Accurate prediction of orthologs in the presence of divergence after duplication

Manuel Lafond¹,², Mona Meghdari Miardan¹ and David Sankoff¹

¹Department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada and ²Department of Computer Science, Université de Sherbrooke, Sherbrooke, Canada

Abstract

Motivation: When gene duplication occurs, one of the copies may become free of selective pressure and evolve at an accelerated pace. This has important consequences on the prediction of orthology relationships, since two orthologous genes separated by divergence after duplication may differ in both sequence and function. In this work, we make the distinction between the primary orthologs, which have not been affected by accelerated mutation rates on their evolutionary path, and the secondary orthologs, which have. Similarity-based prediction methods will tend to miss secondary orthologs, whereas phylogeny-based methods cannot separate primary and secondary orthologs. However, both types of orthology have applications in important areas such as gene function prediction and phylogenetic reconstruction, motivating the need for methods that can distinguish the two types.

Results: We formalize the notion of divergence after duplication, and provide a theoretical basis for the inference of primary and secondary orthologs. We then put these ideas to practice with the HyPPO (Hybrid Prediction of Paralogs and Orthologs) framework, which combines ideas from both similarity and phylogeny approaches. We apply our method to simulated and empirical datasets, and show that we achieve superior accuracy in predicting primary orthologs, secondary orthologs and paralogs.

Availability: HyPPO is a modular framework with a core developed in Python, and is provided with a variety of C++ modules. The source code is available at https://github.com/manuellafond/HyPPO.

Contact: milafond2@uOttawa.ca, mmeghd01@uottawa.ca, sankoff@uottawa.ca

1 Introduction

During the course of evolution, speciation and duplication create pairs of homologous genes, which can be classified into two categories. Two genes are orthologs if they descend from an ancestral gene that has undergone speciation, and paralogs if they result from duplication. Distinguishing orthologs from paralogs is of considerable importance in biology, owing to their functional and evolutionary implications (Koonin, 2005; Gabaldón and Koonin, 2013). Notably, the well-known “orthologs conjecture” states that orthologs tend to perform similar functions, in contrast with paralogs, which tend to diverge in their functional role. The reasoning underlying this conjecture is that when a gene creates a copy in the same genome, then this genome has two genes that are able to perform the same function. Hence, one of these copies may become free of selective pressure, resulting in a higher rate of mutations and facilitating a possible change in functionality. We refer the reader to (Studer and Robinson-Rechavi, 2009; Nehrt et al., 2011; Altenhoff et al., 2012; Thomas et al., 2012; Chen and Zhang, 2012) for some recent developments on this conjecture.

The two traditional methods to predict orthology relations are similarity-based and phylogeny-based (see (Kristensen et al., 2011; Altenhoff and Dessiouz, 2012) for a survey). Similarity-based methods aim to partition the set of genes into “groups” of orthologs, under the assumption that genes that share similar sequences are orthologous. The precise notion of a group depends on the particular method used, since in some cases, these groups include multiple genes from the same species (e.g. recent paralogs). For instance in OrthoMCL (Li et al., 2003), Markov clustering is used to identify groups of orthologs and close paralogs. A similar idea is implemented in Proteinortho using spectral clustering techniques (Lechner et al., 2011). The OMA software is somewhat more stringent, and aims to find cliques of pairwise orthologous genes (Roth et al., 2008). This was extended in later versions with the addition of the GETHOGS algorithm (Altenhoff et al., 2013; Train et al., 2017).

The program is able to potentially find all orthologous gene pairs by grouping genes into hierarchical orthologous groups (HOGs), which are the orthologs that belong to a set of species within a given taxonomic range. Some other similarity-based grouping methods and databases include COG (Tatusov et al., 2003), EggNOG (Powell et al., 2011), InParanoid (O’Brien et al., 2005) and OrthoFinder (Emms and Kelly, 2015). The OMG approach (Zheng et al., 2011) identifies sets of pairwise orthologous genes from a homology graph inferred from synteny.
On the other hand, phylogeny-based methods do not attempt in the first instance to group genes together, but rather aim to identify orthologs by inferring the lowest common ancestor (lca) relationships between genes directly. This can be done by inferring a gene tree and identifying its speciation and duplication events using reconciliation with a species tree (Stolze et al., 2012; Ullah et al., 2015). This approach is however sensitive to errors in the gene tree and the species tree. Methods of this type include notably LOFT (Van der Heijden et al., 2005) and COCO-CL (Jothi et al., 2006). Recently, a number of methods exploit a fundamental property of orthology graphs, in which vertices are genes and edges depict orthology: a graph \( G \) is a valid orthology graph if and only if it is \( P_3 \)-free (Hernandez-Rosales et al., 2012; Hellmuth et al., 2013; Lafond and El-Mabrouk, 2014; Hellmuth et al., 2015; Dondi et al., 2017a; Lafond et al., 2016; Jones et al., 2016; Dondi et al., 2017b), i.e. \( G \) has no path on 4 vertices without a shortcut. The essential reason underlying this characterization is that \( P_3 \)-free graphs have a special tree representation called a co-tree, which turns out to be interpretable as a gene tree that depicts all the lca relationships prescribed by \( G \). These methods usually construct an approximate putative orthology graph and perform a minimum number of modifications on the graph so that it becomes \( P_3 \)-free.

The output from similarity-based and phylogeny-based methods can be different, as some orthologies may be irrelevant from the perspective of grouping genes with similar functions, but will be inferred by phylogenetic methods. Consider the example of Figure 1. The node \( z \) represents an ancestral duplication event in which neofunctionalization hypothetically occurred, where one copy (here the left child of \( z \)) retains its original functionality whereas the other copy (the right child of \( z \)) diverges and starts accumulating mutations at a higher rate. As a result of this event, the \( a \) and \( b \) genes remain similar, as no major divergence event separates the two. However, \( a \) and \( c \) will have differentiated significantly, even though the two genes are orthologous (since their lca is a speciation). Consequently, similarity-based methods are unlikely to mark \( a \) and \( c \) as orthologs, as the two genes do not share sequence similarity. From the point of view of functional annotation, this is not problematic, since the \( a \), \( c \) orthology pair essentially behave as paralogs and may differ in function.

This raises the question as to which type of inference is the most relevant: should we focus on the groups of similar orthologs, or do we need all the orthologs? Ideally, both types of orthologies should be predicted, since similarity groups are useful for function-related applications (Doyle et al., 2010), while the complete set of pairwise relations is useful for phylogenetic reconstructions (Hellmuth et al., 2015).

Of course, post-duplication phenomena other than neofunctionalization as described above can arise. For instance, subfunctionalization occurs when both copies preserve the parental function without diverging, and divergent evolution among the descending paralogs (Zhang, 2003), in which case it is possible for both copies to undergo a new rate of mutations. Nevertheless, the phenomenon of a single copy undergoing functional change has been reported to occur frequently (Lynch and Conery, 2000; Jordan et al., 2004; Woods et al., 2013; Soria et al., 2014; Cardoso-Moreira et al., 2016), as duplication models other than neofunctionalization, for instance adaptive radiation or modified duplication, also predict this type of divergence after duplication. We redirect the reader to (Inman and Kondrashov, 2010, Table 1) for more details.

In this work, we are interested in how the task of predicting orthology is affected when divergence after duplication does occur. We propose an orthology prediction framework that makes the distinction between the primary orthologs, the orthologous gene pairs that have not been separated by an event of duplication followed by an increased rate of mutation, and the secondary orthologs, which consist of the pairs of orthologs which have had at least one such event along their evolutionary path. For example in Figure 1, \( \{a, b\} \) are primary orthologs whereas \( \{a, c\} \) are secondary orthologs. Note that the relevance of making this distinction was also discussed in (Swenson and El-Mabrouk, 2012), where the primary orthologs are called soothologs. Also observe that this categorization is different from the co-orthology relationship, which is usually defined with respect to a given node of a gene tree. We formalize this notion of primary orthology and show that they must form cliques in the orthology graph, thereby providing a formal justification of the clustering steps often performed by orthology prediction methods.

