UNIVERSITÉ FRANCO ITALIENNE | UNIVERSITÀ ITALO FRANCESE

Università Italo-Francese / Université Franco-Italienne,
AGREEMENT FOR THE CO-DIRECTION OF THE Ph.D THESIS:

Sapienza, Università di Roma:
Dept. of Computer Science
Supervisor Tiziana Calamoneri

Université Claude Bernard, Lyon 1:
Ecole doctorale E2M2
Supervisor Marie-France Sagot

Author: **Mattia Gastaldello**

Enumeration Algorithms and Graph Theoretical Models to Address Biological Problems Related To Symbiosis

---

Contents

Abstract

In this thesis, we address two graph theoretical problems connected to two different biological problems both related to *symbiosis* (two organisms live in *symbiosis* if they have a close and long term interaction).

The first problem is related to the size of a *minimum* cover by chain subgraphs of a bipartite graph. A *chain graph* is a bipartite graph whose nodes can be ordered by neighbourhood inclusion.
In biological terms, the size of a minimum cover by chain subgraphs represents the number of genetic factors involved in the phenomenon of *Cytoplasmic Incompatibility* (CI) induced by some parasitic bacteria in their insects host. CI results in the impossibility to give birth to an healthy offspring when an infected male mates with an uninfected female.

In particular, in the first half of the thesis we address three related problems.
One is the enumeration of all the maximal *edge induced* chain subgraphs of a bipartite graph G, for which we provide a polynomial delay algorithm (Algorithm 3 in Section 1.5.1) with a delay of $O(n^2m)$ where n is the number of nodes and m the number of edges of G. In the same section, we show that $\frac{n}{2}!$ and $2^{\sqrt{m}\log m}$ bound the number of maximal chain subgraphs of G and use them to establish the input-sensitive complexity of Algorithm 3.

The second problem we treat is finding the minimum number of chain subgraphs needed to cover all the edges of a bipartite graph. To solve this problem, we provide in Section 1.5.2 an exact exponential algorithm which runs in time $O^*((2+\varepsilon)^m)$, for every $\varepsilon > 0$, by combining Algorithm 3 with the inclusion-exclusion technique [7] (by O^* we denote standard big O notation but omitting polynomial factors). Notice that, since a cover by chain subgraphs is a family of subsets of edges, the existence of an algorithm whose complexity is close to 2^m is not obvious. Indeed, the basic search space would have size 2^{2^m}, which corresponds to all families of subsets of edges of a graph on m edges.

The third problem we approach is the enumeration of all minimal covers by chain subgraphs of a bipartite graph G and show that it is possible to enumerate all such minimal covers of G in time $O([(m+1)|\mathcal{S}|]^{\log((m+1)|\mathcal{S}|)})$ where $\mathcal{S}$ is the number of minimal covers of G and m the maximum number of chain graphs in a minimal cover.

In Section 1.6, we present the relations between the second problem and the computation of the *interval order dimension of a bipartite poset* and in Sections 1.5.1 and 1.5.2 the interpretation of the results in the context of poset and interval poset dimension. Indeed, we can straightforwardly compute the interval dimension of a bipartite poset $\mathcal{P}$

in $O^*((2+\epsilon)^p)$ where p is the number of incomparable pairs of Q.

Finally, in Section 1.7, we extend our result on interval poset dimension to the problem of computing the *poset dimension* of a poset by means of Trotter *split* operation [84] and we obtain a procedure which computes it still in $O^*((2+\epsilon)^{p/2})$.
To improve our results on the *poset dimension* and to perform better than $O(\sqrt{2}^p)$, *i.e.* the minimum time to run the inclusion-exclusion formula on which these results are based, we introduce for each poset an associated graph $\mathcal{GCP}$, called *the graph of critical pairs*. In this way we obtain two algorithms, an exponential and a polynomial space one. These algorithms compute the poset dimension in 2^q and $O(2.9977^q)$ time respectively where q is the number of critical pairs of $\mathcal{P}$ (intuitively, *critical pairs* are the fundamental incomparable pairs to consider).

We then conclude this first part with some open questions related to these problems.

In the second part of the thesis, we deal with the Reconciliation Model of two phylogenetic trees and the exploration of its optimal solutions space. *Phylogenetic tree reconciliation* is the approach commonly used to investigate the coevolution of sets of organisms such as hosts and symbionts. Given a phylogenetic tree for each such set, respectively denoted by H and S, together with a mapping φ of the leaves of S to the leaves of H, a reconciliation is a mapping ϱ of the internal nodes of S to the nodes of H which extends φ with some constraints.

Depending on the mapping of a node and its children, four types of events can be identified [4, 22, 73]: *cospeciation* (when host and parasite evolve together), *duplication* (when the parasite evolves into different species but not the host, and at least one of the new parasite species remains associated with the host), *loss* (when the host evolves into two new species but not the parasite, leading to the loss of the parasite in one of the two new host species) and *host switch* (when the parasite evolves into two new species with one species remaining with its current host while the other switches, that is jumps to another host species).
Given a cost for each kind of event, $C = (c_c, c_d, c_l, c_h)$ respectively, we can assign a total cost to each reconciliation. The set of reconciliations with minimum cost is denoted by $\mathcal{R}(H, P, \varphi, C)$ and its elements are said to represent *parsimonious* reconciliations. However, their number can be often huge.
Without further information, any biological interpretation of the underlying coevolution would require that all the parsimonious reconciliations are enumerated and examined. The latter is however impossible without providing some sort of high level view of the situation.

In this thesis, we approached this problem by introducing two equivalence relations $\sim_1$ and $\sim_2$ (Section 2.6 and Section 2.7) to collect similar reconciliations and reduce the optimal solutions set to a smaller set of representatives of these equivalence classes.

In Section 2.8, we introduce a new distance among optimal reconciliations DH and we compare it with the distances already present in literature. In the same section, we show that we can embed the set of parsimonious reconciliations $\mathcal{R}(H, P, \varphi, C)$ into the discrete k dimensional hypercube $\mathcal{H}^k = \{0,1\}^k$ and that DH coincides with the *Hamming Distance* on $\mathcal{H}^k$. The equivalence distances $\sim_1$,$\sim_2$ and the distance DH are all based on the set of host-switch edges of a reconciliation $\widetilde{\Theta}$ (*i.e.* the edges of P which are mapped to a non-edge of H).

In Section 1.3, we present a series of results on reconciliations based on the conditions $c_c \leq c_d$ and $c_l > 0$ which lead to Theorem 12 which proves that $\widetilde{\Theta}$ characterize the reconciliations.

In the literature, it is known [46, 48, 71, 82] that when host-switch events are forbidden, there is just one optimal reconciliation given by the *least common ancestor map*, $lca : V(P) \to V(H)$, which recursively (starting from the leaf mapping) maps a parasite tree internal node u to the least common ancestor of the host tree nodes $lca(u_1), lca(u_2)$ where u_1, u_2 are the children of u.

To the best of our knowledge, we do not know about any other results like Theorem 12.

In Section 2.9, we present some experimental results to show the efficacy of $\sim_1$ and $\sim_2$ and we comment these results under the light of the chosen cost vector C. The most outstanding results we obtain is in the case of the dataset related to the parasite *Wolbachia* (a bacterium with many interesting aspects, see Section 1.1) where we pass from $\sim 4.08 \cdot 10^{42}$ parsimonious reconciliations to $\sim 1.15 \cdot 10^3$ representatives.

In the second half of Section 2.9, inspired by the work of Robinson and Foulds on distances between phylogenetic trees [78], we present and comment some statistics and histograms on DH and the dimension k of $\mathcal{H}^k$.

We conclude the chapter with some open problems and some comments in Section 2.10.

Introduction

A graph is an abstract representation of objects (its nodes) and the pairwise relations or interactions among them (its edges). This extreme simplicity and flexibility makes graphs and graph theory suitable for many applications and models of real life situations. Some examples that surely do not cover all the application areas are: systems optimization of the distribution of goods (or of informations and data), a website and all the links to its pages, the evolutionary history of species, the metabolic network of some bacteria, the reconstruction of a whole genome or transcriptome from its fragments (DNA or RNA-seq reads), time scheduling for the exploitation of shared resources (such as time, working forces, physical location of objects), social networks and social interactions.

It is clear then the importance of graph theory and algorithms in order to solve real life problems. Indeed, most such problems correspond to classical ones in graph theory such as: network flows, decomposition of graphs, enumeration of graphs or of subgraphs, graph coloring, routing problems and edge or node coverings.

In particular, many of the more recent real-life problems come from biology which, thanks also to the new technologies, provides complex simulations and allows to collect an amount of data that is estimated, for the next future, to be of orders of magnitude greater than any other source of data.

In this thesis, we study some graph theoretical problems which model two specific biological questions, both related to *symbiosis*. Two organisms are said to live in *symbiosis* if they have a strong and long term biological interaction. This phenomenon involves almost all the living entities, with the interactions varying in terms of frequency and type of interaction.

More specifically, we study the following two symbiosis-related problems: the *Cytoplasmic Incompatibility* phenomenon, which is related to some formulations of the cover problem in bipartite graphs, and the *Reconciliation of phylogenetic trees* whose enumeration requires a better exploration of the solutions space. In fact, very often biological problems require to list all the possible solutions, to then discern among them by means of further knowledge based on experimental data, hence, they are a good source of enumeration problems and a good motivation to develop efficient enumeration algorithm.

The first topic we consider, the phenomenon of *Cytoplasmic Incompatibility* (CI), occurs in some biological interactions of parasitic nature between bacteria and the insect

hosts inside which they live. CI consists basically in promoting the spread of the parasite in the host population by discouraging the breeding with uninfected hosts. The better known example of this kind of phenomenon is perhaps the one of mosquitos and of the bacterium *Wolbachia* which was tested and employed for instance in Florida in 2017 as a solution to control the spread of the Zika virus. *Wolbachia* is a parasite which lives in the reproductive organs of most of the insects, a fact that in itself makes the research on *Wolbachia* of high interest for many applications (such as the one cited here). Zika is a virus originally from Africa and extremely dangerous for humans as it is believed to cause birth defects such as abnormally small brains. To address this threat, the U.S. Food and Drug Administration approved the release of male mosquitos artificially infected with specific *Wolbachia* bacterial strains which prevent successful breeding with the local population of mosquitos with the final aim to reduce the population size.

In order to understand the phenomenon of CI, it is necessary to search for the genes in the genome of the bacterium that are responsible for it. The *genome* of an organism is its genetic material in terms of its chromosomes and the DNA constituting them. In the case of *Wolbachia*, there is a single chromosome that is circular. The cost for obtaining a good quality genome for *Wolbachia* has become drastically low due to the recent DNA sequencing technologies and also to its small size (which is $\sim$ 6,000 times smaller than the human genome).

However, such genome contains on average $\sim$ 1,000 genes and at the moment the best approach to identify the genes involved in CI phenomenon is *comparative genomics*, *i.e.* the comparison of different genomes, gene structures *etc.* in order to identify similar features in order to identify the potentially responsible genes.

The approach we employed in the first chapter of the thesis is an approach "external" to the genome of the bacterium in the sense that we do not compare the genomes of similar CI-inducing bacteria such as *Wolbachia* but we start from the observation of the patterns of successful and failed breedings between the hosts infected by different families of the bacterium. In this way, such external approach comes as a support to a comparative genomics approach.

This approach is based on the so-called *Toxin and Antitoxin Model*, described in Chapter 1, which has a graph theoretical formulation in terms of computing the size of a minimum *edge cover by chain subgraphs* in a bipartite graph. A *chain graph* is a bipartite graph whose nodes can be ordered by neighbourhood inclusion.
Intuitively, the bipartite graph with its edges represents the successful breeding between male and female hosts and each chain graph of the cover represents a genetic factor involved in the phenomenon. Based on a parsimonious criterion, we then search for the size of a minimum cover.

To this purpose, we started from the problem of enumerating all maximal chain subgraphs of a given bipartite graph and we proposed an algorithm which efficiently enumer-

ates them.
We then considered the problem of enumerating all the *minimal* covers by chain subgraphs and, separately, the problem of computing the size of a *minimum* cover avoiding the enumeration of all the covers.
The problem of minimal covers (by chain subgraphs) of a bipartite graph is strictly related to the problem of computing the *interval poset dimension* of a bipartite ordered set.
Our solutions and conclusions on these four related problems can be found in [17], in [16] (currently under review) and in this thesis in Sections 1.1-1.6 of Chapter 1.In Section 1.7.2 of the same chapter, we present this relation and the interpretation of our results in the context of poset dimension theory. Finally, we applied and studied the same approach, in the case of the *poset dimension* and we introduced a special graph which helps us to compute it. This is a joint work currently in manuscript version [42].

Symbiosis, as a source of interesting graph theoretical problems, and enumeration algorithms, as solutions to these problems, play both a central role in the thesis. Indeed, not only enumeration is an important aspect for the problem related to CI (as different covers imply different genetic structures) but also the second topic of this thesis is related to enumeration and symbiosis.

Indeed, in this second part of the thesis, we deal with the symbiotic interactions between species from an evolutionary point of view. Again we deal with a graph theoretical model which describes how these symbiotic interactions evolved in time up to the current state. The final challenge then will be to efficiently enumerate and explore the various optimal solutions which come out of this model. Indeed, many possible solutions to the model are equally sound and it happens very often that even a small input dataset gives rise to an exponential number of optimal solutions. Furthermore, the computation of just a single optimal solution is in itself an NP-hard problem.

The model we use to study and describe the coevolution of two sets of species that interact is the *Reconciliation Model* which is based on the *phylogenetic trees* of two symbiotic sets of organisms. A *phylogenetic tree* is a tree graph whose set of leaves represent the currently living species, while the evolution and the ancestry relations are represented by the branchings of the tree.

Without loss of generality, in the following we will explain the model in the case of symbiosis where the two phylogenetic trees model the evolution of a set of parasites and of the host species with which these interact. However the model can be applied also in the case of a set of species and one or a set of their genes, to study the evolution of that gene (or those genes) with respect to the evolution of the species, or in the case of *biogeography*, *i.e.* of the study of how the species are geographically distributed. In this latter case, the phylogenetic tree of the species is compared to the cladogram of a geographical area, *i.e.* to a tree graph which represents the similarities between geographical areas by the branching of the tree, the leaves representing those areas and the tree distance measure

their similarity.

Given two phylogenetic trees, one of a set of parasites and one of their hosts, the Reconciliation Model consists in defining a function, called *reconciliation*, from the nodes of the parasite tree to the nodes of the host tree. This function must be conform to some constraints such as the temporal order of the species which has to be preserved once mapped on the host tree. As part of the input, we have also the mapping of the leaves of the parasite tree to those of the host tree.

A reconciliation permits to model and identify evolutionary events such as *cospeciation* which occurs when the branching of the trees agree on a given node. This situation happens when the host evolves into different species and such change induces an evolution of the parasite as well.

In the case of a bacterium, it may probably happen that it evolves much faster than its host. The reconciliation model can describe this phenomenon by mapping a parasite tree node together with its sibling to the same host tree node. In this case, we speak about a *duplication* event and the number of such adjacent parasite tree nodes can be interpreted as the rate of evolution of the parasite with respect to the host.

The *loss* event occurs when the host speciates and one of the two new species becomes immune to the parasite infection.

Finally, the so-called *host switch* occurs when a parasite infects a host species evolutionarily different from the current one. Current well known examples of this phenomenon are all those viruses which typically affect other species and which then end up by infecting humans, such as *SARS* which originally was a virus affecting bats, *EBOLA* found as well in African bats but also in some Pigs in the Philippines, and finally *Avian Flu* which was a virus transmitted from birds.

For these reasons as well, there is a wide interest in the scientific community on such *host-switch* event, on its frequency and on the rapidity of adaptation of parasites and viruses, the so-called *parasite paradox*. Indeed, it is not clear how a parasite can be highly specialized on the hosts which they infected for generations and then suddenly change its host and develop traits that enable it to detect the new host, to ensure reproductive continuity, to adapt its metabolic system to the new host and to defend itself from its immune system.

A possible resolution of this paradox is that the parasite is conservative in terms of resource use (of its host), *i.e.* ancestral traits developed in the past to survive in some hosts are preserved in its DNA and may help it to survive in the new host if the latter presents analogous conditions. When the parasite is exposed to such a host and ends up infecting it, we have an *host-range expansion* of the parasite. This exposure is actually one explanation for the many host switches of disease vectors; a main reason for these various exposures to a new host is related to the relaxed environmental conditions that

are themselves a consequence of the climate change. For instance, the warming up of the arctic area may explain the appearance in 2008 on the low Arctic Victoria Island of two lungworms *Umingmakstrongylus* and *Varestrongylus* which are both muskoxens parasites; the second one is also a caribou parasite.
Both parasites were found before only on the mainland, however, the warming up of the region has relaxed the geographical acceptable boundaries for the survival of their larvae and the seasonal migrations of the caribou on the mainland may have facilitated their introduction in the island. It is not to be excluded that there was, in the case of *Umingmakstrongylus*, a host switch from caribou to muskoxen and it is not to be excluded that it will happen in the future a host switch of the *Varestrongylus* in the opposite direction.

The host-switch event is perhaps the most interesting one (of the four presented), both in terms of biological implications and graph theoretical ones; indeed, also in terms of the complexity of computing the solutions of the reconciliation model, the presence of host switches is responsible for its NP-hardness as such switches may involve all species (all nodes) in the host tree.

In the second chapter of the thesis, we define two equivalence relations among the reconciliations that are both based on some host-switch properties. We then present two equivalence relations and a new distance between reconciliations, all based on host switches. Furthermore, we show that a knowledge of only the host switches of a reconciliation is enough to reconstruct all the reconciliations under some conditions.

The two equivalence classes group together reconciliations which differ slightly in some host switches. By choosing a canonical representative in the class, we can restrict ourselves to work with a reduced number of objects without losing the possibility of recovering all the original set.
We show that the number of representatives is often much lower compared to the total number of optimal reconciliations, in some cases passing from $\sim 10^{48}$ objects to $\sim 10^{3}$.

Moreover, we discuss a distance based on host switches and we present some experimental results which show some of the characteristics of the new distance. Along the chapter, we also compare our distance with some other recently defined and already known ones underlining their advantages and disadvantages.
We present the results and the definitions of the two equivalence classes in [41] (in this thesis Section 2.1-2.7) and, together with the results and definition of the distance, in [40] (Section 2.8 of the thesis).

We proceed now with an introduction on enumeration algorithms which will conclude the introduction to the thesis.

0.1 Enumeration algorithms

An enumeration algorithm is an algorithm which *lists all* the optimal solutions, without repetition and possibly in a efficient way. We present here after an informal definition of what means solve an enumeration problem (an interested reader may refers to [58] or for a complexity theory dissertation [60, 81]):

Definition 1. *Let A be a problem, x an instance of A and $A(x)$ the set of solutions of A on x.*
An algorithm is said to solve the enumeration problem *$Enum(A)$ if, for each instance x of A, it computes a sequence of elements $y_1, \ldots, y_n$ such that:*

1. $\{y_1, \ldots, y_n\} = A(x)$,
2. $i \neq j \rightarrow y_i \neq y_j$

In real life applications finding a single optimal solution is often not enough. For instance, referring to social networks analysis, a classical problem in the field [52, 57, 85] consist in listing all the communities (the *maximal clique subgraphs*). Another example, this time taken from metabolism biology, are the chemical changes in a metabolic network which can be studied by listing specific *maximal acyclic subgraphs* [9].

Generally speaking, there are contexts as computational biology, where there is often the need of pass through all the space of solutions. Just to cite one classic and fundamental situation: often a model offer different equivalent optimal solution solutions as for instance the number of optimal reconciliation (*i.e.* a model of coevolution of organism) of in Tables 2.3-2.12 of Chapter 2.

To chose among these solutions requires to undergo a phase of exploration of the solution space and to employ further knowledge external to the model to chose the best solution which fit with the additional information.

In all these three situations we have to be able to explore all the solution space, hence we need to have an efficient Enumeration Algorithm to list all the elements of the space.

In designing enumeration algorithms a classical situation in which we may occurs is the risk of a redundant repetition of solutions output and, on the other hand, the risk to cut out many solutions which will be missing. To avoid these risks, there are *ad-hoc* solutions as the introduction of some order among the object to enumerate [57] or, in the case of isomorphic solutions, the choice of a canonical representative which will be the only one outputted among all the isomorphic ones. In Section 0.1.2 we present some more general classical approaches for enumeration algorithms.

0.1.1 Complexity Classes for Enumeration Algorithms

We could regards enumeration algorithms in a classical context and actually we can also have a decision version of the $Enum(A)$ problem of Definition 1, presented informally here under:

Definition 2. *Let A be a problem, x an instance of A,$A(x)$ the set of solutions of A on x.An algorithm is said to* solve the decisional version enumeration problem *$DecisionEnum(A, \mathcal{S})$ if, for each instance x of A and each set $\mathcal{S}$, it can decide if $\mathcal{S}$ equals $A(x)$.*

Posed the problem in a decision version, we could employ all the classical theory of complexity classes $P, NP, co-NP, \#P$ etc.. However, let us observe that it happens very often to have an exponential number of elements to enumerate. Then, we cannot expect in general to develop polynomial algorithms as at least an operation for each element has to be spent. For instance, by the classical results of Moon and Moser [64], the number of maximal clique subgraphs in a graph with n nodes can be $3^{n/3}$. Hence the performance of the algorithm is heavily influenced by the dimension of the output. Yet we need a way to evaluate the goodness of such an algorithms.

Here after we present some definitions, as given in [52], to describe the efficiency of an enumeration algorithm. These definitions are usually referred as *output sensitive* complexities and the belonging of an enumeration algorithm to one of the defined classes, can be already very informative on how it enumerate as we better explain in the following:

Definition 3. *An algorithm is said to run in* polynomial total time *if the time required to output all the solutions can be bound by a polynomial in the input and output size. The class $TotalP$ is the class of all polynomial total time algorithms.*

A stronger notion of total polynomial time is the following:

Definition 4. *An algorithm is said to run in* incremental polynomial time *if given several solutions, it can determine another solution or the non-existence of another one in time bounded by a polynomial in the combined size of the input and the given solutions. Let $IncP$ be the class of all incremental polynomial algorithms.*

It is clear that an algorithm that runs in incremental polynomial time runs also in polynomial total time, hence $IncP \subseteq TotalP$.However, observe that even if an algorithm requires an exponential time to generate a solution s it may be still possible that it runs in incremental polynomial time if there are sufficient polynomially computable solution, *i.e.* we bound the exponential time for s by the exponential number of easier solution to compute.

Any enumeration problem $Enum(A)$ can be put in the following form:

Problem 1. *Let $A(x)$ be the set of all the solutions of a given problem $Enum(A)$ and let $\mathcal{S} \subseteq A(x)$. Decide if $A(x) \setminus \mathcal{S} = \emptyset$ or exhibit an element $s \in A(x) \setminus \mathcal{S}$.*

If we are able to solve Problem 1 in polynomial time for any instance x, then we have an incremental polynomial time procedure to solve $Enum(A)$.

Many problems have been proved to be solvable in incremental polynomial time as for example enumerating all the *circuits of a matroid* [53], enumerating all *minimal edge dominating sets* [45] or *maximal independent sets of an hypergraph* [10].

Definition 5. *An algorithm is said to run in* polynomial delay *if the time spent until the output of the first solution as well as the time between optimal solutions is bound by a polynomial in the input size. We call DelayP the class of all polynomial delay algorithms.*

Observe that a polynomial delay algorithm is also an incremental polynomial algorithm (*i.e.* $DelayP \subseteq IncP$) and it is what is considered *good enumeration algorithm*. Observe that it is also accepted an exponential time delay before the first output.

The algorithm described in Section 1.5.1 can enumerate all *maximal chain subgraphs of a bipartite graph* in polynomial time delay and requires only the space proportional to the input size (see Section 1.6 for problems equivalent to this one as enumerating all *maximal interval order reductions*).

Other enumeration problems which are known to be in $DelayP$ are, for example, the enumeration of all the *bubbles in a direct graph* [6], the *chordal graphs in a graph* [54] and the *ideals of a poset* [55].

Regarding the classes of $DelayP$ and $IncP$ at the moment it is not known if they are well separated [81] (*i.e.* $DelayP \subsetneq IncP$), although we have the following results on the complexity of the other classes:

Proposition 1 ([81])**.** *$P = NP$ if and only if $TotalP = EnumP$.*

Proposition 2 ([81])**.** *If $TotalP = IncP$ then $P = coNP \cap NP$.*

0.1.2 Techniques to Design Enumeration Algorithms

This section is inspired by the methods collected and presented in [2, 58]. Among the classic approaches to enumerate all the solutions of a given problem there is the brute force one which enumerates all the solutions by enlarging a given one and removing the non feasible ones or by a divide and conquer approach. This method is acceptably efficient in the case of contained size problems.

Here after although we present some finer techniques which avoid in a more elegant way the problem of finding the same solution many times and the one of missing a solution.

Backtracking

This approach is often used when the set of solutions $\mathcal{S}$ has the so called *downward closure* property, to be more formal: let U be a set such that $\mathcal{S} \subseteq 2^U$ then for each $X \in \mathcal{S}$ and each $Y \subseteq X$ we have $Y \in \mathcal{S}$. In particular this mean that we can enumerate all the solutions in $\mathcal{S}$ by extending them in all possible ways. We start from the empty set recursively and once reached the end of the current branch we *backtracing* and exploring other paths. The problem of no redundancy of solutions can be solved by adding elements of U to enlarge the current solution by choosing them in an arbitrary but fixed order.
In this way it is possible to avoid redundant solutions by extending with elements greater

than all the current ones, without having to store an exponential number of solutions. An example is given by Algorithm 1 taken from [58].

It is implied in this schema that when we are able to answer in polynomial time to the question: "given X are there any other supersets of $X' \in \mathcal{S}$?" , then there exist a polynomial delay algorithm which can enumerate the elements of $\mathcal{S}$.

Algorithm 1: BACKTRACK(G)

Input: A solution $G \in \mathcal{S}$
Output: All the other solutions containing G.

```
1 output G;
2 foreach e ⪰ max_{g∈G} do
3     if {e} ∪ G ∈ S then
4         BACKTRACK({e} ∪ G)
```

Binary Partition

This schema can be applied when we can explore the set of solutions by partitioning it according to some property and we can output a solution only if the partition is a singleton, *i.e.* a leaf of the recursion tree we are defining in this way. In general, if there exists an oracle which can partition the the set in polynomial time and the height of the tree is bounded by a polynomial in the input size, we have a polynomial delay algorithm. If each partition does not form empty sets, the number of internal nodes is bounded by the number of leaves and then we have a total polynomial time algorithm.

This method have been successfully employed to enumerate for example: st-paths in undirected graphs [36], perfect matchings in bipartite graphs [86], k-trees in undirected graphs [37].

As an example we show how we can enumerate with Algorithm 2 the elements of the set $G[s,t]$ of all the paths from a node s to a node t in a graph G. We partition the current set in two sets: the set of paths which include an edge and the set of paths which do not include it.

To this extend we define the graph $G - v$ as the graph G with the node $v \in V(G)$ removed together with all the edges incident to it, and $G - e$ as the graph G with the edge $e \in E(G)$ removed. Let v be a node in the neighbourhood of s, then $G[s,t] = (G-v)[s,t] \cup \big(G - (s,v)\big)[v,t]$.

We can then define a recursive procedure up to the case when $s = t$ where we reach the leaf of the recursion tree and we are ready to output the solution, as done in Algorithm 2.

Finally to design an efficient algorithm we should assure to do not waste time in descending a branch of the recursive tree without finding a solution at the end or at least

avoiding to pay this cost too many times. We can do this by check if there exists at least a path between s and t in the current graph by performing a DFS (*Deep First Search*).

