A Unified Approach for Reconstructing Ancient Gene Clusters

Jens Stoye and Roland Wittler

Abstract—The order of genes in genomes provides extensive information. In comparative genomics, differences or similarities of gene orders are determined to predict functional relations of genes or phylogenetic relations of genomes. For this purpose, various combinatorial models can be used to identify gene clusters — groups of genes that are co-located in a set of genomes.
We introduce a unified approach to model gene clusters and define the problem of labeling the inner nodes of a given phylogenetic tree with sets of gene clusters. Our optimization criterion in this context combines two properties: *parsimony*, i.e. the number of gains and losses of gene clusters has to be minimal, and *consistency*, i.e. for each ancestral node, there must exist at least one potential gene order that contains all the reconstructed clusters.
We present and evaluate an exact algorithm to solve this problem. Despite its exponential worst-case time complexity, our method is suitable even for large scale data. We show the effectiveness and efficiency on both simulated and real data.

Index Terms—comparative genomics, gene order, gene cluster, gene cluster reconstruction, phylogeny, parsimony, consistency

1 Introduction

The exploration of phylogenetic relationships of several species yields broad information about their evolution. There are different approaches for the reconstruction of phylogeny or comparison of genomes. On lower levels, the nucleotide or protein sequences can be explored. On a higher level, the order of genes within the genome is analyzed to study its structure, where the term *gene* is generally understood to mean any kind of genomic segment or marker.

Genomes with equal gene content can easily be modeled with permutations, where a unique identifier is assigned to each gene. A convenient way to account for the orientation of a gene within the genome is to use signed permutations, where each gene is represented by a signed integer. Given such a model for the gene order, *gene clusters*, parts of the genome that are conserved during evolution, can be defined to study common structures in related genomes. It has been shown that such conserved regions often contain functionally or evolutionary associated genes: Dandekar *et al.* [1] found that proteins encoded by conserved gene pairs appear to interact physically, and Overbeek *et al.* [2] showed that it is possible to predict functional coupling based on gene clusters. In line with these studies, it has been confirmed that genome comparison can be used to identify conserved clusters of functionally related genes [3], [4]. Lathe, Snel and Bork [5] and Tamames [6] suggested that the selective pressure for co-localization goes even beyond co-expression since they found conserved regions spanning more than one operon. Other causes for gene clusters were discussed by Lawrence and Roth [7], in particular horizontal gene transfer. In this respect, the identification of conserved regions might also provide insight into the evolutionary history of genes.

There exists a variety of different definitions of gene clusters based on different models for gene order. Most cluster models can either be formulated as *r-window* clusters, where a region of given size has to contain at least a specified set of genes, or as *max-gap* clusters, where the distance between the regions containing a specified set of genes is never larger than a given gap size. See [8] and [9] for recent surveys of different concepts of gene clusters. We consider a concept that in general includes all these models.

If different species share a gene cluster, it is likely that this is inherited from a common ancestor. Given the phylogeny and the gene clusters of contemporary species, our goal is to reconstruct putative gene clusters for the ancestral species.

Depending on specific models for the gene order and gene clusters, putative ancestral gene clusters derived from different contemporary species can be in contradiction with each other, which we call a *conflict*. If a reconstructed set of clusters contains a conflicting subset, this would exclude the existence of any potential, ancient gene order that contains all these clusters. Hence, a natural aim for a reconstructed set of clusters is to rule out that it contains any conflicts, i.e. we want to ensure *consistency*. This framework was set by Bergeron *et al.* [10] who present an algorithm that reconstructs sets of *conserved intervals* which are consistent. But their approach does not optimize any objective function. Adam *et al.* [11] introduced *parsimony* as an objective function, requiring that the number of formations and losses of gene clusters is minimized. But their algorithm does not always find an exact solution. In addition, a second heuristical step is added to reach consistency.

In this paper, we define a new objective function for the problem of annotating a phylogenetic tree with sets of gene clusters that includes both parsimony and consistency,

• *The authors are with the Faculty of Technology, Bielefeld University, Germany. E-mail: stoye@TechFak.Uni-Bielefeld.de, roland@CeBiec.Uni-Bielefeld.de.*

Manuscript received 25 Sep. 2008; revised 9 Dec. 2008; accepted 11 Dec. 2008; published online 31 Oct. 2025.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-XXX-XXX.

called the *labeling problem*. As a first result, we elucidate an important relation between these two criteria, which is used to solve the problem. Any most parsimonious labeling can be modified to reach consistency and therefore optimality by deleting clusters which are involved in conflicts. To this end, we introduce a first, simple algorithm to find all conflicting subsets, which is a general, central issue in the context of consistency. Furthermore, as our main result, we present a method that always finds an exact solution for the labeling problem. It is based on the Fitch-Hartigan approach to find a most parsimonious labeling of the tree. Then a bounded search is used to transform this into an optimal labeling. Since the search space can grow exponentially with respect to the genome lengths and we want to provide an exact solution, this search strategy has a exponential worst-case time complexity. Two different search strategies are combined to increase the sensitivity. It should be noted that all presented results and algorithms are oracle based and therefore not restricted to a fixed definition of gene clusters. A detailed evaluation on simulated data using different concrete gene cluster models, and a showcase analysis on real data confirm the effectiveness of the method and our definition of optimality.

The paper has been organized in the following way. In Section 2, we give an abstract definition of gene clusters. We then define some concepts, including *consistency*, based on this definition, and formally introduce the *Labeling Problem*. In Section 3, we describe our exact method for finding an optimal solution. Section 4 shows our evaluation on both simulated and real data, before we finish with some discussions and conclusions in Section 5.

An implementation including some specific gene cluster models is available from the web site: http://bibiserv.techfak.uni-bielefeld.de/rococo/.

2 Gene Cluster Reconstruction

In Section 2.1 we introduce a general concept of gene clusters. The term *consistency* is defined in Section 2.2. It is a central point of the *labeling problem* described in Section 2.3.

2.1 Gene Cluster Model

We define an abstract *gene cluster model* that is both, general enough to allow various definitions of different models of gene clusters, and concrete enough to be used in the presented algorithm. In genome comparison, conserved gene clusters are usually defined on sets of genomes. However, for the moment we define the presence of a cluster for just one genome.

Definition 1 (Gene cluster model): A *gene cluster model* is a triple $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ where

> $\mathcal{U}$ is a finite set, called the universal set, i.e. the set of all possible genomes;
> $\mathcal{C}$ is the set of all possible gene clusters; and
> $\sqsubseteq \subseteq \mathcal{C} \times \mathcal{U}$ is a binary relation that determines whether a gene cluster $c \in \mathcal{C}$ is contained in a genome $g \in \mathcal{U}$.

This general model allows the concrete instantiation of well known cluster definitions like *common intervals* or *conserved intervals*:

Example (Common intervals on unsigned permutations): Let N be the length of the genomes and $\mathcal{G}_N := \{1, \ldots, N\}$ the set of all genes. Then the gene cluster model for *common intervals on unsigned permutations* [12] is defined as $(\mathcal{U}_{comm}, \mathcal{C}_{comm}, \sqsubseteq_{comm})$, where

> $\mathcal{U}_{comm}$ is the set of all permutations on $\mathcal{G}_N$;
> $\mathcal{C}_{comm} := \mathcal{P}(\mathcal{G}_N)$ is the power set of $\mathcal{G}_N$; and
> $c \sqsubseteq_{comm} g :\Leftrightarrow$ all genes in c occur contiguously in genome g.

Note that trivial clusters like singletons and clusters of size N would not provide any deeper insight and will be ignored. Also note that this definition applies to signed permutations similarly by simply ignoring the signs.

A common interval of size two is called an *unsigned adjacency*.

Example (Conserved intervals on signed permutations): Again, let N be the length of the genomes and $\mathcal{G}_N := \{1, \ldots, N\}$ the set of all genes. A conserved interval $[aIb]$, defined on signed permutations, consists of two *extremities* a and b with $|a|, |b| \in \mathcal{G}_\mathcal{N}$, and a set of inner elements $I \subseteq \mathcal{G}_N \backslash \{|a|, |b|\}$. We say that $[aIb]$ is contained in a signed permutation π, if and only if in π a is followed by b or $-b$ by $-a$, and in between these extremities, exactly the elements of I occur with arbitrary signs. Then the gene cluster model for *conserved intervals on signed permutations* [13] is defined as $(\mathcal{U}_{cons}, \mathcal{C}_{cons}, \sqsubseteq_{cons})$, where

> $\mathcal{U}_{cons}$ is the set of all signed permutations over $\mathcal{G}_N$;
> $\mathcal{C}_{cons}$ is the set of all conserved intervals over $\mathcal{G}_N$; and
> $[aIb] \sqsubseteq_{cons} g :\Leftrightarrow [aIb]$ is contained in g.

Example (Signed adjacencies on signed permutations): Again, let N be the length of the genomes and $\mathcal{G}_N := \{1, \ldots, N\}$ the set of all genes. Then the gene cluster model for *signed adjacencies on signed permutations* is defined as $(\mathcal{U}_{adj}, \mathcal{C}_{adj}, \sqsubseteq_{adj})$, where

> $\mathcal{U}_{adj}$ is the set of all signed permutations over $\mathcal{G}_N$;
> $\mathcal{C}_{adj}$ is the set of all pairs (a, b) with $|a|, |b| \in \mathcal{G}_N$; and
> $(a, b) \sqsubseteq_{adj} g :\Leftrightarrow$ in g, a is directly followed by b or $-b$ by $-a$.

A signed adjacency (a, b) is equivalent to a conserved interval with no inner elements $[a\{\}b]$.

Other gene cluster models, e.g. based on sequences or circular permutations, can be defined similarly. Different approaches are discussed in Section 5.

2.2 Consistency

A set of gene clusters can be used to represent a set of genomes: all genomes of the universal set that contain the clusters of the set.

Definition 2 (Genome set): Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model and $C \subseteq \mathcal{C}$ a set of gene clusters. The *genome set* of C, denoted by $\mathrm{GS}(C)$, is the subset of genomes in $\mathcal{U}$ that contain all gene clusters in C:

$$\mathrm{GS}(C) := \{g \in \mathcal{U} \mid c \sqsubseteq g \; \forall \; c \in C\}\,.$$

Example: Assume the gene cluster model for common intervals on unsigned permutations with $N = 5$.

Let $C = \big\{\{1,2\},\ \{2,3\},\ \{1,2,3,4\},\ \{4,5\}\big\}$. Then the genome set of C is:

$\mathrm{GS}(C) = \big\{(1,2,3,4,5),\ (5,4,1,2,3),\ (5,4,3,2,1),\ (3,2,1,4,5)\big\}$.

