SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

A brief introduction to molecular phylogeny

av

Måns Magnusson

2010 - No 11

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET, 106 91 STOCKHOLM

A brief introduction to molecular phylogeny

Måns Magnusson

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Jens Lagergren

2010

Abstract

The problem of recreating the evolutionary tree of a species from a set of gene trees has been studied for a long time and through the years a number of models have been presented.
In the year 2000 Jens Lagergren and M.T Hallet published the article *"New Algorithms for the Duplication-Loss Modell"* [1] where they developed algorithms based on parsimony for this problem.
In this essay we will learn the basic concepts of phylogeny, both in terms of biology and mathematics, we will look closer at the article mentioned above and end by making a small study in the evolution of genes.

Contents

1 Acknowledgements

I would like to thank Jens Lagergren for letting me do this work without any prior knowledge in the field, also Tom Britton for taking his time to do the examination.

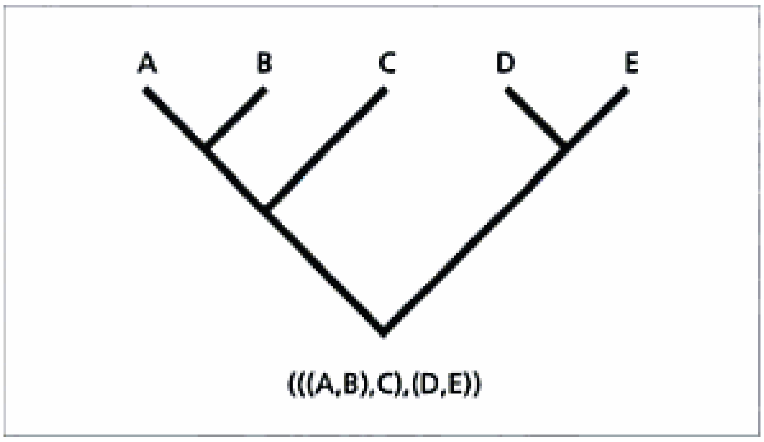
A special thank to Bengt Sennblad at SBC for all his help, to Jens Malmros for helping out with the statistics, Amarendra Badugu for the proof-reading and to my family Carolina and Iker for giving me the time.

2 Introduction

In this essay we will study the problem of reconstructing the evolution of a species by looking at data in the form of genes from the species in question. We will use [1] as our base and look closer on some of the concepts of evolution and one of the algorithms presented in that article.
We begin with a walkthrough in the field of evolutionary biology so that we can understand the technical part.
The structure of the Tree of Life have been altered a lot throughout history when the ways of classification have changed and science has made new methods possible.
The basic idea of the system used is called *Cladistics* where species are classified into groups called *clades* that consists of an ancestor and all of it's known ascendants.

2.1 Molecular Phylogeny

Phylogeny is a way to structure evolutionary relations in a cladistic manner, for example in a group of genes or organisms. To illustrate these relations graph is the general tool to use, usually in forms of binary trees where the leafs represent species for which we have data in some form, these species can be existing or extinct.
Observe that species can mean for example genes or animal species.
The inner vertices represents hypothetical ancestors and if there is a root it represents the common ancestor of all species in the tree.
Two vertices with a common ancestor are closer related than two without a common ancestor.



[2]

As we can see in the picture, it is also possible to represent a tree in a more compact notation called Newick format, this way of writing also gives us full understanding of the relations in the tree.

At the inner vertices the evolutionary tree splits, this is called branching, and we have a speciation.
This means that the population has evolved so much in different directions that parts of the population are not the same anymore.
The subject of telling biologically when a speciation has happened is debated and will not be discussed in this overview.

In traditional phylogeny *morphology* was used to structure relations, i.e physical similarities decides how closely related different species are, but when genetics came into the picture one realized that even if two species were physically similar one of them could be closer related to another species with less physical similarity.

This is explained by looking at homologous molecular data, i.e. strings of DNA/RNA or amino acid sequences, from different species and compare the similarities.
By homology in the context of molecular data we mean sequences with a common ancestor and a common function, for example if the DNA-sequences of two genes in two different species are similar we can assume that they have a common ancestry and hence are homologous.

The sequences compared are of such lengths that there is a very small chance they can have developed into similar sequences independent of each other[3].
We can think of the evolution of a gene as a tree (gene tree) where all the leaves have the root as their common ancestor, we call a this set of homologous genes for a *gene family*.
When we say evolution of genes (sequences of DNA) we refer to mutations of the DNA that occurs in biological processes, that is nucleotides or sequences of nucleotides change over time and effect the performance of the genes.
The *evolutionary distance* between two homologous sequences is a measure of how far they have evolved from each other since their common ancestor.

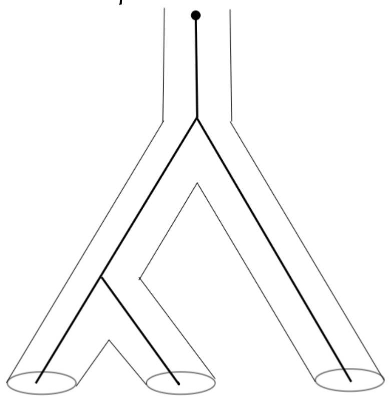
One might think that when building the evolutionary tree of a species, *species trees*, we sequence the whole genome for different species and then compare the genomes of all species to see which ones are most similar.
With the knowledge of today this is almost impossible since aligning sequences is a hard problem and it is not possible to get good results if the aligned sequences are to long.

Instead we can choose a number of gene families and compare what they say about the evolution of the species.

The bigger evolutionary distance that two genes have in a gene family the bigger distance between the species that the genes came from are in the species tree.

We can think of this as the gene trees evolving inside the species tree, following the evolutionary events of the species.
The edges, or branches, in the trees doesn't mean more than the relation as mentioned earlier if nothing else has been declared.
In other case different lengths or edges labeled by numbers could indicate time elapsed or a measure for the amount of evolution.

Example:



This picture shows a gene tree evolving "inside" a species tree. The speciations should be thought of as vertices and the leafs are in the bottom of the picture.

So again, if we follow the evolution of a gene from their common ancestor in the species tree, we can expect to find variations of that gene in all leaves, i.e. the data that is available to us now.
The evolutionary distance between the members of the gene family corresponds to the evolutionary distance between the species that the genes were found in.