Algorithm 2: PATHs(G,s,t,π)

Input: An undirected graph G, two of its nodes s, t and a path π initially empty
Output: All the paths in G from s to t.

```
1  if s = t then
2  |  output π;
3  |  return;
4  Choose an arc e = (s, r);
5  if G − s[r, t] = ∅ then
6  |  PATHs(G − e,s,t,π);
7  |  return;
8  if G − e[s, t] = ∅ then
9  |  PATHs(G − e,r,t,π ∪ {s});
10 |  return;
11 PATHs(G − s,r,t,π);
12 PATHs(G − e,s,t,π);
```

Reverse Search

This method, introduced by an article of Avis et al. [2], requires to turn the set $\mathcal{S}$ of elements to be enumerated into the nodes set of a tree by introducing a *father-children* relation so as to have a DAG. To enumerate all the solutions it is then enough to perform a *Depth First Search* where in each iteration it will be output a solution. If there is a procedure to pass from one solution to the next one then there is no need to store $O(tree\_height)$ solutions, as it is for the Backtracking method, to perform a *Depth First Search* of the tree.

With this method it is possible, for instance, to enumerate all the maximal cliques of a given graph $G_(V, E)$ as done by Tsukiyama et al. [85] and Johnson et al. [52] and then later reviewed by Makino et al. [57]. We present here under this last revision of Makino after few preliminaries.

Impose a fixed but arbitrary total order on the nodes of V, say $V = \{v_1, \ldots, v_n\}$ with $v_i < v_j$ iff $i < j$. Then we can induce an order among induced subgraphs of G by defining $C \succ D$, for $C, D \subseteq G$, if the smallest node in $V(C) \ominus V(D)$ belongs to C (where with $\ominus$ we mean the symmetric difference between sets, *i.e.* in our case $(V(C) \setminus V(D)) \cup (V(D) \setminus V(C))$).

For a clique $K \subseteq G$ let us define $\mathcal{C}(K)$ to be the $\succ$-greater clique among all the maximal cliques extension of K. For a subgraph C of G and for $1 \leq i \leq n$, let us define

the graph C_i defined as the subgraph of C induced by the nodes $V(C) \cap \{v_1, \ldots, v_i\}$. Let R be the $\succ$-greater maximal clique. A *parent-child* relation among maximal cliques can be given then defining the father $P(K)$ of a maximal clique $K \neq R$ to be the maximal clique $\mathcal{C}(K_i)$ such that i is the largest index for which $\mathcal{C}(K_i) \neq K$.

Observe that in this way $P(K)$ is $\succ$-greater than K, hence the induced direct graph in $\mathcal{S}$ is acyclic. Furthermore, since only R has no father according to this definition, it is a tree rooted at R which can be explored by implementing a procedure to compute the i-th children of a maximal clique as in [57].

For sake of completeness, observe that the reverse search approach actually is just a particular application of a general view said *transition graphs* or *super-graph* [60]. This more general approach aims at traverse a graph whose nodes are the element of S to list and the edges are defined by some neighbourhood function; in our example we are traversing a tree with a DFS. In the general case where the graph may have cycles, to avoid redundancy we can construct, for instance, a spanning forest of the graph, or a spanning tree.

This method has been employed implicitly in all the methods which slightly modify a solution to obtain another [52, 85] and in a explicit way in [10, 43, 45].

Cytoplasmic Incompatibility and Chain Graphs

1.1 Introduction and Motivations

Cytoplasmic Incompatibility (CI) refers to one solution adopted by intracellular parasites such as *Wolbachia* or *Cardinium hertigii* [47, 51] to spread themselves among the hosts they infect by interfering with the breeding process, preventing the formation of a healthy offspring with the uninfected host. The phenomenon has been noticed by Marshall [59] in the late 30's who observed that some male mosquito strains of *Culex pipiens* failed to accomplish a breeding with females of another strain, while the breeding between females of the first strain and males of the second one was successful.

We will take as example of this phenomenon the bacterium *Wolbachia* and describe some characteristics, aspects and models related to CI. *Wolbachia* is an endosymbiont (*i.e.* an organism that lives within the cells of another) which lives in the gametes of many insects which act as their hosts [50]. To underline the extent of interest in this phenomenon and in *Wolbachia*, observe that it is believed *Wolbachia* has an incidence of infection of 67% among all the insect species [50] with a slightly higher prevalence among arthropods [77]. These data are interesting for their prospective applications as insect pest control or gene spreading among the population. One of the main scientific interests of Cytoplasmic Incompatibility is the possibility to reduce the population size of the hosts or employ the parasite as a vector to spread some genes among the population. Indeed, as done in experiments with *Culex pipiens* [56] and the medfly *Ceratitis capitata* [56], by releasing in the population a huge quantity of infected males it is possible to reduce the size of the next generation population.

Different approaches to reduce the size of the population are obtained by employing CI-inducing bacteria which may reduce the lifespan of the infected host as experimentally done for the mosquito *A. aegypti* [61].

Finally, another source of interest is the presumed capability of CI to induce a speciation in the host population [88]. Indeed, the impossibility to breed between two populations with incompatible infections may promote a higher genetic variation between them than within each population. Speciation events and the phenomenom of coevolution are also in the center of our interest in Chapter 2.

	Fem wHa	Fem wNo	Fem wRi	Fem (uninfected)
Male wHa	1	0	0	0
Male wNo	0	1	0	0
Male wRi	0	0	1	0
Male (uninfected)	1	1	1	1

Table 1.1: Bidirectional CI in *Drosophilia simulans*

Furthermore, *Wolbachia* may provide a positive effect on the fitness of infected elements in the population for instance immunological capacity to resist to some viruses as for the fruit fly *Drosophila simulans* [69]. This grants stable infection of the bacteria in the population [3] and makes it a good CI vector.

The bacterium *Wolbachia* spreads mostly through the maternal cytoplasm and the phenomenon observed by Marshall is due to the crossing of infected males and uninfected females. The males are in principle seen by the bacterium as dead ends because the male parent does not share its cytoplasm with the offspring and the bacteria cannot spread in the cytoplasm of the children. In turn, the males are useful in promoting the diffusion of the bacterium by preventing the breeding with uninfected females. In this way, the infected females become fitter in the population as they are able to accomplish a successful breading with infected and uninfected males. An uninfected male can successfully breed with both infected and uninfected females as, intuitively, infected females should spread the bacterium as much as possible.

In the literature, this asymmetry between the failure to breed of an infected male with an uninfected female on one hand, and the successful breeding of an uninfected male with an infected female on the other hand is called *unidirectional CI*. There are also cases where crossings of males and females infected with different strains of *Wolbachia* are not possible. A classical example is the one of the fruit fly *Drosophila simulans* [62] presented here in Table 1.1 where a 1 (resp. a 0) indicates a successful (resp. unsuccessful) breeding between a male and a female of the fly infected by three different strains of *Wolbachia*, (denoted as wHa, wNo, wRi). The last column and row are control groups of uninfected individuals.

This impossibility to successfully breed for a host infected by different strains is referred to in literature as *bidirectional CI*.

The exact biological mechanism behind the CI phenomenon remains unknown, although there are studies which support the hypothesis that the development of the embryos is disturbed or interrupted as the paternal genetic information results damaged [13, 18, 87].

The population structure, *i.e.* the geographical and physical properties of the population, is another factor which may influence the phenomenon of Cytoplasmic Incompatibility, the diffusion of the CI-inducing bacterium and the co-existence of different strains of bacteria. In [31], according to the different mathematical models employed (which vary on different descriptions of the geographical distribution of the population and different

diffusion rules), the authors suggest interesting and various aspects involved in the CI diffusion such as invasion threshold for a stable and successful infection or phenomena of bidirectional CI and coexistence of incompatible CI-inducing bacteria in situation of geographically sparse populations.

In this part of my thesis, we will present our scientific contribute to the understanding of the mechanism behind CI starting from the model described in [65] with some excursions on related topics in computer science regarding graph covers and poset dimension theory. More specifically, the following of this chapter is structured as follows: in Section 1.2 we present an overview on the Toxin and Antitoxin Model, which is the CI model we adopt.

In Section 1.3 we introduce the required notation and graph theoretical objects such as Chain Graphs and Interval Orders.

In Section 1.4 we show how the computation of optimal solutions for the Toxin and Antitoxin Model is equivalent to the graph theoretical *Minimum Chain Subgraph Covers Problem*.

Section 1.5.1 provides a new polynomial delay algorithm to enumerate all maximal chain subgraphs of a bipartite graph, and an upper bound on their maximum number. We use the latter result to further establish the input-sensitive complexity of the enumeration algorithm.

In Section 1.5.2 we detail the exact algorithm for finding the minimum size of a minimum chain cover in bipartite graphs, and in Section 1.5.3 we exploit the connection of this problem with the minimal set cover of a hypergraph to show that it is possible to enumerate in quasi-polynomial time all minimal covers by maximal chain subgraphs of a bipartite graph.

Section 1.6 deals with the relation of the results in Sections 1.5.1 and 1.5.2 with some classical problems of partial order theory.

1.2 Models for the Cytoplasmic Incompatibilty

Regarding the mechanism of CI, we know that the host nuclear variation is not responsible for the success of a breeding (just to cite two of many evidences on this, see [29, 44]). We know also that in some species, for instance the parasitic wasp *Nasoni vitripennis* [14] and the fly *Drosophila simulans* [18], the breeding between infected males and uninfected females is jammed by the delay of the paternal chromosome in joining the maternal one and proceed with the development of the embryo.

In general the mechanism behind the phenomenon of CI is not completely clear for some difficulties in terms of genetic tools. For instance, in the case of *Wolbachia*, there are difficulties in identifying involved genes [65]. However, there are methods involving phenotypic data only (*i.e.* which symbiont infects which host) that can give a satisfying primary idea on the number of genes involved (see under the Lock and Key Model and the Toxin and Antitoxin Model for further details). My research in the area is based

on this phenotypic approach employing the Toxin and Antitoxin Model presented here under. We developed an algorithm which can help to cast light on the number of involved genetic factors starting from the observation of the patterns of successful breeds.

In the rest of this section, we present some of the models used in the past and the current state of the art concerning the genetic and molecular mechanism behind CI.

The Modification and Rescue Model

A first widely known explanation of CI is the *modification/rescue* model [88]. This model describes the phenomenon of CI as a "modification" of the male sperm by means of some function *Mod* acting on an infected male. Subsequently, the modified sperm is rescued in an infected female by some *Resc* function. This model was employed for a long time as it has the merit of being simple, although paying the price of the inability of explaining bidirectional CI (if not assuming the independence of the *Mod* and *Resc* functions, hypothesis supported in [75], or assuming some hypotheses on the population structure as in [31]). In this model, the *Mod* and *Resc* mechanisms are assumed to be deeply related and they follow the same evolution, *i.e.* the presence in the population of some elements carrying a *Mod* function suggests the existence in the same population of a carrier of the corresponding *Resc* function as both are supposed to depend on the same genes and hence to diffuse at a same time in the population.

The hypothesis of the independence of the *Mod* and *Resc* functions is, on the contrary, a cornerstone in the Lock and Key Model and the Toxin and Antitoxin Model both presented here under. On the other hand, there are other evolutions of the *Mod* and *Resc* Model towards models closer to experimental observations such as the Mistiming and Goal Keeper models which both still support the dependence of the functions *Mod* and Resc. These two other models are presented after for sake of completeness but are less flexible in terms of representative capabilities than the Toxin and Antitoxin Model which we are using.

The Lock and Key Model

The *Lock and Key Model* [65] gives a description of these *Mod* and *Resc* functions in terms of independent molecules present in sperm and eggs that are called Lock and Key, respectively. These molecules act, as the name itself suggests, as a lock put on the sperm which is successively unlocked by the proper key retained in the egg. Compared to the *Mod* and *Resc* model, the Lock and Key one has the positive aspect of describing *Mod* and *Resc* functions in terms of biological proteins, and has also the capability to explain bidirectional CI supposing that each strain of the bacterium carries its own Lock and Key pair. In this way we can easily explain, for instance, the observations made for the fruit fly *Drosophila simulans* [62] in Table 1.1 by assuming that each strain of bacterium has its own specific Lock/Key pair. However, assuming just two single *loci*, *i.e.* the locations

Line ID (males \ females)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
A	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1
B	1	1	0	1	1	1	1	0	1	1	0	1	1	1	1	0	1	1	1
C	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
D	1	1	1-0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
E	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
F	1	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
G	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
H	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
I	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
J	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
K	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
L	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
M	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
N	0	1	0	1	1	1-0	1	0	0	0	0	1	1	1	1	1	1	1	1
O	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
P	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
Q	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
R	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
S	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1

Figure 1.1: *Culex pipiens* compatibility matrix. The yellow cells are an example of different strains mutually compatible explainable with a single Lock/Key pair. Although the introduction of the third strain, the blue cell, claims the existence of other Lock/Key pairs, more specifically: a second Lock for the strain B whose Key is possessed by the strain A,B but not by the strain C.

of the genetic rules governing the formation of the molecules Lock or Key, is not enough. We should assume the existence of multiple *loci* (hence different types of Lock and Keys molecules) to explain more complex patterns such as the ones observed for the mosquito *Culex pipiens* that was presented in earlier studies of Duron *et al.* [29]. In the matrix of Figure 1.1 where rows represent the male and columns the female strains. In order to explain the values of the yellow and blue colored cells, it is necessary to assume a multiple *loci* model which may express different Lock and different Key molecules at the same time (idea supported by a probable duplication and subsequent modification of the original genetic information, see [65] and under).

In [65], this model is tested on the dataset in Figure 1.1. The results were satisfactory displaying on average a situation where multiple Key *loci* are required in the females and in a higher number with respect to the fewer Lock *loci* required for the males as shown in Figure 1.2 where an 1 is placed if the Lock/Key corresponding to the given column is present in the strain identified by the given row.

The multiple *loci* hypothesis is also in line with current theories on the evolution of CI [20, 21, 32, 65] which claims that females with many Keys are more fit in the population. Indeed they can be fertile with many more partners than uninfected females, fact that gives them an evolutionary advantage; on the contrary for the male host it is convenient not to have a symbiont which imposes many bounds or it will augment the risk of both host and symbiont becoming extinct.

Strain	Lock 1	Lock 2	Lock 3	Lock 4	Lock 5	Lock 6	Lock 7	Lock 8	Key 1	Key 2	Key 3	Key 4	Key 5	Key 6	Key 7	Key 8
A				1			1		1		1	1	1		1	
B	1			1					1	1	1	1	1		1	1
C		1		1	1		1			1		1	1		1	
D				1					1	1	1	1	1		1	1
E				1					1	1	1	1	1		1	1
F			1						1	1	1	1	1		1	1
G		1		1			1		1	1	1	1	1		1	1
H					1	1				1	1		1	1	1	1
I							1		1		1	1			1	
J				1					1	1	1	1	1	1		
K							1			1	1	1	1		1	
L				1					1	1		1	1		1	1
M				1					1	1		1	1		1	1
N				1			1	1	1	1	1	1	1		1	1
O		1		1					1	1	1	1	1		1	1
P				1	1					1	1	1	1		1	1
Q		1		1			1		1	1	1	1	1		1	1
R		1		1			1		1	1	1	1	1		1	1
S		1		1			1		1	1	1	1	1		1	1

Figure 1.2: Figure 4 in [65], which reports: "Best confidence solution of the binary model. The value in each cell indicates whether we infer presence (1) or absence (empty cell) of this factor in this particular *Wolbachia* strain. Gray cells contain a value inferred in at least 90% of the solutions of minimum size"

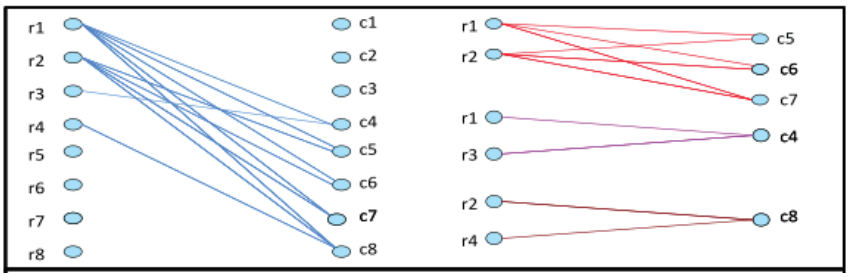

Figure 1.3: Figure 2 in [66], a bipartite graph on the left, and a biclique cover of size 3 on the right.

Computing the number of Lock/Key pairs necessary to explain a given patterns of (successful or failed) breedings can be done by employing some heuristics and algorithms related to biclique covers of bipartite graphs [66]. A *bipartite graph* is a graph $G = (U \cup V, E)$ whose node set can be partitioned into two independent sets U, V. If G is such that $(u, v) \in E$ for all $u \in U, v \in V$ G then it is called a *biclique*. A *biclique cover* is an edge cover of a bipartite graph made of biclique subgraphs (Figure 1.3).

In the case of the Lock and Key Model, the pattern of successful breedings (for instance the one in Figure 1.1) is reduced to a bipartite graph with two sets of nodes, the males and females infected with different strains of bacteria, and where the edges between a pair of nodes represent a successful breeding. A biclique subgraph coincides with a Lock/Key pair of this model. There is an interest in finding minimum biclique covers (the so-

called *bipartite dimension*) which coincide with the minimum number of Lock/Key pairs necessary to explain the breeding pattern.

The Bipartite Dimension or *Biclique cover number* consists exactly in finding the size of a minimum edge cover of a bipartite graph by means of bicliques. The decision problem associated to it is known to be NP-complete [68] (see for instance [1] for a list of equivalent reformulations) while in [66] it has been proved to be in FPT. Deciding if there exists a bipartite cover of size k can thus be done in time $O(2^{k2^{k-1}+3k} + \max(m,n)^3)$ where n is the number of nodes and m the number of edges of the bipartite graph in input.

There is also an interest in the enumeration of all the covers of minimum size for the relation they have with the number of Lock/Key *loci* as each element of the cover corresponds in the model to a Lock/Key pair.

The Toxin and Antitoxin Model

The *Toxin and Antitoxin Model* can be seen as a quantitative version of the Lock and Key model [65]: the Toxin is identified as the Lock put on the sperm and the Key now is the Antitoxin in the egg which can neutralize the Toxin if it is present at least in the same quantity as the corresponding Antitoxin. Indeed, if in the binary version it was only possible to note the presence or absence of a given Lock or Key, in this model we are able to express a quantity of its presence as a non-negative integer. A breeding is successful if in the egg are present all the Antitoxins for all the Toxins and *in enough quantity*. In particular in this model, the symbiont strains vary also on the quantity of Lock/Key produced.

In this model, as in the Lock and Key Model, the genetic factors responsible for the Toxins and the Antitoxins are independent, *i.e.* the Keys are encoded by different genes and alleles than the Locks.

Being an extension of the Lock and Key model, each solution within that model can directly beinterpreted as a solution to this one. The implication is that the minimum number of Lock/Key pairs is always an upperbound for the minimum number of Toxin/Antitoxin pairs.

Two More Concrete Model Yet Less Powerful: The Mistiming Model and The Goalkeeper Model

Another extension of the Modification and Rescue Model is the **Mistiming Model**. This model has a quantitative representation of the *Mod* and *Resc* functions and supports the experimental evidence [14, 18] of the difficulties encountered by the male chromosome to participate in the formation of embryonic cell hindered by the symbiont. In this model, the breeding is successful only if the female chromosome has also a time delay greater than the one of the paternal chromosome.

On the contrary, the successful breeding between infected females and uninfected males

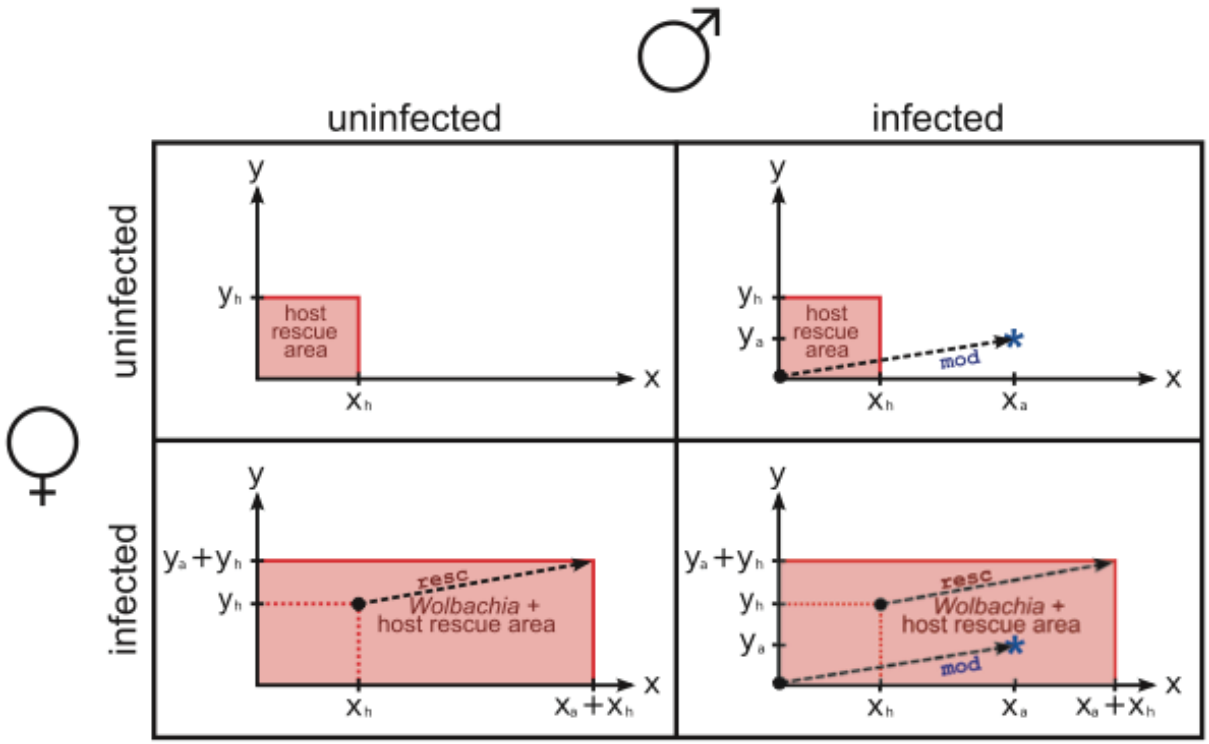

Figure 1.4: Visual example for the Goalkeeper Model: The top right breeding is the only one not successful as the *Mod* point falls outside the Rescue area.

is justified arguing that the chromosome from uninfected males does not suffer the delay related to the infected female but it will suffer just a slowed cell cycle.

This model is then further extended to the **Goalkeeper Model**, which describes each symbiont strain by means of two positive values which were interpreted by Bossan *et al.* as the time delay induced and another not well defined factor. Both factors are present in the same quantity in the sperm and the egg implying then a dependence of *Mod* and *Resc* functions. Furthermore, the model claims for the female a contribution to these two factors which is independent on the infecting symbiont strain and is hence constant among bacterial strains.

The name *Goalkeeper Model* comes from the visual interpretation of these two factors as in this model, the rescue of the offspring is performed (see Figure 1.2) if the point identified on the Cartesian plane by the two factors of the male given only by the symbiont strain, say (x, y), falls within a rescue area. This rescue area is a rectangle which is the convex-hull of the points $(0,0), (x+x_f, 0), (0, y+y_f), (x+x_a, y+y_a)$ where x_a, y_a represent the female host contribution.

The two factors per symbiont strain of this model have the power to explain bidirectional CI, which was not possible with only one factor as in the Mistiming model.

According to the authors in [11], in the Lock and Key Model (and the Toxin and Antitoxin Model we add), it is left unexplained how it is possible to match in a population the independently evolved modifications of Lock and Keys functions when this traits

should be soon extinct if not supported by other evolutionary forces which would make more fit these modified versions. On the contrary, the assumption of the correlation between *Mod* and *Resc* functions in the Goalkeeper Model, prevents the arising of this theoretical problem.

Observe that we can interpret the two quantitative factors of this model as two Toxin/Antitoxin pairs of the Toxin and Antitoxin. In this way it is clear that the Goalkeeper Model results in a special case of the Toxin and Antitoxin Model.

In the following, we introduce the mathematical formulation of the Toxin and Antitoxin Model after some theoretical preliminaries necessary also to prove the relation of this model with the *Chain Subgraphs Cover Problem*.

1.3 Preliminary Notation

In this section, we introduce the notation we use and we assume that the reader is familiar with the standard graph terminology, as contained for instance in [8]. We consider finite undirected bipartite graphs without loops or multiple edges. Bipartite graphs arise naturally in many other applications besides the ones addressed in this thesis since they enable to model the relations between two different classes of objects. For each of the graph problems in this thesis, we denote by n the number of nodes and by m the number of edges of the input graph.

Given a bipartite graph $G = (U \cup W, E)$ and a node $u \in U$, we denote by $N_G(u)$ the set of nodes adjacent to u in G, u excluded, and by $E_G(u)$ the *set of edges incident to* u *in* G. When it is clear the graph G in which the neighbourhood is taken, we may omit it from the notation simply writing $N(u)$ or $E(u)$ respectively. Moreover, given $U' \subseteq U$ and $W' \subseteq W$, we denote by $G[U', W']$ the *subgraph of* G *induced by* $U' \cup W'$. A node $u \in U$ such that $N_G(u) = W$ is called a *universal node*.

Chain Graphs

A central role in this part of the thesis is played by *chain graphs* (also called *difference graphs*).
As we prove in Section 1.4, they are related to the graph interpretation of the quantitative model Toxin and Antitoxin (presented in Section 1.2). Although chain graphs are interesting in themselves as graph theoretical objects, they appear in many other applications. Indeed, not only are they related to other important graph classes such as *threshold* graphs [49] and *interval graphs* [91], but they can also be directly interpreted as the comparability graph of *height one partial orders*.

Hereafter, we introduce chain graphs and some of their properties that will be used in this thesis.

Definition 6. *A bipartite graph is a* chain graph*, or also said* difference *graph, if it does not contain a* $2K_2$ *as an induced subgraph.*

There are several equivalent definitions of chain graphs as shown by the next theorem. For further equivalent characterizations and the omitted proofs, we refer to [49]:

Theorem 1. *[49] The following conditions are equivalent for a bipartite graph* $G = (U \cup V, E)$*:*

1. *G is a chain graph,*
2. *For each two nodes* $u_1, u_2 \in U$ *(or equivalently* V*) either* $N_G(u_1) \subseteq N_G(u_2)$ *or* $N_G(u_1) \subseteq N_G(u_2)$*,* i.e. *there exist a total order for the elements of* U *(equivalently* V*) given by neighbourhood inclusion,*
3. *We can partition the set* U *in the sets* $U_0, \ldots, U_k$*, the set* V *in* $V_0, \ldots, V_k$ *for a positive integer* k *and such that for each* $u \in U_i$*,* $v \in V_j$ *we have that* $(u, v) \in E$ *iff* $i + j > k$*, the sets* $U_1, \ldots, U_k, V_1, \ldots, V_k$ *are said a* degree partition *of* G*.*
4. *G can be constructed from the empty graph by repeatedly adding an isolated node or a node adjacent to all previous ones,*
5. *there exists a function* $w : V(G) \to \mathbb{R}$ *bounded by a positive integer* $T > 0$ *such that for all* $u \in U$*,*$v \in V$ *it holds* $(u, v) \in E(G)$ *iff* $|w(u) - w(v)| \geq T$*.*

In the following we heavily employ item 2. of Theorem 1, and for this reason we introduce the following definition:

Definition 7. *Given a chain subgraph* $C = (X \cup Y, F)$ *of* G*, we say that a permutation* π *of the nodes of* U *is a* neighbourhood ordering *of* C *if* $N_C(u_{\pi(1)}) \subseteq N_C(u_{\pi(2)}) \subseteq \ldots \subseteq N_C(u_{\pi(|U|)})$*.*

Observe that if $X \subset U$, the sets $N_C(u_{\pi(1)}), \ldots, N_C(u_{\pi(l)})$ for some integer $l \leq |U|$, may be empty and, if C is connected, $l = |U| - |X|$. By *largest neighbourhood of* C we mean the neighbourhood of a node x in X for which the set $N_C(x) \subseteq Y$ has maximum cardinality. A set $Y' \subseteq Y$ is a *maximal neighbourhood* of G if there exists $x \in X$ such that $N_G(x) = Y'$ and no node $x' \in X$ is such that $N_G(x) \subset N_G(x')$. Two nodes x, x' such that $N_C(x) = N_C(x')$ are called *twins*.