A naive algorithm to compute $\mathrm{GS}(C)$ would follow the definition and enumerate all possible genomes, and then it would test for each of them whether it contains all clusters in C or not. In general, the cardinality of the universal set can increase exponentially w.r.t. the length of the genomes. Hence, processing all genomes is infeasible for most instances. However, for unsigned permutations, PQ-Trees provide a more sophisticated method to perform this filtering procedure and a compact way to represent its result [14], [15]. Even for signed permutations, a technique shown in [10] allows to use this data structure.

The genome set of a specific set of gene clusters is empty if some of the clusters are in contradiction with others. The following notion, introduced by Bergeron *et al.* [10], formalizes this concept.

Definition 3 (Conflicting set of gene clusters): Let $(\mathcal{U},\ \mathcal{C},\ \sqsubseteq)$ be a gene cluster model and $C \subseteq \mathcal{C}$ a set of gene clusters. C is *conflicting* if $\mathrm{GS}(C) = \emptyset$. C is *minimal conflicting* if it is conflicting and for each $c \in C : \mathrm{GS}(C \backslash \{c\}) \neq \emptyset$.

Note that every conflicting set is either minimal conflicting or it contains minimal conflicting subsets. For a set of gene clusters C, let $\mathrm{Conf}(C)$ be the set of all minimal conflicting subsets of C.

Example: Assume the gene cluster model for unsigned adjacencies on unsigned permutations with $N = 4$.

Let $C = \big\{\{1,2\},\ \{2,3\},\ \{2,4\},\ \{1,3\}\big\}$ be a given set of unsigned adjacencies. C is conflicting, since $\mathrm{GS}(C) = \emptyset$, and it contains exactly two minimal conflicting subsets:

$$C_1 = \big\{\{1,2\},\{2,3\},\{2,4\}\big\}, \text{and}$$
$$C_2 = \big\{\{1,2\},\{2,3\},\{1,3\}\big\}.$$

Following Definition 3, this can be seen, for instance for C_1, as follows:

$$\begin{aligned}
\mathrm{GS}(C_1) &= \mathrm{GS}\left(\big\{\{1,2\},\{2,3\},\{2,4\}\big\}\right) = \emptyset \\
\mathrm{GS}\left(C_1 \backslash \big\{\{1,2\}\big\}\right) &= \mathrm{GS}\left(\big\{\{2,3\},\{2,4\}\big\}\right) \\
&= \big\{(1,3,2,4),(1,4,2,3),\dots\big\} \neq \emptyset \\
\mathrm{GS}\left(C_1 \backslash \big\{\{2,3\}\big\}\right) &= \mathrm{GS}\left(\big\{\{1,2\},\{2,4\}\big\}\right) \\
&= \big\{(1,2,4,3),(3,1,2,4),\dots\big\} \neq \emptyset \\
\mathrm{GS}\left(C_1 \backslash \big\{\{2,4\}\big\}\right) &= \mathrm{GS}\left(\big\{\{1,2\},\{2,3\}\big\}\right) \\
&= \big\{(1,2,3,4),(3,2,1,4),\dots\big\} \neq \emptyset
\end{aligned}$$

Since C_1 and C_2 are the only minimal conflicting subsets of C, we have $\mathrm{Conf}(C) = \{C_1, C_2\}$.

2.3 The Labeling Problem

In a phylogenetic context, given the gene order (and therefore the gene clusters) of contemporary species, we want to infer sets of conserved gene clusters for the ancestral genomes as shown in Fig. 1(a). If we assume that a phylogenetic tree is given and all of the descendants of an ancestral node v share a gene cluster, it is obvious that v should be labeled with this cluster as well. Similarly, if none of the descendants have a specific cluster, the node should not be labeled with that cluster. But what if only a subset of the descendants shows a common structure? And what if different descendants suggest conflicting clusters for the node v?

Since we do not know what exactly has happened during evolution, many answers can be motivated without ever being verified. Even though a reconstruction algorithm may sound very plausible (like the one presented in [10]), a more natural way to ensure explanatory power of the results may be the definition of an objective function that rates a given labeling. This way, we can specify properties of a *good* result and verify whether an algorithm finds one of the *optimal* solutions.

One reasonable and therefore commonly used approach for reconstructing labelings of inner nodes in a given phylogenetic tree is *maximum parsimony*, which assumes that a scenario with a minimal number of changes is most reasonable. In our case, we want to minimize the number of formations and losses of gene clusters.

Definition 4 (Parsimony): Let $(\mathcal{U},\ \mathcal{C},\ \sqsubseteq)$ be a gene cluster model and $T = (V, E)$ be a tree with each leaf $l \in V$ labeled with a set of gene clusters C_l. A labeling $\lambda : V \to \mathcal{P}(\mathcal{C})$ is *most parsimonious* if $\lambda(l) = C_l$ for all leaves and the total weight

$$W(T,\lambda) \ := \sum_{(u,v)\in E} \Big|\big(\ \lambda(u) \cup \lambda(v)\ \big) \backslash \big(\ \lambda(u) \cap \lambda(v)\ \big)\Big|$$

is minimal.

This criterion still allows conflicts, as shown in Fig.1(a) and 1(b). Next, we define a criterion to exclude conflicts.

Definition 5 (Consistency): Let $(\mathcal{U},\mathcal{C},\sqsubseteq)$ be a gene cluster model and $T = (V, E)$ be a tree. A labeling $\lambda : V \to \mathcal{P}(\mathcal{C})$ is *consistent* if for all $v \in V$ the labeling contains no conflicts, $\mathrm{Conf}(\lambda(v)) = \emptyset$.

We consider the problem of finding a labeling that is both parsimonious and consistent.

Definition 6 (Labeling Problem): Let $(\mathcal{U},\mathcal{C},\sqsubseteq)$ be a gene cluster model and $T = (V, E)$ a tree with each leaf $l \in V$ labeled with a set of gene clusters C_l. The *labeling problem* is to find, under all consistent labelings of T, a labeling λ of minimal weight $W(T,\lambda)$. Such a labeling is called *optimal*.

See Fig. 1 for examples of the different criteria.

If a labeling λ of T has minimal weight but is inconsistent, we suppose that it contains too many clusters, and we therefore reach consistency by deleting some conflicting clusters from the labeling of some nodes. The following Theorem 1 verifies that we find an optimal labeling using this strategy. It excludes the existence of any labeling that contains clusters which the parsimonious labeling λ does not include, but nevertheless has a smaller weight. Moreover, Theorem 2 guarantees that *all* optima can be found this way. Fig. 2 illustrates the relationships between the different classes of labelings.

Theorem 1: Let $\lambda^{par} : V \longrightarrow \mathcal{P}(\mathcal{C})$ be a parsimonious labeling for a tree $T = (V, E)$. Then there exists an optimal labeling λ^{opt} with $\lambda^{opt}(v) \subseteq \lambda^{par}(v)$ for all $v \in V$.

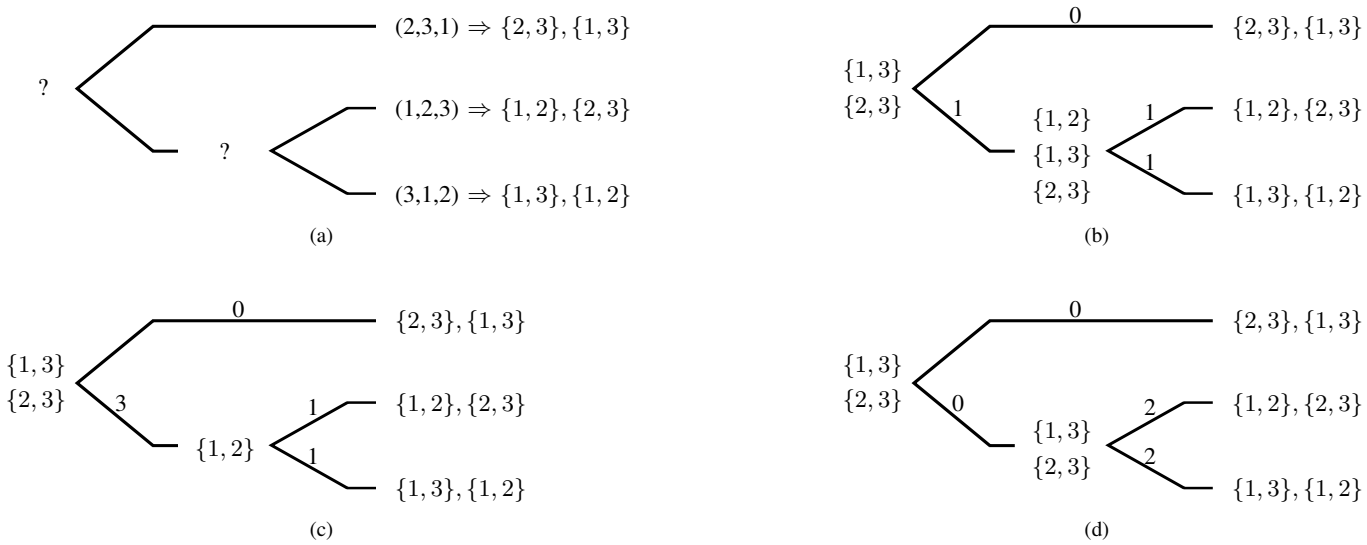


Fig. 1. Example of different labelings for a given instance of the labeling problem. (a) The given tree with labeled leaves and the gene cluster model of unsigned adjacencies with genome length 3. $\mathcal{U}$ = {(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)}; $\mathcal{C} = \big\{\{1,2\}, \{1,3\}, \{2,3\}\big\}$; $\{1,2\} \sqsubseteq$ (1,2,3), $\{1,2\} \not\sqsubseteq$ (1,3,2), $\{1,2\} \sqsubseteq$ (2,1,3), ... (b) A most parsimonious labeling of weight 3 (edges are annotated with their weights). Note that this labeling is inconsistent since $\big\{\{1,2\}, \{1,3\}, \{2,3\}\big\}$ is a (minimal) conflicting set. (c) A consistent labeling of weight 5. Note that this labeling is neither most parsimonious nor optimal. (d) An optimal (i.e. most parsimonious consistent) labeling of weight 4. There is no consistent labeling of smaller weight.