Our algorithmic framework is a hybrid approach that makes use of both similarity clustering and phylogenetic reconstruction. More specifically, we use exact graph partitioning techniques to find the primary orthologs, construct a species tree from the orthology clusters found, and use this species tree to obtain the secondary orthologs. The method also puts to practical use the theory of \( P_3 \)-free orthology graphs developed recently. The framework, which we call HyPPO (for Hybrid Prediction of Paralogs and Orthologs), is implemented as a fully modular pipeline in which each task can have its own independent implementation.

We evaluate HyPPO on both simulated and real empirical datasets. We compare our method with OrthoMCL and OMA-GETHOGS - to the best of our knowledge, the latter is the only other program that can distinguish primary and secondary orthologs. We show that HyPPO achieves superior accuracy in the set of predicted pairwise relations, and is better at finding the primary orthology clusters.

2 Methods

In this section, we first introduce the preliminary notions necessary for the understanding of this paper. We then formalize the notion of divergence after duplication in gene trees and its consequences on distinguishing primary and secondary orthologs. We then present the main algorithmic components of the HyPPO framework. Note that due to space constraints, most of the proofs are relegated to the supplementary material.

2.1 Preliminary notions

In this paper, every tree \( T \) is assumed to be rooted and binary, i.e. each internal node has two children, and each node \( v \) has a unique parent \( p(v) \), with the exception of the root, denoted \( r(T) \), which has no parent. The set of vertices of \( T \) is denoted by \( V(T) \) and its set of leaves by \( L(T) \). A node \( u \) of \( T \) is an ancestor of a node \( v \) if \( u \) is on the path from \( v \) to \( r(T) \).

Then \( v \) is a descendant of \( u \), in which case we write \( v \in_{\text{desc}} u \). If \( v \neq u \), we write \( v < u \). If none of \( v \leq u \) nor \( u \leq v \) holds, then \( u \) and \( v \) are incomparable, and we write \( u \perp v \). The least common ancestor of a set of leaves \( X \subseteq L(T) \), denoted \( lca_T(X) \), is the node \( u \in T \) that is the ancestor of every node in \( X \), and that is the most distant from the root. For convenience, we may write \( lca_T(a, b) \) instead of \( lca_T(\{a, b\}) \). For
Accurate prediction of orthologs in the presence of divergence after duplication

3

a graph $G$, a clique is a set of vertices of $G$ in which each pair shares an edge. A $P_3$ is a path on 3 vertices without a chord.

Given a species tree $T$, we map each node of $T$ to a node of $S$ with the lca-mapping $\sigma: V(T) \rightarrow V(S)$ defined as follows: if $g \in L(T)$, then $\sigma(g) = \sigma(g)$, and otherwise $\sigma(g) = \text{lca}_S(\sigma(l) : l \in L(T) \text{ and } l \leq g)$. By a slight abuse of notation, if $C \subseteq \Gamma$, we denote $s(C) = \text{lca}_S(\sigma(C))$. Note that $s$ depends on the species tree $S$. In case of ambiguity, we may write $s(g, S)$ (or $s(C, S)$) to denote the lca-mapping with respect to $S$. An example of lca-mapping is presented later on in Figure 3.

We use the definition of Fitch (Fitch, 2000) for orthology and paralogy.

Definition 1. With respect to a DS-tree $T$, two genes $a$ and $b$ are orthologs if $(\ell(a), b, a) \subseteq \Sigma$, and paralogs if $(\ell(a), b, a) = \emptyset$.

An orthology graph $G = (V, E)$ is a graph in which $V$ is a set of genes and $we \in E$ if and only if $u$ and $v$ are orthologs. An orthology graph $G$ is called DS-consistent if genes that are pairwise orthologues in particular. $C$ may contain at most one gene from any given species.

2.2 Orthology and divergence after duplication

As mentioned in the introduction, duplication may introduce divergence when one of the child copies is redundant. In this section, our goal is to investigate the implications of this phenomenon on the structure of orthology relations. That is, if divergence after duplication occurs, what are the orthologs that are expected to be recovered using similarity-based methods? Or using phylogeny-based methods? We present the tools we will use to devise methods tailored for this type of evolution. As we will demonstrate in the Results section, the methodology developed here can also be used in case that divergence after duplication does not always occur, or even when it does not happen at all.

We formalize divergence after duplication as follows. Let $T$ be a DS-tree. Suppose that for each $\forall v \in V(T)$ node $v$, it has one child $v_1$ which evolved at a normal rate, and another child $v_2$ that underwent an accelerated pace of evolutionary changes. We will say that the $v_2 \rightarrow v_1$ edge is a divergent duplication edge. Now, we will assume that if, for two leaves $g_1$ and $g_2$, there is a divergent duplication edge on the path between $g_1$ and $g_2$, then $g_1$ and $g_2$ will be considered “not similar”. Conversely, we will assume that if there is no such edge on the path between $g_1$ and $g_2$, then we expect no unexpected differentiation between $g_1$ and $g_2$ to be observed, and thus $g_1$ and $g_2$ will be considered “similar”.

Note that here, we have not defined “similarity” in a quantitative manner. For our purposes, we prefer to leave this notion abstract, and we will assume that divergence after duplication leaves behind some traces that can be used to detect similarity versus non-similarity (for instance, one could compare the sequence of a gene to all its homologues in a given species). This is because our goal is to evaluate the consequences of divergence in the orthology inference process, and not necessarily to detect such divergence. We thus prefer to keep the notion of similarity as general as possible. We do note however that in the future, further investigation will be needed to establish concrete methods to recognize similarity.

We now introduce DAD-trees (for Divergence After Duplication).

Definition 2. A DAD-tree $T$ for $\Gamma$ is a DS-tree for $\Gamma$ in which each $\forall$ node $v$ has exactly one child $v'$ such that the edge $vv'$ is marked as divergent, and the other edges of $T$ are marked as non-divergent.

Two genes $g_1, g_2 \in L(T)$ are called similar if there is no divergent edge on the path between $g_1$ and $g_2$ in $T$. Otherwise, $g_1$ and $g_2$ are called non-similar.

It then becomes quite easy to characterize similarity graphs, using the above notion of “similarity”.

Proposition 3. Let $G$ be the graph in which $V(G) = \Gamma$, and two genes share an edge if and only if they are similar with respect to a DAD-tree $T$. Then each connected component of $G$ is a clique.

Therefore, unlike orthology, this notion of similarity is a transitive relation. Proposition 3 then tells us precisely what we should expect from similarity graphs when divergence after duplication always occurs: we should obtain cliques of genes. Moreover, the genes in these cliques are pairwise orthologous, as two paralogous genes cannot appear as “similar” under the above definition. Hence, the primary orthologies resulting from a DAD-tree form the cliques in the similarity graph. For this reason, we may sometimes call orthology clusters the cliques of primary orthologs. This motivates the first essential step of our orthology prediction framework, which is to transform the similarity graph into a cluster graph. However, this will only identify a subset of all orthologies, as there may be inter-cluster orthologies.

One idea might be to infer orthology between the “closest” inter-cluster genes. However, this may introduce invalid relations or inconsistencies. Here, as in (Hellmuth et al., 2015; Lafond et al., 2016), we take the viewpoint that for a set of relations to be correct, there must be an evolutionary scenario that could have given rise to the observed relations. In other words, there should exist a DS-tree that displays the inferred orthologies, and the speciation nodes on this DS-tree should agree with the species history. This notion of agreement with a species tree is defined as in (Lafond and El-Mabrouk, 2014; Jones et al., 2016).