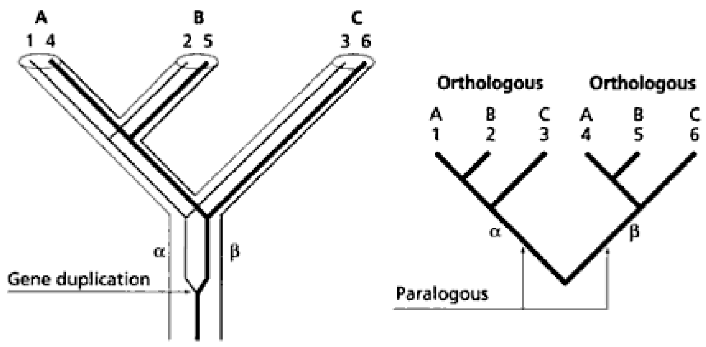
The idea is that if we collect the sequences for a gene family in many different species that share these genes and label the leaves in our gene tree with the found sequences, we can then align these sequences to see which ones are most similar.
We now let the pairs with most similarity have a common ancestor that we label with the aligned sequence from it's descendants, then we can align the ancestors to get new ancestors and so on.
Unfortunately nature is not as simple as one might hope, in the next section we will look closer on what events that can effect the processes described.

2.2 Building Phylogenetic Trees

There are similarities in evolutionary time between different levels of biology from molecules to macro evolutionary patterns, one example of this is the evolution of parasite-host relation[2].
In a similar way we look at gene- and species trees of an organism and make the assumption that there is a relation between the evolution of the genes and the organisms, that the gene- and species trees are isomorphic.
So if we get all the information from one of them we have all the information of the other.
Many species trees has been carefully builded by morphologists during the years so one often compare the results of molecular studies with the morphological trees.

The more information that is gained in the form of sequence data the more differences in the trees are usually discovered and one start to realize that a complex relation reigns.
We want to understand what kind of evolutionary events that causes these changes. There are several processes like lineage sorting and horizontal gene transfer that can give rise to discrepancies but the ones that are important for us are *gene duplications* and *gene loss*.

A duplication is the event when a section of the genome is duplicated, usually during transcription, and keeps on evolving independent of each other.
As a consequence any genes that are in this part also get duplicated, a loss is when a part of the genome is lost.
We can see the duplication of a gene evolving in species tree illustrated in the following picture:



[2]

Notice that in every speciation of the species tree, all genes from the ancestor keeps on existing in all species.
We say that two homologous genes are *orthologous* if their most recent com-

mon ancestor corresponds to a speciation and not a duplication, if it is a duplication we call them *paralogous*.

The problem here is that when the information of one or more genes are lost the gene tree will differ from the species tree, missing information could indicate a loss or simply that we missed some genes during sequencing.
For example in the picture above, if we only get information from gene 1, 5 and 6 and align these, we would think that 5 and 6 are most alike so the species tree will be $(A,(B,C))$ instead of the actual tree which is $((A,B),C)$.
Since it is a fact that events like losses occurs we will need information from several different gene families to reconstruct the history of evolution.

The concept of *reconciling* gene trees with a species tree were introduced by Goodman et. al.[4], where they showed how the differences in the trees can be explained by speciations, duplications and losses.
The method they used is called *Parsimony* where one make the assumption that the species tree that uses minimal number of evolutionary events to explain the gene trees is the actual species tree.

This lead to two optimization problems[5]:

1. The **Duplication** problem that search for the species tree that require the minimal numbers of duplications to reconcile the gene trees.
2. The **Duplication/Loss** problem that search for the species tree that requires the minimal number duplication and losses to reconcile the gene trees.

We are now ready to formulate the main problem for this essay:

Given a set of homologous gene trees but no information about the species tree, how many duplications is needed for the optimal species tree to explain all of the gene trees?.

This is a first step in solving the problem of recreating the evolutionary tree for a species by looking at the gene trees.
We begin by defining the mathematics needed, both the basics and also some math from [1]. Then we will analyze an algorithm from the same article that solves the problem stated above, we will also prove that the algorithm provides us with the right answer.
As mentioned the algorithm gives us the number of duplications but it is possible to modify it to compute the optimal species tree, this will not be included in this essay.

3 Method

3.1 Basic Graph theory

Graphs allow a representation of a possible connection between objects. We will in this section make formal definitions of different graphs and some of there properties. [6] [7]

Definition 3.1.1. *A* graph *is a pair $G = (V, E)$ of a finite set $V_G = \{s, t, u, ...\}$ and a set E_G which elements are subsets whith two elements from V. The elements in V_G is called* vertices *or* nodes *and the elements in E_G are called* edges*.*

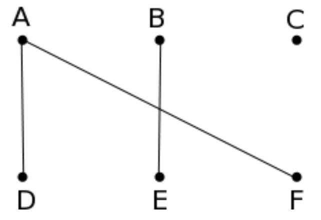
The edge $\{u, v\}$ can be denoted uv or (u, v) and we say that the vertices u and v are connected by edge uv. The graph G can be represented by a diagram where V_G are dots in the plane who are joined by lines if and only if they are connected.

The *degree* of a vertex is the number of edges that connect a node v with other nodes.

Definition 3.1.2. *A* path *in a graph G between the nodes u and u' is a series $S : u = u_0, u_1, u_2, ..., u_{p-1}, u_p = u'$ such that for $0 \leq i \leq p-1$, $u_i u_{i+1} \in E_G$. Moreover $u_i u_{i+1}$ only exist once in S.*
The number p is called the length *of S. We have that an edge is a path with length 1, and a vertex is a path with length 0. The vertices u and u' is called endpoints in S.*

- A *Cycle* in a graph is a path that starts and ends in the same node.
- A graph G is called *connected* if it for every pair of vertices in V_G exists a path that connects them.
- A *subgraph* of G is a subset $V_G' \in V_G$ that includes all the edges from G that has it's endpoints in V_G'.

Example:



This is a graph G, which is not connected, with vertices $V_G = \{$A, B, C, D, E, F$\}$ and edges $E_G = \{$AD, AF, BE$\}$.

The degrees of the vertices in G are as follows:

deg(A)=2, deg(B)=deg(d)=deg(E)=deg(F)=1 and deg(C)=0.