In this chapter, we always consider *edge induced* chain subgraphs of a graph G. Hence, in the following we consider a chain subgraph C of G to be identified by its edges $E(C) \subseteq E(G)$ and in that case its set of nodes will be constituted by all the nodes of G incident to at least one edge in C. With this interpretation, a chain graph is characterized by the set of its edges; for this reason we sometimes abuse the notation *e.g.*: $C \setminus E(D)$, with D a subgraph of G, to denote the chain subgraph induced by edges $E(C) \setminus E(D)$, $C \subseteq E(D)$ or equivalently $C \subseteq D$ to say that C is an edge-induced subgraph of D and $e \in C$ means that $e \in E(C)$.

Definition 8. *A* maximal chain subgraph *C of a given bipartite graph G is a connected chain subgraph such that it is not a connected chain subgraph for any other subgraph of G than itself. We denote by $\mathcal{C}(G)$ the set of all maximal chain subgraphs in G.*

Definition 9. *A set of chain subgraphs $C_1, \ldots, C_k$ is a* cover *for G if $\cup_{1 \leq i \leq k} E(C_i) = E(G)$. We denote by $\mathcal{S}(G)$ the set of all minimal chain covers of a bipartite graph G.*

Observe that, given any cover of G by chain subgraphs $C = \{C_1, \ldots C_k\}$, there exists another cover of same size $C' = \{C'_1, \ldots C'_k\}$ whose chain subgraphs are all maximal; more precisely, for each $i = 1, \ldots, k$, C'_i is a maximal chain subgraph of G and C'_i admits C_i as subgraph.

On n bipartite nodes, the number of different unlabeled chain graphs is 2^n and the number of labeled chain graphs d_n is:

$$d_0 := 1, \qquad d_n = 2 \sum_{k=0}^{n} k! S(n,k), \quad n \geq 1,$$

where $S(n,k)$ are the Stirling numbers of second kind. Both these results were obtained by Peleg and Sun in [74]. In the same paper, an asymptotic evaluation for d_n of $O(n!/(\ln 2)^{n+1})$ can be found.

On the other hand, as we observe also in Section 11, a tight upper bound on the number of maximal chain subgraphs in a connected graph with n nodes is $O((n/2)!)$. This bound comes from the branching tree of the algorithm to enumerate all maximal chain subgraphs of a given bipartite graph and is described in Section 11. It also provides an upper bound in terms of edges, so a connected bipartite graph with m edges can have $O(m^3 2^{\sqrt{m} log(m)})$ maximal chain subgraphs.

To the best of our knowledge we do not know any other results of this kind.

1.4 Toxin and Antitoxin: The graph theoretic interpretation

The Toxin and Antitoxin model claims that a breeding in presence of CI is successful if the egg contains all the antitoxins to all the toxins present in the sperm of the infected male [65]. We can model this by introducing the following operation between positive integer-valued vectors:

Definition 10. *The* quantitative $\otimes$ vector multiplication *is a binary external anti-symmetrical operation between two vectors $U, V \in \mathbb{N}^k$, for a strictly positive integer k, defined as follows:*

$$U \otimes V := \begin{cases} 1 & \text{if } U[i] > V[i] \text{ for some } i \in \{1, 2, \ldots, k\}, \\ 0 & \text{otherwise} \end{cases}$$

Interpreting vectors in $\mathbb{N}^k$ as matrices with a single row in $\mathbb{N}^{1\times k}$, we can directly extend this operation to a matrix multiplication operation as follows:

Definition 11. *The* quantitative row-by-row matrix multiplication $\otimes$ *is defined as follows: given two matrices* $M \in \mathbb{N}^{m\times k}$ *and* $R \in \mathbb{N}^{r\times k}$ *with* m, r, k *positive integers, then* $M \otimes R := C \in \{0,1\}^{m\times r}$ *where* $C_{i,j} = M_{i,\cdot} \otimes R_{j,\cdot}$ *for* $i \in \{1, 2, \ldots, m\}$ *and* $j \in \{1, 2, \ldots, r\}$.

In the biological setting, we are given a matrix $C \in \{0,1\}^{m\times r}$ which has one row assigned to each male host and one column assigned to each female host. The value at the given row and column is a boolean value representing the success or the failure to breed of the corresponding male and female. Our interest is exploring the set of the inverse images of the defined quantitative row-by-row matrix multiplication with the aim to solve the minimization problem:

Problem 2. *Given a binary matrix* $C \in \{0,1\}^{m\times r}$ *where* m, r *are positive integers, determine the minimum positive integer* k *such that there exists two matrices* $M \in \mathbb{N}^{m\times k}$ *and* $R \in \mathbb{N}^{r\times k}$ *such that* $M \otimes R = C$.

The columns of matrix M (resp. R) represent the different Toxin (resp. Antitoxin) quantity present in the infected males (resp. females). The k value represents the minimum number of Toxin/Antitoxin pairs necessary to explain pattern C.

Observe that for each bipartite graph $G = (U \cup W, E)$, we can associate a binary matrix $A(G)$, called the *bi-adjacency matrix of* G and defined by the following rule: let $U = \{u_1, \ldots, u_m\}$, $W = \{w_1, \ldots, w_r\}$ then $A(G)_{i,j} = 1$ iff $(u_i, w_j) \in E(G)$.

It is clear that each bipartite graph has a bi-adjacency matrix and vice versa.

The following lemma sheds light on graph theoretic concepts related to the minimum quantitative solution.

Lemma 1. $G = (U \cup W, E)$ *is a chain graph if and only if there exists a quantitative solution of size* 1 *in* $A(G)$.

Proof. Let G be a chain graph, then we can assume $U = \{u_1, \ldots, u_m\}$ to be ordered by neighbourhood inclusion, *i.e.* $N(u_i) \subseteq N(u_j)$ if $i \leq j$. Let us define $B_1, \ldots, B_n$ as n disjoint subsets of U as follows: $u_x \in B_i$ if and only if $|N(u_x)| = i$ for $1 \leq x \leq m$. Observe that B_0 is the set of isolated nodes of U and it may be empty, in that case we exclude it from the following reasoning. In general, as every node u in a set B_j has the same neighbourhood (being G a chain graph) we can extend the notation of neighbourhood as follows $N(B_j) := N(u)$ for u any node in B_j and for any $0 \leq j \leq n$.

We now construct two matrices $M \in \mathbb{N}^{|U|\times 1}$ and $R \in \mathbb{N}^{|W|\times 1}$ as follows: for $u_x \in B_i$ define $M[x][1] := i$ for $1 \leq x \leq m$ and define for $w_y \in N(B_j) \setminus \cap_{s=1}^{j-1} N(B_s)$ and $1 \leq y \leq r$ $R[y][1] := j - 1$.

The conclusion follows by this chain of implications: $C[x][y] = 1 \Leftrightarrow \exists e = (u_x, w_y) \in E(G) \Leftrightarrow u_x \in B_i$ and $w_y \in \cap_{s=1}^{i} N(B_s) \Leftrightarrow M[x][1] = i$ and $R[y][1] > i \Leftrightarrow M[x][1] \otimes R[y][1] = 1$.

Let us now assume that there exists a quantitative solution of size 1 for $C \in \{0,1\}^{m\times r}$. Then there exist $M \in \mathbb{N}^{m\times 1}$ and $R \in \mathbb{N}^{r\times 1}$ such that $C = M \otimes R$. Let us define the graph $G = (U \cap W, E)$ as follows: let $U = \{u_1, \ldots, u_m\}$, $W = \{w_1, \ldots, w_r\}$ be two sets of respectively m and r distinct elements and let $(u_x, w_y) \in E$ if and only if $M[x][1] \otimes R[y][1] = 1$. We will now prove that G is a chain graph as we can order the neighbourhood of the nodes of U by neighbourhood inclusion as either $N(u_x) \subseteq N(u_z)$ or $N(u_x) \supseteq N(u_z)$ (see point 2 of Theorem 1).

Suppose that $N(u_x) \subseteq N(u_z)$ does not hold and let us show then that $N(u_x) \supseteq N(u_z)$ must hold. If $N(u_x) \subseteq N(u_z)$ then there exists $w_y \in N(u_x) \setminus N(u_z)$ such that $M[x][1] \otimes R[y][1] = 1$ while $M[z][1] \otimes R[y][1] = 0$, this implies that $M[x][1] > R[y][1] > M[z][1]$ and it follows for all $1 \leq l \leq r$ that $w_l \in N(u_z) \Leftrightarrow M[z][1] > R[l][1] \Rightarrow M[x][1] \otimes R[l][1] = 1 \Leftrightarrow w_l \in N(u_x)$,*i.e.* $N(u_x) \supseteq N(u_z)$. □

From the next Theorem, we deduce that there exists a relation between the size of a chain subgraph cover of a bipartite G and the number of columns of two matrices M, R such that $M \otimes R = A(G)$. We can then solve the Problem 2 by computing the minimum size of a chain graph cover for $A(G)$.

Theorem 2. *Let $G = (U \cup W, E)$ be a bipartite graph. Then G has a chain subgraph cover of size k if and only if there exist two matrices $M \in \mathbb{N}^{m\times k}$ and $R \in \mathbb{N}^{r\times k}$ for m, r positive integers such that $M \otimes R = A(G)$.*

Proof. In all the proof let $U = \{u_1, \ldots, u_m\}$, $W = \{w_1, \ldots, w_r\}$ for m, r positive integers. For the only if part of the theorem, let $C_1, \ldots, C_k$ be a chain subgraph cover for G. Without lack of generality (modulo a permutation of the nodes in U, W), we can assume for $1 \leq i \leq k$ that $C_i = (U_i \cup W_i, E_i)$ with $U_i := \{u_1, \ldots, u_{m'}\}$, $W_i := \{w_1, \ldots, w_{r'}\}$ for $m' \leq m, r' \leq r$. By Lemma 1 there exist two matrices $M_i \in \mathbb{N}^{m'\times 1}$ and $R_i \in \mathbb{N}^{r'\times 1}$ for m', r' positive integers such that $M_i \otimes R_i = A(C_i)$. Then for $1 \leq i \leq k$ define column i of matrices $M \in \mathbb{N}^{m\times k}$ and $R \in \mathbb{N}^{r\times k}$ as

$$M[x][i] := \begin{cases} M_i[x][1] & \text{if } u_x \in U_i \\ 0 & \text{otherwise} \end{cases} \qquad R[y][i] := \begin{cases} R_i[y][1] & \text{if } w_y \in V_i \\ \max_{i:x\in U_i} M_i[x][1] & \text{otherwise} \end{cases}$$

Observe that $M[\cdot][i] \in \mathbb{N}^{m\times 1}$ and $R[\cdot][i] \in \mathbb{N}^{r\times 1}$ and that for each chain subgraph C_j we can analogously define $M[\cdot][j], R[\cdot][j]$. The conclusion follows being $C_1, \ldots, C_k$ a chain subgraph cover for G and observing that with this definition $M[x][i] \otimes R[y][i] = 1$ if and only if for some chain C_i we have $u_x \in U_i$, $w_y \in V_i$ and $M_i[x][1] \otimes R_i[y][1] = 1$ *i.e.* $(u_x, w_y) \in E(C_i)$.

The if part of the theorem follows trivially. Assume than that $M \otimes R = A(G)$. Then for each $1 \leq i \leq k$, we define the subgraph $C_i = (U \cup W, E_i)$ of G such that$(u_x, w_y) \in E_i$ if and only if $M[x][i] > R[y][i]$ for $1 \leq x \leq m$ and $1 \leq y \leq r$. Clearly with this construction $A(C_i) = M[\cdot][i] \otimes R[\cdot][i]$ and by Lemma 1 it is hence a chain subgraph of G. The set

$\{C_1, \ldots, C_k\}$ are an edge cover for G as we have that $(u_x, w_y) \in E(G)$ if and only if $C[x][y] = 1$ which occurs if and only if there exists $1 \leq i \leq k$ such that $M[x][i] > R[y][i]$ hence $(u_x, w_y) \in E_i$ by construction of C_i. □

As we are interested in chain subgraph covers of minimum size, in order to reduce their number and to avoid unnecessary redundancies, from now on we will restrict our attention to maximal chain subgraphs and covers of a bipartite graph by maximal chain subgraphs.

1.5 Chian Graphs and Bipartite Edge Covers

In this section, we address the problem of enumerating without repetitions all maximal *edge induced* chain subgraphs of a bipartite graph (Section 1.5.1). If there is no ambiguity, from now on we will refer to them simply as *chain subgraphs* omitting the wording "edge induced". We present, then, in Section 1.5.2 a polynomial space exact algorithm to compute the size of a minimum chain subgraph cover of a bipartite graph and in Section 1.5.3 a quasi-polynomial algorithm to enumerate all the such minimal covers.

1.5.1 Enumerating Maximal Chain Subgraphs

The problem of enumerating in bipartite graphs all subgraphs with certain properties has already been considered in the literature. These concern for instance maximal bicliques for which polynomial delay enumeration algorithms in bipartite [24, 57] as well as in general graphs [23, 57] were provided. In the case of maximal *induced* chain subgraphs, their enumeration can be done in total polynomial time as it can be reduced to the enumeration of a particular case of the minimal hitting set problem [30] (where the sets in the family are of cardinality 4). However, the existence of a polynomial delay algorithm for this problem remains open.

Regarding the problem of enumerating maximal *edge induced* chain subgraphs in bipartite graphs, in [34] the authors deal with it in the form of enumerating minimal interval order extensions of interval orders (see Section 1.6 for the relation between these two problems).

We improve this result by proposing a polynomial space and polynomial delay algorithm to enumerate all maximal chain subgraphs of a bipartite graph. We also provide an analysis of the time complexity of this algorithm in terms of input size. In order to do this, we prove some upper bounds on the maximum number of maximal chain subgraphs of a bipartite graph. This is also of intrinsic interest as combinatorial bounds on the maximum number of specific subgraphs in a graph are difficult to obtain and have received a lot of attention (see for *e.g.* [38, 64]).

We start by proving the following result:

Proposition 3. *Let $C = (X \cup Y, F)$ be a chain subgraph of $G = (U \cup W, E)$, with $X \subseteq U$, $Y \subseteq W$ and $F \subseteq E$, and let $x \in X$ be a node with largest neighbourhood in C. Then C is a maximal chain subgraph of G if and only if both the following conditions hold:*

(i) $N_C(x) = N_G(x)$ is a maximal neighbourhood of G, i.e. there does not exist a node $x' \in X$ such that $N_G(x) \subset N_G(x')$;

(ii) $C \setminus E_G(x)$ is a maximal chain subgraph of $G\big[U \setminus \{x\}, N_G(x)\big]$.

Proof. ($\Rightarrow$) Let $C = (X \cup Y, F)$ be a maximal chain subgraph of $G = (U \cup W, E)$. To prove that *(i)* holds, suppose by contradiction that $N_C(x)$ is not a maximal neighbourhood of G, *i.e.* there exists $x' \in U$ with $N_C(x) \subset N_G(x')$ (possibly $x' = x$). Since $N_C(x)$ is the largest neighbourhood of C, for all $z \in X$, we have $N_C(z) \subseteq N_C(x) \subset N_G(x')$, so we can add to C all the edges incident to x' and still obtain a chain subgraph thereby contradicting the maximality of C.

To prove that *(ii)* holds, first observe that $N_G(x) = Y$ (otherwise we would violate (i) with $x' = x$). By contradiction, assume that $C \setminus E_G(x)$ is not maximal in $G\big[U \setminus \{x\}, N_G(x)\big]$. Then, there exists a chain subgraph C' such that $C \setminus E_G(x) \subset C' \subseteq G\big[U \setminus \{x\}, N_G(x)\big]$. By adding to each one of the previous graphs the edges in $E_G(x)$, we have that the strict inclusion is preserved because the added edges were not present in any one of the three graphs. Since C' with the addition of $E_G(x)$ is still a chain subgraph with $N_G(x)$ as its largest neighbourhood, we reach a contradiction with the hypothesis that C is maximal in G.

($\Leftarrow$) We show that if both *(i)* and *(ii)* hold, then the chain subgraph C of G is maximal. Suppose by contradiction that C is not maximal in G, and let C' be a chain subgraph of G such that $C \subset C'$. Let x be the node with the largest neighbourhood in C. It follows that $N_C(x) \subseteq N_{C'}(x)$. As *(i)* holds, we have that $N_G(x) = N_C(x) \subseteq N_{C'}(x) \subseteq N_G(x)$ from which we derive that $N_{C'}(x) = N_G(x)$, and that $C' \subseteq G\big[U, N_G(x)\big]$ since $N_{C'}(x)$ is a maximal neighbourhood of G, hence the largest neighbourhood of C' (and C by the hypothesis). This implies also that C and C' differ in some node different from x, *i.e.* $C \setminus E_G(x) \subset C' \setminus E_G(x) \subseteq G\big[U \setminus \{x\}, N_G(x)\big]$. Notice that $C' \setminus E_G(x)$ is still a chain subgraph because we simply removed node x and all its incident edges. We then get a contradiction with *(ii)*. □

Proposition 3 leads us to design Algorithm 3 which efficiently enumerates all maximal chain subgraphs of G. It exploits the fact that, in each maximal chain subgraph, a node u whose neighbourhood is largest is also maximal in G (part *(i)* of Proposition 3) and this holds recursively in the chain subgraph obtained by removing node u and restricting the graph to $N_C(u)$ (part *(ii)* of Proposition 3). To compute the maximal neighbourhood nodes, the algorithm uses a function, `computeCandidates`, that, given sets U and W, returns for each maximal neighbourhood $Y \subset W$, a unique node u, called

candidate, for which $N_G(u) = Y$. This means that in case of twin nodes, the function `computeCandidates` extracts only one representative node according to some fixed order on the nodes (*e.g.* the node with the smallest label). If the graph has no edges, the function returns the empty set.

Algorithm 3: Enumerate All Maximal Chain Subgraphs

Input: A bipartite graph $G = (U \cup W, E)$
Output: All maximal chain subgraphs of G

```
1  C ⟵ ∅ ;                    /* C = set of edges of the current chain subgraph */
2  enumerateMaximalChain(U, W, C)
3      Candidates ⟵ computeCandidates(U,W);
4      if Candidates == ∅ then
5          print(C);
6          return;
7      for u ∈ Candidates do
8          U' ⟵ U \ {u}; W' ⟵ W ∩ N_G(u);                    /* reduced graph */
9          F(u) ⟵ {edges of E_G(u) incident to some node in W'};
10         C' ⟵ C ∪ F(u);
11         enumerateMaximalChain(U', W', C');
```

Proposition 4 (Correctness)**.** *Algorithm 3 correctly enumerates all the maximal chain subgraphs of the input graph G without repetitions.*

Proof. Let $G = (U \cup W, E)$ be a bipartite graph. We prove the correctness of Algorithm 3 by induction on $|U|$, *i.e.* we show that all the solutions are output, without repetitions.

When $|U| = 1$, let u be the only node in U. We have that $N_G(u)$ is the only neighbourhood in W, and line 3 returns $\{u\}$ as unique candidate. In line 9, the algorithm reduces the graph of interest. In line 10, the whole $E_G(u)$ is added to the current chain subgraph C. Then the function is recursively recalled, with $U' = \emptyset$ so the condition at line 4 is true and C is printed; it is in fact the only chain subgraph of G, it is trivially maximal and there are no repetitions. Correctness then follows when $|U| = 1$.

Assume now that $|U| = k$ with $k > 1$. As inductive hypothesis, let the algorithm work correctly when $|U| \leq k - 1$.

For each candidate u, the algorithm recursively recalls the same function on a reduced subgraph and, by the inductive hypothesis, outputs all chain subgraphs of this reduced subgraph without repetitions. By Proposition 3, if we add to each one of these chain subgraphs the node u and all the edges incident to u in $G[U, W]$, we get a different maximal chain subgraph of G since each maximal chain subgraph has one and only one maximal neighbourhood and the function `computeCandidates` returns only one representative node. Recall that in the case of twin nodes the algorithm will always consider the nodes in a precise order and so no repetition occurs. Moreover, iterating this process for all

candidates guarantees that all maximal chain subgraphs are enumerated and no one is missed. □

Let $G = (U \cup W, E)$ be a bipartite graph, with $n = |U| + |W|$ and $m = |E|$. Before proving the time complexity of Algorithm 3, we observe that the running time of the function `ComputeCandidates` is $O(nm)$. Indeed, for each node $u_i \in U$, it requires only time proportional to $\sum_{j=1}^{i-1} \big(deg(u_j) + deg(u_i)\big)$ to check whether the neighbourhood of u_i either is included, or includes the neighbourhood of u_j, for each $j < i$.

Proposition 5 (Time Complexity and Polynomial Delay)**.** *Let $G = (U \cup W, E)$ be a bipartite graph. The total running time of Algorithm 3 is $O(|\mathcal{C}(G)|n^2m)$ where $|\mathcal{C}(G)|$ is the number of maximal chains subgraph of G. Moreover, the solutions are enumerated in polynomial time delay $O(n^2m)$.*

Proof. Represent the computation of Algorithm 3 as a tree of the recursion calls of `enumerateMaximalChain`, each node of which stores the current graph on which the recursion is called at line 12. Of course, the root stores G and on each leaf the condition $Candidates == \emptyset$ is true and a new solution is output. Observe that each leaf contains a feasible solution, and that no repetitions occur in view of Proposition 4, so the number of leaves is exactly $|\mathscr{C}(G)|$.

Since at each call the size of U is reduced by one, the tree height is necessarily bounded by $|U| = O(n)$; moreover, on each tree node, $O(nm)$ time is spent for running function `ComputeCandidates`.

It follows that, since the algorithm explores the tree in DFS fashion starting from the root, between two solutions the running time is at most $O(n^2m)$ and the total running time is $O(|\mathcal{C}(G)|n^2m)$. □

Upper bounds on the number of Maximal Chain Subgraphs

In this section, we give two upper bounds on the maximum number of maximal chain subgraphs of a bipartite graph G with n nodes and m edges. The first bound is given in terms of n while the second depends on m. These bounds are of independent interest, however we will use them in two directions. First, they will allow us to determine a (input-sensitive) time complexity of Algorithm 3. Indeed, in Proposition 5, we proved that the total running time of Algorithm 3 is of the form $O(D(n) \cdot |\mathscr{C}(G)|)$, where $D(n)$ is the delay of the algorithm and $|\mathscr{C}(G)|$ is the number of maximal chain subgraphs of G. Thus, a bound on $|\mathscr{C}(G)|$ leads to a bound on the running time of Algorithm 3 depending on the size of the input. Second, the bound on $|\mathscr{C}(G)|$ in terms of edges allows us to compute the time complexity of an exact exponential algorithm for the minimum chain subgraph cover problem in Section 1.5.2.

Bound in terms of nodes

The following lemma claims that a given permutation is the neighbourhood ordering of at most one maximal chain subgraph.

Lemma 2. *Let C_1 and C_2 be two maximal chain subgraphs of $G = (U \cup W, E)$ and let π_1 (resp. π_2) be a neighbourhood ordering of C_1 (resp. C_2). Then, $\pi_1 = \pi_2 \Longrightarrow C_1 = C_2$.*

Proof. The proof proceeds by induction on the number of nodes of U.

If $|U| = 1$ then G has only one maximal chain subgraph and the result trivially holds. Assume now that $|U| > 1$. By Proposition 3, we have that $N_{C_1}(u_{\pi(|U|)}) = N_G(u_{\pi(|U|)}) = N_{C_2}(u_{\pi(|U|)})$. Using again Proposition 3, we obtain that $C_1' := C_1[U \setminus \{u_{\pi(|U|)}\}, N_G(u_{\pi(|U|)})]$ and $C_2' := C_2[U \setminus \{u_{\pi(|U|)}\}, N_G(u_{\pi(|U|)})]$ are maximal chain subgraphs of the graph defined as $G[U \setminus \{u_{\pi(|U|)}\}, N_G(u_{\pi(|U|)})]$. Applying the inductive hypothesis with the permutations restricted to the $|U| - 1$ elements, we have that $C_1' = C_2'$. Finally, since $N_{C_1}(u_{\pi(|U|)}) = N_{C_2}(u_{\pi(|U|)})$, we conclude that $C_1 = C_2$. □

As a corollary, the maximum number of chain subgraphs of a graph $G = (U \cup W, E)$ is bounded by $|U|!$. Since the same reasoning can be applied on W, we have that $|\mathscr{C}(G)| \leq |W|!$ and hence:

$$|\mathscr{C}(G)| \leq \min(|U|, |W|)! \leq \frac{n}{2}!$$

This bound is tight as shown by the following family of graphs that reaches it.

Consider the *antimatching graph with n nodes* $A_n = (U \cup W, E)$ defined as the complement of an $n/2$ edge perfect matching, *i.e.*:

$$U := \{u_1, \ldots, u_{n/2}\}, \quad W := \{w_1, \ldots, w_{n/2}\},$$
$$E := \{(u_i, w_j) \in U \times W : i \neq j\}.$$

It is not difficult to convince oneself that the maximal chain subgraphs of A_n are exactly $(n/2)!$ and that a different permutation corresponds to each of them. In particular, for each permutation π of the nodes of U, the corresponding maximal chain subgraph C_π of A_n can be defined by means of the set of neighbourhoods as follows:

$$N_{C_\pi}(u_i) := \{w_k \text{ s.t. } \pi^{-1}(k) < \pi^{-1}(i)\}.$$

The so-defined graph C_π is a chain subgraph since all the neighbourhoods form a chain of inclusions. Moreover, it is maximal since if we added to the neighbourhood of u_i any one of the missing edges (u_i, w_j) with $\pi^{-1}(j) \geq \pi^{-1}(i)$, we would introduce a $2K_2$ with the existing edge (u_j, w_i) as (u_j, w_j) and (u_i, w_i) are not in E.

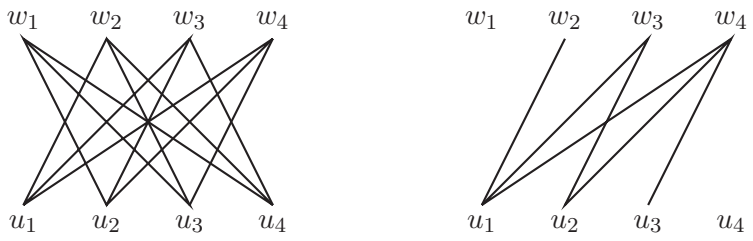

Figure 1.5: a. The graph A_8; b. A maximal chain of A_8 corresponding to the permutation $\pi = (14)(23)$

Bound in terms of edges

Let $T(m)$ be the maximum number of maximal chain subgraphs over all bipartite graphs with m edges. After two preliminary lemmas, we prove that $T(m) \leq 2^{\sqrt{m}\log(m)}$.