Proof: We consider the labeling for each gene cluster c separately:

$$\lambda_c(v) := \begin{cases} 1 & \text{if } c \in \lambda(v), \\ 0 & \text{otherwise.} \end{cases}$$

For contradiction, assume that for each optimal labeling λ^{opt} there is at least one cluster c satisfying the condition $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par}(v) = 0$ for some node $v \in V$. For such a cluster, consider a maximal connected component of vertices in T which satisfy this condition and let $v_1, \ldots, v_k$ be the neighboring nodes that do not satisfy the condition by definition. See Fig. 3(a) and 3(c) for an example.

For each of these nodes v_i, $1 \leq i \leq k$, one of the following cases must hold:

(i) $\lambda_c^{par}(v_i) = 1$, $\lambda_c^{opt}(v_i) = 1$
(ii) $\lambda_c^{par}(v_i) = 1$, $\lambda_c^{opt}(v_i) = 0$
(iii) $\lambda_c^{par}(v_i) = 0$, $\lambda_c^{opt}(v_i) = 0$

We denote the number of nodes satisfying (i) by k_{11}, and the other quantities by k_{10} and k_{00} respectively, such that $k_{11} + k_{10} + k_{00} = k$, and construct the following labelings:

- Let $\lambda^{par,1}$ be the labeling we get by substituting the inner zeros of λ^{par} by ones (cf. Fig. 3(b)).
- Let $\lambda^{opt,0}$ be the labeling we get by substituting the inner ones of λ^{opt} by zeros (cf. Fig. 3(d)).

Then the following equations hold:

$$W(T, \lambda^{par,1}) = W(T, \lambda^{par}) + k_{00} - k_{10} - k_{11} \quad (1)$$
$$W(T, \lambda^{opt,0}) = W(T, \lambda^{opt}) + k_{11} - k_{10} - k_{00} \quad (2)$$

We now consider the following cases:

Case $k_{00} < k_{11}$: In this case, (1) yields $W(T, \lambda^{par,1}) < W(T, \lambda^{par})$. This contradicts the parsimony of λ^{par}.

Case $k_{00} > k_{11}$: In this case, (2) yields $W(T, \lambda^{opt,0}) < W(T, \lambda^{opt})$. Since the weight of $\lambda^{opt,0}$ is smaller and it is consistent as well, this contradicts the optimality of λ^{opt}.

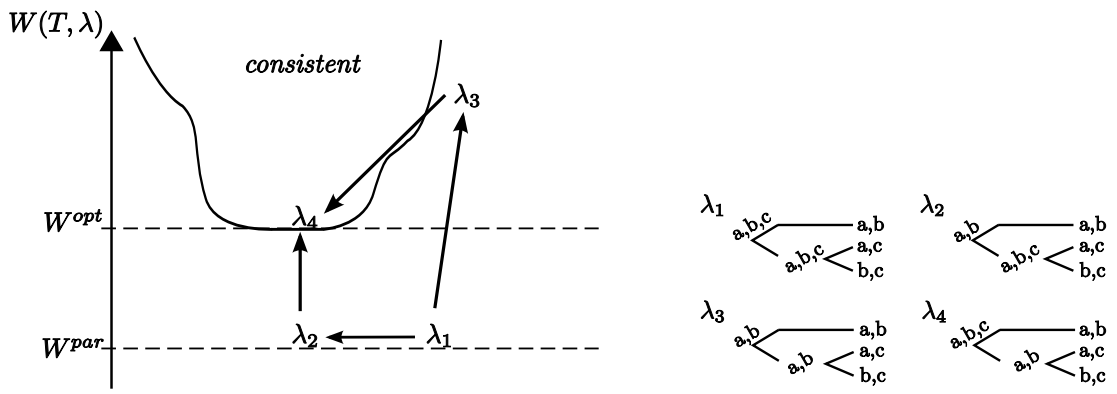
Case $k_{00} = k_{11}$: If $k_{10} > 0$, we again get $W(T, \lambda^{par,1}) < W(T, \lambda^{par})$, which contradicts the parsimony of λ^{par}. Otherwise, we have $W(T, \lambda^{opt,0}) = W(T, \lambda^{opt})$. Hence, $\lambda^{opt,0}$ is optimal. If there is no other node v with $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par,1}(v) = 0$, this contradicts the assumption that there is no such optimal labeling. If there is another such node, we continue the whole argumentation recursively with $\lambda^{opt,0}$ until we eventually end up in one of the above contradictions. □

Theorem 2: Let $\lambda^{opt} : V \longrightarrow \mathcal{P}(\mathcal{C})$ be an optimal labeling for a tree $T = (V, E)$. Then there exists a parsimonious labeling λ^{par} with $\lambda^{opt}(v) \subseteq \lambda^{par}(v)$ for all $v \in V$.

Proof: The proof is similar to the proof of Theorem 1 with switched roles of λ^{par} and λ^{opt}.

Let λ^{opt} be an optimal labeling and assume that for each parsimonious labeling λ^{par} there is at least one cluster c with $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par}(v) = 0$ for some node $v \in V$. Given the same definitions of $\lambda^{opt,0}$ and $\lambda^{par,1}$ as in the proof of Theorem 1, (1) and (2) hold and we get the same contradictions, except for the last case:

Case $k_{00} = k_{11}$: If $k_{10} > 0$, we again get $W(T, \lambda^{par,1}) < W(T, \lambda^{par})$, which contradicts the parsimony of λ^{par}. Otherwise, we have $W(T, \lambda^{par,1}) = W(T, \lambda^{par})$. Hence, $\lambda^{par,1}$ is parsimonious. If there is no other node v with $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par,1}(v) = 0$, this contradicts the assumption



(a) (b)

Fig. 2. Example for the different classes of labelings and their relations. (a) The space of all possible labelings for a given problem instance including four labelings explicitly: λ_1 to λ_4. The labelings are arranged according to their weight $W(T, \lambda)$ and whether they are consistent (e.g. λ_4) or not (e.g. λ_1, λ_2, and λ_3). Within the set of consistent labelings, the ones with minimal weight W^{opt} are optimal (e.g. λ_4). It is always possible to get an optimal labeling by deleting some gene clusters from a most parsimonious labeling of weight W^{par} (e.g. λ_1 or λ_2). A deletion of a cluster is depicted by an arrow. (b) The actual labelings λ_1 to λ_4 for the clusters a, b and c. We assume the set $\{a, b, c\}$ to be minimal conflicting. Hence λ_4 is consistent, whereas λ_1, λ_2 and λ_3 are not.

that there is no such parsimonious labeling. If there is another such node, we continue the whole argumentation recursively with $\lambda^{par,1}$ until we eventually end up in one of the other contradictions. □

3 METHOD

Theorem 1 motivates a strategy to find an optimal labeling by deleting clusters from a possibly inconsistent parsimonious labeling. In order to do this, we have to

1) compute a parsimonious labeling,
2) find the conflicts, and to
3) identify the nodes and the corresponding clusters that have to be excluded.

We will use a variant of the Fitch-Hartigan algorithm to solve the first step (Section 3.1), a recursive strategy to find all minimal conflicting subsets (Section 3.2), and a branch and bound approach for the last step (Section 3.3) — each using the general definition of a gene cluster model. In Section 3.4, we describe how to reconstruct an optimal labeling that preferably contains many clusters.

3.1 Fitch and Hartigan

The subproblem of finding a parsimonious labeling can be reduced to a series of parsimony problems for binary variables. Therefore, we consider the labeling for each gene cluster c separately:

$$\lambda_c(v) := \begin{cases} 1 & \text{if } c \in \lambda(v), \\ 0 & \text{otherwise.} \end{cases}$$

Obviously, a cluster that is not contained in any of the given gene orders will not be reconstructed for any node. Hence, we can restrict the set of all gene clusters $\mathcal{C}$ to the clusters present in the given genomes. For each of these clusters, we calculate a most parsimonious zero-one annotation. *Parsimony* in this case means that we minimize the weight

$$W_c(T, \lambda_c) := \sum_{(u,v)\in E} \left\{ \begin{array}{ll} 1 & \text{if } \lambda_c(u) \neq \lambda_c(v) \\ 0 & \text{otherwise} \end{array} \right\}.$$

Note that the individual weight corresponding to cluster c is denoted by w, whereas the total weight is denoted by W.

Minimizing $w(T, \lambda_c)$ for all clusters c implies minimality for $W(T, \lambda)$ and therefore parsimony of λ:

$$\begin{aligned} \sum_{c\in\mathcal{C}} w(T, \lambda_c) &= \sum_{c\in\mathcal{C}} \sum_{(u,v)\in E} \left\{ \begin{array}{ll} 1 & \text{if } \lambda_c(u) \neq \lambda_c(v) \\ 0 & \text{otherwise} \end{array} \right\} \\ &= \sum_{(u,v)\in E} \sum_{c\in\mathcal{C}} \left\{ \begin{array}{ll} 1 & \text{if } \lambda_c(u) \neq \lambda_c(v) \\ 0 & \text{otherwise} \end{array} \right\} \\ &= \sum_{(u,v)\in E} \Big| \{\, c \in \mathcal{C} \mid \lambda_c(u) \neq \lambda_c(v) \,\} \Big| \\ &= \sum_{(u,v)\in E} \Big| \big(\, \lambda(u) \cup \lambda(v) \,\big) \setminus \big(\, \lambda(u) \cap \lambda(v) \,\big) \Big| \\ &= W(T, \lambda) \end{aligned}$$

Minimizing the individual weight for a given tree is a special case of the well known *Small Parsimony Problem*, which can be solved by the linear time algorithm of Fitch and Hartigan [16], [17]. (For further discussion of the method see [18] and [19].) Notice that the more common algorithm, described by Fitch [16], is less comprehensive than the more generalized version of Hartigan [17], which finds *all* parsimonious labelings and is not limited to binary trees. We use the latter algorithm for zero-one instances as given in Alg. 1.