A connected subgraph H of G have vertices $V_H = \{$A,D,F$\}$

An example of a path P of length 2 in G is $P =$ D,A,F

Definition 3.1.3. *A* Directed Graph *D is a set of vertices, V_D, and a set of directed edges, A_D, called Arcs.*
We denote an arc pointing from u to v like $\overline{(u,v)}$, or in the graphical notation as an arrow from u pointing at v.
Arcs are defined in the same way as edges in a graph with the difference that an arc is a ordered pair $\overline{(u,v)}$ whereas an edge of a graph is a unordered pair (u,v).
Moreover if $v, x, y \in V_D$ we define the indegree *for a node v as the number of arcs on the form $\overline{(x,v)}$ in D, and the* outdegree *for v as the number of arcs on the form $\overline{(v,y)}$ in D.*

For a directed graph D we say that E_D are the the edges of the underlying non-directed graph with the same set of vertices like D.

3.2 Trees

Definition 3.2.1. *A* Tree *is a connected graph without any cycles.*

Definition 3.2.2. *A* Rooted Tree *is a pair, (T, r), of a tree T and a vertex $r \in V_T$, called the root of T.*
Now let (T, r) be a rooted tree and let $s, s' \in V_T$, we say that

$s \leq s'$ *if and only if s' is on the path that unites s and r.*

It is clear that the relation $\leq$ has the following properties:

- ***Reflexive:*** *For all s we have that $s \leq s$.*
- ***Antisymmetric:*** *For all s and s' we have that if $s \leq s'$ and $s' \leq s$ we have $s = s'$.*
- ***Transitive:*** *For all s, u, v where $s \leq u$ and $u \leq v$ we have that $s \leq v$.*

So the relation $\leq$ is an order relation *on V_T of the tree T and for this order the root r is the largest element.*

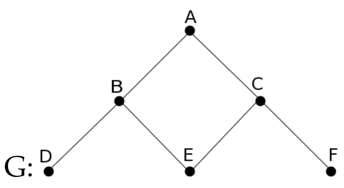
For a rooted tree T we say that for two nodes $u, v \in V_T$, $u \neq v$, v is the parent of u if $u \leq v$, we also say that u is a child of v.
The *leaves* of a rooted tree is defined to be the nodes $u \in V_T$ such that there are no nodes $v \in V_T$, $v \neq u$ such that $v \leq u$.
When each parent of a rooted tree has exactly two children we say that the tree is a *Binary Tree*.

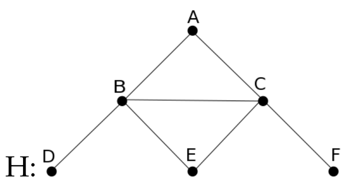
3.3 Math from article

In this part we will look closer at the definitions made in [1].

Definition 3.3.1. *A* Pseudo tree *is a graph G given together with a set $L_G \in V_G$, called the* leafs *of G, so that every cycle of G contains a leaf.*
The other nodes of G, $V_G \setminus L_G$, are called inner nodes of G.

Example Pseudo Tree:





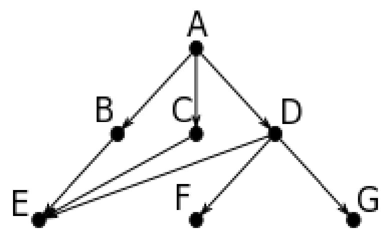
Observe that in a tree there is a definition for what a leaf is, but for a pseudo tree we choose the set of leafs.
Here is an example of two graphs with common leaf set $L_G = L_H = \{D,E,F\}$, the left graph is a pseudo tree while the right one is not.

Definition 3.3.2. *A* Leaf Path *in a pseudo tree G is a path that starts in a non-leaf, ends in a leaf, l, and contains no other leafs than l.*

Definition 3.3.3. *A* Rooted Pseudo Tree *is a* ***directed*** *graph D given together with a vertex $r \in V_D$, called the root, and a set $L_D \subseteq V_D$ (the leafs) every path of maximum length starts in r and ends in a leaf and only leafs have indegree ≥ 2. The vertices of $V_D \setminus L_D$ is called the inner vertices of D.*

Example Rooted Pseudo Tree:

T:



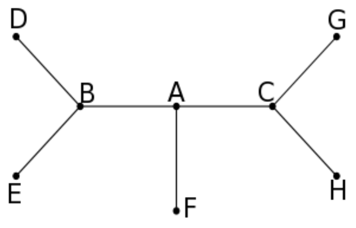
If we choose the leaf set to be $L_T = \{E, F, G\}$ and root $r = A$ this is a rooted pseudo tree.

Definition 3.3.4. *A* Directed Tree *is a rooted pseudo tree D such that the underlying non-directed graph of D is a tree.*

Definition 3.3.5. *A* Binary Pseudo Tree *is a pseudo tree where all internal vertices have degree* 3*.*

Example of a Binary Pseudo Tree:

T:



A Binary Pseudo Tree with leaves $L_T = \{D, E, F, G, H\}$

Definition 3.3.6. *A* Rooted Binary Pseudo Tree *is a directed pseudo tree where one node called the root has indegree* 0 *and outdegree* 2 *and the rest of the internal vertices have indegree* 1 *and outdegree* 2*.*

We can here see that the main difference of a pseudo tree and a tree is that we allow the leafs to have degree $\geq$1, that is they can have multiple ancestors.
The idea with this is if we find species in the gene trees that belong to the same leaf in the species tree we can unite them in the same leaf.

Definition 3.3.7. *Let T be a rooted binary pseudo tree and let $v \in V_T$, then any vertex reachable from v by a directed path is a* descendant *of v (that is v is a descendant of v) and we say that the leaf set $L_T(v) \subseteq L_T$ is the set of leaves of T reachable from v by directed paths.*

Definition 3.3.8. *If T is a directed tree and $L \subseteq L_T$, we define the least common ancestor, **LCA**, of L in T as follows:*

(i) If $L = \{l\}$, then the LCA is l

(ii) Otherwise, the LCA is the vertex v such that

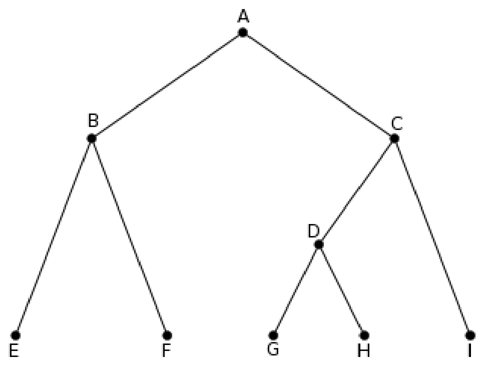
$$L \subseteq L_T(v) \text{ but } L \nsubseteq L_T(u)$$

for any descendant $u \neq v$ of v.

