100	32.26	100	34.35	100	31.95	100

Table S11: Prediction accuracies for unambiguous, ambiguous, and novel atom instances when the deletion/insertion ratio varies and $r = 1.4$. "NA" indicates that the denominator was 0. The first number in each column associated with the result for each ancestor using different methods shows the percentage of atoms fall into that category (unambiguous, ambiguous, or novel) out of all the atoms.

	S^{u}_{adj}								S^{a}_{adj}								S^{n}_{adj}							
	dupcar				infinite				dupcar				infinite				dupcar				infinite			
del/ins	HD		HM		HD		HM		HD		HM		HD		HM		HD		HM		HD		HM	
0	73.88	1.51	87.11	2.11	77.06	1.77	84.89	3.59	23.85	90.74	11.13	74.65	19.37	58.17	12.19	71.17	2.27	100.00	1.76	100.00	3.57	99.05	2.92	98.50
1.5	66.01	3.21	77.21	3.39	39.48	3.20	60.24	3.53	33.79	87.38	22.09	69.91	57.76	21.72	36.06	35.32	0.20	100.00	0.70	100.00	2.76	98.56	3.70	98.26
3	65.06	4.96	75.00	5.17	37.40	4.91	55.85	5.24	34.83	91.97	24.41	74.08	59.85	24.68	40.25	35.32	0.11	100.00	0.59	100.00	2.75	98.52	3.90	98.54
4.5	63.31	4.56	71.06	4.80	30.36	4.52	47.54	4.83	36.63	94.78	28.48	75.51	67.02	23.31	48.85	31.04	0.06	100.00	0.47	100.00	2.62	98.81	3.61	98.79
6	63.08	6.74	65.44	6.44	24.46	6.60	33.90	6.26	36.91	97.41	34.42	79.81	73.77	21.27	63.64	24.42	0.00	100.00	0.14	100.00	1.77	98.72	2.46	98.98

Table S12: Prediction accuracies values for unambiguous, ambiguous, and novel adjacencies when the deletion/insertion ratio varies and $r = 1.4$. "NA" indicates that the denominator was 0. The first number in each column associated with the result for each ancestor using different methods shows the percentage of adjacencies fall into that category (unambiguous, ambiguous, or novel) out of all the adjacencies.