In lines 11 and 13 of Alg. 1, there may be an arbitrary choice of labeling a node v either with $\lambda_c(v) = 0$ or $\lambda_c(v) = 1$. Preferring $\lambda_c(v) = 1$ yields a labeling containing

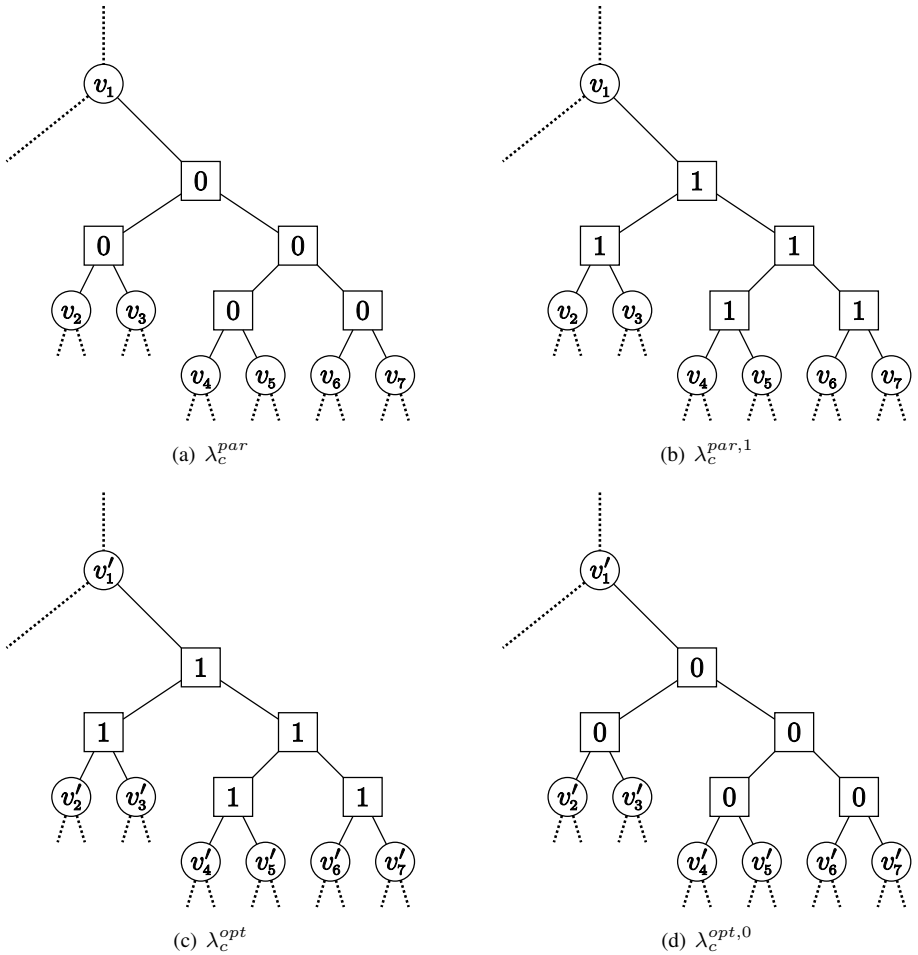


Fig. 3. An example for the connected component used in the proof of Theorems 1 and 2. The same subtree is depicted with the different labelings. The squared nodes are the inner nodes, and the round nodes are the surrounding v_i.

more clusters. However, the more clusters are reconstructed, the more conflicts can occur. We suggest two strategies with different aims. Later, in Section 3.4, we will describe how both can be combined in a two-phase approach. The two strategies are:

Sparse variant: To avoid conflicts in the first place, we choose $\lambda_c(v) = 0$ whenever possible.

Dense variant: To reconstruct as many clusters as possible, we choose $\lambda_c(v) = 1$ whenever possible.

3.2 Finding minimal conflicting sets

Given a set of gene clusters C, a naive way to compute $\mathrm{Conf}(C)$, the set of all minimal conflicting subsets of C, would be exhaustive search: Enumerating all subsets of C and testing for each of them whether it is minimal conflicting. The following observation motivates a more sophisticated approach.

Observation 1: Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model and $C \subseteq \mathcal{C}$ a minimal conflicting set of gene clusters. Then there is no conflicting subset $C' \subset C$.

Based on this, our algorithm for finding all minimal conflicting subsets of a given set of clusters C consists of three steps.

1) First, we identify a conflicting subset $M \subseteq C$ by starting with $M = \emptyset$ and then incrementally expanding M by elements of C until the genome set $\mathrm{GS}(M) = \emptyset$.
2) Now we successively discard elements from M for which $M \backslash \{c\}$ still contains a conflicting set. After this step, we know that $\mathrm{GS}(M) = \emptyset$ and for all $c \in M : \mathrm{GS}(M \backslash \{c\}) \neq \emptyset$. Hence M is a minimal conflicting set by definition.
3) Finally, we have to find all remaining minimal conflicting subsets of C. We start a recursive search for each $c \in M$ searching for minimal conflicting subsets of $C \backslash \{c\}$. This way we exclude the possibility that the same conflicting set might be found again in the recursive calls. Furthermore, Observation 1 guarantees that we do not miss any minimal conflicting set.

This strategy is detailed in Alg. 2.

The efficiency of Alg. 2 depends on the actual gene cluster model. Hence in this general context, we analyze the running time in terms of the number of GS-calculations, which we denote by #GS. Let C be a set of gene clusters and m the number of minimal conflicting subsets of C. If C does not contain any conflict, GS is computed once. Otherwise, in the first two loops GS is computed $\mathcal{O}(|C|)$ times and in the last

Algorithm 1 Fitch-Hartigan for Zero-One Instances
$\mathrm{FitchBin}(T) \longrightarrow \lambda_c$

Given: A tree $T = (V, E)$ with each leaf $l \in V$ labeled with $\lambda_c(l) \in \{0, 1\}$.
Output: A labeling $\lambda_c : V \to \{0, 1\}$ with minimal $w(T, \lambda_c)$.

//Bottom-up phase:
for each leaf l **do**
  $\mathrm{U}(l) := \{\lambda_c(l)\}$
  $\mathrm{L}(l) := \emptyset$
for each unlabeled node v whose children $u_1, \ldots, u_{\ell(v)}$ are labeled **do**
  $k(b) := |\{u_i \mid b \in \mathrm{U}(u_i)\}|$
  $K := \max_{b \in \{0,1\}}\{k(b)\}$
  $\mathrm{U}(v) := \{b \mid k(b) = K\}$
  $\mathrm{L}(v) := \{b \mid k(b) = K - 1\}$
//Top-down refinement:
assign any $b \in \mathrm{U}(r)$ to the root node r: $\lambda_c(r) := b$.
for each unrefined internal node u whose parent node v is already refined to $\lambda_c(v) = a$ **do**
  refine u to $\lambda_c(u) := b$, with any $b \in B(u, a)$:

$$B(u,a) := \begin{cases} \{a\} & \text{if } a \in \mathrm{U}(u), \\ \{a\} \cup \mathrm{U}(u) & \text{if } a \in \mathrm{L}(u), \\ \mathrm{U}(u) & \text{otherwise.} \end{cases}$$

Algorithm 2 Search for Conflicting Subsets
$\mathrm{SearchConf}(C) \longrightarrow \mathrm{Conf}(C)$

Given: A set of gene clusters $C = \{c_1, c_2, \ldots, c_{|C|}\} \subset \mathcal{C}$.
Output: $\mathrm{Conf}(C)$, the set of min. conflicting subsets of C.

$\mathrm{ConfList} \longleftarrow$ empty list
$\mathrm{SearchConfRec}(C, \mathrm{ConfList})$
return ConfList

Procedure $\mathrm{SearchConfRec}(C, \mathrm{ConfList})$

Given: A set of gene clusters $C = \{c_1, c_2, \ldots, c_{|C|}\} \subset \mathcal{C}$.
if $\mathrm{GS}(C) = \emptyset$ **then**
  $M := \emptyset$
  $i = 1$
  while $\mathrm{GS}(M) \neq \emptyset$ and $i \leq |C|$ **do**
    $M := M \cup \{c_i\}$
  for each $c \in M$ **do**
    if $\mathrm{GS}(M \backslash \{c\}) = \emptyset$ **then**
      $M := M \backslash \{c\}$
  if $M \notin$ ConfList **then**
    append M to ConfList
    //recursion
    for each $c \in M$ **do**
      $\mathrm{SearchConfRec}(C \backslash \{c\}, \mathrm{ConfList})$

loop there are $\mathcal{O}(|C|)$ recursive calls with m decreased by one:

$$\begin{aligned} \#\mathrm{GS}(0) &= 1 \\ \#\mathrm{GS}(m) &= \mathcal{O}\big(|C| + |C|\,\#\mathrm{GS}(m-1)\big) \\ &= \mathcal{O}\big(|C|^m\big). \end{aligned}$$

The space requirement S comprises the recursion stack of size $\mathcal{O}(m|C|)$ times some space s_{model} for the input, the output, and the calculation of GS which depends on the actual model and its implementation.

Further research might explore the existence of a more efficient method to compute all minimal conflicting subsets. Even so, the following example concerning adjacencies shows that the number of minimal conflicting subsets can be exponential with respect to the genome length and to the size of C. Similar examples can be constructed for all gene cluster models discussed in this paper. Hence, in general, the time and space requirements of the computation of the minimal conflicting sets is exponential.

Example: Assume the gene cluster model for unsigned adjacencies on unsigned permutations with genome length $N = 2n + 1$ for any $n > 0$. Consider the following set of adjacencies:

$$\begin{aligned} C = &\big\{\{g, g+1\} \mid g = 1, \ldots, N-1\big\} \\ &\cup\ \big\{\{g, g+2\} \mid g = 1, 3, \ldots, N-2\big\} \cup \{1, N\} \end{aligned}$$

We can force each pair of genes g and $g + 2$ (for all $g = 1, 3, 5, \ldots, N - 2$) to be connected in the genome either directly by using the adjacency $\{g, g + 2\}$, or via $g + 1$ using the adjacencies $\{g, g + 1\}$ and $\{g + 1, g + 2\}$. This way, we have 2^n possibilities to build a chain from 1 to N. If we add the adjacency $\{1, N\}$, this would yield a cycle in the genome. Because cycles are not allowed in the model, this is a conflict. Furthermore, if we take out any of the adjacencies, we break the cycle as well as the conflict. Hence, all considered subsets of C are minimal conflicting.

As a result, C contains at least $2^{\lfloor N/2 \rfloor}$ minimal conflicting subsets. This number is also exponential in the size of C since $|C|$ is linear in N.

Sometimes one gene cluster causes multiple conflicts. In the following, we are interested in those clusters that are involved in most of the conflicts.

Definition 7 (Conflict index): Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model, $C \subseteq \mathcal{C}$ a set of gene clusters and $c \in C$. The *conflict index* of c w.r.t. C is defined as:

$$\mathrm{CI}(C, c) := |\{C' \in \mathrm{Conf}(C) \mid c \in C'\}|.$$

Example: Assume the gene cluster model for unsigned adjacencies on unsigned permutations with $N = 4$.