So for the tree T and a set of leafs, $L \in L_T$, we say that the LCA of L is the node v in T from which we can reach all of the leafs in L via directed paths, and we can not reach all leafs in L from any descendant of v.

Example Least Common Ancestor:

T:



Examples of LCA:s for some different leaf sets in T:
LCA$(\{E, F\}) = B$, *LCA*$(\{G, I\}) = C$, *LCA*$(\{E, H\}) = A$

- We denote a disjoint union by $\uplus$.
- By a *gene tree* we mean a rooted binary pseudo tree.
- By a *simple gene tree* we mean a rooted binary tree.
- By a *unrooted gene tree* we mean a binary pseudo tree.
- By a *species tree* we mean a rooted binary tree.

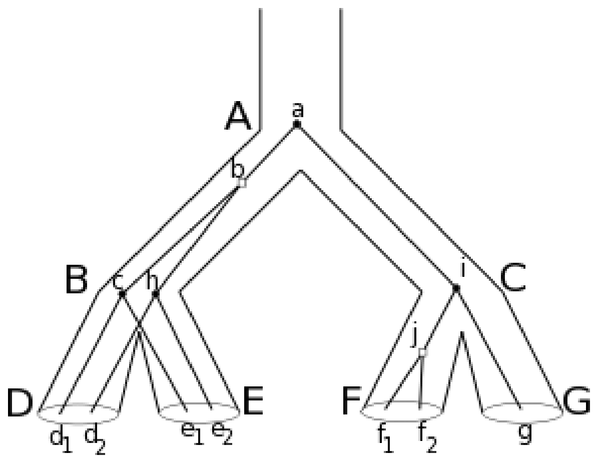
Definition 3.3.9. *We say that a gene tree T and a species tree S are **Compatible** if $L_T \subseteq L_S$.*

Definition 3.3.10. *Let T be a gene tree, S a compatible species tree and let v be a vertex in T.*
An ***LCA Mapping*** *$\phi : V_T \to V_S$ is defined as follows:*

$$\phi(v) \textit{ is the } LCA \textit{ of } L_T(v) \textit{ in } S$$

Assume that T is a gene tree and S a compatible species tree.
Then if we call the set of leaves in a gene tree T reachable from v for M so $M = L_T(v)$. Since T and S are compatible we know that M has a corresponding set in S that we can call N.
So if N has the LCA u in S then $\phi(v) = u$.

LCA-mapping:



The LCA-mappings for some vertices in T:
$\phi(c) = \phi(h) = \phi(b) = B$, $\phi(j) = F, \phi(i) = C$, $\phi(a) = A$

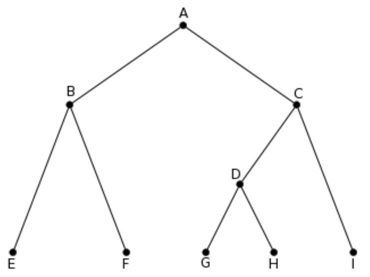
The algorithm described later is based on dynamic programming and to be able to return the optimal solution we must have all the *partitions* of the species tree among the gene trees.

Definition 3.3.11. *Let T be a unrooted gene tree (Binary Pseudo tree), and let $e = (x, y) \in E_T$ be an edge in T.*
We define the ***Partition*** *of e as $\mathcal{P}_T(e) = \{L_1, L_2\}$, where L_1 and L_2 are the leafs reachable by leaf paths in $T \setminus e$ from x and y respectively.*
We denote all partitions of T as $\mathcal{P}_T = \cup_{e \in E_T} \mathcal{P}_T(e) \cup \{L_T\}$.

Similarly if T is a (simple) rooted gene tree, then for each arc $a = \overline{(x, y)} \in A_T$, $\mathcal{P}_T(a) = L_T(y)$ (this is similar to the unrooted case since $L_T(y)$ is the set of leafs reachable from y by leaf paths).
Moreover $\mathcal{P}_T = \{\mathcal{P}_T(a) : a \in A_T\} \cup \{L_T\}$ and for any set of arcs $F \subseteq A_T$, $\mathcal{P}_T(F) = \cup_{e \in F} \mathcal{P}_T(e)$.

Example Partitions:

T:



This tree is described by $T = ((E, F), ((G, H), F))$ and the partitions of T is:

$$\mathcal{P}_T = \{(E, F), (G, H), (G, H, I), (E, F, G, H, I), E, F, G, H, I\}$$

Definition 3.3.12. *Let T be a gene tree and S a compatible species tree. Then we define the* ***Duplication Cost*** *$\theta(T, S)$ like:*

$$\theta(T, S) = |\{x : \exists \overline{(x, y)} \in A_T, \phi(x) = \phi(y)\}|$$

Moreover let $\theta(T_1, \ldots, T_r, S)$ denote the sum $\sum_{i=1}^{r} \theta(T_i, S)$ over all duplications needed for S to explain the gene trees $T_1 \ldots T_r$, and let δ denote the tree S that minimizes $\theta(T_1, \ldots, T_r, S)$, so δ is the optimal species tree.

So $\theta(T, S)$ is the number of duplications needed to explain the gene tree T under the species tree S.
The way to understand this is that whenever we have a duplication they happen in between speciations in a reconciled tree, so at these events the LCA mapping will be the same for both of the duplicated nodes.

4 Algorithm

In this section we will study definitions and the dynamic programming algorithm that computes the number of duplications needed for the optimal species tree to explain the input gene trees.
The algorithm runs in polynomial time which means that the time it takes to make a run is limited and changes with the size of the input.
For computer scientists this is important and the time complexity is proved in [1] but we are satisfied by just making the statement.

When the algorithm is given all the partitions from the optimal species tree among the gene trees it will produce the number of duplications needed for the optimal species tree to describe all the given gene trees.
We will discuss how to get all partitions in the last section.
From now on, in this part, we will assume that all partitions from the gene trees, $\mathcal{P}$, includes the partitions from the optimal species tree $\mathcal{P}_\delta$.
As mentioned it is possible to modify the algorithm so that it produces the optimal specie tree δ but this will not be done here.