Lemma 3. *Let $G = (U \cup W, E)$ be a bipartite graph. Then $|\mathscr{C}(G)| \leq |U| \cdot T(m - |W|)$.*

Proof. In view of how Algorithm 3 works and of Proposition 3, at the beginning, there at most $|U|$ candidates. For each candidate x, we can build as many chain subgraphs as there are in $G[U \setminus \{x\}, N_G(x)]$. We claim that this latter graph has at most $m - |W|$ edges. Indeed, in order to construct $G[U \setminus \{x\}, N_G(x)]$, we remove from G exactly $|E_G(x)|$ edges when deleting x from U, and $|W| - |N_G(x)|$ nodes (each one connected to at least a different edge as G is connected) when reducing W to $N_G(x)$. Observing that $|E_G(x)| = |N_G(x)|$, in total we remove at least $|W|$ edges. It is not difficult to see that $T(m)$ is increasing with m. Hence, the proof follows from the fact that the number of chain subgraphs of $G[U \setminus \{x\}, N_G(x)]$ is bounded by $T(m - |W|)$. □

By the next Lemma, we have that the maximum on n of the auxiliary function $\frac{n}{2} \cdot 2^{\sqrt{m-\frac{n}{2}}\log(m-\frac{n}{2})}$ is reached when $n/2$ is minimum (note that trivially for a bipartite graph we have $n/2 > \sqrt{m}$).

Lemma 4. *The real-valued function $F(x) := x \cdot 2^{\sqrt{m-x}\log(m-x)}$ is decreasing in the interval $[\sqrt{m}, m-1]$.*

Proof. The derivative of $F(x)$ is given by:

$$
\begin{aligned}
&- x\left(\frac{\log(m-x)}{2\sqrt{m-x}} + \frac{1}{\sqrt{m-x}}\right)2^{\left(\sqrt{m-x}\log(m-x)\right)} + 2^{\left(\sqrt{m-x}\log(m-x)\right)} \\
&= -\frac{\left(x\log(m-x) + 2x - 2\sqrt{m-x}\right)2^{\left(\sqrt{m-x}\log(m-x)\right)}}{2\sqrt{m-x}}
\end{aligned}
$$

Then the derivative is negative whenever $\left(x\log(m-x) + 2x - 2\sqrt{m-x}\right) \geq 0$.

Observe that $\log(m-x) \geq 0$ for $x \leq m-1$, while for $x \geq 0$ we have:

$$2x - 2\sqrt{m-x} \geq 0 \iff x \geq \frac{-1+\sqrt{1+4m}}{2} = -\frac{1}{2} + \sqrt{m+\frac{1}{4}}$$

and:

$$\sqrt{m} \geq -\frac{1}{2} + \sqrt{m+\frac{1}{4}}$$

□

We are now able to prove the main theorem of this subsection:

Theorem 3. *Let $G = (U \cup W, E)$ be a bipartite graph with n nodes and m edges; then $|\mathscr{C}(G)| \leq 2^{\sqrt{m}\log m}$, i.e. $T(m) \leq 2^{\sqrt{m}\log m}$.*

Proof. Assume w.l.o.g that $|U| \leq |W|$. The proof is by induction on m. Note that for $m = 1$ the theorem trivially holds.

Applying the inductive hypothesis and Lemma 3, we have:

$$|\mathscr{C}(G)| \leq |U|T(m-|W|) \leq \frac{n}{2}2^{\left(\sqrt{m-\frac{1}{2}n}\log\left(m-\frac{1}{2}n\right)\right)}.$$

Since the function $F(x) := x2^{\sqrt{m-x}\log(m-x)}$ is decreasing in the interval $[\sqrt{m}, m-1]$, the maximum of $\frac{n}{2}2^{\sqrt{m-\frac{n}{2}}\log(m-\frac{n}{2})}$ is reached when $n/2$ is minimum. Note that trivially for a bipartite graph we have $n/2 > \sqrt{m}$. Hence,

$$|\mathscr{C}(G)| \leq \sqrt{m}\,2^{\sqrt{m-\sqrt{m}}\log(m-\sqrt{m})}.$$

Let $A := \sqrt{m} - \sqrt{m-\sqrt{m}}$ and $B := \frac{m-\sqrt{m}}{m}$. Then:

$$\begin{aligned} |\mathscr{C}(G)| &\leq \sqrt{m}\,2^{(\sqrt{m}-A)\log(mB)} \\ &= 2^{\sqrt{m}\log m} \times \sqrt{m}\,2^{\log B(\sqrt{m}-A) - A\log m}. \end{aligned}$$

Let us show that $Z := \sqrt{m}\,2^{(\sqrt{m}-A)\log B - A\log m} \leq 1$ by showing that $\log Z \leq 0$:

$$\begin{aligned} \log Z &= \log(\sqrt{m}) + (\sqrt{m}-A)\log B \;-\; A\log m \\ &= (1-2A)\log(\sqrt{m}) + (\sqrt{m}-A)\log B \\ &\leq 0 \end{aligned}$$

considering that $B < 1$ and $1/2 < A \leq 1$ since:

$$A = \frac{1}{1+\sqrt{B}} = \frac{1}{1+\sqrt{1-\frac{1}{\sqrt{m}}}}$$

□

By this bound on the number of maximal chain subgraphs we trivially obtain an input-sensitive bound on the time complexity for Algorithm 3:

Corollary 1. *The (input-sensitive) complexity of Algorithm 3 is bounded by* $O^*(2^{\sqrt{m}log(m)})$.

1.5.2 Minimum Chain Subgraph Cover

In this section, we address the *minimum chain subgraph cover* problem. This asks to determine, for a given graph G, the minimum number of chain subgraphs that cover all the edges of G. This has already been investigated in the literature as it is related to other well-known problems such as maximum induced matching (see *e.g.* [12, 19]). For bipartite graphs, the problem was shown to be NP-hard [91].

Calling m the number of edges in the graph, we provide an exact exponential algorithm which runs in time $O^*((2+\varepsilon)^m)$, for every $\varepsilon > 0$ (by O^* we denote standard big O notation but omitting polynomial factors). To obtain this result, we exploit Algorithm 3, the bound obtained in Theorem 3 and the inclusion/exclusion method [7, 38] that has already been successfully applied to exact exponential algorithms for many partitioning and covering problems. Notice that, since a chain subgraph cover is a family of subsets of edges, the existence of an algorithm whose complexity is close to 2^m is not obvious. Indeed, the basic search space would have size 2^{2^m}, which corresponds to all families of subsets of edges of a graph on m edges.

Notice also that the minimum chain subgraph problem has already been investigated in the literature as it is related to other well-known problems such as maximum induced matching (see *e.g.* [12, 19]). For bipartite graphs, the problem was shown to be NP-hard [91].

The first step consists in expressing the problem as an inclusion-exclusion formula over the subsets of edges of G.

Proposition 6. *[7] Let $c_k(G)$ be the number of chain subgraph covers of size k of a graph G. We have that:*

$$c_k(G) = \sum_{A \subseteq E} (-1)^{|A|} a(A)^k$$

where $a(A)$ denotes the number of maximal chain subgraphs not intersecting A.

Exploiting this result, we can design an exact algorithm which counts the number of chain subgraph covers of size k with a time complexity given in the following theorem:

Theorem 4. *Given a bipartite graph G with m edges, for all $k \in \mathbb{N}^*$ and for all $\varepsilon > 0$, the number of chain subgraph covers of size k, denoted with $c_k(G)$, can be computed in time $O^*((2+\epsilon)^m)$.*

Proof. Let $G = (U \cup W, E)$ be a bipartite graph, $k \in \mathbb{N}^*$ and fix an $\varepsilon > 0$. Using the formula of Proposition 6, c_k can be computed in time $\sum_{i=0}^{m} \binom{m}{i} C(i)$, where $C(i)$ is the time complexity needed to compute $a(A)$, $|A| = i$.

Notice that to compute $a(A)$ for a given $A \subseteq E$, one can naively compute all maximal chain subgraphs of $G' = (U \cup W, E \setminus A)$ and, for each of them, check whether it is

maximal in G. Using this fact and Corollary 1, $C(i)$ can be determined in time $O(n^2m \cdot 2^{\sqrt{m-i}\log(m-i)})$.

Thus we have that $c_k(G)$ can be computed in time $\sum_{i=0}^{m} \binom{m}{i} n^2 m \cdot 2^{\sqrt{m-i}\log(m-i)}$. Observe now that since $2^{\sqrt{m-i}\log(m-i)} = o((1+\varepsilon)^m)$, there exists a constant n_ε such that for all $m > n_\varepsilon$, $2^{\sqrt{m-i}\log(m-i)} < (1+\varepsilon)^m$.

We have that:

$$\begin{aligned}
\sum_{i=0}^{m} \binom{m}{i} n^2 m \cdot 2^{\sqrt{m-i}\log(m-i)} &= n^2 m \cdot \left(\sum_{i=0}^{m-n_\varepsilon-1} \binom{m}{i} 2^{\sqrt{m-i}\log(m-i)} + \sum_{i=m-n_\varepsilon}^{m} \binom{m}{i} 2^{\sqrt{m-i}\log(m-i)} \right) \\
&\leq n^2 m \cdot \left(\sum_{i=0}^{m-n_\varepsilon-1} \binom{m}{i} (1+\varepsilon)^{m-i} + n_\varepsilon m^{n_\varepsilon} 2^{\sqrt{n_\varepsilon}\log(n_\varepsilon)} \right) \\
&\leq n^2 m \cdot \left(\sum_{i=0}^{m} \binom{m}{i} (1+\varepsilon)^{m-i} + n_\varepsilon m^{n_\varepsilon} 2^{\sqrt{n_\varepsilon}\log(n_\varepsilon)} \right) \\
&\leq n^2 m \cdot (2+\varepsilon)^m + n^2 n_\varepsilon m^{1+n_\varepsilon} 2^{\sqrt{n_\varepsilon}\log(n_\varepsilon)} \\
&= O^*((2+\varepsilon)^m)
\end{aligned}$$

The last step follows by recalling that G is connected and thus $n = O(m)$. □

We conclude by observing that the size of a minimum chain cover is given by the smallest value of k for which $c_k(G) \neq 0$.

1.5.3 Enumeration of Minimal Chain Subgraph Covers

In this section, we prove that the problem of enumerating all minimal chain subgraph covers can be polynomially reduced to the problem of enumerating all the minimal set covers of a hypergraph. This reduction implies that there is a quasi-polynomial time algorithm to enumerate all minimal chain subgraph covers. Indeed, the result in [39] implies that all the minimal set covers of a hypergraph can be enumerated in time $N^{\log N}$ where N is the sum of the input size (*i.e.* $n+m$) and of the output size (*i.e.* the number of minimal set covers).

Let $G = (U \cup W, E)$ be a bipartite graph, $\mathcal{C} = \mathcal{C}(G)$ be the set of all maximal chain subgraphs of G and $\mathcal{S} = \mathcal{S}(G)$ be the set of minimal chain subgraph covers of G. Notice that the minimal chain subgraph covers of G are the minimal set covers of the hypergraph $\mathcal{H} := (V, \mathcal{E})$ where $V = E$ and $\mathcal{E} = \mathscr{C}$. Unfortunately, the size of $\mathcal{H}$ might be exponential in the size of G plus the size of $\mathcal{S}$. Indeed not every maximal chain subgraph in $\mathcal{C}$ will necessarily be part of some minimal chain subgraph cover. To obtain a quasi-polynomial time algorithm to enumerate all minimal chain subgraph covers, we need to enumerate only those maximal chain subgraphs that belong to a minimal chain subgraph cover.

Given an edge $e \in E$, let $\mathcal{C}_e$ be the set of all maximal chain subgraphs of G containing e.

We call an edge $e \in E$ *non-essential* if there exists another edge $e' \in E$ such that $\mathscr{C}_{e'} \subset \mathscr{C}_e$. An edge which is not non-essential is said to be *essential*. Note that for every non-essential edge e, there exists an essential edge e_1 such that $\mathscr{C}_{e_1} \subset \mathscr{C}_e$. Indeed, by applying iteratively the definition of a non-essential edge, we obtain a list of inclusions $\mathscr{C}_e \supset \mathscr{C}_{e_1} \supset \mathscr{C}_{e_2} \ldots$, where no $\mathscr{C}_{e_i}$ is repeated as the inclusions are strict. The last element of the list will correspond to an essential edge.

The following lemma claims that if a maximal chain subgraph C contains at least one essential edge, then it belongs to at least one minimal chain subgraph cover.

Lemma 5. *Let C be a maximal chain subgraph of a bipartite graph $G = (U \cup W, E)$. Then C belongs to a minimal chain subgraph cover of G if and only if C contains an essential edge.*

Proof. ($\Rightarrow$) Let C belong to a minimal chain subgraph cover M and assume that C contains no essential edges. Given $e \in C$, e therefore being non-essential, there exists an essential edge e' such that $\mathscr{C}_{e'} \subset \mathscr{C}_e$. Moreover, $e' \notin C$. As M is a cover, there exists $C' \in M$ such that $e' \in C'$. Thus, $C' \neq C$, $C' \in \mathscr{C}_{e'} \subset \mathscr{C}_e$, hence $e \in C'$. Since for every edge $e \in C$, there exists $C' \in M$ containing it, we have that $M \setminus \{C\}$ is a cover, contradicting the minimality of M.

($\Leftarrow$) Assume C contains an essential edge e. Let $\mathscr{C}' = \{D \in \mathscr{C}(G) : e \notin D\}$. Note that $\mathscr{C}' = \mathscr{C} \setminus \mathscr{C}_e$. We show that $\mathscr{C}' \cup \{C\}$ is a cover. Suppose on the contrary that there exists $e' \in E \setminus E(C)$ and e' is not covered by $\mathscr{C}'$ and thus $\mathscr{C}_{e'} \cap \mathscr{C}' = \emptyset$. This implies that $\mathscr{C}_{e'} \subseteq \mathscr{C} \setminus \mathscr{C}' = \mathscr{C}_e$ and as e is essential, we obtain $\mathscr{C}_{e'} = \mathscr{C}_e$ from which we deduce that $e' \in C$. Thus, $M = \mathscr{C}' \cup \{C\}$ is a cover and clearly it contains a minimal one. Finally, we conclude by observing that, since by construction C is the only chain subgraph of M that contains e, it belongs to any minimal cover contained in M. □

It follows that the set of maximal chain subgraphs that can contribute to a minimal chain cover is $\tilde{\mathcal{C}} = \cup \mathcal{C}_e$ where the index e runs over all the essential edges of G. In the following, we show how to detect essential edges. This problem then consists in detecting all the couples e_1, e_2 such that $\mathcal{C}_{e_1} \subseteq \mathcal{C}_{e_2}$ before enumerating all useful maximal chain subgraphs.

Theorem 5 later in this section provides an efficient way to detect these couples. In order to prove it we need first some preliminary results.

Let $\mathcal{M}_e$ the set of all edges $e' \in E$ inducing a $2K_2$ in G together with e.

Fact 1. *Let $C = (X \cup Y, F)$ be a maximal chain subgraph of a bipartite graph $G = (U \cup W, E)$, and let $z \in X$, $e = \{u, w\} \in E$ be such that for every $e' \in E_C(z)$, we have $e \notin \mathcal{M}_{e'}$. Then at least one of the following holds:*

a) $w \in N_G(z)$.

b) $u \in \cap_{y \in N_C(z)} N_G(y)$.

Proof. The proof follows directly by observing that for any $e' = \{z, y\} \in C$ then as $e \notin \mathcal{M}_{e'}$, either $\{z, w\} \in E(G)$ or $\{u, y\} \in E(G)$. □

Observe that in the previous claim, we can re-write (2) in the form:

b') $N_C(z) \subseteq N_G(u)$.

Lemma 6. *Let C be a maximal chain subgraph of a bipartite graph $G = (U \cup W, E)$ and let $e \in E$ be such that for all $e' \in E(C)$, it holds that $e \notin \mathcal{M}_{e'}$. Then $e \in C$.*

Proof. Let $C = (X \cup Y, F)$ be a maximal chain subgraph of $G = (U \cup W, E)$ and w.l.o.g., let $u_1, \ldots, u_{|X|} \in X \subseteq U$ such that $N_C(u_1) \subseteq N_C(u_2) \subseteq \ldots \subseteq N_C(u_{|X|})$. We can furthermore assume that C is connected.

From the hypothesis, let $e = \{u, w\}$ in E be such that for all $e' \in E(C)$, it holds $e \notin \mathcal{M}_{e'}$. Finally, assume $e \notin C$.

The proof runs by contradiction: we will show that

$$w \in \cap_{x \in X} N_G(x) \tag{1.1}$$

must hold. Although, this contraddicts the maximality of C as in this way we could add e and all the other edges in $E_G(w)$ to C and still obtain a chain subgraph (with $N_G(w)$ as the largest neighbourhood of C).

In order to prove (1.1), using Fact 1 with $z = u_{|X|}$, we have that at least one among a) and b) must hold. Observe that b) cannot hold as otherwise we have straight away (1.1) (interchanging the roles of Y and X, and w and u) observing that $N_C(u_{|X|}) = N_G(u_{|X|}) = Y$ by point (i) of Proposition 3. Thus, a) must hold, *i.e.* $w \in N_G(u_{|X|})$.

If we now show that $w \in \cap_{k=j}^{|X|} N_G(u_k) \Rightarrow w \in N_G(u_{j-1})$, we prove the claim since together with the just proved $w \in N_G(u_{|X|})$ this leads to (1.1):

$$w \in \bigcap_{k=1}^{|X|} N_G(u_k) = \bigcap_{x \in X} N_G(x)$$

We conclude the proof by showing the validity of $w \in \cap_{k=j}^{|X|} N_G(u_k) \Rightarrow w \in N_G(u_{j-1})$.

Assume then that $w \in \cap_{k=j}^{|X|} N_G(u_k)$ and we deduce $w \in N_G(u_{j-1})$ applying again Fact 1 with $z = u_{j-1}$ and showing that b'), hence b), cannot hold.

Indeed, supposing by contradiction that b') holds, it yields $N_C(u_{j-1}) \subseteq N_G(u)$. By this assumption and using the maximality of C, we deduce that $u \in X$ with the following arguments: $N_C(u)$ has to contain at least $N_C(u_{j-1})$, and hence there exists $\tilde{k} \geq j - 1$ for which $u = u_{\tilde{k}}$ otherwise we could add the related edges.

Although $u \in X$ implies that we could contradictorily extend C to C' by adding at least e, were C' has the following list of neighbourhoods:

$$N_{C'}(u_k) := N_C(u_k) \qquad \text{for } k \neq \tilde{k}$$
$$N_{C'}(u_k) := N_C(u_k) \cup \{w\} \qquad \text{for } k = \tilde{k}$$

and C' is a chain graph since $N_C(u_{\tilde{k}}) \cup \{w\} \subseteq N_C(u_k)$ for all $k > \tilde{k} \geq j-1$ by the fact that $w \in \cap_{k=j}^{|X|} N_G(u_k)$ and by the maximality of C. □

Using Lemma 6 we can now prove the following result.

Theorem 5. *Given a bipartite graph $G = (U \cup W, E)$, for any two edges $e, e' \in E$, $\mathcal{C}_e \subseteq \mathcal{C}_{e'}$ if and only if $\mathcal{M}_e \supseteq \mathcal{M}_{e'}$.*

Proof. ($\Rightarrow$) Given two edges $e, e' \in E$, suppose that $\mathcal{C}_e \subseteq \mathcal{C}_{e'}$, and assume on the contrary that there exists $f \in \mathcal{M}_{e'}$ and $f \notin \mathcal{M}_e$. Then there exists a maximal chain subgraph C' containing e and f (as they do not form a $2K_2$ in G) but not e' ($f \in \mathcal{M}_{e'}$). Hence, $C' \in \mathcal{C}_e$ but $C' \notin \mathcal{C}_{e'}$, contradicting the assumption that $\mathcal{C}_e \subseteq \mathcal{C}_{e'}$.

($\Leftarrow$) Suppose now $\mathcal{M}_e \supseteq \mathcal{M}_{e'}$. Let $C \in \mathcal{C}_e$. By definition, none of the edges of $\mathcal{M}_e$ appears in C. Hence, e' does not form a $2K_2$ with any edge in C in the graph G (as $\mathcal{M}_e \supseteq \mathcal{M}_{e'}$). By Lemma 6 $e' \in C$. Thus, $\mathcal{C}_e \subseteq \mathcal{C}_{e'}$. □

Notice that, given an edge $e = (u, w) \in E$, $u \in U$ and $w \in W$, it is easy to determine the set $\mathcal{M}_e$. We just need to start from E and delete all edges that are incident either to u or to w, as well as all edges at distance 2 from e (that is all edges $e' = (u', w')$ such that either u' is adjacent to w or w' is adjacent to u). Checking whether $\mathcal{M}_e \supseteq \mathcal{M}_{e'}$ is also easy: it suffices to sort the edges in each set in lexicographic order, and then the inclusion of each pair can be checked in linear time in their size, that is in $O(m)$. It is thus possible to enumerate in polynomial delay only those maximal chain subgraphs that contain at least one essential edge by slightly modifying Algorithm 3 as shown in the pseudo-code

here after.

Algorithm 4: Enumerate All Maximal Chain Subgraphs with a not empty intersection with a given set of edges E'

Input: A bipartite graph $G = (U \cup W, E)$, a set of edges $E' := \{u_1w_1, ..., u_kw_k\}$

Output: All maximal chain subgraphs of G that intersect E'

```
1  C ⟵ ∅
2  enumerateMaximalChain(U, W, C)
3      Candidates ⟵ computeCandidates(U, W)
4      if Candidates == ∅ then
5          print(C);
6          return;
7      if {u_1, ..., u_k} ⊆ U then
8          Candidates ⟵ Candidates \ {u ∈ U : N_G(u) ∩ {w_1, ..., w_k} = ∅}
9      for u ∈ Candidates do
10         U' ⟵ U \ {u}; W' ⟵ W ∩ N_G(u);          /* reduced graph */
11         F(u) ⟵ { edges of E_G(u) incident to some node in W'};
12         C' ⟵ C ∪ F(u);
13         enumerateMaximalChain(U', W', C');
```

Theorem 6. *Given a bipartite graph $G = (U \cup W, E)$, one can enumerate all its minimal chain subgraph covers, i.e. all the elements in $\mathcal{S}$, in time $O([(m+1)|\mathcal{S}|]^{\log((m+1)|\mathcal{S}|)})$.*

Proof. We first construct the hypergraph $\mathcal{H} = (V, \mathcal{E})$ where $V := E'$ is the set of essential edges of G and $\mathcal{E} := \mathscr{C}_{ess}$ is the set of maximal chain subgraphs of G that contain at least one essential edge. This takes time $O(n^2m|\mathscr{C}_{ess}|)$. Applying then the algorithm given in [39], one can enumerate all minimal set covers of $\mathcal{H}$ (*i.e.* all minimal chain subgraph covers) in time $O((|\mathcal{H}| + |\mathcal{S}|)^{\log(|\mathcal{H}|+|\mathcal{S}|)}) = O((|\mathscr{C}_{ess}| + |\mathcal{S}|)^{\log(|\mathscr{C}_{ess}|+|\mathcal{S}|)})$. The total running time is thus $O(n^2m|\mathscr{C}_{ess}| + (|\mathscr{C}_{ess}| + |\mathcal{S}|)^{\log(|\mathscr{C}_{ess}|+|\mathcal{S}|)})$. Notice now that since by Lemma 5, every maximal chain subgraph in $\mathscr{C}_{ess}$ belongs to at least one minimal chain subgraph cover, we have that $|\mathscr{C}_{ess}| \leq m|\mathcal{S}|$. Finally, we obtain that the total running time is $O(n^2m^2|\mathcal{S}| + (m|\mathcal{S}| + |\mathcal{S}|)^{\log(m|\mathcal{S}|+|\mathcal{S}|)}) = O([(m+1)|\mathcal{S}|]^{\log((m+1)|\mathcal{S}|)})$ □

1.6 Chain Graphs and Interval Orders

There is an interesting connection between chain graphs and interval orders. In this section, we look at the previously presented results under the light of this relation, and show how we can computation the *interval order dimension* of a poset and how we can enumerate *minimal interval order extensions* and *maximal interval order reductions* of a bipartite poset. First, we will briefly recall these notions and this relation.

1.6.1 Preliminaries on Poset Theory

A *partially ordered set* (or in short *poset*) is a pair $(P, \leq_P)$ where P is a set, called the *ground set*, and $\leq_P \subseteq P \times P$ is a binary, reflexive, transitive and anti-symmetric relation on P and referred as *partial order on* P. A partial order is an *interval order on* P if there exist two functions $l, r : P \to \mathbb{R}$ such that $x \leq_P y$ iff $r(x) \leq l(y)$, while P is said a *total* or *linear* order iff either $x \leq_P y$ or $y \leq_P x$ for all $x, y \in P$. A partial order $Q = (Q, \leq_Q)$ is said to *extend* P or *to be an extension of* P if $x \leq_P y$ implies $x \leq_Q y$. A linear extension of P is an extension of P which is also a linear order. A *bipartite poset* is a poset $H = (U \cup V, \leq_H)$ such that $\leq_H \subseteq U \times V$.

The *interval order dimension* of a bipartite poset H, denoted by $Idim(H)$, is the minimum number k of interval order extensions whose intersection gives H.

We can view H as a bipartite undirected graph, said the *comparability graph* $G(H)$ *of* H, with nodes $U \cup V$ and edges given by $\{(u, v) \in E(G(H)) : u \in U, v \in V \text{ and } u \cong_H v\}$. A bipartite poset is an interval order if and only if its comparability graph is a chain graph [91]. Hence each interval extension of H can be viewed as a chain graph (edge) completion of $G(H)$, *i.e.* a chain graph with the same node set of $G(H)$ which has $G(H)$ as subgraph. Thus $Idim(H)$ coincides with the minimum number of chain graph completions of $G(H)$ whose intersection gives $G(H)$.

The *bipartite complement* of a bipartite graph $D = (U \cup V, E))$ is the graph $B(D) := (U \cup V, E')$ where E' are all the non-edges of D across the two partitions.

1.6.2 Computation of the Interval Order Dimension

If C is a chain subgraph of $G(H)$, also its bipartite complement is a chain graph (as the bipartite complement of a $2K_2$ is a $2K_2$), so we have that $Idim(H)$ coincides with the size of a minimum chain subgraph cover of the bipartite complement of $G(H)$.

All this is contained in the following result of [91] where by abuse of notation the bipartite complement of $G(H)$ is denoted by $B(H)$ (instead of $B(G(H))$) and the size of a minimum chain subgraph cover of $B(H)$ is denoted by $ch(B(H))$:

Proposition 7 ([91])**.** *Let H be a bipartite poset. Then $Idim(H) = ch(B(H))$.*

From this we have a straightforward interpretation of our results on the computation of the size of a minimum chain subgraph cover (see Theorem 4):

Corollary 2. *Given a bipartite poset H where $B(H)$ has m edges, for all $k \in \mathbb{N}^*$ and for all $\varepsilon > 0$, we can compute $Idim(H)$ in time $O^*((2+\epsilon)^m)$.*

1.6.3 Enumeration of Minimal Extensions and Maximal Reductions of Interval Order

In the same way, enumerating all the maximal chain subgraphs of a bipartite graph $G(H)$ (*i.e.* our first problem) is equivalent to list all the maximal interval order reductions of the bipartite order H and enumerating all maximal chain subgraphs of $B(H)$ is equivalent to list all minimal interval order extensions of the bipartite order H as minimal interval order extensions of bipartite posets are still bipartite posets (substantially a proof of this is present in the proof of Lemma 4 in [91]).