Let $C = \big\{\{1,2\}, \{2,3\}, \{2,4\}, \{1,3\}\big\}$ be a given set of unsigned adjacencies. C contains exactly two minimal conflicting subsets:

$$\begin{aligned} \mathrm{Conf}(C) = \Big\{ &\big\{\{1,2\},\{2,3\},\{2,4\}\big\}, \\ &\big\{\{1,2\},\{2,3\},\{1,3\}\big\}\Big\} \end{aligned}$$

The clusters $\{1, 2\}$ and $\{2, 3\}$ are contained in both minimal conflicting sets. Therefore, their conflict index is $\mathrm{CI}\big(C, \{1,2\}\big) = \mathrm{CI}\big(C, \{2,3\}\big) = 2$. The other two clusters appear just once, hence, $\mathrm{CI}\big(C, \{2,4\}\big) = \mathrm{CI}\big(C, \{1,3\}\big) = 1$.

3.3 Branch and Bound Search

Given an inconsistent labeling, we want to delete some clusters for some nodes until we reach consistency. In this process, we have to find a sequence of deletions that increases the total weight $W(T, \lambda)$ as little as possible. In general, we consider all possible such sequences using a recursive approach: If after a deletion, a consistent labeling is reached, no further deletions are needed and the current labeling is saved as a potential optimum. Otherwise, further deletions are performed in a recursive branch and bound manner: For each cluster $c \in \mathcal{C}$ and for each node v, we make a copy of the labeling, delete cluster c from the labeling of node v, and start a new recursion with the new labeling. In the end, from all potential optima, one with lowest weight is reported as an actual optimum.

Obviously, for a most parsimonious labeling a deletion can not decrease the weight. But in general, during the exploration of the search space, a deletion of a cluster c for a node v could create a labeling whose weight can be decreased by further deletions of the same cluster c in nodes neighboring v. See Fig. 4 for an example. Since we are interested in labelings with lowest possible weight, we directly perform these further deletions in a re-optimization step. To this end, we perform the Fitch-Hartigan algorithm for the current cluster with a fixed value for the actual node: $\lambda_c(v) = 0$. For this, the tree does not have to be traversed completely, but just as long as any changes occur as described in Alg. 3.

The following Lemma guarantees parsimony of the re-optimized labeling w.r.t. the fixed values for deleted clusters.

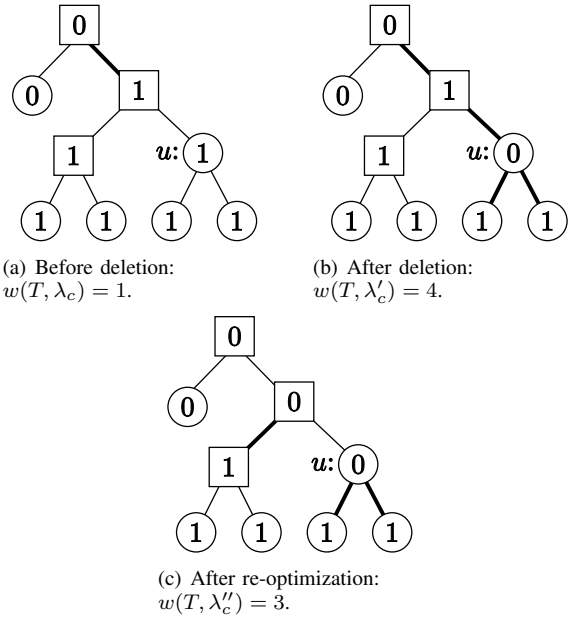
Lemma 1: Let $T = (V, E)$ be a tree with each node $v \in V' \subseteq V$, including all leaves, labeled with a value b_v. The following variant of the Fitch-Hartigan algorithm yields a most parsimonious labeling w.r.t. the given constraints, i.e. $\lambda_c(v) = b_v$ for all $v \in V'$.

- Bottom-up:
  1) If the processed node v is in V', than $\mathrm{U}(v) := \{b_v\}$ and $\mathrm{L}(v) := \emptyset$.
  2) Otherwise, compute $\mathrm{U}(v)$ and $\mathrm{L}(v)$ as defined in Alg. 1.
- Top-Down:
  1) If the processed node v is in V', then $\lambda_c(v) := b_v$.
  2) Otherwise, compute $\lambda_c(v)$ as defined in Alg. 1.

Proof: A node with a fixed value can be seen like a leaf. Hence, the tree with this node as a leaf is parsimonious. The subtree rooted at this node is parsimonious, since the calculation of $B(u, a)$ and the following choice of $\lambda_c(u)$ guarantee parsimony for both possible values of $\lambda_c(v) = a$, independently of $\mathrm{U}(v)$ and $\mathrm{L}(v)$ (cf. [17]).

Hence, the entire tree is parsimonious under the given constraint of the fixed values. □

As in the computation of a parsimonious labeling (Alg. 1), the result of the re-optimization does not have to be unique. Since Theorem 1 also holds for instances that include values fixed to zero, all possible re-optimization results allow to find an optimum by further deletions.



(a) Before deletion: $w(T, \lambda_c) = 1$.
(b) After deletion: $w(T, \lambda'_c) = 4$.
(c) After re-optimization: $w(T, \lambda''_c) = 3$.

Fig. 4. Example for the re-optimization after deleting a cluster. A tree T and different labelings for a specific cluster c are shown. Fixed values are depicted in round nodes. Thin edges add 0 to the weight $w(T, \lambda_c)$ and bold edges add 1. (a) T is labeled with a most parsimonious labeling λ_c with weight $w(T, \lambda_c) = 1$. (b) Deleting c for node u increases the weight to $w(T, \lambda'_c) = 4$. (c) The re-optimization decreases the weight to $w(T, \lambda''_c) = 3$.

Algorithm 3 Re-Optimization after Deletion $\mathrm{ReOpt}(T, u)$

Given: A tree $T = (V, E)$ and a node $u \in V$ in which the labeling changed

fix the new value of u
//bottom-up phase:
for each node v in a bottom-up fashion, starting with u **do**
  perform bottom-up step of Lemma 1
  if $\mathrm{U}(v)$ and $\mathrm{L}(v)$ remain unchanged **then**
    do not process any further parents of v
//top-down phase:
for each node v where
- $\mathrm{U}(v)$ or $\mathrm{L}(v)$ changed during the bottom-up phase, or
- $\lambda_c(p)$ for parent p of v changed during the top-down phase

do
  perform top-down step of Lemma 1

We perform the re-optimization after each deletion, before starting the recursion. This guarantees that the weight can not be decreased in any recursive call. Therefore the current recursion is stopped once the current weight exceeds the weight of the best solution found so far, like shown in Fig. 5.

Obviously, the weight of a (consistent or inconsistent) most

parsimonious labeling or a known optimum (if we want to find further optima) can be used as lower bounds. Once a consistent labeling of such a minimal weight is found, the search can be stopped completely.

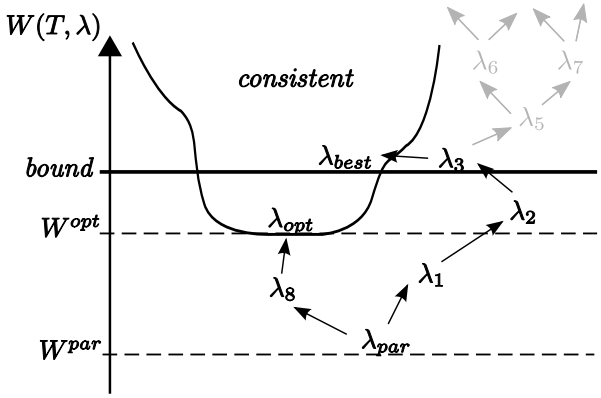


Fig. 5. Example for the bounding. The space of all possible labelings for a given problem instance is shown including same labelings explicitly. The labelings are arranged according to their weight $W(T,\lambda)$ and whether they are consistent (e.g. λ_{best}, λ_{opt}) or not (e.g. λ_1 to λ_8). Once a consistent labeling λ_{best} is found, all labelings with higher weight are discarded (shown in gray).

To find consistent solutions with low weight, to be used as upper bounds, preferably quickly, we start the recursions in a promising order. To annihilate as many conflicts as possible by one deletion, we first compute the conflict index for each node for each cluster and select the candidates in descending order of their conflict index. Moreover, the reoptimization step often causes deletions of the same cluster for neighboring nodes. This again can annihilate more conflicts without a further increase of the weight. Hence, we consider the number of conflicts a cluster causes in the overall tree. Since the exactness of the algorithm does not depend on the order of the recursions, the desired order does not have to be determined exactly. To save time in this sorting step, we just estimate the conflict index instead of computing it exactly. A simple bounding of the recursion depth during the search for conflicting subsets (Alg. 2) already gives a suitable speed up, although the search has to be repeated if not all conflicts have been covered by the preceding search. A limitation to depth two results in a quadratic time complexity for each bounded search. Experiments show that this yields the best tradeoff for estimation quality and running time. Further, more sophisticated estimation strategies are under consideration.

The overall strategy is detailed in Alg. 4.

Theorem 3: Alg. 4 solves the labeling problem exactly.

Proof: The exactness of the algorithm follows from Theorem 1, Lemma 1 and the above description. □

3.4 Two Phase Approach

As mentioned in Section 3.1, the computation of a most parsimonious labeling does not always yield a unique result.

Algorithm 4 Recursive Search for an Optimal Labeling $\mathrm{FindOpt}(T,\lambda_{par}) \longrightarrow \lambda_{opt}$

Given: A Tree T and a most parsimonious labeling λ_{par}.
Output: An optimal labeling λ_{opt}.

```
1: λ_opt ← null, W(T, λ_opt) ← ∞
2: FindOptRec(T, λ_par, λ_opt)
3: return λ_opt
```

Procedure $\mathrm{FindOptRec}(T,\lambda,\lambda_{opt})$

Given: A Tree $T=(V,E)$, a labeling λ, and λ_{opt}, the consistent labeling with the lowest weight found so far.

```
4: if λ is consistent then
5:    λ_opt ← λ
6: else
7:    for each cluster c ∈ C
      in descending order of Σ_{v∈V} CI(λ(v), c) do
8:       for each node v ∈ V
         in descending order of CI(λ(v), c) do
9:          λ'(v) ← λ(v)\{c}
10:         ReOpt(T, v) (Alg. 3)
11:         if W(T, λ') < W(T, λ_opt) then
12:            FindOptRec(T, λ', λ_opt)
```

Depending on the decisions made, the recursive search described in the previous section may give different labelings, all of which are optimal. We already introduced the two extremes, the sparse and the dense variant. On the one hand, the dense variant finds more clusters than the sparse. On the other hand, more clusters accompany more conflicts, which makes the branch and bound search for an optimum slower. We combine the advantages of both: In the first phase, the sparse variant is used to get a parsimonious labeling containing as few conflicts as possible, which can then be solved by the branch and bound search very quickly. In the second phase, the dense variant is used to reconstruct as many clusters as possible. This tends to result in many conflicts, which have to be solved by Alg. 4. But the branch and bound search can be accelerated by the result of the first phase:

- The recursion can be bounded by the known optimal weight (in line 11).
- The search can be completely stopped, once a consistent labeling with the optimal weight is found (in line 4).