4.1 Preparations

Definition 4.1.1. *We say that a species tree S is $\mathcal{P}$-restricted if and only if $\mathcal{P}_S \subseteq \mathcal{P}$*

That is if the partitions of the species tree is a subset of all partitions from all gene trees.

Definition 4.1.2. *Let $\mathcal{P}^{(i)}$ denote the subsets from $\mathcal{P}_T$ with size i, that is:*

$$\mathcal{P}^{(i)} = \{A \in \mathcal{P} : |A| = i\}$$

Also let $\mathcal{P}^{(\leq i)}$ denote the subsets from $\mathcal{P}$ with size $\leq i$, that is:

$$\mathcal{P}^{(\leq i)} = \{A \in \mathcal{P} : |A| \leq i\}$$

4.2 Pseudocode

The algorithm below takes as input a set of genetrees $T_1, \ldots, T_r$ with partitions $\mathcal{P}$ such that the optimal species tree is $\mathcal{P}$-restricted, and computes the number of duplications, $\theta(T_1, \ldots, T_r, \delta)$, needed to explain the gene trees.
In the algorithm d should be interpreted as the number of duplications needed at vertex a in a species tree S with leaf set $L_S(a) = A$ and childrens a_1 and a_2 such that $L_S(a_1) = A_1$ and $L_S(a_2) = A_2$.

```
**Input:** Gene trees $T_1, \dots, T_r$ s.t. $L_{T_i} \subseteq L$ and $\mathcal{P}$ s.t. $\cup_{A \in \mathcal{P}} A = L$ and $L \in \mathcal{P}$
for $i = 1$ to $|L|$ do
  for each $A \in \mathcal{P}^{(i)}$ do
    Let $M(A) := \min\{M(A_1) + M(A_2) + d\}$ where
    $A_1 \uplus A_2 = A,\ A_1, A_2 \in \mathcal{P}^{(\leq i)}$ and
    $d = |\{x : \exists\overline{(x,y)} \in A_{T_i};\ 1 \leq i \leq r;\ L_{T_i}(x), L_{T_i}(y) \subseteq A;\ L_{T_i}(x), L_{T_i}(y) \nsubseteq A_1;\ L_{T_i}(x), L_{T_i}(y) \nsubseteq A_2\}|$
  end for
end for
**Output:** M(L)
```

L includes the partitions from δ and we assumed that the gene trees also includes all of these partitions.
The algorithm always save the results of the previous subproblems $M(A_1)$ and $M(A_2)$ so we need to prove that $M(A)$ is the optimal solution to the subproblem.

Definition 4.2.1. *Let S be a species tree and T a gene tree, also let $A \in \mathcal{P}_S$. We define $\psi_{T,S}(A)$ to be the number*

$$\psi_{T,S}(A) = |\{x : \exists\overline{(x,y)} \in A_T : \phi(x) = \phi(y), \text{and } A \subseteq L_S(\phi(x))\}|$$

(We can think of this as the duplication cost for a subtree of S with leaf set A.)
Moreover we define $\psi_{T_1,\dots,T_r}(A,\mathcal{P})$ to be the minimum of $\sum_{i=1}^{r} \psi_{T_i,S}(A)$, taken over all $\mathcal{P}$-restricted species trees S such that $A = L_S$.

Observation

If the partitions of the optimal species tree δ is a subset of the partitions of the gene trees then:

$$\theta(T_1, \dots, T_r, \delta) = \psi_{T_1,\dots,T_r}(L, \mathcal{P})$$

So an optimal solution to the subproblem is an optimal solution to the original problem.

We show in the following theorem that the algorithm computes the optimal solutions to the subproblems.
Notice that $M(A)$ in the algorithm corresponds to $\psi_{T_1,\dots,T_r}(A,\mathcal{P})$ in 4.2.1

Theorem of minimum number of duplications from [1]:

Theorem 1. *If $A \in \mathcal{P}^{(i)}$ then*

$$\psi_{T_1,\ldots,T_r}(A,\mathcal{P}) = min\{\psi_{T_1,\ldots,T_r}(A_1,\mathcal{P}) + \psi_{T_1,\ldots,T_r}(A_2,\mathcal{P}) + d(A,A_1,A_2))\}$$

where the minimum is taken over all $A_1, A_2 \in \mathcal{P}^{(\leq i-1)}$ such that $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$ and where $d(A,A_1,A_2)$ equals:

$$|x : \exists\overline{(x,y)} \in A(T_i); 1 \leq i \leq r; L_T(x), L_T(y) \subseteq A; L_T(x), L_T(y) \nsubseteq A_1; L_T(x), L_T(y) \nsubseteq A_2|$$

Proof. We start by proving the inequality

$$\psi_{T_1,\ldots,T_r}(A,\mathcal{P}) \leq min\{\psi_{T_1,\ldots,T_r}(A_1,\mathcal{P}) + \psi_{T_1,\ldots,T_r}(A_2,\mathcal{P}) + d(A,A_1,A_2)\} \quad (1)$$

Assume that S_1 and S_2 are $\mathcal{P}$-restricted species trees such that

$$A_1 = L_{S_1}, A_2 = L_{S_2}$$

then

$$\psi_{T_1,\ldots,T_r}(A_1,\mathcal{P}) = \sum_{i=1}^{r} \psi_{T_i,S_1}(A_1)$$

and

$$\psi_{T_1,\ldots,T_r}(A_2,\mathcal{P}) = \sum_{i=1}^{r} \psi_{T_i,S_2}(A_2)$$

respectively according to 4.2.1.
Let S be the rooted binary tree obtained by attaching S_1 as a left subtree and S_2 as right subtree to root a.
Since S_1 and S_2 are $\mathcal{P}$-restricted, and moreover since $A \in \mathcal{P}$, then S is $\mathcal{P}$-restricted.
By the construction $L_S = A$ it is straightforward to verify that

$$\sum_{i=1}^{r} \psi_{T_i,S}(A) = \psi_{T_1,\ldots,T_r}(A_1,\mathcal{P}) + \psi_{T_1,\ldots,T_r}(A_2,\mathcal{P}) + d(A,A_1,A_2))$$

This equality gives (1).