We can then interpret the results on the enumeration of maximal chain subgraphs (Proposition 4 and Proposition 5) in this context as follows:

Corollary 3. *Let $H = (U \cup V, \leq_H)$ be a bipartite poset with $n = |U| + |V|$ and $|R_{\leq_H}| - n = m$. Then:*

1. *We can enumerate its maximal interval order reductions in polynomial time delay with a delay of $O(\min\{|U|, |V|\}^2 m)$.*
2. *We can enumerate its minimal interval order extensions in polynomial time delay with a delay of $O(\min\{|U|, |V|\}^3 \cdot \max\{|U|, |V|\})$.*

Proof. The first result is a straightforward interpretation of Proposition 4 and Proposition 5 in this context, while the second result comes from having to run our algorithm on $B(H)$ instead of $G(H)$ hence the number of edges pass from m to $|U| \cdot |V| - m$ hence $O(n^2(|U| \cdot |V| - m)\}) = O(n^2 \cdot |U| \cdot |V|)$. Finally in both the results we employ the observation that we can run our algorithm on the smaller of the two partitions substituting n^2 with $\min\{|U|, |V|\}^2$. □

Observe that it is not surprising that enumerating minimal extensions is more complicated than enumerating maximal reductions as it happens in [34] for counting minimal extensions and maximal reductions of N-free orders (*i.e.* the posets $(P, \leq_P)$ such that there are no $x, y, z, w \in P$ with $x \leq_P y$, $x \leq_P w$ and $z \leq_P w$).

Furthermore, recall that for general posets P, in [34] it has already been proven that the number of minimal interval order extensions and maximal reductions can be computed polynomially in their number, but the proposed dynamic programming algorithm implies the avoidance of repeated solutions by storing all the past solutions leading to an incremental polynomial algorithm.

1.7 Computing the Poset Dimension

In this section we deal with the problem of computing the poset dimension of a poset $\mathcal{P} = (X, \leq_{\mathcal{P}})$ adapting to this case the methods of our work on the interval poset dimension and the relation with chain subgraph covers of bipartite graph (see Section 1.6).

The *poset dimension*, denoted by $dim(\mathcal{P})$, is the minimum number of linear extensions of $\mathcal{P}$ such that their intersection gives $\mathcal{P}$. A family of linear extensions of $\mathcal{P}$ whose intersection gives $\mathcal{P}$ is said a *realizer of* $\mathcal{P}$. In the following, we write $LinExt(\mathcal{P})$ for the set of linear extensions of $\mathcal{P}$. If for $x, y \in X$ we have $x \not\leq_{\mathcal{P}} y$ and $y \not\leq_{\mathcal{P}} x$ then (x, y) (hence (y, x)) is said an *incomparable pair* and we write $x \parallel y$. The set of all incomparable pairs of $\mathcal{P}$ is denoted with the symbol $Inc(\mathcal{P})$. A linear extension $L \in LinExt(\mathcal{P})$ *reverses* an incomparable pair (x, y) if $x >_L y$ and L reverses a subset of incomparable pairs in $W \subseteq Inc(\mathcal{P})$ if it reverses all the incoparable pairs in W.

The Naive Method

We want to compute the poset dimension of $\mathcal{P}$ with the aim of the inclusion/exclusion formula with the same polinomial space approach used for the interval order dimension whose bottleneck, on the other hand, is the time for listing all linear order extensions of $\mathcal{P}$ (see Corollary 2 an Section 1.6 for the interval poset dimension case where we list all the interval order extensions). Observe that:

Proposition 8. *[35] Let $\mathcal{R}$ be a family of linear extensions of a poset $\mathcal{P} = (X, \leq_{\mathcal{P}})$. The following statements are equivalent:*

1. *$\mathcal{R}$ is a realizer of $\mathcal{P}$,*
2. *for every $(x, y) \in \mathrm{Inc}(\mathcal{P})$ there exists a linear extension $L \in \mathcal{R}$ which reverse (x, y),*

We can then count the realizers of $\mathcal{P}$ of size k, denoted with $r_k(\mathcal{P})$ with the following formula:

$$r_k(\mathcal{P}) = \sum_{W \subseteq Inc(\mathcal{P})} (-1)^{|W|} \tilde{N}(W)$$

where $\tilde{N}(W)$ is the number of k-tuple linear extensions which do not reverse any incomparable pairs in W. Moreover:

$$\tilde{N}(W) = |\tilde{s}(W)|^k$$

where $\tilde{s}(W)$ are the linear extensions of $\mathcal{P}$ which do not reverse any incomparable pair in W.

Obseve now that if a linear extension L does not reverse a pair (x, y) then it holds $x \leq_L y$, it means:

$$\tilde{s}(W) := \{L \in LinExt(\mathcal{P}) : x \leq_L y \text{ for all } (x, y) \in W\} = LinExt(poset(\mathcal{P} \cup W))$$

where $poset(P \cup W)$ denotes the poset, if it exists, $\mathcal{Q} = (X, \leq_{\mathcal{Q}})$ such that $\leq_{\mathcal{P}} \cup W \subseteq \leq_{\mathcal{Q}}$ and which minimize $| \leq_{\mathcal{Q}} |$ (the existence and unicity of $\mathcal{Q}$ is granted by taking the intersection of all the posets which extend the relation $\leq_{\mathcal{P}} \cup W$, if there are any). For a simple condition for its existence see Theorem 8. If it does not exists, we set $LinExt(poset(\mathcal{P} \cup W)) = \emptyset$.

To count the elements of $\tilde{s}(W)$, as done for the interval order dimension, we can list all linear extensions of $poset(\mathcal{P} \cup W)$. Let us focus then on computing the number of linear extensions of a general poset $\mathcal{P} = (X, \leq_{\mathcal{P}})$.

The problem of counting linear extensions of a poset is $\#P$-complete [15]. We compute $|s(W)|$ directly enumerating all linear extension of $\mathcal{P}$; this enumeration can be done with constant delay [67]. As the number of linear extension is trivially never greater than $2^{|Inc(\mathcal{Q})|/2}$ [33], then this enumeration can be done in time $\sqrt{2}^p$ with p the number of incomparable pairs of $\mathcal{P}$. The total running time $T(p)$ to compute $r_k(\mathcal{P})$ is then:

$$T(p) = \sum_{j=0}^{p} \binom{n}{j} \left(\sqrt{2}\right)^{p-j} = (1+\sqrt{2})^p \sim 2.41^p$$

In the next section we present a faster, but still employing polinomial space method.

The Split Method

Let us recall here that by the *split* operation defined by Trotter in [83, 84] we have that:

Theorem 7. *Let $\mathcal{P} = (P, \leq_{\mathcal{P}})$ be a poset. Then $dim(\mathcal{P}) = Idim(split(\mathcal{P}))$.*

where $split(\mathcal{P})$ is the bipartite poset $(P \times \{0,1\}, \succeq)$ with $(x,a) \preceq (y,b)$ iff $a = 0$,$b = 1$ and $x \leq_{\mathcal{P}} y$.

Then, with our approach to compute interval dimension of bipartite posets, we can actually compute the poset dimension of any poset. However observe that if we start with a poset $\mathcal{P}$ whose comparability graph has m edges and n nodes, we must apply our procedure to the bipartite complement of $split(\mathcal{P})$ which has $n^2 - m$ edges. In this way we obtain a procedure which runs in time $O^*((2+\epsilon)^{n^2-m}) = O^*((2+\epsilon)^{p/2})$ where p is the number of incomparable pairs of $\mathcal{P}$.

1.7.1 Reversing Critical Pairs

With the inclusion/exclusion approach and counting incomparable pairs we cannot hope to perform better than $O(\sqrt{2}^p)$, *i.e.* the time to compute the formula running through all the subset of incomparable pairs which do not contain both (x,y) and (y,x) for $x \not\cong_{\mathcal{P}} y$ (as in that case there are no linear extensions).

To this extent we present here the concept of *critical pairs* [35, 84] (see Figure 1.6 and definition here under) which, intuitively, are the strictly necessary incomparable pairs to be reversed by a realizer of $\mathcal{P}$ (see Proposition 9 here under).

Definition 12 ([35])**.** *Let $\mathcal{P} = (X, \leq_{\mathcal{P}})$ be a partial order and let $x, y \in X$. The incomparable pair (x,y) is said a* critical pair *if for each $u, w \in X$ it holds that:*

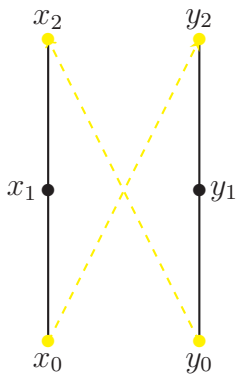

Figure 1.6: (x_0, y_2) and (y_0, x_2) are critical pairs.

1. *if* $u <_{\mathcal{P}} x$ *then* $u <_{\mathcal{P}} y$,

2. *if* $y <_{\mathcal{P}} w$ *then* $x <_{\mathcal{P}} w$.

The set of all critical pairs of $\mathcal{P}$ *is denoted by* $\mathcal{CP}(\mathcal{P})$ *or only* $\mathcal{CP}$ *when there is no ambiguity.*

Proposition 8 holds also if we restrict the incomparable pairs to only critical pairs:

Proposition 9. *[35] Let* $\mathcal{R}$ *be a family of linear extensions of a poset* $\mathcal{P} = (X, \leq_{\mathcal{P}})$. *The following statements are equivalent:*

1. $\mathcal{R}$ *is a realizer of* $\mathcal{P}$,

2. *for every* $(x, y) \in \mathcal{CP}(\mathcal{P})$ *there exists a linear extension* $L \in \mathcal{R}$ *which reverses* (x, y).

Writing $\mathcal{CP} = \{cp_1, cp_2, \ldots, cp_q\}$, observe now that the inclusion/exclusion formula give us:

$$r_k(\mathcal{P}) := \sum_{W \subseteq \{cp_1, \ldots, cp_q\}} (-1)^{|W|} N(W)$$

where $N(W)$ is the number of k-tuple linear extensions which do not reverse any critical pairs in W.

Furthermore we have that:

$$N(W) = |s(W)|^k$$

where $s(W)$ are the linear extensions of $\mathcal{P}$ which do not reverse any critical pair in W which might be empty.

The inclusion/exclusion approach will lead us to check if $r_k(\mathcal{P}) \neq 0$ to verify the existence of a realizer of $\mathcal{P}$ of size k, we show here after how to speed this process. Intuitively, the idea is passing from computing $|s(W)|$, by listing all linear extension of P which do not reverse critical pairs in W, to listing only equivalence classes of them

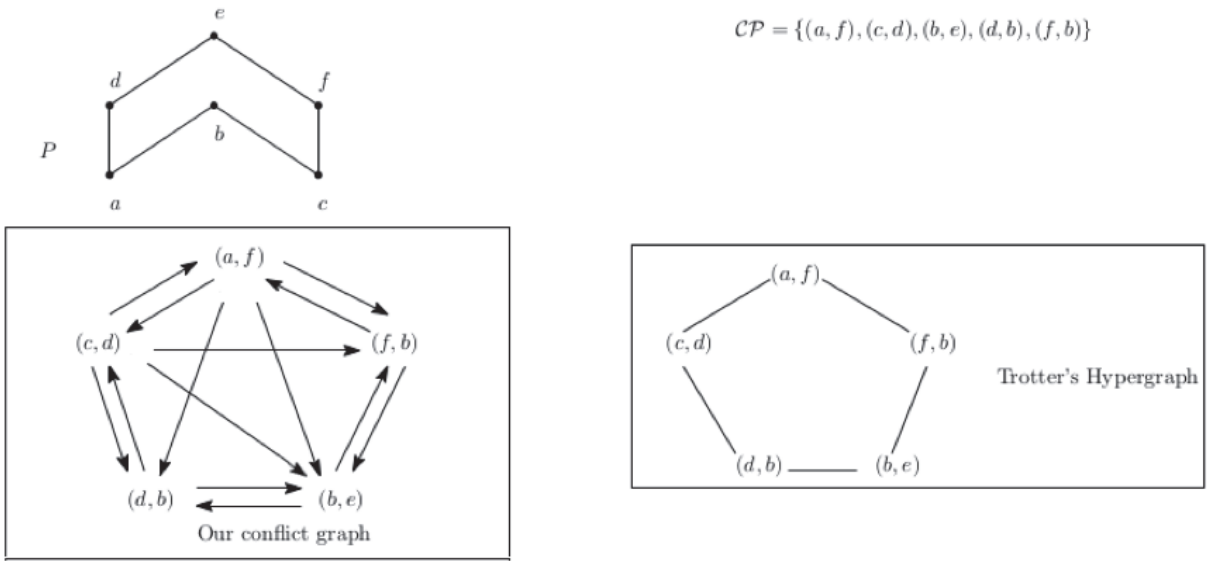

Figure 1.7: (top): A poset $\mathcal{P}$ and its set of critical pairs $\mathcal{CP}$. (bottom left): The set of its critical pairs and the graph of critical pairs of $\mathcal{P}$. (bottom right): Trotter's (hyper)graph .

(based on the exact set of critical pairs reversed) which do not reverse the critical pairs in W. In this way we have less object to list. We extend then the equivalence relation to realizers of $\mathcal{P}$. Due to the difficulties in identifying these equivalence classes, we solve the problem by introducing an auxiliary directed graph $\mathcal{GCP}(\mathcal{P})$ where we can "forget" about the linear extension and work only with the set W. Realizers of $\mathcal{P}$ correspond to acyclic covers of $\mathcal{GCP}(\mathcal{P})$. Denoting then with $\tilde{r}_k(\mathcal{P})$ the number of equivalence classes of realizers of $\mathcal{P}$ of size k, clearly we end up verifying $r_k(\mathcal{P}) \neq 0$ by verifying $\tilde{r}_k(\mathcal{P}) \neq 0$ which require to list the acyclic covers of $\mathcal{GCP}$.

We introduce now the graph $\mathcal{GCP}(\mathcal{P})$ and afterwards we justify its definition.

Definition 13 (Graph of Critical Pairs)**.** *We denote with $\mathcal{GCP}(\mathcal{P})$ or only $\mathcal{GCP}$ the graph* Graph of Critical Pairs *of $\mathcal{P}$, $\mathcal{GCP} = (\mathcal{CP}, E)$, where $((x,y),(u,w)) \in E$ iff $x \leq_P w$.*

The concept of alternating cycle, here under, together with the following propositions should shed light on the relation between linear extensions of $\mathcal{P}$ and directed acyclic subgraphs of $\mathcal{GCP}(\mathcal{P})$:

Definition 14 ([35])**.** *Let $\mathcal{P} = (X, P)$ be a poset. An alternating cycle in (X, P) is a sequence $\{(x_i, y_i) | 1 \leq i \leq k\}$ of incomparable pairs with $x_i \leq y_{i+1}$ (cyclically) for $i \in \{1, \ldots, k\}$. The integer k denotes the length of the cycle.*

An alternating cycle is strict if $x_i \leq y_j$ iff $j = i + 1$.

Theorem 8 ([35])**.** *Let $\mathcal{P} = (P, X)$ be a poset and let $S \subseteq \mathrm{Inc}(\mathcal{P})$. Then the following statements are equivalent:*

1. *There exists a linear extension L of $\mathcal{P}$ which reverse all the incomparable pairs in S.*

2. *S does not contain an alternating cycle.*

3. *S does not contain a strict alternating cycle.*

Lemma 7. *A simple cycle in $\mathcal{GCP}$ corresponds to an alternating cycle in $\mathcal{P}$.*
A chordless cycle in $\mathcal{GCP}$ corresponds to a strict alternating cycle in $\mathcal{P}$.

Proof. Let $C = ((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k), (x_1, y_1)$ be a simple cycle in $\mathcal{GCP}$, then we can define the sequence $C' = ((x_1, y_1), \ldots, (x_k, y_k))$ and one can check that C' is alternating cycle in $\mathcal{P}$.
C is a chordless cycle in $\mathcal{GCP}$ if and only no two nodes of the cycle are connected by an edge of $\mathcal{GCP}$ which does not belong to the cycle, hence if and only if $x_i \leq_{\mathcal{P}} y_j$ holds only for $j = i + 1$. □

It is clear that the structure of $\mathcal{GCP}$ can help us identifying linear extensions of $\mathcal{P}$. In particular, by Theorem 8 a linear extension which reverses all the critical pairs in W exists if and only if $\mathcal{GCP}[W]$, *i.e.* the subgraph of $\mathcal{GCP}$ induced on the nodes in W, is a directed acyclic subgraph of $\mathcal{GCP}$.

Since there may be many linear extensions which reverse the same set of critical pairs W, it is natural to introduce an equivalence classes on $LinExt(\mathcal{P})$:

Definition 15. *Let $L, L' \in LinExt(\mathcal{P})$, then $L \sim L'$ iff they reverse exactly the same subset of critical pairs. If this is the case, an equivalence class of linear extensions will be denoted by $[L]$ or $[L']$. The set of equivalence classes of linear extension of $\mathcal{P}$ will be denoted with $[LinExt(\mathcal{P})]$.*

By Definition 15 we can think of a linear extension as the set of critical pairs it reverses and by Lemma 7, Theorem 8 and Proposition 9 we can interpret a realizer as a cover of $\mathcal{GCP}$ by directed acyclic subgraphs.

Finally, observe that not all the directed acyclic subgraphs of $\mathcal{GCP}$ correspond to an element of $[LinExt(\mathcal{P})]$; for instance, for the poset of Figure 1.8, the set $\{(z, y)\}$ does not corresponds to any equivalence class of linear extensions as we can reverse at the same time (z, y) and (b, a) or (z, y) and (a, b)). However, to a maximal directed acyclic subgraph corresponds surely a $[L] \in [LinExt(\mathcal{P})]$ as in this case $[L]$ reverses *exactly* the corresponding set of critical pairs. We can than restrict to consider maximal directed acyclic subgraphs of $\mathcal{GCP}$ and covers of maximal directed acyclic subgraphs (as if $\mathcal{GCP}$ can be covered by k directed acyclic subgraphs it can be covered by k maximal ones).

All this reasoning is condensed in the following corollary: :

Corollary 4. *$\mathcal{P}$ has a realizer of size k iff $\mathcal{GCP}$ can be covered by k maximal directed acyclic subgraphs.*

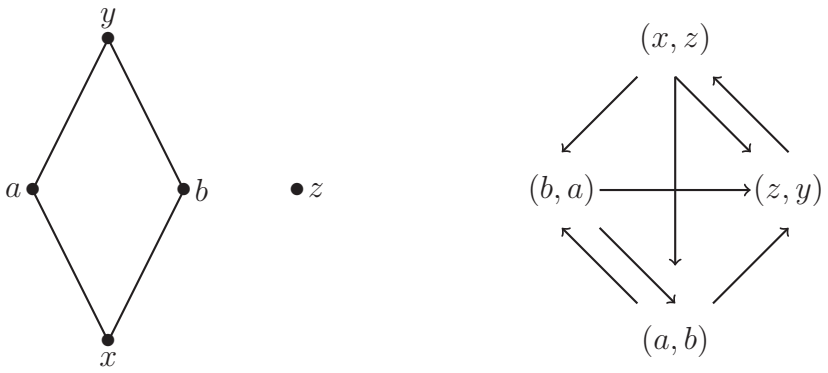

Figure 1.8: A poset $\mathcal{P}$ (left) and its graph of critical pairs $\mathcal{GCP}(\mathcal{P})$ (right). The set $\{(z, y)\}$ does not corresponds to any equivalence class of linear extensions as we can reverse at the same time (z, y) and (b, a) or (z, y) and (a, b).

In the following section we show how we can compute the poset dimension of $\mathcal{P}$ employ $\mathcal{GCP}$.

1.7.2 Computing $r_k(\mathcal{P})$ Employing Critical Pairs

In this section we present two algorithms which determine the existence of a realizer of a given size k for $\mathcal{P}$.One algorithm employ exponential space in the number of critical pairs and the other polynomial space.

Both algorithms employ implicitly or explicitly the results of Section 1.7.1 which led us to see linear extensions of $\mathcal{P}$ by the set of critical pairs they reverse.

We start presenting the exponential space algorithm:

Exponential Space

If it is acceptable the employment of exponential space, we can compute the values for $s(W)$ (of the exclusion/inclusion formula in the previous section) by filling up a table $s : 2^{\mathcal{CP}} \times \{0, 1, 2, \ldots, q\} \to \mathbb{N}$ where after having imposed an arbitrary order on the critical pairs $\mathcal{CP} = \{cp_1, cp_2, \ldots, cp_q\}$, the value $s(W, i)$ is the number of maximal directed acyclic induced subgraphs of $\mathcal{GCP}$ which do not contain any node in W and all the nodes in $\{cp_{i+1}, cp_{i+2}, \ldots, cp_q\} \setminus W$.
The set $\{cp_{i+1}, cp_{i+2}, \ldots, cp_q\} \setminus W$ is considered empty for $i = q$.

Observe that $s(W) = s(W, q)$ and we can recursively compute the values $s(W, i)$ for $1 \leq i \leq q$ (see for instance Section 4.3.1 in [38]):

$$s(W, i) = \begin{cases} s(W, i-1) & \text{if } cp_i \in W \\ s(W, i-1) + s\big(W \cup \{cp_i\}, i-1\big) & \text{if } cp_i \notin W \end{cases}$$

The basic values for $i = 0$ is given by:

$$s(W,0) = \begin{cases} 1 & \text{if } \mathcal{GCP}[\mathcal{CP} \setminus W] \text{ is a maximal directed acyclic subgraph of } \mathcal{GCP} \\ 0 & \text{otherwise} \end{cases}$$

By using the inclusion/exclusion formula we can then compute the number of covers of size k (made of maximal acyclic subgraphs) of $\mathcal{GCP}$ as:

$$\tilde{r}_k(\mathcal{P}) = \sum_{W \subseteq \{cp_1,\ldots,cp_q\}} (-1)^{|W|} |s(W)|^k$$

if $\tilde{r}_k(\mathcal{P})$ is strictly positive then there exists a k cover of maximal directed acyclic subgraphs of $\mathcal{GCP}$, *i.e.* there exists a realizer of size k for $\mathcal{P}$ by Corollary 2. This procedure results in a $O(2^q) = O(2^{|\mathcal{CP}|})$ time algorithm.

Polynomial Space

We have seen in Section 1.7.1 that there exists a realizer of $\mathcal{P}$ of size k if there exists a directed maximal acyclic cover of $\mathcal{GCP}$ of size k.

Observe that the complement of a maximal directed acyclic subgraph is a minimal feedback vertex set and there exists a non trivial algorithm which can enumerate all minimal feedback vertex set in time $O(1.9977^q)$ [76]. Then with the same inclusion/exclusion formula of the previous section, we can compute on the fly the value of $|s(W)|$ simply by listing all maximal directed acyclic induced subgraphs of $\mathcal{GCP}[\mathcal{CP} \setminus W]$ and check if they are maximal in $\mathcal{GCP}$ in time $O(1.9977^q)$.

The total time for this algorithm is then given by:

$$T(q) = \sum_{j=0}^{q} \binom{n}{j} (1.9977)^{q-j} = (2.9977)^q$$

We collect the results of both the algorithms in the following theorem:

Theorem 9. *Let $\mathcal{P}$ be a poset and let q be the number of its critical pairs. Then we can verify the existence of a realizer for $\mathcal{P}$ of size k in time $O(2^q)$ or employing polynomial space in time $O^*(2.9977^q)$.*

1.7.3 Not Reversing Some Critical Pairs

We have seen in Section 1.7.1 that there is a relation between maximal directed acyclic induced subgraphs of $\mathcal{GCP}$ and linear extension of $\mathcal{P}$ based on the critical pairs they reverse. In the same section we have introduced in Definition 15, an equivalence relation on linear extension according to the set of critical pairs they reverse. The polynomial

space algorithm presented in Section 1.7.2, enumerate all the maximal directed acyclic induced subgraphs of $\mathcal{GCP}[\mathcal{CP} \setminus W]$ for $W \subseteq \mathcal{CP}$.

We want in this section to discuss the possibility of improving the polynomial space algorithm by just enumerating all directed acyclic induced subgraphs whose node set corresponds *exactly* to the critical pairs reversed by all the linear extensions in an equivalence class $[L] \in [LinExt(\mathcal{P})]$.

Indeed, we have seen that not all the directed acyclic subgraph corresponds to equivalence classes (see Figure 1.8).

For the inclusion/exclusion approach, in particular, we are interested in the enumeration of all directed acyclic induced subgraphs in $\mathcal{GCP}[\mathcal{CP} \setminus W]$ which do not reverse any of the critical pairs in W or better, we want to identify all the subsets $S \subseteq \mathcal{CP} \setminus W$ which corresponds to linear extensions of $\mathcal{P}$ that reverse all the critical pairs in S and none in W.

The graph $\mathcal{GCP}$ is missing at least an information, indeed: if $(a, b), (b, a) \in \mathcal{CP}$ as in Figure 1.8, we cannot at the same time have $(a, b) \in \mathcal{CP} \setminus (W \cup S)$ and not reverse $(b, a) \in W$ or we cannot simultaneously have (a, b) and (b, a) in W and not reverse any of them.

With this observation, we introduced the classes of twin nodes of $\mathcal{P}$ (Figure 1.9 shows an example).

Definition 16. *A* twin *element in a poset $\mathcal{P} = (X, P)$ is an element $x \in X$ such that for an element $y \in X$, $y \parallel x$ it holds: $z \leq_P x$ iff $z \leq_P y$ and $x \leq_P z$ iff $y \leq_P z$ for all $z \in X$. In particular x, y are indistinguishable as element of $\mathcal{P}$.*

Observe that for twin nodes it holds trivially:

Fact 2. *x, y are twins in $\mathcal{P}$ if and only if $(x, y), (y, x) \in \mathcal{CP}(\mathcal{P})$.*

In the inclusion/exclusion formula we have to compute the number of equivalence classes of linear extension of $\mathcal{P}$ which do not reverse any critical pair in $W \subseteq \mathcal{CP}$.

In the next proposition we see that the fact that not reversing a critical pair (x, y) force another critical pair (u, w) to be reversed imply that x, y are twins and $(u, w) = (y, x)$:

Lemma 8. *Let $(x, y), (u, w) \in \mathcal{CP}(\mathcal{P})$. If there exists no $L \in LinExt(\mathcal{P})$ such that $x \leq_L y$ and $u \leq_L w$ then $(u, w) = (y, x)$.*

Proof. By the non existence of L and by Theorem 8 we have that $\{(y, x), (w, u)\}$ contains a strict alternating cycle and in particular $y \leq_P u$ and $w \leq_P x$. By definition of critical pair if it were $y <_P u$ then $x <_P u$ which is absurd since $w \leq_P x <_P u$ against $u \parallel_P w$. We have to conclude that $y = u$ and with analogous reasoning that $x = w$. □

Hence twin nodes can create problems as we cannot simply restrict to the subgraph $\mathcal{GCP}[\mathcal{CP} \setminus W]$ and compute all the directed acyclic graphs here or we might reverse $(x, y) \in W$ if the directed acyclic subgraph does not contain (y, x).

We can solve this problem by temporary removing the twins imposing an arbitrary but fixed order among twin elements: First of all note that the *twin* status is an equivalence relation among elements of X and for a twin element x let denote with $[x]_t$ its equivalence class.

Let τ be the number of twin elements in an equivalence class. In this case imposing an arbitrary but fixed linear order $Q = ([x]_t, \leq_Q)$ to the elements of $[x]_t$ and denoting with $\overline{\mathcal{P}}^Q = poset(P \cup \leq_Q)$ (*i.e.* the poset $\mathcal{P}$ with all the twin elements ordered by an arbitrary but fixed total order Q, see the right graph of Figure 1.9), we reduce the number of critical pairs by $\tau(\tau - 1)$.

We can then apply our algorithm to the graph $\mathcal{GCP}(\overline{\mathcal{P}}^Q)$ indeed: trivially there exist a realizer of size k for $\mathcal{P}$ if and only if there exist a realizer of size k for $\overline{\mathcal{P}}^Q$ as either $\mathcal{P}$ has dimension 1 (hence is a total order and has no twin nodes) or it has dimension at least 2 so we can take a realizer of $\overline{\mathcal{P}}^Q$ with two linear extension L, L' one that extends Q and one that extends Q^d (the total order on the same ground set of Q with order given by $x \leq_{Q^d} y$ iff $x \leq_Q y$).