Definition 4. Let $T$ be a DS-tree and $S$ be a species tree. Moreover, let $v \in V(T) \setminus L(T)$ be an $S$ node of $T$, and let $v_1$ and $v_2$ be its children. Then we say that $v$ is $S$-consistent if $s(v) \neq s(v_1)$ and $s(v) \neq s(v_2)$.

Furthermore, we say that $T$ is $S$-consistent if each of its $S$ nodes is $S$-consistent.

In the context of our work, the tree $T$ is usually unknown, and only the orthology graph $G$ is known. We use the existence/non-existence of a tree $T$ corresponding to the relations in $G$ to validate the orthology graph.

Definition 5. An orthology graph $G$ is $S$-consistent if there exists a DS-tree $T$ such that the following two conditions hold:

1. $ab \in E(G)$ if and only if $a$ and $b$ are orthologs in $T$;
2. $T$ is $S$-consistent.

The property of $S$-consistency has been investigated in depth in the recent years, and admits a convenient graph-theoretic characterization. First, we need to define a triplet. Let $T$ be a tree, and let $x, y, z \in L(T)$ be three distinct leaves. We say that $T$ contains the triplet $xy\{z\}$ if $\text{lca}_T(x, y) < \text{lca}_T(x, z) = \text{lca}_T(y, z)$.

Theorem 6 (Lafond and El-Mabrouk, 2014). An orthology graph $G$ is $S$-consistent if and only if the following two conditions hold:

1. $G$ is $P_3$-free, i.e. has no path on 4 vertices without a chord;
2. for every $x, y, z \in V(G)$ such that $\sigma(x), \sigma(y)$ and $\sigma(z)$ are distinct and such that $xy, yz \in E(G)$ but $xz \notin E(G)$, we have that $\sigma(x)\sigma(y)\sigma(z)$ is a triplet of $S$. 
Gene sequences  

1. Similarity graph  
2. Cliques of primary orthologs  
3. Species tree  
4. Inter-cluster orthologs

Fig. 2: The four essential steps of the HyPPO pipeline. First, a similarity graph is constructed from the gene sequences. This graph is refined into orthology clusters, which are cliques of primary orthologs. These clusters are used to infer a species tree, which is then used to find the secondary orthologs (which we also call inter-cluster orthologs).

Therefore, $S$-consistent graphs can be characterized by the absence of $P_s$’s and a set of forbidden $P^*$’s given by the structure of the species tree. Given the set of primary orthology clusters, our goal is to infer the inter-cluster orthologs in a way that the resulting relations could be generated from a DAD-tree in a $S$-consistent manner.

### 2.3 The HyPPO orthology framework.

We may now describe the main ingredients of our orthology inference framework. The flow of the pipeline is illustrated in Figure 2. Starting only from the DNA or amino acid sequences of a given set of genes (or proteins), our ultimate goal is to identify the cliques of primary orthologs, and the orthology relations between these cliques. Recall that this framework is modular, and that an implementation can be provided for each step individually. We first provide an outline of the tasks that each step must perform, and then proceed with the algorithmic details of how steps 2-4 were realized in the default implementation of our framework.

**Step 1.** We start by building a tentative similarity graph, in which two genes share an edge if they are “similar”. The pairwise similarity scores between the genes are typically used for this step. For instance, the graph may contain an edge $xy$ iff the BLAST score between $x$ and $y$ is above a certain threshold. The edges can also be weighted by this score.

**Step 2.** As stated in Proposition 3, we expect the above similarity graph to contain only cliques. In practice, this might of course not always be the case and so in this step, we partition the graph into cliques of pairwise orthologous genes, which are assumed to be primary orthologs. Note that virtually every similarity-based inference method performs this task, although it should be observed that our requirement of pairwise orthology is strict, and no in-paralogs are allowed in the inferred groups.

**Step 3.** In order to obtain the inter-cluster relations in a $S$-consistent manner, a species tree $S$ is needed. If known, this step can be omitted, but otherwise, a species tree can be inferred from the orthology clusters. Indeed, as we will show, some phylogenetic signal can be extracted from these clusters.

Note that in Figure 2, Step 3 points back to Step 2. This is because the framework allows reinferring the orthology clusters, but this time with the additional information of the species tree. The new clusters can then be used to refine the species tree, and this process can be looped until convergence. Although our framework supports such a loop, we did not use it in our experimentations, and leave this exploration for future work.

**Step 4.** In this final step, we assume that the orthology clusters and the species tree are known, but the orthologous gene pairs containing genes from distinct clusters are missing. Our goal is to recover these inter-cluster relations in a way to ensure $S$-consistency whilst preserving the properties of a DAD-tree. That is, we assume that in their history, two genes from two distinct clusters are separated by a divergent duplication edge, and that the predicted orthology relations can be explained by this type of history.

In the rest of this section, we describe how Steps 2, 3 and 4 were implemented in HyPPO.

#### 2.3.1 Prediction of orthology clusters.

As partitioning a graph into cliques is a difficult algorithmic problem, most of the orthology prediction methods that perform clustering use a heuristic in order to deal with, thousands, or even millions, of genes. In our case, we assume that genes are partitioned into families, which do not usually exceed a few hundred genes. In some cases, this makes it possible to use exact algorithms instead of heuristics. In fact, we provide two implementations for gene clustering: an exact method and a greedy heuristic. The exact method can be used on gene families with a reasonable number of genes (up to 200), and the heuristic for the other families.

The exact algorithm solves the cluster editing problem, which asks for the minimum number of edges to add or delete in a given graph so that every connected component of the resulting graph is a clique. We used the BBH graph, in which vertices are genes and edges are Bidirectional Best Hits (BBH), where two genes $x$, $y$ form a BBH if $y$ is the gene of $\sigma(y)$ of maximum score with $x$, and vice-versa. The cluster editing problem is well-studied, and there are multiple methods that can find an optimal solution in an acceptable amount of time, including ILP formulations (Hartung and Hoos, 2015; Böcker et al., 2011) and fixed-parameter algorithms (Böcker et al., 2009). We used the ILP method (see Böcker et al., 2011) for details, ensuring that genes $a$ and $b$ from the same species would never belong to the same cluster by making the cost of adding the $ab$ edge infinite.

The greedy heuristic proceeds in an agglomerative fashion using the pairwise scores. This can be seen as an adaptation of UPGMA with the restriction that clusters with a common species can never be joined. More precisely, at the start we treat each individual gene as a cluster. Then, we merge the two clusters $C_1$ and $C_2$ by maximizing the average score of $a, b$ (i.e. $C_1$ and $C_2$ have the two closest genes among all possible pairs of clusters). The algorithm stops when no more clusters can be merged. Note that we also tested other join criteria, for instance joining the two clusters that minimize the average distance as in UPGMA, but the above algorithm yielded the best results.

#### 2.3.2 Inference of a species tree from orthology clusters.

We now show how orthology clusters may guide the construction of a species tree. In order to avoid confusion between species and clusters, we shall denote species with a hat symbol, e.g. $\hat{C}$ is a species. Consider the example provided in Figure 3. Here, the set of species $\sigma(C_1)$ appearing in cluster $C_1$ forms a subset of the species $\sigma(C_2)$ from cluster $C_2$. This...
Fig. 3: Two orthology clusters $C_1$ and $C_2$, and two putative species trees $S_1$ and $S_2$. The genes are named according to their species, e.g., $\sigma(a_1) = A$. Each internal node $v$ of $T_1$ (resp. $T_2$) is labeled by its lea-mapping $\sigma(v)$ (resp. $\sigma(v)$). The tree $T_1$ is the DAD-tree that could have given rise to $C_1$ and $C_2$ if $S_1$ was the true species tree, assuming that $C_1$ was born at the root of the history of the $C_2$ genes. $T_2$ is the DAD-tree for $C_1$ and $C_2$ under the assumption that $S_2$ is the true species tree. In the case of $T_2$, a loss in the clade $[D, E]$ is required to explain the birth of the $C_1$ cluster at a time prior to $\text{leaves}(C_1) = y'$. Suggests that $C_1$ was born somewhere during the evolution of the $C_2$ cluster. Moreover, the duplication that gave rise to $C_1$ should have occurred in a species that existed at least as far back in time as the lca of $\sigma(C_1)$ in the true species tree $S$. This means that in the true gene tree for $C_1 \cup C_2$, the duplication node $d$ must satisfy $\sigma(d) \supseteq \sigma(C_1)$ (where $\sigma$ is with respect to the true species tree $S$). We argue that this suggests that the true species tree $S$ contains the clade $(A, B, C)$, meaning that some node $u$ of $S$ has exactly $A, B$ and $C$ as its descendant leaves.