We now prove the inequality

$$\psi_{T_1,\ldots,T_r}(A,\mathcal{P}) \geq min\{\psi_{T_1,\ldots,T_r}(A_1,\mathcal{P}) + \psi_{T_1,\ldots,T_r}(A_2,\mathcal{P}) + d(A,A_1,A_2)\} \quad (2)$$

Assume that S is a $\mathcal{P}$-restricted species tree such that:

$$A = L_S \text{ and } \psi_{T_1,\ldots,T_r}(A,\mathcal{P}) = \sum_{i=1}^{r} \psi_{T_i,S}(A)$$

Let a be the root of S. Let S_1 and S_2 be the left and the right subtrees of S at a; moreover, let $A_1 = L_{S_1}$, and $A_2 = L_{S_2}$. It is straightforward to verify that

$$\psi_{T_1,\ldots,T_r}(A,\mathcal{P}) = \sum_{i=1}^{r} \psi_{T_i,S}(A) = \sum_{i=1}^{r} \psi_{T_i,S_1}(A_1) + \sum_{i=1}^{r} \psi_{T_i,S_2}(A_2) + d(A, A_1, A_2)$$

This equality yeilds (2). $\square$

5 Simulation Of Gene Trees

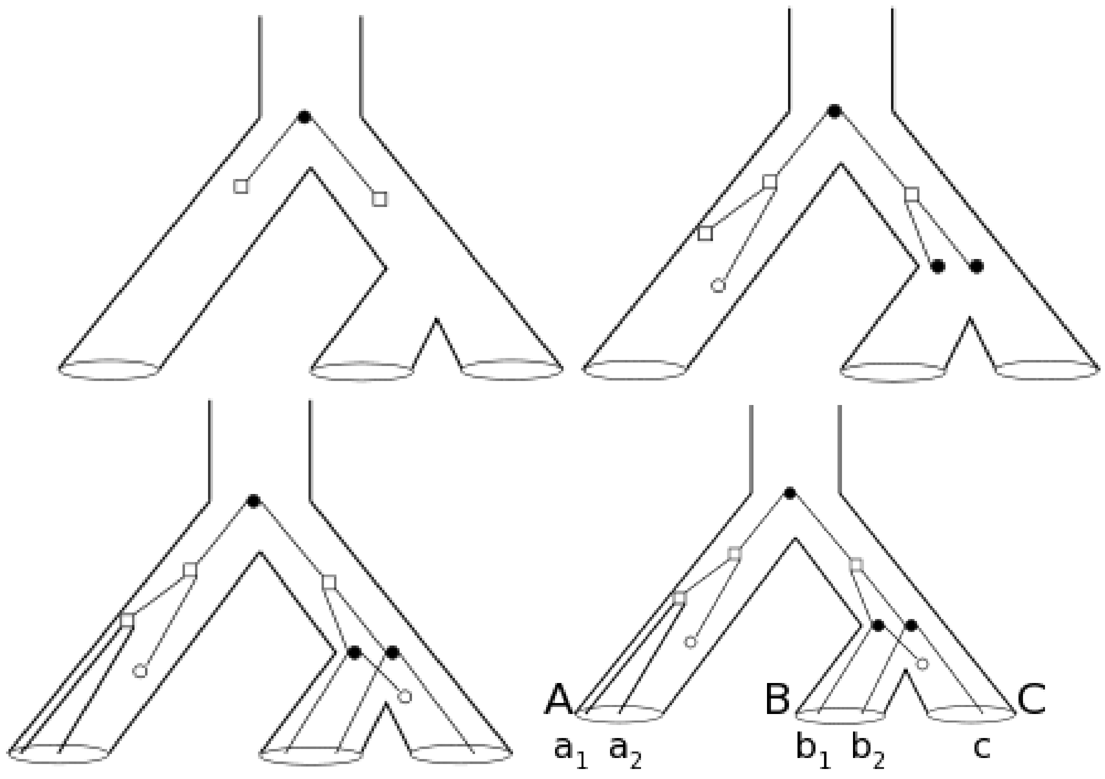
As we have seen earlier in the algorithm part we need all partitions from the optimal species tree among the gene trees, but how can we know anything about the partitions of an unknown species tree?
In the article [1] there is a whole section about how to choose the set of partitions $\mathcal{P}$.
Here we wanted to try another approach by simulating gene families according to an evolutionary model, with different parameters to see how often we found all partitions for a known species tree. We used a program called Aladen that was made by Bengt Sennblad at SBC for simulation of gene trees.

5.1 The Gene Evolution Model

Aladen uses the Gene Evolution Model, GEM described in [8] and [9], to generate gene trees in a species tree, the result is a gene tree with leaves labeled according to the leaves in the species tree.
GEM takes the same parameters as a linear birth- death model described by Kendall [10], that is a birth parameter λ that represents gene duplications and a death parameter μ that represents gene loss, but GEM is modified to develop in the form of trees.
Now consider a species tree S where each arc $\overline{(x,y)}$ is labeled with a time $t(y)$.
GEM let genes evolve along the arcs of S starting from the root of S, when they reach a speciation vertex in S they split into two new genes that evolve independently. At every time step there is a chance for a duplication or loss according to the variables λ and μ.
A duplication give rise to two new genes that develop independently and a loss removes that gene. Finally when all genes reaches the leaves of the species tree we have a gene tree.

Example:

GEM simulating a gene tree in a species tree:

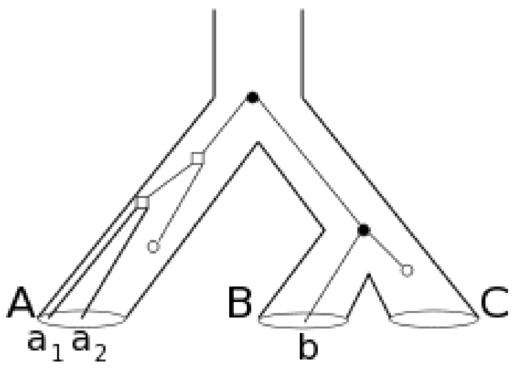


In this simple example we can see how GEM is generating a gene tree inside a given species tree. Filled circles represents speciation events, white squares duplications and white circles are losses.
The leaves in the gene tree are labeled according to which leaf in the species tree it came from.

In this way the generated gene trees can have anything between zero and many leaves in each of the species (leaves) in the species tree. Following the definitions above we let all leafs from one species be represented by one.