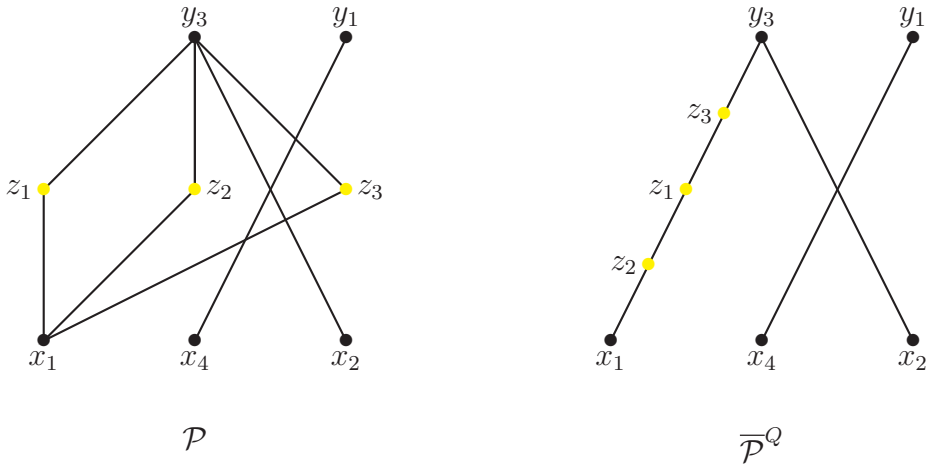

Figure 1.9: (left): A poset $\mathcal{P}$, z_1, z_2, z_3 are twin elements. (right): The poset $\overline{\mathcal{P}}^Q$ where Q is the linear order defined by $z_2 \leq_Q z_1 \leq_Q z_3$.

This process can be done also when there is more than one equivalence class of twin nodes hence, if $\tau_1, \ldots, \tau_l$ are the size of all these different equivalence classes, the enumeration of directed acyclic subgraphs will run on a $\mathcal{GCP}$ with $q - \sum_{i=1}^{l} \tau_i(\tau_i - 1)$ critical pairs.

Denoting with $\overline{\mathcal{GCP}}^Q$ the twin-free graph obtained in this way, we are left then with the following problems:

Problem 3. *Is there a bijection between equivalence classes in $[LinExt(\mathcal{P})]$ and directed acyclic induced subgraphs of $\overline{\mathcal{GCP}}^Q$ (where among them we count also the empty graph) given by the exact set of critical pairs which are reversed by all the linear extensions of an equivalence class?*

Conjecture 1. *Let $S, W \subseteq \mathcal{CP}(\overline{\mathcal{P}}^Q)$ and let S be alternating cycle-free. Then there exists a linear extension $L \in LinExt(\overline{\mathcal{P}}^Q)$ such that L reverses all the critical pairs in S and none in W.*

1.8 Conclusions and Open Problems

In this chapter, we studied different problems related to the Toxin and Antitoxin Model for CI which lead us to the enumeration of maximal chain subgraphs and chain subgraph covers in bipartite graphs, interval order dimension and to extend our results to poset dimension. This work raises many questions:

Problem 4. *Is it possible to enumerate the minimal chain covers of a graph in polynomial delay?*

Indeed, our problem is more constrained than an arbitrary instance of the set cover of a hypergraph as we have solved it. A future goal is to better exploit the connections between these two problems.

Secondly, the bound $T(m)$ on the number of maximal chain subgraphs of a bipartite graph with m edges is not tight. It would be interesting to determine the exact value of $T(m)$. We have the following conjecture:

Conjecture 2. *Let $G = (U \cup W, E)$ be a bipartite graph with m edges; then $T(m) \leq \frac{1+\sqrt{1+4m}}{2}!$.*

This bound would be reached in the case of the antimatching graph A_n (see figure 1.5).

Finally we recallthe two open question related to poset dimension and that we have already presented at the end of the previous section. In particular in Problem 3, we wonder if the equivalence classes of linear extensions of a poset $\mathcal{P}$ are in bijection with directed acyclic subgraphs of the graph of critical pair $\mathcal{GCP}(\mathcal{P})$ (when $\mathcal{P}$ has no twin elements, Definition 16).

We have then formalized in Conjecture 1 the possibility of enumerating in a faster and more efficient way (with polynomial space) the direct acyclic induced subgraphs of $\mathcal{GCP}(\mathcal{P})$ which corresponds to specific linear extensions of $\mathcal{P}$. List these subgraphs is necessary for the inclusion/exclusion formula we use.

Cophylogeny and Reconciliations

2.1 Introduction

It is rare to find in nature an organism which does not live in *symbiosis*, a greek word which literally means "living together". We speak about symbiosis in any situation where different organisms have a stable biological interaction during a period or possibly for all their life. There are different kinds of symbiosis and so we speak about, for example, *mutualistic, commensalistic or parasitic* symbiosis depending on the benefits or drawbacks the single organisms have from the interaction.

Sometimes organisms in symbiosis may influence one the evolution of the other as for instance the case of *Wolbachia* and the bidirectional Cytoplasmic Incompatibility that it induce on its host and which may trigger an evolution of the host species (see [88] and the introductive paragraph of Chapter 1); in this case we speak about *coevolution*. The concept of coevolution has a long history: from Charles Darwin, who mentioned in its *On The Origin of Species* an evolutionary interaction between flowering plants and insects, it still attracts a wide interest [72, 79].

A *phylogenetic tree* T is a leaf-labelled rooted full binary tree that models the evolution of a set of species from their most recent common ancestor (placed at the root) to the current derived species (placed at the leaves). The internal nodes of the tree correspond to speciation events (a speciation occurs when a species evolves in divergent ways to other species). The tree is rooted so a partial order among its nodes is intrinsically assumed and it corresponds to the ancestor and descendant relations.

A classical approach to study the coevolution of organisms is to consider their phylogenetic trees together trying to identify and model coevolutionary events that could have possibly occurred. Without loss of generality and for sake of simplicity, from here on we will set in the special relation of host/parasite. The evolutionary events can be described and understood, for instance in the case of a parasite and its host, by mapping the phylogenetic tree P of the parasite into the phylogenetic tree H of the host [89]. We characterize then some evolutionary events according to this mapping.

The model of host-parasite evolution we rely on is based on the ones described in [82] and [4] and it will be presented in Section 2.3 after few introductory notions on our notation.

The rest of this chapter is organized as follows: in Section 2.2 we present the notation we use, some definitions and some basic results which will be employed throughout the rest of the chapter.
In Section 2.4 we present EUCALYPT, the algorithm we use and that we modified to compute the experimental results of Section 2.9 which we also comment them suggesting some interpretations and heuristics.
This results describes the potentialities of the two equivalence classes $\sim_1$ and $\sim_2$ among reconciliations and presented in Section 2.6 and Section 2.7 respectively. We also present some histograms and statistics on the distance DH between optimal reconciliations and introduced in Section 2.8. We end up the chapter by a conclusive section where we pose some open problems and conjectures.

2.2 Preliminaries and Notation

A *binary rooted tree* T is *full* if every node in the tree have either 2 or 0 children. We denote by $V(T)$ and $E(T)$ the set of its nodes and edges, respectively, and with $L(T)$ the leaves set of T.

If $v \in V(T)$ and v is not the root of T, then $p(v)$ denotes its father and $s(v)$ its sibling.

Given two nodes $u, v \in V(T)$, u is said an *ancestor* of v, denoted by $u \succeq_T v$, if either $u \equiv v$ or v is contained in the subtree rooted at u. If either $u \succeq_T v$ or $v \succeq_T u$, then we call them *comparable nodes* and write $u \cong_T v$. We say that u and v are *incomparable* if they are not comparable and write $u \ncong_T v$.

The *least common ancestor* $lca(S)$ of a set of nodes $S \subseteq V(T)$ is the $\succeq_T$-minimal node u s.t. $u \succeq_T s$ for all $s \in S$.

We denote by $path_T(u, v) = (t_1, \ldots, t_j)$ the (unique) ordered sequence of nodes of T that must be traversed to reach v from u; of course $t_1 \equiv u$ and $t_j \equiv v$. The length of $path_T(u, v)$ is $j - 1$ and it is denote as $|path_T(u, v)|$.

Let T_v be the subtree of T rooted at v and let $e = (u, v)$ be an edge of T, we introduce here the tree T^e (see Figure 2.10) as the subtree of T obtained from T:

- eliminating all nodes and edges of T_v;

- if u is not the root of T, substituting edges $(p(u), u)$ and $(u, s(v))$ with the single edge $(p(u), s(v))$, otherwise simply eliminating edge $(u, s(v))$.

When there is ambiguity on the edge $e = (u, v)$ we denote the tree T^e as T^u.

2.3 The Reconciliation Model

The model we use and that we present here under is the *reconciliation model*. This model can be employed not only to study the coevolution of a host and a parasite, but also

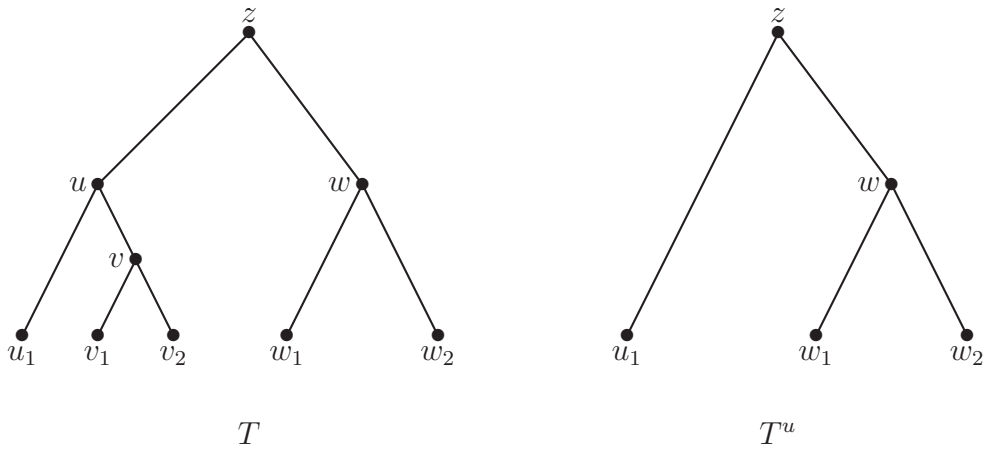

Figure 2.10: A tree T with an edge $e = (u, v)$ (left) and the reduced tree T^e (right).

for the evolution of a gene with respect to a species or the evolution of a species on a geographical territory [4, 46, 82].

In this model four events can occur, and they can be informally described as follows:

cospeciation occurs when both the parasite and the host speciate,

duplication occurs when the parasite speciates but the host does not,

loss occurs when the host speciates but the parasite does not follows both the branches,

host switch occurs when the parasite speciates and one of its children "jumps" to an incomparable host.

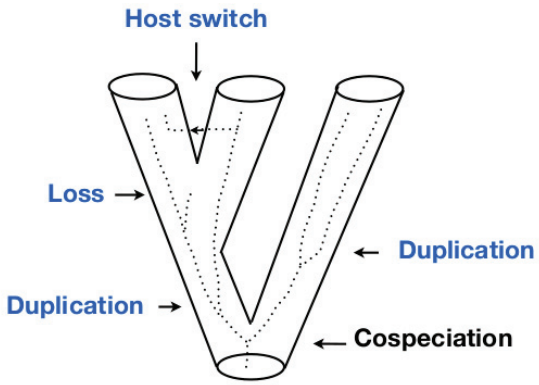

Figure 2.11: The nodes of the thinner parasite tree P are mapped to the nodes of the "tubolar" host tree H. According to the mappings of these nodes, four types of events are presented: cospeciation, duplication, loss and host switch.

This model requires a pair of full binary rooted trees, H and P , and a function

$\varphi : L(P) \to L(H)$, which maps every leaf of P to a leaf of H and which represents the parasitic interaction between host and parasite species.

For sake of simplicity, in the rest of the chapter we consider H, P and φ fixed if not differently specified.

Given H, P and φ, we can model co-evolutive events by extending φ from all the nodes of P to all the nodes of H, under some constraints:

Definition 17 ([82]). *Given two phylogenetic trees P, H and a leaf mapping function $\varphi : L(P) \to L(H)$, the function $\varrho : V(P) \to V(H)$ is said a* reconciliation of the two phylogenetic trees H and P with leaf mapping φ *(or simply a* reconciliation *when H, P, φ are clear) if ϱ satisfies the following constraints:*

1. *For each leaf $l \in L(P)$ it holds $\varrho(l) = \varphi(l)$,*
2. *For each node $p \in V(P) \setminus L(P)$, if $q, r \in V(P)$ are the children of p then $\varrho(p)$ is not a proper descendant of $\varrho(q)$ or $\varrho(r)$ and at least one of them is a descendant of $\varrho(p)$.*

Given a reconciliation we can partition the set of internal nodes of P as follows:

Definition 18 ([82]). *Let $\varrho : V(P) \to V(H)$ be a reconciliation and let $\Delta, \Theta, \Sigma \subseteq V(P) \setminus L(P)$ be defined by the following rules for a node $p \in V(P) \setminus V(L)$ with children r, q:*

$p \in \Sigma$ *if and only if $\varrho(p) = lca\{\varrho(q), \varrho(r)\}$ and $\varrho(q), \varrho(r)$ are incomparable and we say that p* is assigned to a cospeciation event*;*

$p \in \Theta$ *if and only if $\varrho(p)$ is incomparable with either $\varrho(q)$ or $\varrho(r)$ (called the* jumping child of p*) and we say that p* is assigned to an host-switch event *;*

$p \in \Delta$ *if and only if $p \notin \Sigma \cup \Theta$ and we say that p* is assigned to a duplication event*.*

So, every internal node is assigned an event among cospeciation, duplication or host switch and the set $V(P) \setminus L(P)$ results partitioned as $\Delta \cup \Theta \cup \Sigma$.

Furthermore observe that we could have defined, by exclusion, the nodes in Δ also as:

$p \in \Delta$ if and only if $\varrho(p) \succ_h lca\{\varrho(q), \varrho(r)\}$ or $\varrho(p) = lca\{\varrho(q), \varrho(r)\}$ and $\varrho(q), \varrho(p)$ are comparable.

We privileged the other form as we wanted to underline the fact that we are partitioning the sets of internal nodes of P.

An edge of P in

$$\widetilde{\Theta}(\varrho) := \{(u, v) \in \Theta \times V(P) : p(v) = u \text{ and } \varrho(u), \varrho(v) \text{ are incomparable}\},$$

is called *host-switch edge of* ϱ.
We write $\widetilde{\Theta}$ instead of $\widetilde{\Theta}(\varrho)$ when there is no doubt about the reconciliation ϱ involved.

We can then assign a number of events of type *loss* to the edges of P as follows: for an edge $e = (u, v) \in E(P)$ where $u \preceq_P v$ we define preliminarily $I_\varrho(e) := \{x \in V(H) : \varrho(u) \prec_H x \prec_H \varrho(v)\}$, then the number of losses associated to e is given by:

$$loss_\varrho(e) := \begin{cases} I_\varrho(e) + 1 & \text{if } u \notin \Sigma,\ \varrho(u), \varrho(v) \text{ are comparable and } \varrho(u) \neq \varrho(v), \\ I_\varrho(e) & \text{otherwise.} \end{cases}$$

The number of losses of ϱ is defined as $|\Lambda| := \sum_{e \in E(P)} loss_\varrho(e)$.

This definition of losses is based on [82] and [4].

In literature the event-based models may vary in the number and type of event classifications (see [89] for a more complicated model in terms of events), and in consequence this changes the size of the partition of the internal nodes of P according to the number of event. However in general, independently of the refinement of the partition [89, 90], it is usual to assign a cost for each of the possible events, to obtain a global cost for the reconciliation summing up the costs for all the nodes:

$$cost(\varrho) := c_s|\Sigma| + c_d|\Delta| + c_h|\Theta| + c_l|\Lambda|$$

where $C = (c_s, c_d, c_h, c_l)$ is a real valued quartuple which defines the cost assigned to each event, called *cost vector*. For the rest of the chapter, we consider C to be a fixed vector.

We are now ready to present the following problem:

Definition 19. *Given H, P and the leaf mapping φ, the* reconciliation problem *consist in enumerating all the reconciliations $\varrho : V(P) \to V(H)$ with minimum cost. Such reconciliations are said* most parsimonious *reconciliations.*

Definition 20. *We denote with $\mathcal{R}(H, P, \varphi, C)$ the set of the most parsimonious reconciliations $\varrho : V(P) \to V(H)$, in which the costs of the events are given by C, and whose leaves are connected through the mapping $\varphi : L(P) \to L(H)$.*

Finally, it is also possible to associate to each reconciliation ϱ a vector $E_\varrho = (e_c, e_d, e_s, e_l) = (|\Sigma|, |\Delta|, |\Theta|, |\Lambda|)$ [5], called *event vector*, where e_c, e_d, e_s and e_l denote the number of cospeciations, duplications, host switches and losses, respectively, that are in ϱ.

Extension of the Model and Alternative Models

Note that in the reconciliation model we have presented the parasite nodes are allowed to be mapped only on host tree nodes, but there are other models [28, 89] where they are allowed to be mapped also to edges of H or the host tree is extended to a tree $\tilde{H}$ with

some artificial nodes to create some temporal layers defined by the height of the nodes and host switch are allowed only between contemporary nodes; this temporal information sometimes is provided by a function $V(P) \to \mathbb{R}$ which gives the time of speciation.

For sake of completeness, note that the reconciliation model is not the only possible model to study coevolutions of organisms. There are also probabilistic models which simulate the development of the parasite species within the host tree H by a *birth-and-death process* and where the edged of H are provided with a length (representing their time duration) and estimated duplication and loss rates [27].

Cyclic and Acyclic reconciliations

Passing from a model without host switches (for instance the reconciliation model where the host-switch cost c_h is infinite or extremely high), to a model where they are allowed, we permit to the parasite lineage to jump to and fro the host lineage with the possibility of creating some time inconsistencies and cycles as we do not have any time information on the nodes if not the one given by the order of the tree.

For this reason, among the reconciliations, we need to distinguish between *time feasible* and *time unfeasible* ones. They are also said *acyclic* and *cyclic* respectively as their definition is given in terms of cyclic and acyclic graphs as follows:

Given a reconciliation $\varrho : V(P) \to V(H)$, let $G(\varrho) = (V, E)$ be the digraph defined as $V = V(H)$ and $E = E(H) \cup D$ where:

$$D := \bigcup_{\substack{(u,v),(u',v') \in \overline{\Theta} \\ u \preceq_P u'}} \{p(\varrho(u)), p(\varrho(v))\} \times \{\varrho(u'), \varrho(v')\}$$

Definition 21 ([80])**.** *Let $\varrho : V(P) \to V(H)$ be a reconciliation. Then ϱ is said to be a* time feasible *(or* acyclic*) reconciliation if the graph $G(\varrho)$ does not contain any direct cycle.*

ϱ is said time unfeasible *or* cyclic *if it is not time feasible.*

When host switches are allowed, finding an optimal time feasible reconciliation is NP-hard as proved in [70, 82] with a reduction from the *Minimum Feedback Arc Set*. However, if we do not restrict to time feasible reconciliations, the problem becomes tractable and it is possible to enumerate all the optimal reconciliations in polynomial time with a polynomial space complexity algorithm as Eucalypt [25].

In the next section we present this algorithm that we modify to count the number of equivalence classes $\sim_1$ and $\sim_2$ in Section 2.9. The equivalence classes are introduced in Section 2.6 and Section 2.7.

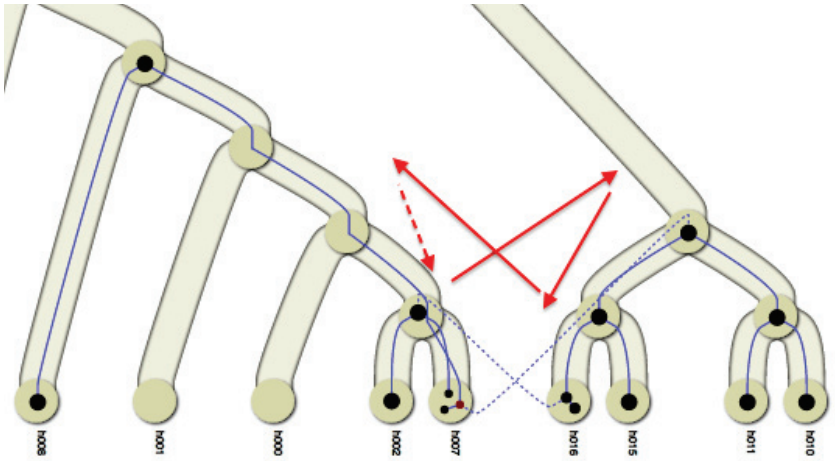

Figure 2.12: A time unfeasible reconciliation: the red node has to be mapped in H at least to the father of the current node.

2.4 Eucalypt

The optimal solutions for the Reconciliation Model we presented in Section 2.3 can be computed by a dynamic programmig approach.

In this section we present the specific optimal subproblems which are solved to obtain the final solution, we present the data structures and peculiarities of the implementation of Eucalypt [25], the algorithm we have employed to enumerate all the most parsimonious reconciliation in $\mathcal{R}(H, P, \varphi, c)$ and to collect the experimental results of Section 2.9.

We start from the formulation of the subproblems:

Definition 22. *Let $h \in V(H)$. We denote with $\underline{\mathcal{R}}(H, P, \varphi, C, h)$ the set of most parsimonious reconciliations $\varrho : V(P) \to V(H)$ with leaf mapping φ with the further condition of mandatory mapping the root of P to node h.*

With this definitions it is clear that, taken an optimal reconciliation $\varrho \in \mathcal{R}(H, P, \varphi, c)$, letting q, r be the children of the root node z of P, for the recoonciliations $\varrho_q := \varrho_{|P_q}$, $\varrho_r = \varrho_{|P_r}$ we have that: $\varrho_q \in \underline{\mathcal{R}}(H, P_q, \varphi, C, \varrho(q))$, $\varrho_r \in \underline{\mathcal{R}}(H, P_r, \varphi, C, \varrho(r))$ and letting $e \in \{c, d, h\}$ we have that:

$$cost(\varrho) = cost(\varrho_q) + c_l \cdot loss_\varrho((\varrho(z), q)) + c_e + c_l \cdot loss_\varrho(\varrho(z), r)) + cost(\varrho_r).$$

Finally observe that $\varrho \in \underline{\mathcal{R}}(H, P, \varphi, C, \varrho(root(p)))$ we can then compute with the dynamic programming paradigm the set of optimal reconciliations by storing and computing the information contained in matrix $D(p, h)$, which intuitively coincide with $\underline{\mathcal{R}}(H, P_p, \varphi, C, h)$, and whose row and columns are indexed by the nodes of the parasite tree $V(P)$ and of the host tree $V(H)$ respectively.

This is precisely what Eucalypt [25] and similar approaches do (see also [4, 82, 89]).

Moreover, EUCALYPT employs a second data structure, $\widetilde{D}(p,h)$ defined as the set of optimal reconciliations in $\varrho \in \underline{\mathcal{R}}(H,P_p,\varphi,C,h')$ for $h' \preceq_H h$ which minimizes the cost:

$$cost(\varrho) + c_l \cdot loss_\varrho((h,h')).$$

With this second data structure we than have that the reconciliations $\varrho \in \underline{\mathcal{R}}(H,P,\varphi,C,h)$ are the reconciliations which minimize the costs:

$cost(\varrho_q)+c_e+cost(\varrho_r)$ for $\varrho_q \in \underline{\mathcal{R}}(H,P_q,\varphi,C,h_q)$, $\varrho_r \in \underline{\mathcal{R}}(H,P_r,\varphi,C,h_r)$ and $h_q,h_r \preceq_H h$.

The Information in $D(p,h)$ and $\widetilde{D}(p,h)$

We have already mentioned that the information in $D(p,h)$ are the necessary ones to reconstruct all the reconciliations in $\underline{\mathcal{R}}(H,P_p,\varphi,C,h)$ and that $\widetilde{D}(p,h)$ is computed once filled the values for $D(p,h')$ with $h' \preceq_H h$.
In Section 2.7, we describe the modifications done to the algorithm to compute the number of $\sim_2$-equivalence classes and in order to do so we need to specify better the structure of $D(p,h)$.

In fact, the values for $D(p,h)$ and $\widetilde{D}(p,h)$ are filled with a *post-order* traversal of the nodes of P and H and the optimal reconciliations which map p to h are computed starting from the optimal mapping of the children p_1,p_2 of p as we have seen.

It suffice then to keep track of the mapping of the children p_1,p_2, say to the nodes $h_1,h_2 \in V(H)$ respectively, and to which of the matrices D $\widetilde{D}$, to be able to reconstruct the optimal reconciliation recursively by accessing the values of $D(p_i,h_i)$ or $\widetilde{D}(p_j,h_j)$.

Finally for a given imposed mapping of p to h, there may be more than one optimal *pair* of mappings for the children of p; then the value in $D(p,h)$ (or $\widetilde{D}(p,h)$) is either a *simple solution*, *i.e.* a pair of pointers for example $(D(p_1,h_1),\widetilde{D}(p_2,h_2))$ together with the cost of this solution and the event associated or the values in $D(p,h)$ (or $\widetilde{D}(p,h)$) is a list of pairs of pointers to simple solutions all with the same optimal cost and we call it a *multiple solution* (see Figure 2.13).

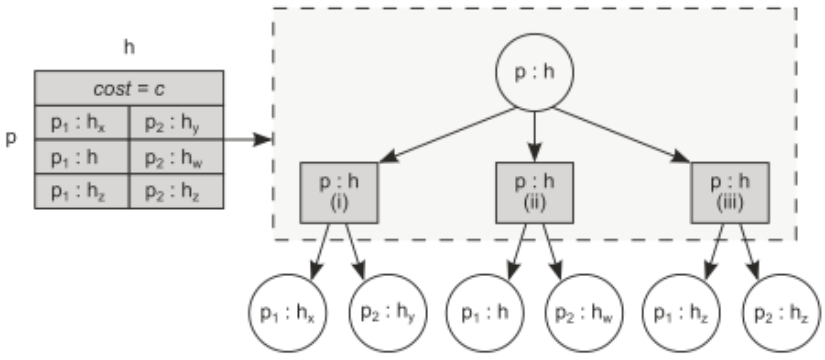

Figure 2.13: Figure 2 in [25]: Schematic representation of a multiple solution cell in $D(p,h)$ (or $\widetilde{D}(p,h)$) with three alternative mappings for the children p_1,p_2 of p.

In practice EUCALYPT computes the optimal reconciliations in $D(p,h)$ after three different branching done in a first step, selecting the best speciation, duplication and host-switch solutions with the only imposition of mapping p to h and in a second step the most parsimonius among them.

Here under we sketch these three selections as they are done in EUCALYPT for the value of $D(p,h)$. In the following let $p, p_1, p_2 \in V(P)$ be such that $p(p_1) = p = p(p_2)$ and $h, h_1, h_2 \in V(H)$ such that $p(h_1) = h = p(h_2)$.

The optimal mappings which associate a speciation event are the pairs $(left, right)$ which give the minimum cost for $c(left) + c_s + c(right)$ where the only possible pairs are $(\widetilde{D}(p_1,h_1), \widetilde{D}(p_2,h_2)), (\widetilde{D}(p_1,h_2), \widetilde{D}(p_2,h_1))$, (*i.e* to induce a speciation event p_1, p_2 has to be mapped to incomparable nodes and h has to be the least common ancestor of them, see Definition 18).