To justify this claim, observe that some species trees may require losses to explain the clusters, in a way that is analogous to gene tree and species tree reconciliation (see Doyon et al., 2011) for more details on this topic. Consider the two species trees $S_1$ and $S_2$ from Figure 3. If $S_1$ is the true species tree and $d$ is the duplication node that gave birth to the $C_1$ cluster, then $\sigma(s(d)) \subseteq \sigma(C_2)$, as argued in the previous paragraph. $T_1$ is an example of an $S_1$-consistent DAD-tree that satisfies this condition. If $S_2$ is the true species tree, then $T_2$ does not meet this requirement anymore. The tree $T_2$ does satisfy $\sigma(s(d)) \supseteq \sigma(C_2)$ and $y'$. However, $\sigma(C_1), \sigma(C_2)$ has $A, B, C, D$ and $E$ as its descendants, but $D$ and $E$ do not appear in $C_1$. This can be explained by a gene loss prior to the ancestor of $D$ and $E$, resulting in a loss in the clade $[D, E]$. The tree $T_1$ does not require a loss to be explained, and therefore its corresponding species tree is preferred. Observe that any species tree that contains the $(A, B, C)$ clade leads to a loss-less history, suggesting that this clade should be in the species tree.

Of course, the situation gets more difficult when $\sigma(C_2)$ is not a subset of $\sigma(C_1)$, or when there are more than two clusters. Below, we will consider the problem of reconstructing a species tree from an arbitrary set of clusters while minimizing losses.

Let $C$ be an orthology cluster. For convenience, we will make no distinction between $C$ and its set of species $\sigma(C)$, as this has no bearing on the inference procedure. For a species tree $S$ and $x \in V(S)$, the clade of $x$ is defined as $\text{clade}(x) = \{ y \in L(S) : x \text{ is a descendant of } y \}$. The set of clades of $S$ is $\text{clades}(S) = \{ \text{clade}(x) : x \in V(S) \}$. For a given cluster $C$, let $S_C$ be the subtree of $S$ rooted at $\text{leaves}(C)$. The number of losses of $C$ with respect to $S_C$, denoted $\ell_C(S_C)$, is defined as the minimum number of clades to remove from $\text{clades}(S_C)$ so that the union of the remaining clades is exactly $C$. Another way to view $\ell_C(S)$ is as follows. Call a non-root node $v$ of $S_C$ maximal C-free if $\text{clade}(v) \cap C = \emptyset$. Then $\ell_C(S)$ is the number of maximal $C$-free nodes in $S_C$. As an example, the reader can verify in Figure 3 that $\ell_C(S) = 1$ and $\ell_C(C_2) = 0$. We may now formally define our species tree reconstruction problem.

**The Species Tree Using Clusters (STUC) problem:**

**Given:** a set of orthology clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$.

**Find:** a species tree $S$ that minimizes $\sum_{i=1}^{k} \ell_C(S_C)$.

Denote by $OPT(\mathcal{C})$ the minimum number of losses of a species tree for the STUC instance $\mathcal{C}$. Note that $\mathcal{C}$ might not be sufficient to determine the complete species tree exactly, as in Figure 3 where any species tree containing the $(A, B, C)$ clade is optimal. However, the more clusters there are, the more precisely $S$ can be inferred.

We do not know whether the STUC problem is NP-hard, although we suspect that it is. Nevertheless, we borrow ideas from (Mirarab et al., 2014) in order to limit the space of possible species trees to those containing a predefined set of splits. For $X \subseteq \Sigma$ and $A, B \subseteq X$ we consider $X$ a partition of $X$ into two non-empty subsets. We say that $S$ contains the split $(A, B)$ if there is a node $v$ with children $v_1, v_2$ such that $A = \text{clade}(v_1)$ and $B = \text{clade}(v_2)$. Suppose that we are given a set of allowable splits $\mathcal{X} = \{A_1, B_1, \ldots, A_k, B_k\}$, where for each $i$, $A_i$ and $B_i$ are disjoint subsets of $\Sigma$. Then we may ask for a species tree $S$ that solves the STUC problem under the condition that every split contained in $\mathcal{X}$ appears in $S$. In the supplementary material, we show that this restricted version can be solved in polynomial time by dynamic programming, and then we discuss how a reasonable set $\mathcal{X}$ can be constructed.

**Theorem 7.** The STUC problem with species trees restricted to the allowable splits $\mathcal{X}$ can be solved in time $O(|\mathcal{X}|^2 OPT(\mathcal{C}))$.

**2.3.3 Inference of inter-cluster orthology relations.**

In this final step, we assume that the orthology clusters $\mathcal{C}$ and the species tree $S$ are known, but the orthologous gene pairs containing genes from distinct clusters are missing. Our goal is to recover these inter-cluster relations in a way to ensure $S$-consistency whilst preserving the properties of evolution under divergence after duplication. Note that unlike the last section, here we do not confound a cluster $C$ and its set of species $\sigma(C)$.

Consider the orthology clusters $C_1$ and $C_2$ in Figure 3. As we have argued in the previous section, the ancestral gene of $C_1$ was born off a duplication node $d$ satisfying $\sigma(d) \supseteq \sigma(C_1)$. One implication of this is that if $\sigma(g_2) < \sigma(C_1)$, then $g_2 \in C_2$, and it is not possible for a gene of $C_1$ to be orthologous to $g_2$. As an example, take the gene $g_2 \in C_2$ in Figure 3, and assume that $S_1$ and $T_1$ are the true species tree and gene tree, respectively. We have $\sigma(g_2) < \sigma(C_1)$, and so $g_2$ is orthologous with no gene of $C_1$. However, this is not true for $d_2 \in C_2$ for example, because $d_2$ is a secondary ortholog with all of $C_1$. This is because $\sigma(d_2) \supseteq \sigma(C_1)$.

In our implementation, when inferring inter-cluster relations between $C_1$ and $C_2$, we simply make as many pairs of genes orthologous whilst preserving $S$-consistency. As above, we also require that if $\sigma(g_2) < \sigma(C_2)$, then $g_2$ is paralogous to all the genes in $C_2$ (and vice-versa). This results in Algorithm 1. This works as described above in the case of $k = 2$ clusters. For larger $k$, we merge the clusters iteratively in a greedy manner.