The overall procedure is summarized in Alg. 5. Note that each phase separately finds an optimum.

Theorem 2 guarantees that for all optimal labelings there is a parsimonious labeling that can be used to find it. Furthermore, the Fitch-Hartigan algorithm finds all parsimonious labelings if all solutions are reported. Hence, the above procedure can easily be adopted to find *all* optima. Certainly, it would not be practical to actually compute all optima. Nevertheless, since no optimum is excluded a priori, it would be possible to search for some particular optima that obey certain additional criteria.

3.5 Efficiency

For the gene cluster models defined in Section 2.1, one genome of length n contains $\mathcal{O}(n^2)$ gene clusters. Therefore,

Algorithm 5 Two-Phase Approach
$\mathrm{TwoPhase}(T) \longrightarrow \lambda_{dense}^{opt}$

Given: A tree $T = (V, E)$ with each leaf $l \in V$ labeled with a set of gene clusters C_l.
Output: An optimal labeling λ_{dense}^{opt}.

1: *//Phase 1:*
2: $\lambda_{sparse} \longleftarrow \bigcup_{c \in \mathcal{C}}$ sparse variant of $\mathrm{FitchBin}(T, c)$
3: $\lambda_{sparse}^{opt} \longleftarrow \mathrm{FindOpt}(T, \lambda_{sparse})$
4: *//Phase 2:*
5: $\lambda_{dense} \longleftarrow \bigcup_{c \in \mathcal{C}}$ dense variant of $\mathrm{FitchBin}(T, c)$
6: $\lambda_{dense}^{opt} \longleftarrow \mathrm{FindOpt}(T, \lambda_{sparse})$ bounded by $W(T, \lambda_{sparse}^{opt})$

k genomes contain $\mathcal{O}(kn^2)$ clusters. For the computation of a most parsimonious labeling, one Fitch-Hartigan call is used for each cluster. Hence, the running time and space requirements for this first step are in $\mathcal{O}(k^2n^2)$.

However, the computation of an optimal labeling is not polynomial, since the search space for the branch and bound search grows exponentially w.r.t. the number of conflicting clusters.

In our evaluation on simulated data (cf. Section 4), the sparse variant very rarely yields any conflicts. Hence, in most cases the weight of the optimum is equal to the weight of the parsimonious labeling. Therefore the bound for the second phase is very low and the recursion tree is shallow. Hence, the first consistent labeling λ_{sparse}^{opt} is found very quickly, and in the considered case this is optimal. However, for some instances the dense labeling λ_{dense} in the second phase comprises an infeasible amount of conflicts. In this case, we have to be content to compute the sparse optimum only. This already gives suitable results, like detailed in the following section.

4 Results

Since the output of our algorithm consists of gene clusters of extinct species, a direct verification on biological data is hardly possible. Therefore, we use simulated data to evaluate the effectiveness of our method. Furthermore, we show its capability on real data.

4.1 Evaluation on Simulated Data

For this evaluation, we generated simulated gene order data using the program Rose-GEvolutionS, available from the web site http://bibiserv.techfak.uni-bielefeld.de/rose/. In our simulation, we assigned a gene order of length 100 to the root of a balanced, binary tree with eight leaves and performed several different rearrangement operations along the edges. To compare different scenarios, we varied the number of operations per edge and the type of operations (inversions and transpositions with uniform length distribution). Then we used the rearranged gene orders at the leaf nodes and the same tree topology for the reconstruction. We have implemented the gene cluster model for *common intervals*, *conserved intervals*, and *signed* and *unsigned adjacencies*. After the computation of an optimum, for each reconstructed cluster at each node, we checked its existence in the simulated (true) gene order to count the number of true positives (TP) and false positives (FP). We counted the number of clusters which were present in the simulated order, but not reconstructed by the algorithm as false negatives (FN) and the rest as true negatives (TN). Then we calculated the specificity, the sensitivity and the precision (also known as positive predictive value), averaged over ten runs:

$$\text{Specificity} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}}$$

$$\text{Sensitivity} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Independent of the actual setup, the number of simulated and reconstructed clusters was very small compared to the number of all considered clusters. This entailed a very high number of true negatives compared to the number of false positives. Therefore, the specificity was always close to one. Since this is not meaningful, we omitted the commonly used ROC curve, which includes the sensitivity and the specificity. Instead we studied the sensitivity and the precision to answer the questions "How many of the ancestral clusters does our method reconstruct?" and "How many of the reconstructed clusters are correct?".

Since the overall evaluation comprised several simulations for many different parameters and models, we chose the small genome length of 100. Table 1 shows stable sensitivity and precision for increasing genome length for a constant ratio of genome length and evolutionary rate.

TABLE 1
The results of the sparse variant for increasing genome length and evolutionary rate using unsigned adjacencies.

Genome length	Inversions per edge	Precision	Sensitivity	Running time (seconds per run)
100	20	0.99	0.36	0.5
250	50	1.00	0.34	3.2
500	100	1.00	0.34	11.6
1000	200	1.00	0.34	47.5
2000	400	1.00	0.34	199.4
4000	800	1.00	0.35	910.6

Fig. 6 shows the overall results of our method for the gene cluster model of unsigned adjacencies. In Fig. 6 (left), we see a high precision which is slowly decreasing for higher evolutionary distances. The sensitivity is lower and declining faster. In comparison to the sparse optimum, the precision of the dense optimum is a bit lower, but the sensitivity is higher. Thus, reconstructing more clusters comes at the cost of their correctness. Fig. 6 (right) identifies the following relation: The closer a node is to the root of the tree, the greater is the

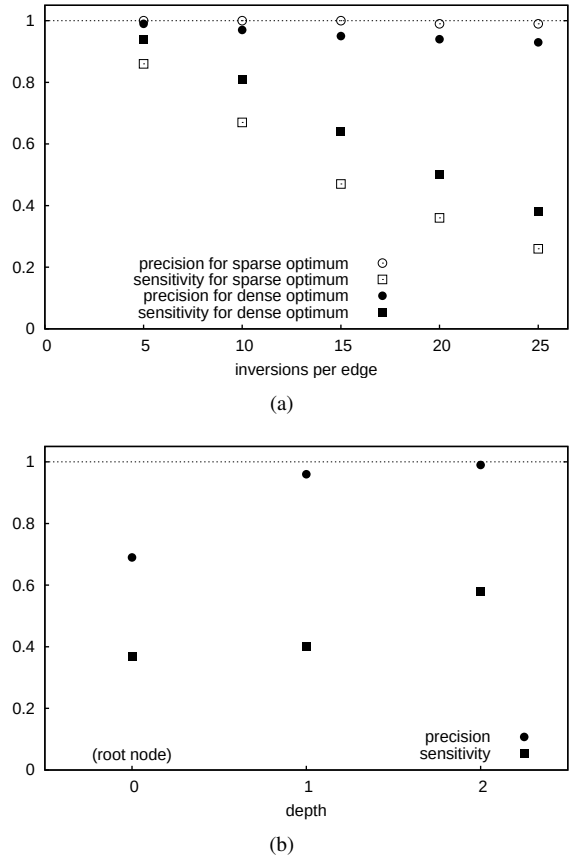


Fig. 6. The precision and sensitivity of Alg. 5 using unsigned adjacencies. (a) Results for sparse and dense optimum for different evolutionary rates. (b) Results for different levels in the tree. (Three inversions per edge.)

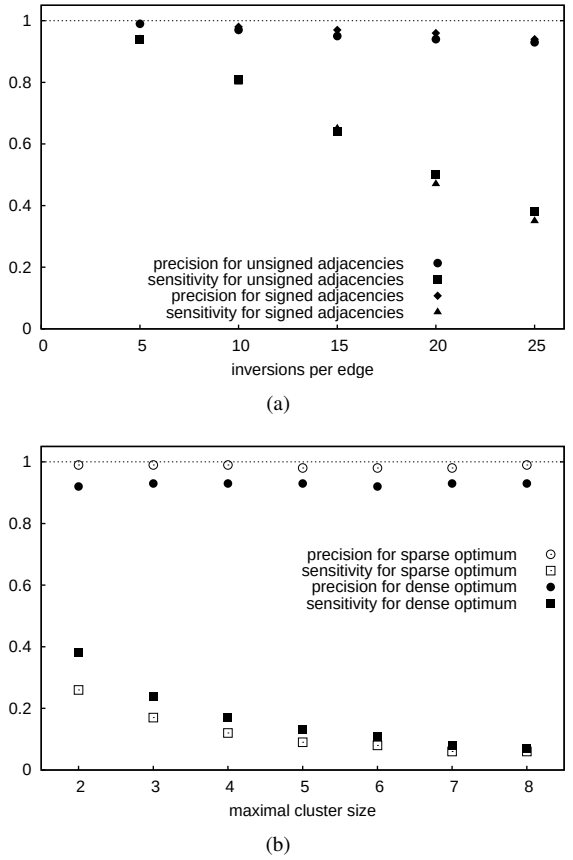


Fig. 7. The precision and sensitivity of Alg. 5 using different gene cluster models. (a) Results for the dense optimum for unsigned adjacencies and signed adjacencies. (b) Results for common intervals of different sizes. (25 inversions per edge.)

distance to the given information at the leaves, and the greater is the inaccuracy.

We got similar results for other gene cluster models. For example, Fig. 7 (left) compares the results for unsigned and signed adjacencies. When we include the orientation, the precision decreases less steeply. Simultaneously, the sensitivity is much lower. We see a similar behavior for clusters with increasing sizes in Fig. 7 (right). In general, the more complex the model (with signs, bigger size), the fewer clusters are found. However, this only affects the sensitivity, not the precision.

In Fig. 8, different rearrangement scenarios are compared. The quality of the results is declining with the amount of transpositions. A reason for this may be the fact that one inversion breaks two adjacencies, whereas a transposition breaks three. An analysis with corresponding evolutionary rates validates this hypothesis (data not shown).

Fig. 9 shows the running time behavior of the two phases of our reconstruction method (Alg. 5). The time for the reconstruction is decreasing for increasing evolutionary distances in both phases (sparse and dense optimum), because less clusters were found and less conflicts had to be solved.