This means that if we look at our example above we have a species tree on the form $S = (A,(B,C))$ with partitions $\mathcal{P}_S = \{(B,C),(A,B,C),A,B,C\}$, the generated gene tree is on the form $T = ((a_1,a_2),((b_1,b_2),c))$.
Since we are only interested in finding the partitions from the species tree, a_1 and a_2 gives us the same information, same for b_1 and b_2, so this gene tree gives us all the partitions from the species tree.

An example of the case when we don't find the partitions could be:



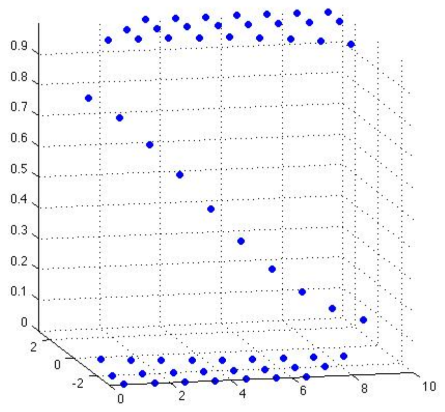
With partitions $\mathcal{P}_T = \{(a,b), a, b\}$

5.2 Experiment

In our simulation we generated genes with GEM in the species tree for Yeast that have 17 leafs.
We generated 100 gene trees at a time and started with low values of the birth/death parameters, for each value of the birth parameter we changed the death parameter in a small interval around the value of birth parameter to see how the evolution of the 100 gene trees behaved.
A program named parsePartition, see appendix, was written in the language Python by the author to treat the data and see how often the generated set of gene trees together had all the partitions from the species tree.
parsePartition also counts how many of the generated trees that in themselves includes all partitions from the species tree and how many of them that were bijections of the species tree.

5.3 Results and Discussion

The simulations were made 10 times for each choice of parameter and then we made Logistic Regression in Matlab to plot the probability of finding all partitions in the species tree among the surviving gene trees.
In the plot below the ”x”-axis, that goes from 0-10, show the birth parameter, the ”y”-axis the difference of the birth- and death parameters and the ”z”-axis show the probability of finding all partitions among the gene trees.

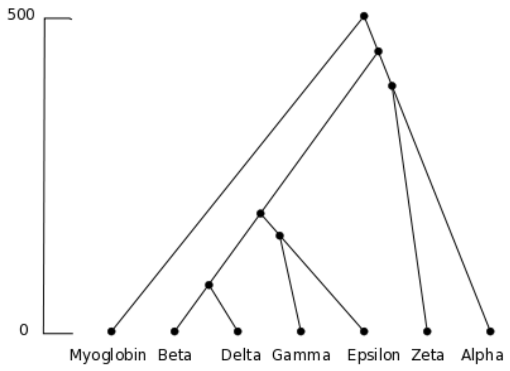


The decreasing line through the system follows $y = 0$, which means that both parameters has the same values.
When discussing evolution, researchers often assume that the parameters in the birth and death process are equal but that must not necessarily be true, so we tried to simulate the evolution for different more or less realistic values of the parameters to see how the trees would behave.

We can see in the picture that when death>birth we almost never find all the partitions, this would mean that some of the species have lost genes during evolution.
When birth>death we can expect to almost always find all partitions from the species tree.
When birth=death we see a decreasing chance of finding all the parameters when the values of the parameters increase.

Evolution of homologous genes are slow and gradual and tends to change about the same rate in many related organisms. The rate of change for different genes depend on how the structure where they are located is constrained, for example the gene that codes for cytochrome c has evolved al-

most as much in animal as in plants and fungi, about 10% change of amino acid sequence every 200 million years. The change of hemoglobins in vertebrates is bout 10% every 55 million years [11].
This can be counted by looking at the fossil record for some species where it is very well preserved and then compare the difference in homologous genes for these species.
Genes from ancient duplications can be found in all life and is therefore a good way of measure evolution.
For example the globin family family of vertebrates that have myoglobin as there anscestral globin and has then developed into the hemoglobins($\alpha, \zeta, \beta, \delta, \gamma$ and ϵ) by mutations.



This is an illustration of the evolution of globins. The Scale is in millions of years, length of edges represent time.

Since we assume that we can recreate the evolutionary tree for species if we can find all the partitions among the gene trees, results like this can help us to estimate our chances to succeed depending on the size of our molecular data.

Future studies might for example be to try and recreate the species tree with different amounts of molecular data and see how often we get the correct tree by using this method.