More specifically, we first pick the cluster $C$ for which $\sigma(C)$ is the lowest in the species tree, breaking ties arbitrarily. We then ask: what other cluster $C_j$ could have spawned the $C$ cluster? We call $C_j$ the favorite cluster of $C$, and assume that we are given a function $\text{fav}(C, \mathcal{C})$ that finds the favorite cluster of $C$ among a set of clusters $\mathcal{C}$. We then merge $C$ with $C_j$ by making all possible orthologies - namely, we make $e_j \in C_j$ orthologous to every gene of $C$ iff $\sigma(e_j) \supseteq \sigma(C)$. Afterwards, $C_j$ is redefined as $C_j \cup C$, and we proceed with the next cluster. We show that Algorithm 1 preserves $S$-consistency as desired.
Algorithm 1 Algorithm to infer inter-cluster orthologs.
1. procedure inferInterClusters(G, %= (C₁, . . . , Cn), fsc) → %
2. G is the initial orthology graph, % is the set of clusters,
3. and fsc is a function that finds favorites. We assume that in
4. %, two vertices share an edge iff they belong to the same cluster.
5. The output is an orthology graph obtained from G.
6. while % has at least two non-empty clusters do
7. Let Ci ∈ % such that s(Ci) ≤ s(Cj) or s(Ci) = s(Cj)
8. holds for every Ci ∈ %.
9. Let Cj = fsc(Ci, %).
10. for gj ∈ Cj such that σ(gj) ⊆ s(Ci) do
11. Add an edge in % between gj and every vertex in Cj.
12. Cj ← Cj ∪ C and % ← %

Theorem 8. The orthology graph obtained from Algorithm 1 is S-consistent.

In our implementation, we used the following simple fsc function: for each x ∈ Ci, find the
gene y ∈ Γi, C of maximum score. Then x votes for the cluster that contains y. The favorite cluster of C is then the one
with the highest number of votes. Only the members of the genes initially
in %, as given in the input, are allowed to vote.

3 Results

We compare the default implementation of the HyPPO pipeline with two
other orthology prediction methods: OMA-GETHOGS and OrthoMCL.
To our knowledge, OMA-GETHOGS is the method that is the most
similar to ours, as it outputs both the 1-to-1 orthology groups, and the
pairwise orthology relations. The groups can be interpreted as the primary
orthologs. As for OrthoMCL, it is a similarity-based clustering method
that focuses on grouping orthologs, and so it is not expected to find
all the secondary orthologs. Nevertheless, we include it for the sake of
comparison, as OrthoMCL is one of the most popular orthology prediction
tools. HyPPO was tested under two settings: one in which the species tree is
unknown, and the other in which the true simulated species tree is provided.
OMA-GETHOGS was provided the true species tree in all experiments.¹

We performed our experiments on both simulated and real empirical
data sets. In the simulated datasets, the true orthologs and the primary
orthologs are known. This is not true for the real datasets. As (Altenhoff
et al., 2013), we used a set of gold standard, manually curated, gene
trees from the SwissTree database (Wapinski et al., 2007). These
trees are annotated with duplication and speciation events, and so
the underlying orthology/paralogy relations can be considered as “gold
standard” relations. We did not, however, include the analysis of the
primary vs secondary orthologs, as those are not determined in the
real datasets. Note that in both simulated and real datasets, the genes
were already separated into homologous families. All the programs were
executed on the separated gene families, thereby avoiding predictions
between non-homologous genes. For HyPPO, the sequences were aligned
with MAFFT (Katoh et al., 2002), and the similarity values for the first
step were obtained from the pairwise identity percentages computed from
these alignments (number of identical base pairs divided by the length of
the longest sequence, not counting gaps).

¹ The program does not require a species tree as input, but OMA-
GETHOGS produced an error message when not given one - for reasons
that are yet to be determined. Nevertheless, better results are expected
when the true species tree is known.

3.1 Performance metrics

In our experiments, we address two questions: is the set of all pairwise
relations predicted correct? And, can the method distinguish the primary
orthologs from the secondary orthologs? For the pairwise relations, we
used precision, recall and accuracy as our performance metrics. The
precision P and recall R calculated from the orthology relations as follows:
P = TP /
FN + FP and R = TP /
FN + TP. Here, TP means True Positive (the number of relations correctly predicted as orthology), FP means False Positive (a paralogy relation incorrectly predicted as orthology) and FN means False Negative (an orthology relation left undetermined or predicted as paralogy). As for accuracy, it
consists in the total number of relations predicted correctly (including both
orthologs and paralogos) divided by n, the total number of relations².

In the simulated datasets containing divergent duplication edges, we
also evaluated the ability to recover the primary orthologs. The metric we
used is the cluster score defined as follows: let P₁ and P₂ be two partitions
of a given set of elements. Here, P₁ would be the predicted
primary orthologs, and P₂ the “true” simulated primary orthologs. Take a
graph in which the vertices are P₁ ∪ P₂ (i.e. one vertex for each subset),
add an edge x₁x₂ between each pair x₁ ∈ P₁, x₂ ∈ P₂ and weigh this
edge by |x₁ ∩ x₂|. Then the cluster score between P₁ and P₂ is a
value of the maximum matching in this graph, divided by the number of
genes. Hence the cluster score is 1 if P₁ and P₂ are equal. The idea
behind this score is that this value yields the proportion of elements from
P₁ that must be moved from one set to another in order to obtain P₂ (if the
score is c, then a fraction of 1 − c elements from P₂ need to be moved).

3.2 Simulated datasets

For our simulations, we used SimPhy (Mallol et al., 2015) to generate
a set of species trees along with a set of gene trees evolving within
these species trees. As SimPhy incorporates speciation and duplication
events in its simulations, the true sets of orthologs and paralogs are
known. We generated 40 species trees and 400 gene trees - 10 gene
families for each species tree. The number of species in each species
tree was chosen uniformly at random between 30 and 50. The gene trees
were subject to the events of speciation, duplication and losses. We then
used the INDELIBLE (Fletcher and Yang, 2009) module provided with
SimPhy to simulate the evolution of gene sequences on each gene tree.
The nucleotide evolutionary model was selected at random for each gene
gene family, each evolving under a General-Time Reversible model (GTR)
with rates sampled from a 6-dimensional Dirichlet distribution. The only input
to the programs was the set of extant gene sequences and, depending on
the method, the true simulated species tree.

In order to evaluate the impact of the rates of duplications, losses and
substitutions, we used four sets of parameters for the simulations,
yielding four datasets named as follows: Standard, Eventful, Fast and
Slow. Standard uses default parameters, Eventful has an increased rate
of duplications and losses, Fast has high mutation rates and Slow has low
mutation rates. For each dataset, 10 species trees were generated. In both
the Standard and Eventful datasets, the tree-wide substitution rate was
set to 0.000005 (the default value in SimPhy). In the Default dataset,
the duplication and loss rates were set to 0.000005 each, and to 0.000001 in
the Eventful dataset. Higher duplication rates led to trees with too many
genes in the trees (averaging in the thousands, making our analysis too
computationally intensive) while lower values yielded gene trees with too
few paralogos. In the Fast dataset, the duplication/loss rates were set as in

² Note that Ortho-MCL does not predicts paralogs, aside from the recent
ones. In our computation of accuracy, we interpreted as paralogy the set
relations not predicted as orthology - otherwise, leaving these relations as
undetermined made the accuracy too low for comparison.
the Standard setting, but the tree-wide substitution rate was increased to 0.00005. In the Slow dataset, this parameter took value 0.0000005. The Fast dataset led to difficulties in alignments, due to the presence of many substitutions and very large gaps, whereas the Slow dataset had a small amount of substitutions, resulting in sequences with fewer differences.

These simulations do not consider the phenomenon of divergence after duplication. That is, the children of a node in the generated gene trees do not differentiate from each other, and both copies continue to evolve under the same model. Therefore we also simulated divergent edges by choosing, for every node in the simulated gene trees, one child edge arbitrarily and multiplying its length by some factor $\ell$. Thus for each gene tree and each $\ell \in \{1, 2, 8, 50\}$, we used INDELIBLE to generate a set of extant gene sequences obtained by multiplying the length each divergent edge by $\ell$. Each gene family gave rise to four sets of sequences, resulting in a total of 1600 gene families to analyze. In the cases where $\ell > 1$, the primary orthologs can be deduced from the divergent duplication edges.