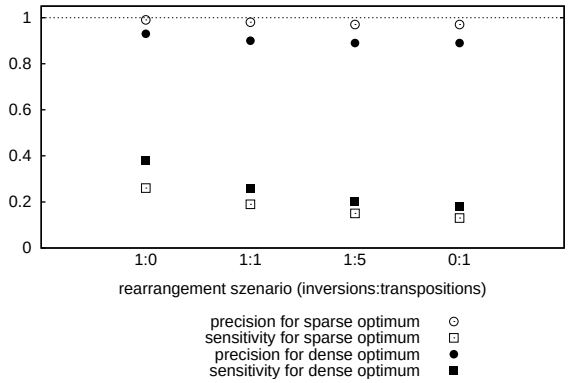


Fig. 8. The precision and sensitivity of Alg. 5 using unsigned adjacencies for different rearrangement scenarios. (25 inversions per edge.)

4.2 Experimental Results on Bacterial Genomes

To illustrate the results that can be achieved on real data, we tested our method on a set of nine bacterial genomes. The underlying data was taken from the NCBI database [20], [21]: We created the phylogenetic tree shown in Fig. 10 using the NCBI Entrez Taxonomy Database, and we extracted the gene

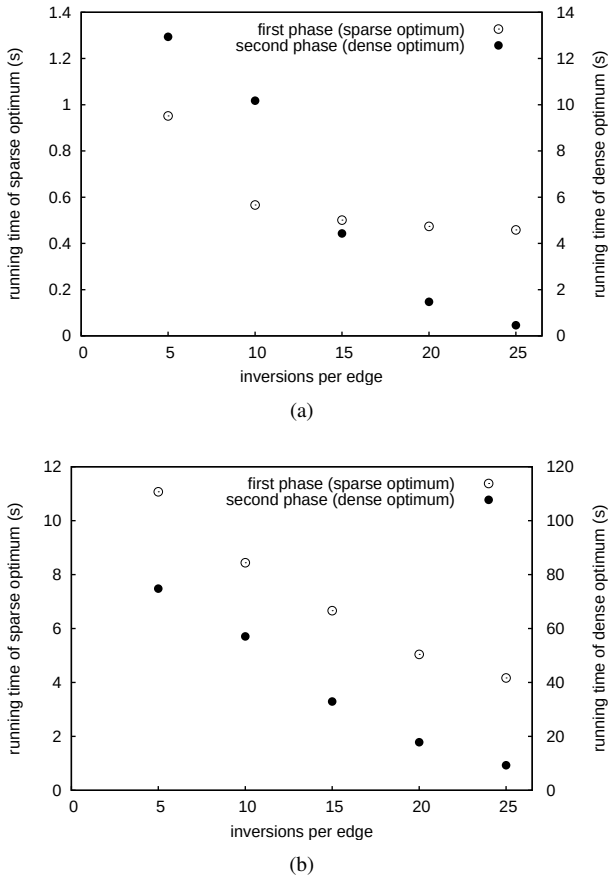


Fig. 9. The running times of Alg. 5 for different evolutionary rates on a 900MHz Sparc processor with 4GB of main memory. (a) Running times for unsigned adjacencies. (b) Running times for signed adjacencies.

order information from the NCBI Entrez Genome Database. As gene orders, we used sequences of *clusters of orthologs* (COGs) [22], [23]. However, this raw sequence data did not fit the cluster models which we have implemented. On the one hand, there were COG numbers occurring more than once in some genomes, and, on the other hand, to some segments of the genome, no or more than one COG number was assigned. Instead of deleting these positions from the sequence, which could induce false clusters, we chose a more rigorous approach. Clusters that contained such a position were not taken into consideration for the reconstruction. To this end, in a preprocessing step, we substituted each of these positions by a marker, whereas consecutive positions were substituted by one single marker. During the scan of a genome sequence for all the clusters it contains, we skipped all clusters that contained a marker. The genomes and the corresponding sequence lengths are listed in Fig. 10.

We computed an optimal labeling using the dense approach for the gene cluster model of *unsigned adjacencies*. The computation took 5 minutes and 30 seconds on a 900MHz Sparc processor with 4GB of main memory.

The result is summarized in Table 2. Often several adjacencies overlap such that the exact order of the corresponding genes is determined. In this case, we merged these adjacencies (e.g. $(a,b),(b,c),(c,d),\ldots$) into one *chain* (e.g. $[a,b,c,d,\ldots]$). The fact that we found only few long chains may have several reasons. First of all, regions with exact conservation of the gene order may be rare. Additionally, due to the preprocessing, any duplicated or ambiguously annotated gene breaks a chain.

TABLE 2
The number of reconstructed clusters for the different internal nodes. For example, for *Corynebacterineae* 121 adjacencies have been reconstructed. These can be merged to 76 chains, 26 of which contain more than two, and four of which contain more than four genes.

	Number of reconstructed clusters			
Node	Adjacencies	Chains	Chains>2	Chains>4
Corynebacterineae	121	76	26	4
Micrococcineae	98	55	19	4
Actinomycetales	97	60	16	5
Bacteria	247	135	59	11
Total	563	326	120	24

Amongst others, we found clusters solely composed of genes which belong to a known gene cluster which was found in *cyanobacteria* and *eubacteria* [24]. The labelings for the different internal nodes regarding these clusters varied only in some details. In Table 3, adjacencies common to all nodes are summarized in chains. The genes of these chains were contained in all genomes and were reconstructed for all internal nodes in the indicated order.

The original complete cluster from [24] appeared in smaller sub-clusters in the reconstruction. On the one hand, as already discussed above, some of these separations were caused by the preprocessing. A more sophisticated preprocessing, such as an orthology assignment of duplicated genes based on this result, would allow to find larger parts of the cluster. On the other hand, some parts of the cluster were separated by several intervening genes in some of the genome sequences. Fig. 11 shows the organization of the reconstructed chains in the input sequences and in the reconstruction results for the internal nodes. This may give hints on how the cluster evolved during evolution and on the function of the intervening genes.

5 Conclusion

We have introduced a unified approach to model gene clusters that allows to define a variety of different specific models. Based on this abstract concept, we have defined the problem of reconstructing sets of gene clusters at the inner nodes of a given phylogenetic tree. The thereby optimized objective function includes two criteria: consistency and parsimony. We could show that it is always possible to find all (consistent) optima using a branch and bound search starting with a (possibly inconsistent) most parsimonious labeling. The presented exact algorithm combines two phases. In the first phase a sparse variant is used to find an optimum very quickly. In a second phase, this result is used to speed up the dense variant, which is more sensitive but still precise.

Due to the correctness of our algorithm, the evaluation of our method refers to the characteristics of our definition of

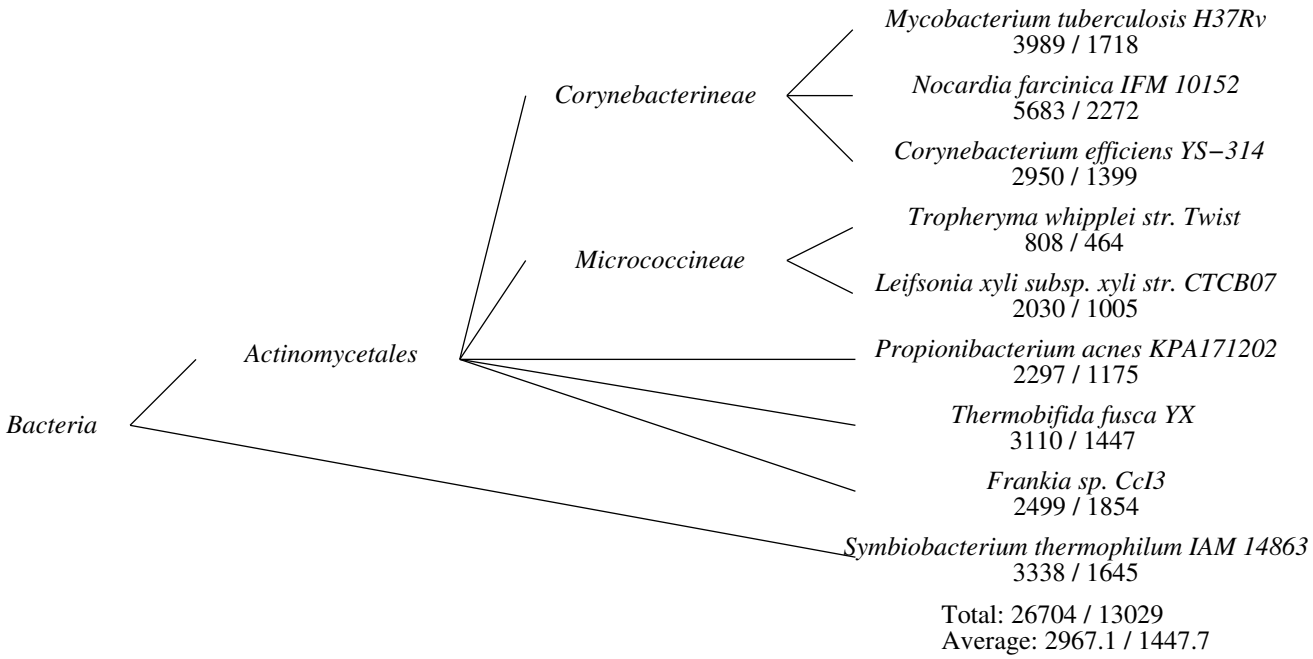


Fig. 10. The phylogenetic tree, names and lengths of the used COG sequences. The lengths are given for the original sequences from the database, and for the preprocessed sequences: before/after preprocessing.