Appendices

A parsePartition

```
import sys
import os
from string import ascii_letters

letters=ascii_letters
allFound=False
bijection=False
allPartitions = []
tmpTree=[]
realbijection=0
spPartNr=16

def usage():
"""Displays a usertext"""
print """
Usage: parsePartition <Speciestree> <Genetrees> <Resultfile>

This program takes a file with a speciestree and one file
with 0 to many genetrees and checks if the partitions of
the speciestree exists in the genetrees.
It will also tell if there is a bijective image of the
speciestrees among the genetrees."""

def wrongInput():
"""Makes shure that the input has been made in
the correct way"""
length=len(sys.argv)
if length==4:
        return 0
elif length==3:
        return 1
else:
        return 2

def makeDict(speciesTree):
"""Makes a dictionary from the input species tree with
 a boolean for each element."""
f=open(speciesTree,'r')
tree=f.read()
parsePartition(tree,0)
newTree=tuple(allPartitions)
treeDict=dict([(x,False) for x in newTree])
f.close()
return treeDict
```

```
def sortPartition(partition):
"""Takes a partition, delets all doubles and order
the elements by name."""
newPartition=[]
for i in partition:
        if i not in newPartition:
                newPartition.append(i)
newPartition.sort()
return newPartition

def makeIntersection(bigPart,partition):
"""Makes the intersection of two lists with partitions."""
if len(partition)>0:
        for i in partition:
                if i not in bigPart:
                        bigPart.append(i)
return bigPart

def uniteLeafs(partition):
"""Inserts the partition but if it is already found
the new one is deleted."""
newPart=sortPartition(partition)
partStr=''
i=0
while i<len(newPart):
        partStr=partStr+newPart[i]
        i=i+1
if len(partStr)>4:
        newPart=[partStr]
else:
        newPart=[]
return newPart

def getTree(tree,spTree):
"""Picks the trees out of the file, counts them and run
parsePartition.
getTree also checksif all partitions in the speciestree
can be found in one of the genetrees."""
global bijection,tmpTree,realbijection
f=open(tree,'r')
trees=0
bijections=0
for line in f:
        tmpTree=[]
        trees=trees+1
        partitions,i=parsePartition(line,0)
        if checkAll(tmpTree,spTree):
                bijections=bijections+1
                bijection=True
                if len(tmpTree)==spPartNr:
                        realbijection=realbijection+1
                        print tmpTree
                        print "Treenumber:",trees
f.close()
return trees,bijections

def checkAll(partitions,speciesDict):
"""Checks if all the partitions in the speciesTree have
been found."""
for i in speciesDict:
        speciesDict[i]=False
for i in partitions:
        if i in speciesDict:
                speciesDict[i]=True
for i in speciesDict:
        if speciesDict[i]==False:
                return False
return True

def getToken(treeStr, index):
"""Reads the symbol where the index is and returns it or the
leaf if there is one."""
i=index
symbol = treeStr[i]
if symbol == '[':
        while symbol != ']':
                i=i+1
                symbol=treeStr[i]
        if symbol==']':
                i=i+1
                symbol=treeStr[i]
                return symbol,i
elif symbol in letters:
        leaf = ''
        while symbol in letters:
                leaf=leaf + symbol
                i=i+1
                symbol=treeStr[i]
        if symbol == '[':
                while symbol != ']':
                        i=i+1
                        symbol=treeStr[i]
        return(leaf,i)
else:
        return symbol, i

def parsePartition(tree, index_in):
```

```
"""Goes trough a tree with the leaf-partitions and makes a
list of partitions."""
global allPartitions,tmpTree
index=index_in
#To have something when start reading a tree:
symbol=' '
myPart=[]
if len(tree)<=14:
#To get rid of trees with only one leaf.
        return [],0
while symbol != ')':
#When this sign occur we have a partition end.
        index=index+1
        symbol,index=getToken(tree,index)
        if symbol == '(':
#This is the begining of a new partition, so we make a new one.
                newPart,index=parsePartition(tree,index)
                myPart=makeIntersection(myPart,newPart)
                #Add the new partition to original.
                symbol,index=getToken(tree,index)
        elif symbol[0] in letters:
        #This part detects and combine the leaves.
                myPart.append(symbol)
                #Because we found a letter we found a leaf.
                        symbol,index=getToken(tree,index)
                while symbol not in ['(',')']:
                        if symbol[0] in letters:
                                tmpPart=[symbol]
                                #Because makeIntersection
                                #expects two lists.
                                myPart=makeIntersection(myPart,tmpPart)
                                symbol,index=getToken(tree,index)
                        else:
                                index=index+1
                                symbol,index=getToken(tree,index)
                symbol,index=getToken(tree,index-1)
        elif symbol==';':
                return myPart,index
partitionString=uniteLeafs(myPart)
allPartitions=makeIntersection(allPartitions,partitionString)
tmpTree=makeIntersection(tmpTree,partitionString)
return myPart,index+1

def main():
global allPartitions
if wrongInput()==2:
        usage()
        sys.exit(2)
print "Input:",wrongInput()
```

```
spTree = makeDict(sys.argv[1])
allPartitions=[]
trees,bijections=getTree(sys.argv[2],spTree)
name=str(sys.argv[1])+" "+str(sys.argv[2]+"\n")
first="There were "+str(trees)+" genetrees, "+
str(bijections)+ " bijections and "+str(realbijection)+
" true bijections in the file.\n"
if wrongInput()==0:
        filename=sys.argv[3]
        f=open(filename,'a')
        f.write(name)
        f.write(first)
        if checkAll(allPartitions,spTree):
                f.write("-parseParitions found all
                partitions from the speciestree in
                the genetrees.\n")
                f.write("\n")
        else:
                f.write("-parsePartitions did not find all
                partitions in the speciestreee in
                 the geenetrees.\n")
                f.write("\n")
                f.close()
elif wrongInput()==1:
        print name
        print first
        if checkAll(allPartitions,spTree):
                print "-parseParitions found all partitions
                from the speciestree in the genetrees.\n"
        else:
                print "-parsePartitions did not find all
                partitions in the speciestreee in
                the geenetrees.\n"
                print "\n"

if __name__ == '__main__':
main()
```

B Logistic Regression

Matlab code:

```
% Fit the model:
[b dev stats]=glmfit([data(:,1) data(:,4)],data(:,3),'binomial')
% get predicted values:
yhatu=glmval(b,uniquepred,'logit');
% Scatterplot:
scatter3(ZU(:,1),ZU(:,2),ZU(:,3),'filled')
```

References

[1] J. Lagergren and M. T. Hallet, “New algorithms for the duplication-loss model,” *RECOMB 2000*, pp. 138–146, 2000.

[2] R. Page and E. Holmes, *Molecular Evolution A Phylogenetic Approach*. Blackwell Science Ltd., 1998.

[3] M. Zvelebil and O. J. Baum, *Understanding Bioinformatics*. Garland Science, 2008.

[4] M. Goodman and et. al, “Fitting the gene lineage into it’s species lineage:a parsimony strategy illustrated by cladograms constructed from globin sequences.,” *Systematic Zoology*, vol. 28, pp. 132–163, 1979.

[5] R. Guigo, I. Muchnik, and F. T. Smith, “Reconstruction of ancient molecular phylogeny,” *Molecular Phylogenetics And Evolution*, vol. 6, pp. 189–213, 1996.

[6] J.-P. Barthelemy and A. Guenoche, *Trees And Proximity Representations*. John Wiley Sons Ltd., 1998.

[7] N. Biggs, *Discrete Mathematics*. Oxford University Press, 2002.

[8] L. Arvestad, A.-C. Berglund, J. Lagergren, and B. Sennblad, “Bayesian gene/species tree reconciliation and orthology analysis using MCMC,” *Bioinformatics*, vol. 19 Suppl 1, pp. i7–15, 2003.

[9] L. Arvestad, J. Lagergren, and B. Sennblad, “The gene evolution model and computing its associated probabilities,” *J. ACM*, vol. 56, no. 2, pp. 1–44, 2009.

[10] D. G. Kendall, “On the generalized ”birth-and-death” process,” *Ann. Math. Stat.*, vol. 19, pp. 1–15, 1948.

[11] J. Ringo, *Fundamental Genetics*. Cambridge University Press, Cambridge, 2004.