For the duplication event, the optimal mappings are obtained by the left right pairs $(left, right)$ which minimize the cost $c(left) + c_d + c(right)$. A duplication event occurs when either:

$$\begin{cases} (D(p_1,h), D(p_2,h)), (D(p_1,h), \widetilde{D}(p_2,h_1)), & \\ (D(p_1,h), \widetilde{D}(p_2,h_2)), & \text{the mapping of } p_1, p_2 \text{ are comparable in } H \\ (\widetilde{D}(p_1,h_1), D(p_2,h)), & \\ (\widetilde{D}(p_1,h_2), D(p_2,h)) & \\ \text{or} & \\ (\widetilde{D}(p_1,h_1), \widetilde{D}(p_2,h_2)), (\widetilde{D}(p_1,h_2), \widetilde{D}(p_2,h_1)) & \text{they are not comparable and } h \neq h_1, h_2 \end{cases}$$

Eventually, the host-switch event is computed as mapping one of the two children to the subtree rooted at h and the other to another node of H $h' \ncong h$ keeping, among the pairs $(left, right)$, $(\widetilde{D}(p_1,h), D(p_2,h')), (D(p_1,h'), \widetilde{D}(p_2,h))$ for $h' \ncong h$, the ones which minimize the cost $c(left) + c_h + c(right)$.

The pairs stored in $D(p,h)$ are then the optimal ones among these three pre-selections event-based.

Complexities

We have already mentioned in the previous section that if host switches are allowed computing an optimal acyclic reconciliation is NP-hard [82].

However, if we do not restrict to acyclic solutions the problem becomes tractable: a solution can be computed in $O(nm)$ time where m and n are the number of parasite and host tree nodes, respectively.

While to enumerate all the optimal (not necessarily acyclic) reconciliations, EUCALYPT takes $O(n^3m)$ time and space [4, 25] by performing for each optimal reconciliation a DFS following the optimal children mapping in $D(p,h)$ and $\widetilde{D}(p,h)$ keeping track in a side list of the alternatives mappings for the children mapping so as to explore each time a different combination until exhaustion.

2.5 Preliminary Lemmas

We prove in this section some useful properties and lemmas on reconciliations.

The following fact follows directly from the impossibility that all children of a parasite node jump to incomparable host tree nodes.

Fact 3. *Given any reconciliation $\varrho : V(P) \to V(H)$, for each node $u \in V(P)$:*

a. there exists a leaf $l \in V(P)$ such that $l \preceq_P u$ (i.e. $l \in P_u$) and $\varrho(l) = \varphi(l) \preceq_H \varrho(u)$ (i.e. $\varrho(l) \in H_{\varrho(u)}$);

b. if edge $(u,v) \in E(P)$ (where u is the parent of v) is a host switch (i.e. $\varrho(u) \not\cong_H \varrho(v)$) then there exist a leaf $l' \in V(P)$ such that $l' \preceq_P u$ (i.e. $l' \in P_u$) but $\varrho(l') = \varphi(l') \not\cong_H \varrho(u)$ (i.e. $\varrho(l') \notin H_{\varrho(u)}$).

The so-called *lca* mapping of a parasite node $p \in P$ [46] is defined as $lca(p) := lca_H(\varphi(L(P_p)))$, *i.e* the least common ancestor in H of all species which contain a parasite descended from that parasite node. This function can be inductively computed with φ as base of the induction [82]. It is known that the *lca* mapping induces a most parsimonious reconciliation in a model without host switches [46, 48, 63, 71].

Theorem 10. *Let (H, P, φ) be a scenario. In a model where the only allowed events are cospeciation, duplication and loss, there exist only one optimal reconciliation given by the* lca *mapping $lca(p) := lca_H(\varphi(L(P_p)))$.*

The function *lca* is *monotonic*, *i.e.* for all $u, v \in V(P)$, if $u \preceq_P v$ then $lca(u) \preceq_H lca(v)$ and it plays a central role in many of the lemmas of this section.

We continue with an observation on models were host switches are not allowed.

Fact 4. *In a model where the only events allowed are cospeciation,duplication and loss, any reconciliation $\varrho : V(P) \to V(H)$ is such that for any $u, v \in V(P)$ s.t. $u \succeq_H v$ we have $\varrho(u) \succeq_H \varrho(v)$. Hence, for any $u \in V(P)$, $\varrho(u) \succeq_H lca(u)$.*

If host switches are allowed and we restrict to consider only optimal reconciliations, the situation changes, as highlighted by the following result.

Lemma 9. *Let be given an optimal reconciliation $\varrho \in \mathcal{R}(H, P, \varphi, c)$. If $c_l > 0$ and $c_d \geq c_c$, for any $u \in V(P)$, we have $\varrho(u) \preceq_H lca(u)$.*

Proof. The case $|V(P)| = |L(P)| = 1$ is trivially true. Let us then assume that the tree P has at least an internal node u. Preliminarily observe that it cannot be $\varrho(u) \ncong_H lca(u)$; indeed, in view of part *a.* of Fact 3, there exists $l \in L(P_u)$ such that $\varrho(u) \succeq \varrho(l)$. In view of this and of the fact that $lca(u)$ must lie in H on the path between $\varrho(l)$ and the root of H, it follows that necessarily $\varrho(u) \cong_H lca(u)$.

Hence, suppose by contradiction that $\varrho(u) \succ_H lca(u)$. In this case, node u cannot be associated to a host-switch event; indeed, by part *b.* of Fact 3, it would exist $l' \in L(P_u)$ such that $\varrho(l') \ncong_H \varrho(u)$, in contradiction with $\varrho(u) \succ_H lca(u) \succeq_H \varphi(l') = \varrho(l')$.
It follows that u is associated to an event mapping both the u's children in $H_{\varrho(u)}$.

Without loss of generality, let u be such that it is the lowest node for which the absurd hypothesis $\varrho(u) \succ_H lca(u)$ holds (*i.e.* no descendant node $w \prec_P u$ is such that $\varrho(w) \succ_H lca(w)$) and let v, w be the children of u (observe that such children always exist as at least for $f \in L(P)$ we have $\varrho(f) = \lambda(f) = lca(f)$). For v and w we have that $\varrho(v) \preceq_H lca(v)$ and $\varrho(w) \preceq_H lca(w)$. Then we have $\varrho(w) \preceq_H lca(w) \preceq_H lca(u) \prec_H \varrho(u)$ and the same holds for v.

Recalling that u is associated to either a duplication or a co-speciation, we then get a reconciliation with smaller cost by moving $\varrho(u)$ to $lca(u)$. Indeed in this way, we reduce the costs of at least $c_l * |path_H(lca(u), \varrho(u))|$ (see Figure 2.14), reducing the losses along the edges (u, v) and (u, w) but having to pay the cost of losses along the edge $(p(u), u)$ and possibly turning the event associated to u from duplication to co-speciation reducing the cost of $c_d - c_c \geq 0$.
This is in contradiction with the optimality of ϱ. □

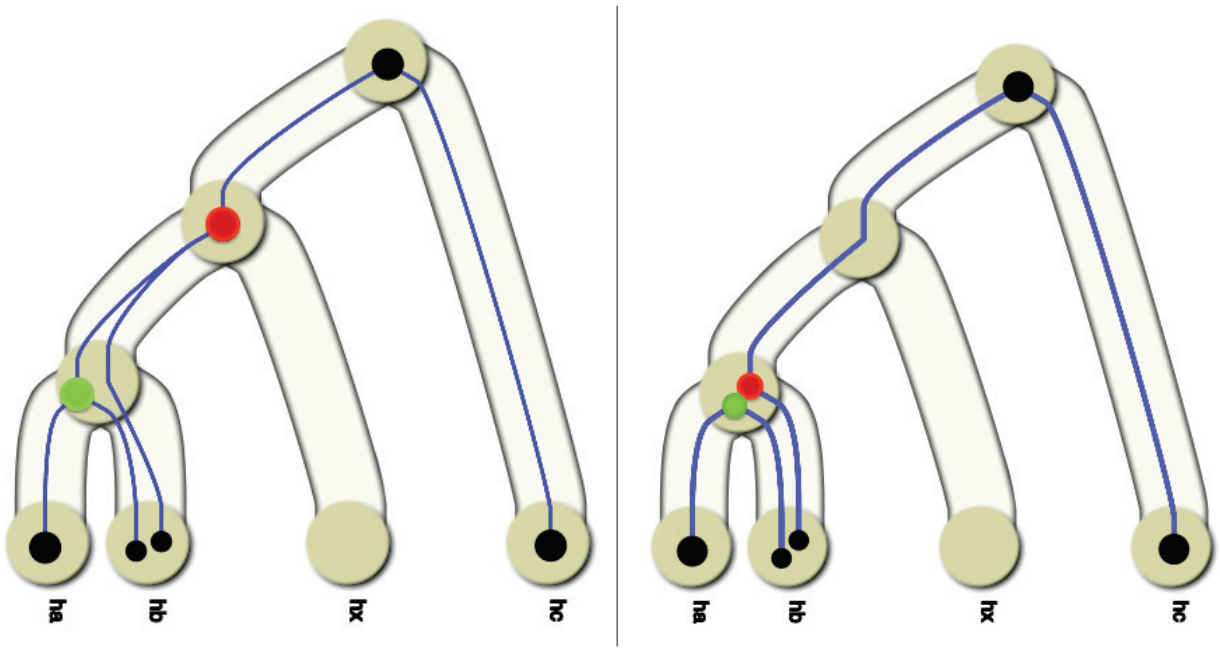

Figure 2.14: Two reconciliation with the same event vector except for the number of losses: the left one has 3 losses while the right one only 2.

Here after we present a result related to nodes adjacent to more than one host-switch nodes; this result will be fundamental in Section 2.7:

Lemma 10. *Let $\varrho \in \mathcal{R}(H, S, \varphi, C)$ and let $c_l > 0$. For each edge (u, v) associated by ϱ to a host switch s.t. $(p(u), u)$ is also associated a host-switch event, it hold: $\varrho(u) = \varrho(s(v))$.*

Proof. Suppose by absurd that $\varrho(s(v)) \prec_H \varrho(u)$. Then we can define a more parsimonious reconciliation ϱ' which agrees with ϱ on all the nodes but u which is mapped in ϱ' to $\varrho(s(u))$. In this way, the only events which may change are the ones associate to $p(u)$ and u. This is not the case as both the nodes remain associated to an host-switch event as $\varrho'(p(u)) = rho(p(u)) \not\cong_H \varrho(u) \succeq_h \varrho(s(v)) = \varrho'(u)$ hence $\varrho'(p(u)) \not\cong_H \varrho'(u)$ and $\varrho'(v) = \varrho(v) \not\cong_H \varrho(u) \succeq_H \varrho(s(v)) = \varrho'(u)$ hence $\varrho'(u) \not\cong_H \varrho'(v)$.

However, ϱ' spare the cost of $c_l * |path_H(\varrho(u), \varrho(s(v))| > 0$ as $\varrho'(s(v)) = \varrho'(u)$. □

Projections of Reconciliations

Notice that, for each host switch (u, v) of a given reconciliation ϱ in $\mathcal{R}(H, P, \varphi, C)$, two reconciliations are naturally induced: $\varrho^u := \varrho_{|V(P^u)}$ and $\varrho_v := \varrho_{|V(P_v)}$. It is not difficult to convince oneself that ϱ^u and ϱ_v are valid reconciliations. We now prove that the cost of ϱ can be expressed in terms of the costs of ϱ^u and ϱ_v, and that these two reconciliations remain optimal.

Lemma 11. *(Reconciliation Decomposition Lemma) Let $(u, v) \in \widetilde{\Theta}(\varrho)$ (with $p(v) = u$) for $\varrho \in \mathcal{R}(H, P, \varphi, C)$, and let $c_l > 0$ and $c_c \leq c_d$. Then we have that:*

$$cost(\varrho) = cost(\varrho^u) + c_s + cost(\varrho_v).$$

Proof. Remind that the total cost of ϱ is given by $cost(\varrho) = \sum_{i \in \{c,d,s,l\}} e_i \cdot c_i$. The contribution to the cost of the nodes of P_v is unaltered both in $cost(\varrho)$ and in $cost(\varrho_v)$ and host-switch edge (u, v) contributes to $cost(\varrho)$ with exactly c_s. So, we focus our attention on P^u, and show in the following that the costs of $\varrho_{|V(P)\setminus(V(P_v)\cup\{u\})}$ (referred simply as ϱ in the following) and of ϱ^u are exactly the same, so concluding the proof.

Notice that the removal of u (performed to obtain P^u) may affect only the number of loss events associated to new edge $(p(u), s(v))$ and the event associated to $p(u)$ (while all the events associated by ϱ^u to the other nodes of P^u remain unaltered w.r.t. ϱ). We divide the rest of the proof into two cases, according to the fact that the event associated to $p(u)$ by ϱ is a host-switch event or not.

Case 1. the event associated to $p(u)$ by ϱ is not a host-switch event.

The number of loss events of ϱ^u along edge $(p(u), s(v))$ are the sum of the loss events of ϱ along edge $(p(u), u)$ and along edge $(u, s(v))$ plus 1 which is given by the lacking of node u in the case $\varrho(p(u)) \neq \varrho(u) \neq \varrho(s(v))$. On the other hand, for ϱ, this plus 1 is

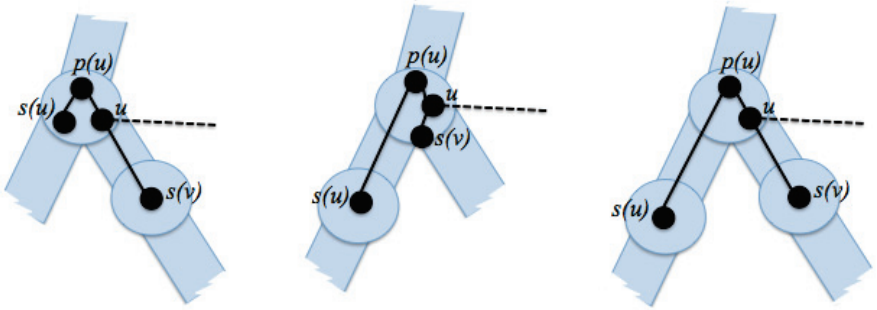

Figure 2.15: The three situation that can occur during case 1. in the proof of Lemma 11 when the event associated to $p(u)$ by ϱ is not a host switch.

given only if $\varrho(u) \neq \varrho(s(v))$ or by $\varrho(p(u)) \neq \varrho(u)$ in the cases when $p(u)$ is associated to a duplication event. It is not difficult to convince themselves that the three situations of Figure 2.15 are all the possible situations of comparability between the mappings of the nodes $p(u), u, s(u), s(v)$, that the missing plus 1 is always balanced and that the rightmost one is not parsimonious as by $c_c \leq c_d$ and $c_l > 0$ we can decrease the cost of ϱ by c_l mapping u to $s(\varrho(s(u)))$ ($\varrho(s(v))$ in the picture).

Case 2. the event associated to $p(u)$ by ϱ is a host-switch event.
We must distinguish two situations: either the host-switch edge is $(p(u), u)$ (left picture in Fig. 2.16) or it is $(p(u), s(u))$ (right picture in Fig. 2.16).

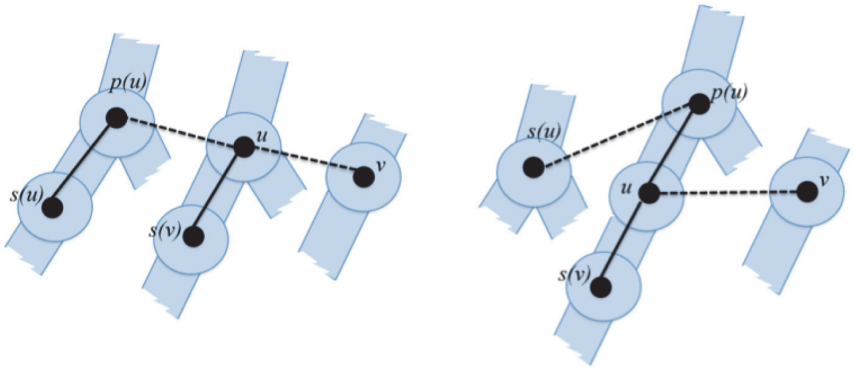

Figure 2.16: The two situations that can occur during the proof of Lemma 11 when the event associated to $p(u)$ by ϱ is a host switch.

In the first case the introduction in P^u of edge $(p(u), s(v))$ does not affect the event associated to $p(u)$ by ϱ as $\varrho(p(u)) \not\cong_H \varrho(u)$ and $\varrho^u(p(u)) \not\cong_H \varrho^u(s(v))$.

The only loss events in this case are the ones associated to edge $(u, s(v))$, but $\varrho(u) = \varrho(s(v))$ by Lemma 10.

In the second case, the event associated still does not change since it holds still $\varrho(p(u)) \not\cong_H \varrho(s(v))$. The plus 1 loss in ϱ^u due to the lack of u along the edge $(\varrho(p(u)), \varrho(s(v)))$

when $\varrho(p(u)) \neq \varrho(u) \neq \varrho(s(v))$, is payed by the plus 1 loss for being u associated to an host-switch event and $\varrho(u) \neq \varrho(s(v))$.

In all cases, the claim follows. □

We introduce now the following definition which we use in Lemma 12 to show how we can decompose an optimal reconciliation.

Definition 23. *Let $\varrho : V(P) \to V(H)$ be a reconciliation. An edge $(u, v) \in E(P)$ associated to a host switch by ϱ is said to be a* lowest host switch *if no other edge in P_u is associated to a host switch by ϱ.*

Lemma 12. *Assume that $c_l > 0$ and $c_d \geq c_c$. Let be given a lowest host switch (u, v)(where u is the parent of v) in an optimal reconciliation $\varrho \in \mathcal{R}(H, P, \varphi, C)$; then $\varrho^u \in \mathcal{R}(H, P^u, \varphi_{|V(P^u)}, C)$ and $\varrho_v \in \mathcal{R}(H, P_v, \varphi_{|V(P_v)}, C)$, i.e. ϱ^u and ϱ_v are optimal reconciliations.*

Proof. We have already mentioned that ϱ_v and ϱ^u are valid reconciliations. We will prove now separately that they are both optimal.

Reconciliation ϱ^u is optimal.
Suppose by contradiction that there exists one reconciliation $\sigma \in \mathcal{R}(H, P^u, \varphi_{|V(P^u)}, C)$ such that $cost(\sigma) < cost(\varrho^u)$.

We preliminarily show that it cannot be $\sigma(s(v)) \ncong_H \varrho(v)$. Indeed, assume instead that $\sigma(s(v)) \ncong_H \varrho(v)$; then it is possible to construct a new reconciliation ϱ' mapping P to H in the following way:
- $\varrho'(w) = \varrho(w)$ for each $w \in P_v$;
- $\varrho'(w) = \sigma(w)$ for each $w \in P^u$;
- $\varrho'(u) = \sigma(s(v))$.

It is easy to see that ϱ' is a valid reconciliation from P to H, that ϱ' assigns a host-switch event to u and that the event associated to $p(u)$ does not change w.r.t. σ. By Lemma 11 its total cost is:

$$cost(\varrho') = cost(\sigma) + c_s + cost(\varrho_v) < cost(\varrho^u) + c_s + cost(\varrho_v) = cost(\varrho)$$

in contradiction with the optimality of ϱ. Hence it should be $\sigma(s(v)) \cong_H \varrho(v)$. We show in the following that also this case does not hold. If $\sigma(s(v)) \preceq_H \varrho(v)$ then, in view of part *a.* of Fact 3 there exists a leaf $l \in L(P_{s(v)})$ such that $\varphi(l) = \sigma(l) \preceq_H \sigma(s(v)) \preceq_H \varrho(v)$. Now, since (u, v) is a lowest host switch w.r.t. ϱ, for $P_{s(v)}$ Lemma 4 holds, and $\varrho(s(v)) \succeq_H lca(s(v))) = lca(\varphi(L(P_{s(v)})))$, so in particular $\varrho(s(v)) \succeq_H \varphi(l)$. We have reached a contradiction then reminding that (u, v) is a host switch in ϱ, hence $\varrho(v) \ncong_H \varrho(s(v))$, but $\varrho(v) \succeq_H \varphi(l)$ and $\varrho(s(v)) \succeq_H \varphi(l)$.

Assume now, vice-versa, that $\varrho(v) \preceq_H \sigma(s(v))$; in view of Lemma 9, it holds that $\sigma(s(v)) \preceq_H lca(s(v))$.

then $\varrho(v) \preceq_H \sigma(s(v)) \preceq lca(s(v)) \preceq_H \varrho(s(v))$ thanks to Lemma 9, Fact 4 and the hypothesis that (u, v) is a lowest host switch for ϱ; this is a contradiction since (u, v) is a host switch in ϱ and hence $\varrho(v) \not\cong \varrho(s(v))$.

Reconciliation ϱ_v is optimal.
Again, we proceed by contradiction assuming that there exists $\tau \in \mathcal{R}(H, P_v, \varphi_{|V(P_v)}, C)$ such that $cost(\tau) < cost(\varrho_v)$.

By Lemma 9, Fact 4 and that (u, v) is a lowest host switch we have $\tau(v) \preceq_H lca(v) = \varrho(v) \not\cong \varrho(u) \succeq \varrho(s(v))$, hence $\tau(v) \not\cong_H \varrho(s(v))$ and then we can construct a more parsimonious ϱ' as:
- $\varrho'(w) = \varrho(w)$ for each $w \in P^u$;
- $\varrho'(w) = \tau(w)$ for each $w \in P_v$;
- $\varrho'(u) = \varrho(s(v))$.
Hence $\varrho'(u) \not\cong_H \varrho'(v)$ and by Lemma 11 we have:

$$cost(\varrho') = cost(\varrho^u) + c_s + cost(\tau) < cost(\varrho^u) + c_s + cost(\varrho_v) = cost(\varrho)$$

against the optimalilty of ϱ.

□

2.6 $\sim_1$ Equivalence Class

Let $\varrho \in \mathcal{R}(H, P, \varphi, C)$ and let $u \in V(P) \setminus \{root(P))\}$ such that $(u, v) \in \widetilde{\Theta}$ (where $p(v) = u$). Furthermore let us assume for the moment that $p(u) \notin \Theta(\varrho)$.

Consider the nodes in $path_H(\varrho(p(u)), \varrho(s(v)))$, the node u could have been mapped onto any of these nodes without changing the cost of the reconciliation, as proved by the following result:

Lemma 13. *Given two optimal reconciliations $\varrho, \sigma \in \mathcal{R}(H, P, \varphi, C)$ with $c_l > 0$ and $c_c \leq c_d$, such that:*

- *there exists an edge $(u, v) \in \widetilde{\Theta}(\varrho) \cap \widetilde{\Theta}(\sigma)$ (where $p(v) = u$) and,*
- *$\varrho(w) = \sigma(w)$ for each $w \neq u$, and*
- *$p(u)$ is not associated to an host-switch event.*

If $\varrho \neq \sigma$, i.e. *$\varrho(u)$ and $\sigma(u)$ are mapped onto two different nodes of $path_H(\varrho(p(u)), \varrho(s(v))))$, the costs associated to ϱ and σ are the same.*

Proof. The hypotheses of Lemma 11 hold, so we can decompose the cost of ϱ and σ as:

$$cost(\varrho) = cost(\varrho^u) + c_t + cost(\varrho_v) \text{ and } cost(\sigma) = cost(\sigma^u) + c_t + cost(\sigma_v).$$

In view of the hypothesis of equality between ϱ and σ outside u, $\varrho^u = \sigma^u$ and $\varrho_v = \sigma_v$. Furthermore, the event associate to $p(u)$ by ϱ and σ is the same and observe that it could have been the only one to change. This is enough to prove that $cost(\varrho) = cost(\sigma)$. □

In the setting of previous lemma, observe that we can relax the assumptions letting $p(u)$ be associated with any event but, if it were $(p(u), u) \in \widetilde{\Theta}$ (hence $p(u) \in \Theta$) then, by Lemma 10, $path_H(\varrho(p(u)), \varrho(s(v)))) = \{\varrho(s(v))\}$ and the result may be trivial.

Hence, we assume, in this case, that it is $(p(u), s(u))$ the host-switch edge, *i.e.* we should relax the assumption assuming $\varrho(u) \cong_H \varrho(p(u))$.

However $p(u)$, which now is allowed to be a host-switch event, may play the role of u in Lemma 13 as well.

We introduce the following definition:

Definition 24. *Let $\varrho : V(P) \to V(H)$ be a reconciliation and let $u, v \in V(P)$ such that $p(v) = u$ and $(u, v) \in \widetilde{\Theta}$. We denote with $\alpha_\varrho(u)$ (or simply $\alpha(u)$ if ϱ can be omitted) the* least ancestor of u not associated to a host-switch event *such that $\varrho(\alpha(u)) \succeq_H \varrho(u)$. Furthermore we denote with $\omega_\varrho(u)$ (or simply $\omega(u)$ if ϱ can be omitted) the* greatest descendent of u not associated to a host switch *such that $\varrho(\omega(u)) \preceq_H \varrho(u)$.*

Then $path_H(\varrho(\alpha(u)), \varrho(\omega(u)))$ is called the slide of u w.r.t. ϱ. *If the slide of w, for $w \in V(P)$, is the same of u, then we say that u* and w share the same slide w.r.t ϱ.

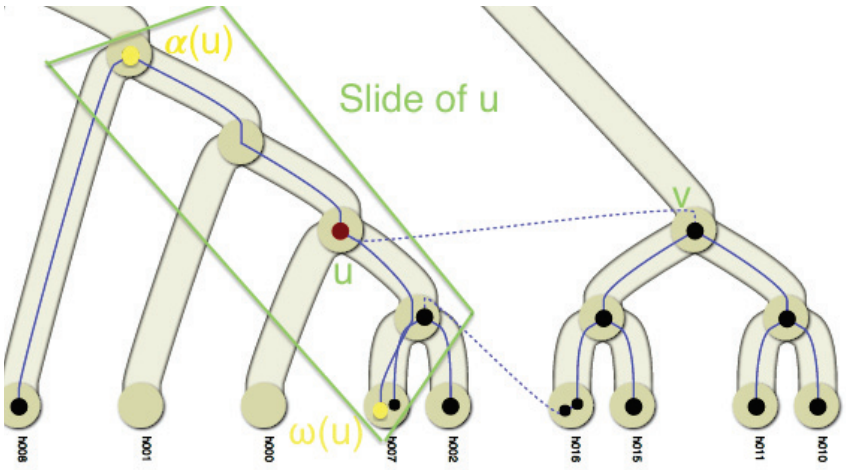

Figure 2.17: A node u, its slide determined by $\alpha(u)$ and $\omega(u)$.

Observe that while $\alpha(u)$ may not exist, $\omega(u)$ always exists as at least a leaf node of P is not associated to a host-switch event (see Fact 3).
Trivial examples are when all the ancestors of u are associated to host-switch events up to the root of P or when $(p(u), u) \in \widetilde{\Theta}$.

We can then extend Lemma 13 as follows:

Proposition 10. *Let $\varrho, \sigma \in \mathcal{R}(H, P, \varphi, c)$ with $c_l > 0$ and $c_c \leq c_d$ and let $u \in V(P)$ be such that $\alpha_\varrho(u) = \alpha_\sigma(u)$ exist and are both associated to the same event, $\omega_\varrho(u) = \omega_\sigma(u)$ and $\varrho(w) = \sigma(w)$ for all the node $w \notin path_P(\alpha(u), \omega(u)) \setminus \{\alpha(u), \omega(u)\}$.*
Then the costs associated to ϱ and σ are the same.

Proof. Primarily observe that ϱ and σ differ only on the nodes in $\pi = path_P(\alpha(u), \omega(u)) \setminus \{\alpha(u), \omega(u)\}$ and that, by definition of $\alpha(\cdot)$ and $\omega(\cdot)$, for both the reconciliations these nodes are all associated to host-switch events mapped into the nodes of $path_H(\varrho(\alpha(u)), \varrho(\omega(u))) = path_H(\sigma(\alpha(u)), \sigma(\omega(u)))$.