The average accuracy, precision, recall and cluster scores, taken over the set of all simulated gene trees, for every possible value of $\ell$, are achieved at $\ell = 50$. STD stands for Standard and EV for Eventful. The comparison of precision and recall is shown in Figure 7 for $\ell = 8$. The other values of $\ell$ yielded similar results. One can see that OMA typically attains better precision values. This is at the cost of a much lower recall, as HyPPO outperforms the other methods in this aspect. Therefore, the orthologs predicted by OMA are truly orthologs slightly more often, but the method tends to miss more of the orthology relations.

### 3.3 Empirical datasets

We used 8 gene trees from the gold standard SwissTree database. This included the Popeye domain family (POP), the NOX 'ancestral-type' subfamily NADPH oxidases (NOX), the V-type ATPase beta subunit (VATB), the Serine incorporator family (SERC), the Sulfatase-modifying factor 1 family (SUMF), the HOX cluster genes family 9-14 (HOX), the Asterix family (ARX) and the Cited family (CITE). These gene families
were chosen because they belong to Eukaryotes and, according to the curators, are not suspected to have undergone horizontal gene transfer.

Figure 6 shows the accuracy of each method for each gene tree. Contrary to the simulated datasets, HyPPO does not outperform OMA on every single dataset, as it performs poorly on the VATB and SERC families. Notably, OrthoMCL attains almost perfect accuracy on the VATB family (0.96) whereas HyPPO is not much more than half as accurate (0.59). The reason appears to be that most gene pairs are orthologous (95.7%), since the duplications are low in this tree. HyPPO infers multiple orthology clusters joined together by duplications, resulting in many false paralogs (we predict 47% pairs as orthologous), whereas OrthoMCL puts almost all the genes into one big orthology cluster, making its predictions more accurate. A similar phenomenon appears to occur with SERC. Nevertheless, HyPPO achieves similar or superior accuracy on the other trees, and so it remains competitive. Observe that as in the simulations, when HyPPO is given the species tree (provided by SwissTree), the accuracy increases.

As for precision and recall, the four methods attained similar values. Once again, OMA offers slightly better precision at the cost of lower recall. The averages over all 8 trees are presented in Table 2.

4 Discussion

Although the results of HyPPO are promising, there is still room for improvements. Each step of the pipeline could be implemented in a more sophisticated manner, and the DAD model around which HyPPO is built could be more realistic. When a duplication node is not followed by accelerated mutations, there is no special post-duplication divergent edge, and it could be argued that the orthology clusters should contain in-paralogs. In the future, we might thus consider merging the clusters that are share a high enough degree of similarity. We have also ignored the effects of convergent evolution, even though it is possible that two independent divergent duplication events lead to similar genes. If this occurs, one cannot guarantee that similar genes are orthologs, and that primary orthologs form cliques.

On the practical side, we assume that the genes are already partitioned into homologous families, which allows the computation of a multiple sequence alignment and eases the clustering phase. Future versions of the framework should be able to predict orthologs and paralogs from whole proteomes, as OrthoMCL and OMA-GETHOGS do. However, one could also argue that partitioning genes into homologous families and predicting orthology are two different tasks. The aforementioned programs perform both simultaneously. A better approach might be to use methods dedicated to homology clustering, and then refine the orthology/paralogy relations within each family using HyPPO.

Also, the reconstruction of a species tree from the clusters is the main bottleneck of HyPPO in terms of time. To give a rough idea, in the worst case, some datasets resulted in a set of about 3000 allowable splits, which took our algorithm 4-5 hours to solve. This motivates the need to improve the complexity of the dynamic programming formulation given in this paper. Also, in the near future we plan to evaluate the quality of the predicted species tree. Note however that our current algorithm may take many arbitrary decisions, and we must refine this process before performing a careful species tree analysis. Finally, it will be interesting to include a wider range of orthology prediction software in a future comparative study, and to establish a procedure to analyze primary orthologs on real datasets - for instance by predicting these orthologs by comparing the ratio of branch lengths following a duplication.

5 Conclusion

In this work, we have introduced the notion of orthology in the presence of divergence after duplication and have addressed the problem that

Fig. 6: Accuracy of the methods for each tree in our empirical dataset. The last column shows the average accuracy of the methods.

Table 2. Average accuracy, precision and recall over the set of 8 trees from SwissTree.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyPPO</td>
<td>0.86</td>
<td>0.94</td>
<td>0.78</td>
</tr>
<tr>
<td>HyPPO + Species Tree</td>
<td>0.90</td>
<td>0.95</td>
<td>0.88</td>
</tr>
<tr>
<td>OMA-GETHOGS</td>
<td>0.83</td>
<td>0.90</td>
<td>0.71</td>
</tr>
<tr>
<td>OrthoMCL</td>
<td>0.80</td>
<td>0.85</td>
<td>0.68</td>
</tr>
</tbody>
</table>
some orthologs may behave as paralogs from a functional point of view. We therefore propose to make a distinction between the orthologs that have or have not been separated by a duplication divergent event. As we have shown, the HyPP framework is able to distinguish paralogs, primary orthologs and secondary orthologs with better accuracy than its competitors. This is true even if divergence after duplication does not always occur. Our work is far from over though, as HyPP offers many opportunities to improve these results even further. Notably, it remains to evaluate alternative algorithms for each step in our pipeline, and to identify which ones yield the best results. Finally, our work raises some interesting questions from a theoretical standpoint. The complexity of inferring a species tree from gene clusters is left open, and it remains to evaluate how many clusters are necessary to reconstruct a high-quality species tree.

**Funding:** ML was financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

**References**


Supplementary material

Proof of Proposition 3
It suffices to show that for any triple of distinct vertices $u,v,w$, $uw \in E(G)$ and $vw \in E(G)$ imply that $uw \in E(G)$. Let $E_u$, $E_v$, and $E_w$ denote the set of edges on the unique $u-v$, $v-w$, and $w-u$ paths in $T$, respectively. In $T$, one can go from $u$ to $v$ by first going from $u$ to $v$, then from $v$ to $u$. This implies that $E_u \subseteq E_v \cup E_w$. As none of the edges in $E_v \cup E_w$ is marked as divergent, by definition, we have that $u$ and $w$ are similar, and thus $uw$ is an edge of $G$.

An algorithm for STUC with allowable splits
We describe how to solve the STUC with allowable splits using dynamic programming. Given $X \subseteq \Sigma$, a set of clusters $C$ and allowable splits $A$, denote by $OPT(X, C, A)$ the minimum number of losses in a species tree $S$, but only counting the lost clades that are strict subsets of $X$, and subject to the conditions that (1) $X \in clades(S)$ and (2) $S$ has only splits from $A$. To rephrase more formally, let $I_{S}(C, X)$ be the number of maximal $C$-tree nodes $v$ of the species tree $S$, with the additional condition that $clade(v) \subseteq X$. Then $OPT(X, C, A)$ is the minimum of $\sum_{S} I_{S}(C, X)$ over all possible species trees that contain the clade $C$ (and only splits from $A$). Define $OPT(X, C, A) = \infty$ if no such species tree exists. Clearly, $OPT(X, C, \emptyset)$ yields a solution to our restricted STUC instance, since every loss occurs at a clade at a strict subset of $X$ and any species tree contains the $X$ clade. We would like to devise a recurrence formula for $OPT(X, C, A)$.

Towards this goal, first let $A \cup B$ be a split of $X \subseteq \Sigma$ and let $C \in \mathcal{C}$. Define the cost $I_{A,B}(C)$ of $C$ with respect to $A \cup B$ as

$$I_{A,B}(C) = \begin{cases} 1 & \text{if } C \cap (\Sigma \setminus X) \neq \emptyset, \text{and exactly one of } A \cap C = \emptyset \text{ or } B \cap C = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

In words, we count a loss for $I_{A,B}(C)$ if $C$ has something outside $X = A \cup B$, and $C$ loses all of $A$ or all of $B$. We can now state our recurrence:

**Lemma 1.** If $|X| = 1$, then $OPT(X, C, A) = 0$. If $|X| > 1$, then either $OPT(X, C, A) = \infty$ or $OPT(X, C, A)$ is equal to

$$\min_{\mathcal{C}/X} \left( OPT(A, C) + OPT(B, \emptyset, C) + \sum_{C \in \mathcal{C}/X} I_{A,B}(C) \right)$$

where $spd(X)$ is the set of all possible splits of $X$.