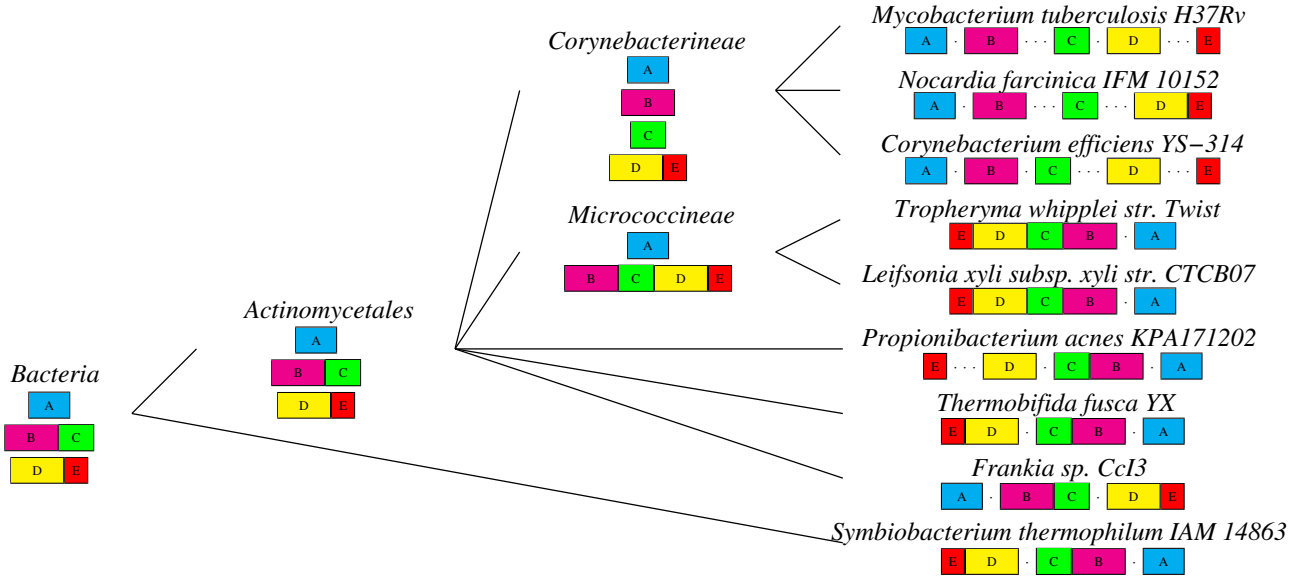


Fig. 11. The organization of the chains listed in Table 3. A dot between chains indicates that these chains are separated by one gene. Three dots indicate a separation by more than one gene (in fact between three and thirty).

optimality. All results satisfy optimality. They span a big range of sensitivity. However, all reconstructed labelings show a high precision, i.e. most of the predictions are correct. This is especially important if results are to be validated biologically, as such analyses require high effort. A high precision makes a biological verification more promising or feasible at all.

So far, our implementation includes adjacencies, common intervals, and conserved intervals on permutations. Given a single gene order for each leaf, it is trivial to compute the corresponding labeling for each leaf, i.e. the set of all clusters contained in the genome. This step can easily be generalized to multichromosomal genomes by using the same computation for each chromosome separately. An implementation of a gene cluster model based on strings without multiplicity as defined by Landau *et al.* [14] (i.e. allowing deletions of genes), or based on circular permutations using PC-Trees [25] is readily possible as well.

Further advanced extensions will include models based on strings with multiplicity (i.e. allowing duplicated genes and gene families). At first sight, the question of consistency might become obsolete if we allow each gene to occur as often as necessary: We could simply concatenate all clusters. However,

TABLE 3
Adjacencies that were contained in all genomes and reconstructed for all internal nodes. The overlapping adjacencies are summarized in chains.

Chain	COG number	Product name
A	0051	ribosomal protein S10
	0087	ribosomal protein L3
	0088	ribosomal protein L4
	0089	ribosomal protein L23
B	0185	ribosomal protein S19
	0091	ribosomal protein L22
	0092	ribosomal protein S3
	0197	ribosomal protein L16/L10E
	0255	ribosomal protein L29
	0186	ribosomal protein S17
C	0093	ribosomal protein L14
	0198	ribosomal protein L24
	0094	ribosomal protein L5
D	0096	ribosomal protein S8
	0097	ribosomal protein L6P/L9E
	0256	ribosomal protein L18
	0098	ribosomal protein S5
	1841	ribosomal protein L30/L7E
	0200	ribosomal protein L15
E	0201	preprotein translocase subunit SecY

this would yield longer and longer gene orders, the more clusters are reconstructed. To avoid this artifact, we would have to bound the multiplicity for each gene. This could be done for each node in the tree by using the maximal multiplicity of a gene in the descendent genomes.

Like most parsimony based approaches, our method suffers from some weaknesses. A main drawback is the simple weighting scheme. However, it is well known that edge and character dependent weight functions can be added to the model [26]. It remains to be seen if our results also apply for such extensions. It is also arguable that all clusters are weighted individually rather than considering whether a cluster is lost completely or just split up into sub-clusters. But, in fact, we do account for this implicitly if the sub-clusters are contained in the model as well. Because either the cluster and all its sub-clusters get lost (and increase the weight), or the cluster and only a few sub-clusters get lost. Further research might explore more sophisticated solutions.

Beside extending the models and the method, our main focus in the future will be on the comparison of different cluster models and the study of the evolution of gene clusters.

ACKNOWLEDGMENTS

The authors wish to thank Pina Krell for the implementation of the gene order evolution simulation Rose-GEvolutionS. The work of Roland Wittler was supported by the DFG Graduiertenkolleg Bioinformatik (GK 635).

REFERENCES

[1] T. Dandekar, B. Snel, M. Huynen, and P. Bork, "Conservation of gene order: a fingerprint of proteins that physically interact." *Trends Biochem Sci*, vol. 23, no. 9, pp. 324–8, 1998.

[2] R. Overbeek, M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev, "The use of gene clusters to infer functional coupling," *Proc. Natl. Acad. Sci. USA*, vol. 96, no. 6, pp. 2896–2901, 1999.

[3] X. He and M. H. Goldwasser, "Identifying conserved gene clusters in the presence of homology families," *J. Comp. Biol.*, vol. 12, no. 6, pp. 638–656, 2005.

[4] S. Böcker, K. Jahn, J. Mixtacki, and J. Stoye, "Computation of median gene clusters," in *Proc. of the 12th Annual International Conference on Research in Computational Molecular Biology, RECOMB 2008*, ser. Lecture Notes in Computer Science, vol. 4955, 2008, pp. 331–345.

[5] W. C. Lathe III, B. Snel, and P. Bork, "Gene context conservation of a higher order than operons," vol. 25, no. 10, pp. 474–479, 2000.

[6] J. Tamames, "Evolution of gene order conservation in prokaryotes," *Genome Biology*, vol. 2, pp. research0020.1–research0027.11, 2001.

[7] J. G. Lawrence and J. R. Roth, "Selfish Operons: Horizontal Transfer May Drive the Evolution of Gene Clusters," *Genetics*, vol. 143, no. 4, pp. 1843–1860, 1996.

[8] R. Hoberman and D. Durand, "The incompatible desiderata of gene cluster properties," in *Proceedings of the 3rd RECOMB Satellite Workshop on Comparative Genomics, RECOMB-CG 2005*, ser. LNBI, vol. 3678. Berlin/Heidelberg: Springer Verlag, 2005.

[9] A. Bergeron, C. Chauve, and Y. Gingras, *Bioinformatics Algorithms: Techniques and Applications*, ser. Wiley Book Series on Bioinformatics. Wiley, 2007, ch. Formal models of gene clusters.

[10] A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve, "Reconstructing ancestral gene order using conserved intervals," in *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics, WABI 2004*, ser. LNBI, I. Jonassen and J. Kim, Eds., vol. 3240. Berlin: Springer Verlag, 2004, pp. 14–25.

[11] Z. Adam, M. Turmel, C. Lemieux, and D. Sankoff, "Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution," *J. Comp. Biol.*, vol. 14, no. 4, pp. 436–445, 2007.

[12] T. Uno and M. Yagiura, "Fast algorithms to enumerate all common intervals of two permutations," *Algorithmica*, vol. 26, no. 2, pp. 290–309, 2000.

[13] A. Bergeron and J. Stoye, "On the similarity of sets of permutations and its applications to genome comparison," *J. Comp. Biol.*, vol. 13, no. 7, pp. 1340–1354, 2006.

[14] G. M. Landau, L. Parida, and O. Weimann, "Gene proximity analysis across whole genomes via PQ trees," *J. Comp. Biol.*, vol. 12, no. 10, pp. 1289–1306, 2005.

[15] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms," *J. Comput. Syst. Sci.*, vol. 13, no. 3, pp. 335–379, 1976.

[16] W. M. Fitch, "Toward defining the course of evolution: Minimum change for a specific tree topology," *Syst. Zool.*, vol. 20, no. 4, pp. 406–416, 1971.

[17] J. A. Hartigan, "Minimum mutation fits to a given tree," *Biometrics*, vol. 29, no. 1, pp. 53–65, 1973.

[18] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. New York, NY: Cambridge University Press, 1997.

[19] L. Parida, *Pattern Discovery in Bioinformatics: Theory and Algorithms*, ser. Mathematical and Computational Biology Series. Chapman & Hall/CRC, 2007.

[20] D. L. Wheeler, C. Chappey, A. E. Lash, D. D. Leipe, T. L. Madden, G. Schuler, T. A. Tatusova, and B. A. Rapp, "Database resources of the national center for biotechnology information," *Nucleic Acids Res.*, vol. 28, no. 1, pp. 10–14, 2000.

[21] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler, "Genbank," *Nucleic Acids Res.*, vol. 28, no. 1, pp. 15–18, 2000.

[22] R. L. Tatusov, E. V. Koonin, and D. J. Lipman, "A genomic perspective on protein families," vol. 278, pp. 631–???, 1997.

[23] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin, and D. A. Natale, "The cog database: an updated version includes eukaryotes," *BMC Bioinformatics*, vol. 4, p. 41, 2003.

[24] M. Sugitaa, H. Sugishitaa, T. Fujishiroa, M. Tsuboia, C. Sugitaa, T. Endob, and M. Sugiura, "Organization of a large gene cluster encoding ribosomal proteins in the cyanobacterium synechococcus sp. strain pcc 6301: comparison of gene clusters among cyanobacteria, eubacteria and chloroplast genomes," *Gene*, vol. 195, no. 1, pp. 73–79, 1997.

[25] W.-L. Hsu and R. McConnell, "PC-trees," in *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sahni, Eds., 2003.

[26] P. L. Erdös and L. A. Székely, "On weighted multiway cuts in trees," *Mathematical Programming*, vol. 65, no. 1, pp. 93–105, 1994.

Jens Stoye studied applied computer science in the natural sciences at Bielefeld University, where he received the PhD degree in 1997 on a topic related to multiple sequence alignment. After postdoctoral positions at the University of California at Davis and the German Cancer Research Center in Heidelberg, he became head of the Algorithmic Bioinformatics Group at the Max Planck Institute of Molecular Genetics in Berlin. Since 2002, Dr. Stoye has been a professor of genome informatics at Bielefeld University.

Roland Wittler recieved the Diploma degree in computer science from Bielefeld University, Germany. He is now a Ph.D. student at the Genome Informatics group in Bielefeld and a scholar of the DFG Graduate College Bioinformatics. His research focuses on cominatorial models for gene clusters.