**Proof.**
We prove the Lemma by induction on $|X|$. As a base case, suppose $|X| = 1$. Then for any species tree $T$ and any cluster $C$, $I_{C}(X, C)$ must be zero since $T$ does not contain a $C$-tree node $v$ such that $clade(v) \subseteq X$ (as otherwise, $clade(v) = \emptyset$). Now suppose that the Lemma holds for any strict subset of $X$, and assume $OPT(X, C, \emptyset) = \infty$. Let $S$ be a species tree containing a node $v$ satisfying $OPT(X, C, A) = X$ and having every split in $A$, chosen so that $S$ minimizes $\sum_{C \in \mathcal{C}} I_{S}(C, X)$. Let $v_1$, $v_2$ be the children of $v$ and let $\{A, B\} = \{clade(v_1), clade(v_2)\}$. Then $\{A, B\} \in \mathcal{spd}(X) \cap \Sigma$.

Since in the recurrence, the range of splits over which we take the minimum contains $\mathcal{A}$, it suffices to prove that $OPT(X, C, A) = OPT(A, C) + OPT(B, \emptyset, A) + \sum_{C \in \mathcal{C}/X} I_{A,B}(C)$. For some $C \in \mathcal{C}$, let $F_C$ be the set of maximal $C$-tree nodes that belong to $\{v_1, v_2\}$. Then $OPT(X, C, A) = \sum_{C \in \mathcal{C}/X} I_{S}(C, X) = \sum_{C \in \mathcal{C}/X} \left( |F_C| + |F_B| + |F_B| \right)$. To prove the Lemma, we will show that $\sum_{C \in \mathcal{C}/X} F_C = OPT(A, C)$, $\sum_{C \in \mathcal{C}/X} F_B = OPT(B, A)$, and $\sum_{C \in \mathcal{C}/X} F_B = OPT(B, A)$.

To achieve this, we first show that the subtrees of $S$ rooted at $v_1$ and $v_2$ must incur $OPT(A, C)$ and $OPT(B, A)$ losses, respectively. Consider a species tree $S'$ that contains a node $v$ satisfying $clade(v) = A$, that has every split in $A$, and such that $\sum_{C \in \mathcal{C}/X} I_{S}(C, A) = OPT(A, C)$. Note that $S'$ exists since, in particular, $S$ is a possible solution. Let $S_{v_1}$ be the subtree of $S'$ rooted at $v_1$. In $S$, replace the $S_{v_1}$ subtree rooted at $v_1$ by the $S_{v_2}$ subtree. Observe that this new tree obtained has cost $OPT(A, C) + \sum_{C \in \mathcal{C}/X} F_B + \sum_{C \in \mathcal{C}/X} F_B$, implying that $\sum_{C \in \mathcal{C}/X} F_B = OPT(A, C)$. By a similar reasoning, $\sum_{C \in \mathcal{C}/X} F_B = OPT(B, A)$.

So far, we have shown that $OPT(X, C, A) = OPT(A, C) + OPT(B, A) + \sum_{C \in \mathcal{C}/X} I_{A,B}(C)$. It remains to argue that $\sum_{C \in \mathcal{C}/X} F_B = \sum_{C \in \mathcal{C}/X} I_{A,B}(C)$. Fix some $C$. If $s(C) < v$ or $s(C) \supseteq v$, then neither of $v_1$ and $v_2$ can be a maximal $C$-free node. If $s(C) = v$, the same holds, because $s(C) = v$ implies that $C$ contains an element from both $clade(v_1)$ and $clade(v_2)$. In both these cases, $F_B = \emptyset$. Accordingly, $I_{A,B}(C) = 0$ in these cases (because when $s(C) \leq v$, we have $C \cap (\Sigma \setminus X) = \emptyset$, and when $s(C) \cap v$, both $C \cap A$ and $C \cap B$ are empty). Assume that $s(C) > v$. It follows that $C \cap (\Sigma \setminus X) \neq \emptyset$, and also that $C \cap X \neq \emptyset$. The latter implies that $v_1$ and $v_2$ cannot both be maximal $C$-free nodes. If only $v_1$ is $C$-free maximal, then $F_B = \{v_1\}$. Accordingly, $I_{A,B}(C) = 1$, since $C \cap A = \emptyset$. The same holds if only $v_2$ is maximal $C$-free. Finally, if none of $v_1$ and $v_2$ are maximal $C$-free, then $F_B = \emptyset$, and accordingly $I_{A,B}(C) = 0$. In every case, we have equality between $|F_B|$ and $I_{A,B}(C)$. This shows that $\sum_{C \in \mathcal{C}/X} F_B = \sum_{C \in \mathcal{C}/X} I_{A,B}(C)$, concluding the proof.

Observe that $spd(X) \cap \emptyset$ can be computed in time $O(|\mathcal{A}| \cdot |X|)$, as it suffices to iterate over every element of $A$ and check whether the split partitions $X$.

In the next section, we prove Theorem 7, i.e. that the above dynamic programming formulation leads to an algorithm that runs in time $O(|\mathcal{A}| \cdot |X|)$.

We construct the set $A$ in a heuristic fashion as follows. We build a collection $D$ of subsets of $\Sigma$, and $A$ consists of all the pairs $(D_1, D_2)$ where $D_1, D_2 \in D$ and $D_1 \cap D_2 = \emptyset$. To construct $D$, sort the species of $\Sigma$ by non-increasing order of frequency, where the frequency of $x$ is the number of clusters of $\Sigma$ that contain $x$. This results in an ordering $(x_1, \ldots, x_m)$ of $\Sigma$. Then for any $i, j \in [m]$ with $i < j$, add the set $\{x_i, x_{i+1}, \ldots, x_j\}$. For good measure, we also add every cluster $C_i \in \Sigma \cap D$, concluding the construction. The idea behind the ordering is that if a set of species has a similar frequency, it is possibly because they always appear together in the same clusters. This set may thus be part of some split of $S$.

**Proof of Theorem 7**
Consider the algorithm that starts computing $OPT(\Sigma, \emptyset)$ and recursively computes $OPT(X, C, A)$ for the necessary values of $X$ according to the formulation of Lemma 1. Assume that $OPT(X, C, A)$ is computed once for each value of $X$, and stored in memory if this value is required later on. The value of $OPT(X, C, A)$ only needs to be computed if $A$ contains a pair of the form $[X, Y]$. Therefore, at most $2|\mathcal{A}|$ computations of $OPT(X, C, A)$ are necessary. The time required to compute $OPT(X, C, A)$, without counting the time for the recursive calls, corresponds to the time required to compute $I_{A,B}(C)$ for each $C \in \mathcal{C}$ and each $(A, B) \in \mathcal{spd}(X) \cap \emptyset$. The value $I_{A,B}(C)$ only needs to compute intersections over subsets of $\Sigma$ and thus takes time $O(|\mathcal{A}|)$.

Since there are at most $|\mathcal{A}|$ splits in $spd(X) \cap \emptyset$, the time required for
OPT(X, Y, k) is \(O(|\mathcal{A}|^2|\Sigma|^2)\). As mentioned before, X can take \(2^{|\mathcal{A}|}\) values, and the overall complexity is \(O(|\mathcal{A}|^2|\Sigma|^2)\).

**Proof of Theorem 8**

First note that if \(\sigma(x) = \sigma(y)\), then no edge will ever be added between \(x\) and \(y\), due to the condition in line 5. Hence no two genes from the same species are ever matched edges. Consider an iteration, some cluster \(C\) is merged with its favorite cluster \(C_j\). Then either \(x\) becomes orthologous with every member of \(C\), or with none of them. Moreover, these relations between \(x\) and \(C\) remain until the end of the algorithm.

Now, suppose that vertices \(x, y, z\) introduce a \(P_j\) in \(G\) (with edges \(xy\) and \(yz\)) at the \(i\)-th iteration, for some \(i \in [k - 1]\). Assume that \(\sigma(x), \sigma(y), \sigma(z)\) are all distinct. We must prove that \(\sigma(x) \oplus \sigma(y) \oplus \sigma(z)\) is a triplet of \(S\), as required by \(S\)-consistency. Let the clusters \(C_1, \ldots, C_h\) be the clusters at the beginning of the \(i\)-th iteration (in particular, there are \(i - 1\) empty clusters due to line 7). To avoid ambiguity, if we need to refer to some cluster \(C_j\) at the beginning of the \(h\)-th iteration, we will write \(C^{(h)}_j\) (hence, under our notation, we use \(C_j\) as a shorthand for \(C^{(i)}_j\)).

Now, let \(C\) be the cluster chosen at the \(i\)-th iteration, and let \(C_j\) be the favorite cluster of \(C\) chosen at this iteration. Since only edges between \(C\) and \(C_j\) are added at this \(i\)-th iteration, \(C\) and \(C_j\) must both contain one of \((x, y, z)\).

Assume that \(x, y, z\) all belong to a distinct cluster at the \(i\)-th iteration, say the clusters \(C, C_j\) and \(C_k\). Then there is an edge going from \(y\) to \(z\) at least one of \(C\) or \(C_j\). This edge was added in an iteration prior to \(i\), implying that one of \(C\), \(C_j\) or \(C_k\) must have been emptied, a contradiction. We may therefore assume that \(C\) and \(C_j\) contain all of \((x, y, z)\). We check all the possible locations of \(x, y, z\) and \(z\):

- If \(x, z \in C\) and \(y \in C_j\), then \(\sigma(y) \oplus \sigma(z) \leq s(C)\) (because every \(y - C\) edge must have been added by the algorithm). Moreover, \(\sigma(x) \leq s(C)\) (by the definition of \(s(C)\)). This implies \(\sigma(x) \oplus \sigma(z) \leq s(C)\). Hence \(\sigma(y) \oplus \sigma(z) \geq s(C)\), which implies \(\sigma(y) \oplus \sigma(z) \in \sigma(S)\).
- If \(x, y \in C\) and \(z \in C_j\), \(z\) is a neighbor of \(y\) but not \(x\). \(z\) is a neighbor of every member of \(C\), or of none of them. Hence \(z\) cannot form a \(P_j\) with \(x\) and \(z\). Similarly, we cannot have \(z, y \in C\) and \(x \in C_j\).
- If \(x \in C\), \(y \in C_j\), then \(\sigma(x) \leq s(C)\) and \(\sigma(y) \leq s(C)\) (otherwise \(x\) would be an edge). This implies \(\sigma(x) \oplus \sigma(y) \in \sigma(S)\), as required. The same idea works if \(x \in C\) and \(x, y \in C_j\).

Finally, suppose that \(y \in C\) and \(x, z \in C_j\). Then \(\sigma(x) \oplus \sigma(z) \in \sigma(S)\). For the sake of contradiction, assume w.l.o.g. that \(\sigma(y) \oplus \sigma(z)\) is a triplet of \(S\) (the other contradicting triplet is \(\sigma(y) \oplus \sigma(z)\), and the same proof applies). Since \(x\) and \(z\) do not share an edge, they did not belong to the initial set of given clusters. Then there must have been some iteration \(h < i\) such that \(C^{(h)}_j\) was the favorite of some cluster \(C^{(h)}_j\), with either \(x \in C^{(h)}_j\) or \(z \in C^{(h)}_j\) or vice-versa. Suppose \(x \in C^{(h)}_j\) or \(z \in C^{(h)}_j\). As no edge was added between \(x\) and \(z\), we had \(\sigma(x) \leq s(C^{(h)}_j)\). Thus \(s(C^{(h)}_j)\) is a common ancestor of \(\sigma(x)\) and \(\sigma(z)\). Since \(\sigma(y) \sigma(z)\) is a triplet of \(S\) and \(\sigma(x)\), \(\sigma(z) \oplus s(C)\), we have \(s(C^{(h)}_j) > s(C)\). However, the algorithm iterates over the clusters in non-decreasing order with respect to \(S\), a contradiction since \(C\) should have been handled before \(C_j\) at the \(h\)-th iteration.

This proves that every \(P_j\) satisfies the requirement of Theorem 6. Now, suppose that vertices \(x, y, z\) introduce a \(P_j\) at the \(i\)-th iteration (with edges \(wx, xy, yz\)), at which point \(C\) is merged with its favorite \(C_j\). Observe that as above, we may assume that \(C\) and \(C_j\) contain all of \((w, x, y, z)\). Notice also that \(C_j\) cannot contain only one vertex from this \(P_j\); this vertex must either be adjacent to the 3 others, or to none of them, both cases preventing a \(P_j\). Also, \(C_j\) cannot contain exactly two of these vertices. Indeed, if \(w, x \in C_j\) and \(y, z \in C\), then since \(xy\) is an edge, \(x\) shares an edge with every member of \(C\). This includes \(x, z\), a contradiction. One can check that the type of argument holds for the other five cases \(w, x, y \in C_j\) and \(w, x, y \in C\) and \(w, x \in C, w, y \in C_j\) and \(w, x \in C, y \in C_j\) and \(w \in C, x \in C_j\) and \(w \in C, y \in C_j\) and \(w \in C, z \in C_j\).

It follows that \(C_j\) must have 3 vertices. In particular, either we have both \(x, z \in C_j\) or \(x, z\) is in \(C_j\). Suppose \(x, y \in C_j\) and \(w \in C, z \in C_j\). Notice that \(\sigma(w) \leq s(C), \sigma(y) \leq s(C)\) and \(\sigma(z) \oplus s(C)\). Note that this implies \(\sigma(w) \neq \sigma(y)\) (moreover, \(\sigma(x) \neq \sigma(y)\)). Since \(xy\) and \(yz\) are edges. This fact implies \(\sigma(w) \sigma(y) \sigma(z)\) is a triplet of \(S\), and that the \(x, y, z\) should be forbidden, contradicting the fact that every \(P_j\) is persistent. It remains to cover the case when only one of \(x\) or \(y\) is in \(C_j\). Suppose, w.l.o.g. that \(x \in C, w, y \in C_j\). Note that \(\sigma(w) \oplus s(C), \sigma(y) \oplus s(C)\) and \(\sigma(z) \leq s(C)\). As we did before, consider the iteration \(h\) at which \(w\) and \(z\) both ended up in \(C_j\). Then either \(w \in C^{(h)}_j\) and \(z \in C^{(h)}_j\) for some \(i\) or vice-versa. If \(w \in C^{(h)}_j\) and \(z \in C^{(h)}_j\), then \(\sigma(w) \leq s(C^{(h)}_j)\). Thus \(C^{(h)}_j\) is a common ancestor of \(\sigma(w)\) and \(\sigma(z)\), which implies \(s(C^{(h)}_j) > s(C)\) since \(\sigma(w) \leq s(C)\) and \(\sigma(z) \leq s(C)\). This contradicts the ordering of the iterations of the algorithm. If \(z \in C^{(h)}_j\) and \(w \in C^{(h)}_j\), then \(\sigma(z) \leq s(C^{(h)}_j)\). \(C^{(h)}_j\) is a common ancestor of \(\sigma(w)\) and \(\sigma(z)\), and we reach the same contradiction. This concludes the proof.