Let $\pi = \{p_1, \ldots, p_n\}$ with $p_1 \succeq_P \cdots \succeq_P p_n$ and, for any $1 \leq i \leq n$, let us denote with ϱ_i and σ_i the nodes $\varrho(p_i)$ and $\sigma(p_i)$ respectively. Then from the definition of $\alpha(\cdot)$ and $\omega(\cdot)$ we have that for any $i \leq j$ it holds $\varrho_i \succeq_H \varrho_j$ and $\sigma_i \succeq_H \sigma_j$.

Without lack of generality let us assume that $\varrho_1 \succeq_H \sigma_1$. Now let us identify ϱ by the tuple $(\varrho_1, \ldots, \varrho_n)$ and σ by the tuple $(\sigma_1, \ldots, \sigma_n)$ which represent the different mapping of the nodes $(p_1, \ldots, p_n)$ in H. In the same way we can define for $1 \leq i < n$ the tuple $(\varrho_1, \ldots, \varrho_i, \sigma_{i+1}, \ldots, \sigma_n)$ and introduce the associated reconciliation τ_i. Letting $\tau_0 := \sigma$ and with $\tau_n := \varrho$, it is clear that by Lemma 11 and arguing as in the proof of Lemma 13 we obtain $cost(\tau_i) = cost(\tau_{i+1})$ for $1 \leq i < n$ which conclude the proof. □

The previous result leads us to consider as equivalent all the reconciliations that, for each host switch (u, v), map u on a different node of the slide of u.

Definition 25. *Two reconciliations $\varrho, \sigma : V(P) \to V(H)$ are in the same $\sim_1$-equivalence class iff the following holds:*

1. *For each node in $V(P)$ they assign the same event.*
2. *For each $u \in V(P)$ not associated to an host-switch event or such that $\alpha(u)$ does not exist, $\varrho(u) = \sigma(u)$.*
3. *For each $u \in V(P)$ associated to an host-switch event such that $\alpha_\varrho(u), \alpha_\sigma(u)$ exists, both $\varrho(u)$ and $\sigma(u)$ lie on the slide of u w.r.t ϱ (which coincide with the slide of u w.r.t σ).*

If these three conditions hold, then we write $\varrho \sim_1 \sigma$.

Intuitively, $\sim_1$-equivalence class collects all the reconciliations which agree on all the nodes of $V(P)$ except the ones which have a slide of length grater than 1.

If there are more nodes sharing the same slide, then they can be mapped to any node of the slide as far as their image respect their original order $\succeq_P$ (so as to have a valid reconciliation).

More specifically, for each reconciliation which has a slide of length l shared by n nodes, there are at least $\binom{n-l-1}{l-1}$ other reconcilations with the same cost and that differ only on these n nodes.

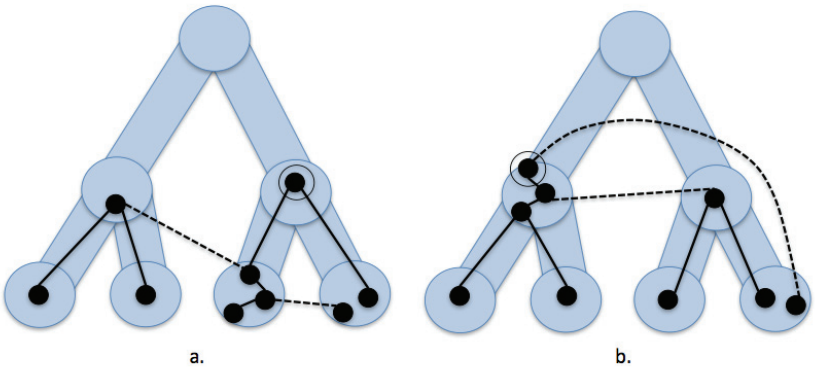

Figure 2.18: Two reconciliations with the same event vector that nevertheless are rather different.

Finally, taken a reconciliation with k slides each one of length l_h and each one shared by n_k nodes, the number of reconciliation in its $\sim_1$-equivalence class is $\prod_{i=0}^{k} \binom{n_k - l_k - 1}{l_k - 1}$.

The following fact claims an interesting property of equivalent reconciliations w.r.t. relation $\sim_1$ and the classification of an optimal reconciliation by its event vector E_ϱ:

Fact 5. *Let $\varrho : V(P) \to V(H)$ be a reconciliation. Then all the reconciliations in $[\varrho]_{\sim_1}$,* i.e. *all the reconciliations in the $\sim_1$-equivalence class of ϱ, have the same cost and the same event vector E_ϱ.*

Observe that by $\sim_1$-equivalence class the partition of $\mathcal{R}(H, P, \lambda, c)$ is finer than the one induced by the event vector and now we may distinguish between reconciliations as the two in Figure 2.18.

However we would like to know the size of this partition: here after and in Section 2.9 we explain how to compute this size and we show some experimental results.

Counting $\sim_1$-Equivalence Classes

In this section we explain how to count the number of $\sim_1$-equivalence classes by enumerating one representative reconciliation for each of them. Observe that, by definition of $\sim_1$, the only difference between reconciliations in the same class are the nodes $u \in V(P)$ for which $\alpha(u)$ exists and if we can impose a univoque mapping for them, we identify a representative for each class. With the aid of the next Lemma we show that we can enumerate all these representative reconciliations if they are characterize by the following property:

$$\text{If } (u, v) \in \widetilde{\Theta}(\varrho^*) \text{ with } p(v) = u, \text{ then } \varrho^*(u) = \varrho^*(s(v)).$$

This is how we collected the data obtained from the experimental results displayed in Section 2.6.

Lemma 14. *Given a reconciliation $\varrho \in \mathcal{R}(H, S, \varphi, C)$ with $c_l > 0$, let $u_0, u_1, \ldots, u_n$ be host-switch nodes such that $\varrho(u_0) \succeq_H \varrho(u_1) \succeq_H \cdots \succeq_H \varrho(u_n) \succeq_H \varrho(u)$ where u is not a*

host-switch node. Furthermore let us assume that either $(p(u_0), u_0)$ *is a host-switch edge or* u_0 *be the root of* P*. Then* $\varrho(u_i) = \varrho(u)$ *for* $0 \leq i \leq n$*.*

Proof. Suppose by absurd that there exists a index j such that $\varrho(u_j) \prec_H \varrho(u_0)$. We can then define a new reconciliation ϱ' which agrees on ϱ on all the nodes but $u_0, u_1, \ldots, u_n$ for which we define $\varrho'(u_i) := \varrho(u)$ for $0 \leq i \leq n$. Observe that ϱ' and ϱ induce the same events on the nodes of P as we only change the mapping of u_i from $\varrho(u_i)$ to $\varrho(u)$ but $\varrho(s(u_i)) \not\cong_H \varrho(u_i) \succeq \varrho(u)$ and so all the nodes u_i preserve their host-switch event status together with the node $p(u_0)$ which either do not exist or for it still holds $(p(u_0), (u_0)) \in \widetilde{\Theta}(\varrho')$.

However, while in ϱ, we pay the cost of at least c_l, being $\varrho(u_j) \neq \varrho(u_{j-i}) = \varrho(u_0)$, in ϱ' we do not. Then we reach a contradiction with the optimality of ϱ. □

Since each reconciliation belongs to an equivalence class and each equivalence class has an unique ϱ^* reconciliation, to count the equivalence classes we can simply enumerate only these ϱ^* reconciliations which satisfy the mentioned property. In order to do so, we can simply impose (in EUCALYPT at time of filling the dynamic matrices $D(p, h)$ and $\widetilde{D}(p, h)$, see Section 2.4) that each node $u \in \Theta$ with $(u, v) \in \widetilde{\Theta}$ is mapped to the same node $\varrho(s(v)) \in V(H)$ of its comparable child $s(v)$ (*i.e.* we are actually mapping u to $\omega(u)$, as this son can be associated to an host-switch event as well).

Observe that, for a node $u \in \Theta$, $(u, v) \in \widetilde{\Theta}$, it may or may not exist the node $\alpha(u)$. In the latter case, either all its ancestor are associated to host-switch events up to the root of P (if there are any) or along this path from u to the root of P there is an edge mapped to a host-switch edge. For these nodes Lemma 14 grants that there is no harm in imposing that their mappings coincide with the ones of the comparable child in H. All the other nodes are not affected by this imposition as by Proposition 10 the cost of ϱ^* is the same of ϱ.

We collect this reasoning in the following corollary:

Corollary 5. *Let* $c_l > 0$ *and* $c_c \leq c_d$*. To count and enumerate one optimal reconciliation in* $\mathcal{R}(H, S, \varphi, C)$ *for each* $\sim_1$*-equivalence class it suffice to impose that the mappings of any node* $u = p(v)$ *such that* $(u, v) \in \widetilde{\Theta}$ *has to coincide with the mapping of* $s(v)$*.*

While the reconciliation ϱ^* is a good reconciliation to enumerate or count the number of $\sim_1$-equivalence classes, it is not the best representative of the class. Indeed, we introduce here after another property to identify a reconciliation $\varrho \in [\varrho]_{\sim_1}$ whose knowledge permits to reconstruct any reconciliation in the equivalence class:

Definition 26. *Let* $\varrho \in \mathcal{R}(H, P, \varphi, c)$*. We call* ϱ^+ the canonical reconciliation in $[\varrho]_{\sim_1}$ *if it satisfies the following condition:*

$$\textit{for each } \sigma \in [\varrho]_{\sim_1} \textit{ and } u \in V(P) \textit{ it holds } \varrho^+(u) \succeq_H \sigma(u).$$

The motivation for this definition is that, among all the reconciliations in $[\varrho]_{\sim_1}$, the nodes on which they may differ are the nodes $u \in V(P)$ for which there exists a slide, *i.e.* it is well defined $path_H(\varrho(\alpha(u)), \varrho(\omega(u)))$. In this case, we know that it is always possible to find a reconciliation $\varrho^* \in [\varrho]_{\sim_1}$ for which $\varrho^*(u) = \varrho^*(\omega(u))$. On the other hand, it is not always possible to find a reconciliation $\varrho^+ \in [\varrho]_{\sim_1}$ such that $\varrho^+(u) = \varrho^+(\alpha(u))$. (for instance when $\alpha(u)$ is associated to a cospeciation event and $c_c < c_d$).

In this sense, the canonical reconciliation permits to know, independently of the cost vector, all the valid mappings of a node u along its slide which are $path_H(\varrho^+(u), \varrho^+(\omega(u)))$.

We conclude this section by observing that a $\sim_1$-equivalence class may contain both cyclic and acyclic solutions as shown in Figure 2.19.

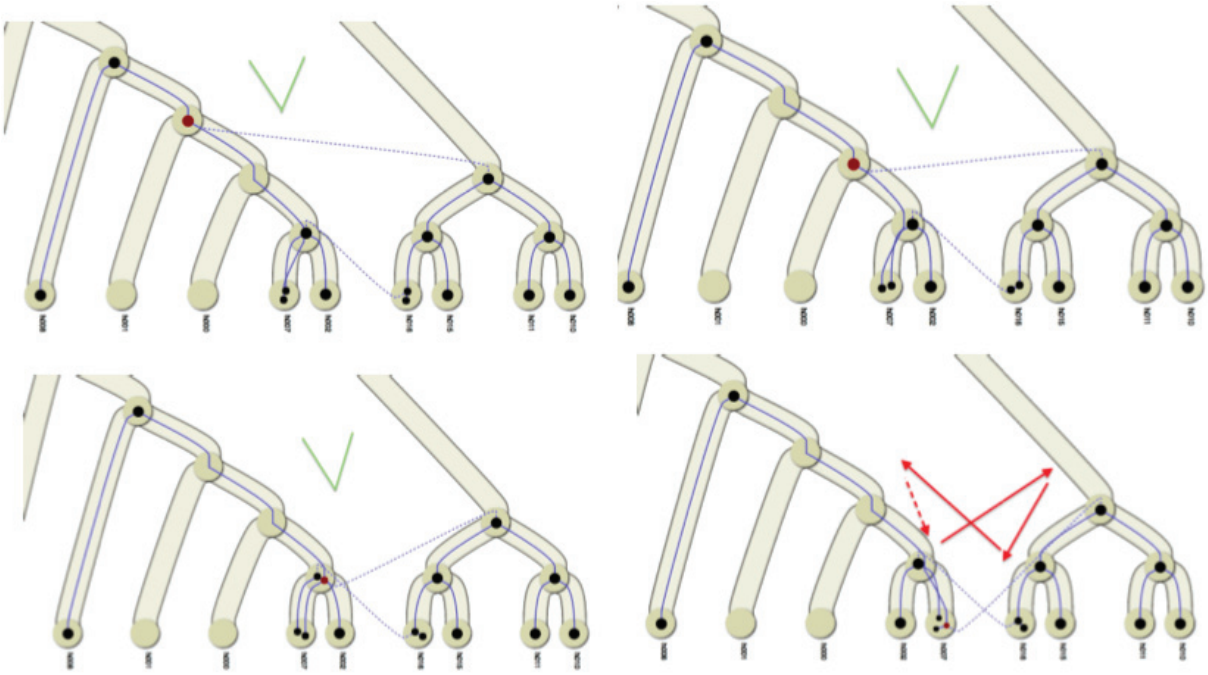

Figure 2.19: Four reconciliations in the same $\sim_1$-equivalence class. The bottom right one is cyclic and it is the reconciliation ϱ^* of the class.

2.7 $\sim_2$ Equivalence Class

We now propose another equivalence relation between optimal reconciliations. This equivalence class permits to the reconciliations in the same class to differ in some nodes associated to host-switch events similarly to the previous $\sim_1$ equivalence class but this time we focus our attention to host-switch nodes for which $\alpha(\cdot)$ is not defined.

Assume there are two siblings v and w in P that are mapped by φ on two incomparable nodes $\varphi(v)$ and $\varphi(w)$ in H. If host switches are allowed, any reconciliation can equivalently map $p = p(v) = p(w)$ on a node that is either comparable with $\varphi(v)$ and incomparable

with $\varphi(w)$ or vice-versa. All these solutions are equally feasible, and there is no reason to distinguish them (see Figure 2.20).

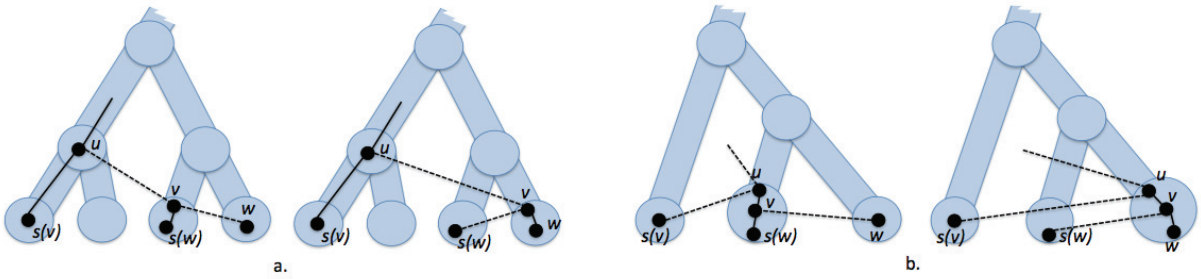

Figure 2.20: Alternative mappings of the node u with same cost.

Inspired by Lemma 14, we introduce the following definitions to identify the nodes which are allowed to differ in this new $\sim_2$-equivalence class for optimal reconciliations.

Definition 27. *Let $\varrho : V(P) \to V(H)$ be a reconciliation and let $u \in \Theta$. Let $z \in \Theta \setminus \{root(P), u\}$ be such that:*

- *all the nodes along the path between z and u are in Θ,*
- *all the edges between consecutive nodes in $path_P(z, u)$ are mapped by ϱ either to reflexive edges or host-switch edges and, among these edges,*
- *the edge incident to z has to be mapped to a non-reflexive host-switch edge.*

If z is the $\succeq_P$-greatest node in $V(P)$ satisfying these conditions, we call it the head of u in ϱ *and denote it with $\chi(u)$.*

Observe that trivially not all the nodes in Θ have a head node, as for instance: a node u for which $\alpha(u)$ exists (see Definition 24) or if all the ancestors of u up to the root of P are in Θ, then by definition $\chi(h)$ does not exist.

Definition 28. *Let $\varrho : V(P) \to V(H)$ be a reconciliation. A* bundle node u of ϱ *is a node in $V(P) \setminus \{root(P)\}$ such that:*

1. *$u, p(u) \in \Theta$ (*i.e. *ϱ assign to u and to its father $p(u)$ an host-switch event),*
2. *its mapping by ϱ in H coincide with one of the mapping of its children,*
3. *there exists $\chi(u)$, the head of u in $\varrho(u)$.*

Observe that Lemma 14 claims that under the setting $c_l > 0$, we can drop point 2. in the previous definition; in particular, if u is a bundle node, a node v, with $p(v) = u$, is a bundle node if and only if $v \in \Theta$.

Definition 29. *Two reconciliations $\varrho, \sigma : V(P) \to V(H)$ are in the same $\sim_2$-equivalence class iff the following holds:*

1. *They induce the same events on the nodes,*
2. *They have the same bundle nodes,*
3. *They agree on the nodes which are not associated to host-switch event and to non-bundle nodes.*

If these three conditions hold, then we write $\varrho \sim_2 \sigma$.

Directly from point 1. of the definition we have the following:

Fact 6. *Let* $\varrho : V(P) \to V(H)$ *be a reconciliation. Then all the reconciliations in* $[\varrho]_{\sim_2}$ *assign the same events to the nodes in* $V(P)$*. In particular all the reconciliations have the same cost and the same event vector* E_ϱ*.*

Observe that for a bundle node u_0 and under the hypothesis $c_l > 0$, by Lemma 14 the mapping of u must coincide with the one of $\omega(u)$:

Fact 7. *Let* $\varrho \in \mathcal{R}(H, P, C, \varphi)$ *and let* $c_l > 0$*. For each bundle node* u *of* ϱ *in* $V(P)$ *it holds* $\varrho(u) = \varrho(\omega(u))$*.*

In the following we employ this last fact to count all the different $\sim_2$-equivalence classes as the bundle nodes are the only one allowed to differ among $\sim_2$-equivalent reconciliations. The following concept describes an invariant of the reconciliations in the same class:

Definition 30. *Let* u *be a bundle node of* ϱ *(see Figure 2.21) and let* $f = \omega_\varrho(u)$ *(*i.e *the oldest non-bundle node of* ϱ *descendant of* u*).*
Then f *is said a* frontier-node of u w.r.t. ϱ *and also a* frontier-node *of* ϱ*.*
Furthermore, the set of pairs $(f, \varrho(f))$ *of frontier-nodes* f *of* u *w.r.t.* ϱ *is called the* frontier of ϱ underneath u *and denoted as* $\Phi(u, \varrho)$*.*
The set of all pairs $(f, \varrho(f))$*, for* f *a frontier-node of* ϱ*, is said the* frontier *of* ϱ *and denoted as* $\Phi(\varrho)$*.*
We will denote the set of all these frontiers by $\Phi(\mathcal{R}) := \bigcup_{\varrho \in \mathcal{R}(H,P,\varphi,C)} \Phi(\varrho)$*.*

Fact 7 implies that bundle nodes has to be mapped to a node of the frontier.

Computing $\sim_2$-Equivalence Classes

In this section we explain how we can count the number of $\sim_2$ equivalence classes of a set of optimal reconciliations. We use this method to show (in Table 2.3-2.9.1) the potentialities of enumerating only a representative for each $\sim_2$-equivalence class. For simplicity reasons, we consider only the counting of the number of these representatives, although the ideas presented in this section may be applied also to enumerate them.

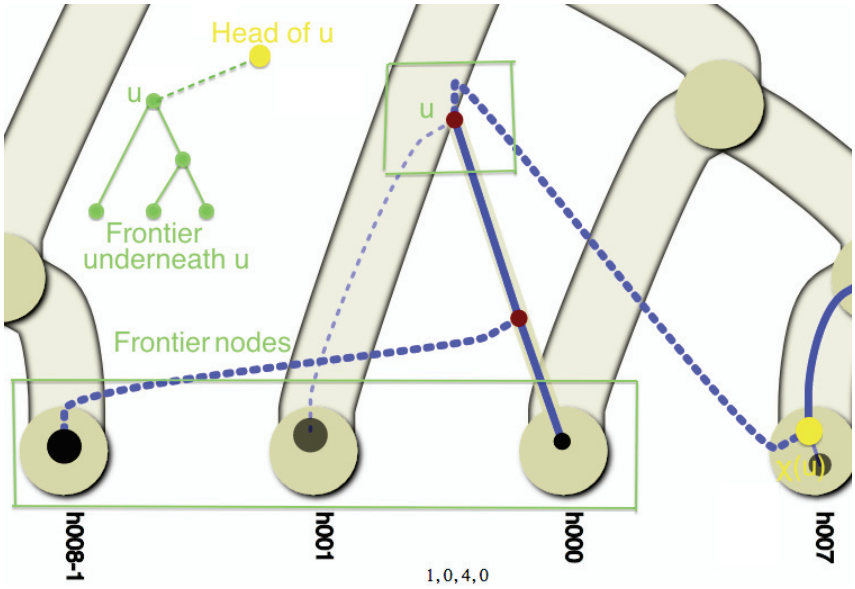

Figure 2.21: A bundle node u of a reconciliation ϱ, the *frontier of* ϱ *underneath* u , $\chi(u)$ (the *head of* u *in* ϱ), a tree in the forest .

Let us start considering the forest $\Gamma_P(\varrho)$ of subtrees of P induced by the bundle nodes of ϱ and let us denote with $\Gamma_P(\mathcal{R})$ the union of all these forests.
Furthermore, taken a tree T in the forest $\Gamma_P(\varrho)$ and denoted with

$$\Phi(\varrho, T) := \{(f, \varrho(f)) \in \Phi(\varrho) : f \in V(T) \text{ is a frontier-node of } \varrho\},$$

it holds trivially that:

$$\Phi(\varrho) = \bigcup_{T \in \Gamma_P(\varrho)} \Phi(\varrho, T)$$

It is clear that all the nodes in a tree T have the same head $\chi(T) \in V(P)$ and so we call $\chi(T)$ the *head of* T *in* ϱ. we extend the concept of frontier to the non-bundle node $\chi(T)$ by calling it also *the frontier underneath* $\chi(T)$ *in* ϱ and the *frontier of* T *in* ϱ.

The following fact stems directly from the definition of $\sim_2$ equivalence:

Fact 8. *Let* $\varrho, \sigma : V(P) \to V(H)$ *be two* $\sim_2$*-equivalent reconciliations, then:*

1. $\Gamma_P(\varrho) = \Gamma_P(\sigma)$,
2. $\varrho_{P \setminus \Gamma_P(\varrho)} = \sigma_{P \setminus \Gamma_P(\varrho)}$,
3. *if* $\varrho \neq \sigma$ *then* $\varrho_{\Gamma_P(\varrho)} \neq \sigma_{\Gamma_P(\varrho)}$,
4. $\Phi(\varrho, T) = \Phi(\sigma, T)$ *for each tree* T *in* $\Gamma_P(\varrho)$*, in particular* $\Phi(\varrho) = \Phi(\sigma)$.

We will prove here after how it is possible to compute the number of $\sim_2$-equivalence classes for a given set of optimal reconciliations $\mathcal{R}(H, P, \varphi, C)$ by exploiting point 3. of

Fact 8. Indeed, if identifies a bundle node and we impose only one mapping for it then we will count only one optimal reconciliation in each equivalence class $[\varrho]_{\sim_2}$.

It is clear, then, the importance of identifying all the trees in $\Gamma_P(\mathcal{R})$ and for each tree T of these, it is important to identify all its possible frontiers $\Phi(\varrho, T)$ and count just one alternative mapping which realize that frontier.

We have seen in Section 2.4, that the partial solutions are build bottom up in EUCALYPT by subsequent choices of the optimal mappings of the children of the current node. Assume the notation $D(p, h)$ of [25] and Section 2.4 to denote the information stored on the most parsimonious mappings of the node $p \in V(P)$ into the node $h \in V(H)$. Recall that this information always comes in pairs (possibly many and obviously with the same cost, see Figure 2.13 and Figure2.22) of optimal mappings for the children p_1, p_2 of p.

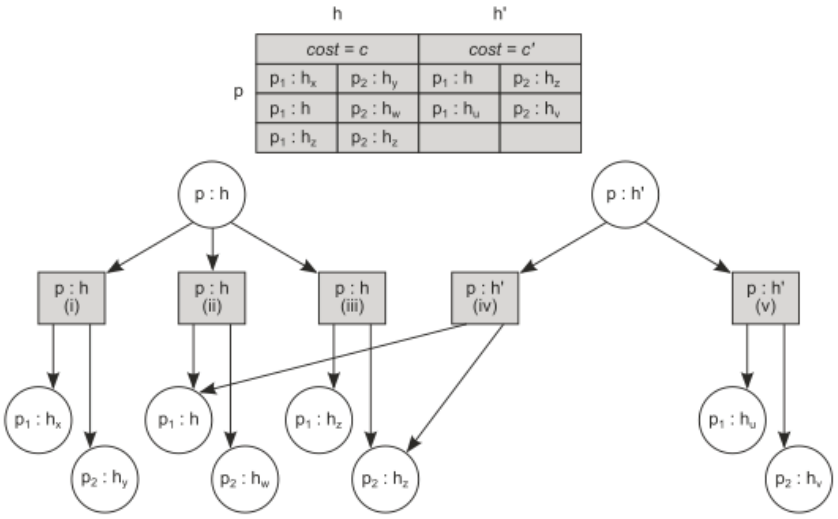

Figure 2.22: Figure 3 in [25]: An example of two cells $D(p, h)$ and $D(p, h')$ (or $\widetilde{D}(p, h)$ and $\widetilde{D}(p, h')$).

To compute the number of $\sim_2$-equivalence classes, we modify the code of EUCALYPT to perform two actions: firstly we enrich the content of $D(p, h)$ and $\widetilde{D}(p, h)$ with the information of the frontiers underneath p from the frontiers underneath p_1, p_2, in this sense we "lift" the frontier. This information of the frontiers is stored in a dictionary which associate to each frontier the number of $\sim_2$-subsolutions at (p, h) with this frontier, *i.e.* the $\sim_2$-equivalence classes in $\underline{\mathcal{R}}(H, P_p, \varphi, C, h)$. The computation is done recursively for all the nodes of P correctly initializing the non host-switch nodes.

Secondly we modify EUCALYPT to identify the head nodes $\chi(T)$ and correctly compute the number of $\sim_2$-equivalence classes in $\mathcal{R}(H, P, \varphi, C)$. Observe that, by Lemma 14 and assuming $c_l > 0$, in Definition 29 we may omit item 2. as it is implied by 1.
With this setting in mind, it is clear that the nodes feasible of being bundle nodes are all the nodes u such that $u, p(u) \in \Theta$ and such that $\chi(u)$ exists.

For the moment we postpone the identifications of the head of u (hence the identification of the roots of the trees in $\Gamma_P(\varrho)$) and focus on "lifting" the information of the frontiers (from the frontier nodes to the head of the bundle nodes).

Augmenting the Information in $D(p,h)$ and $\widetilde{D}(p,h)$

More precisely, to each data structure $D(p,h)$ (or $\widetilde{D}(p,h)$), we add a counter $eqClasses$ for the number of different $\sim_2$-subsolutions. This counter is filled combining the information on the frontiers and combining the counters for all the optima pairs of mapping of the children stored in $D(p_1,h_1)$, $D(p_2,h_2)$ and their values in $\widetilde{D}$, where p_1,p_2 are children nodes of p and h_1,h_2 are some descendant nodes of h in H.
Just to give an example here and leave the full explanation to Lemma 15 and Lemma 16: if $D(p,h)$ is a simple solution and the associated event is not an host switch, then there is actually no frontier underneath p, but, for initialization purposes (in the case p will a fronter node for $p(p)$) we assign to $eqClasses$ the value $D(p_1,h_1).eqClasses \cdot D(p_2,h_2).eqClasses$ as every $\sim_2$-subsolution is build by combining a $\sim_2$-subsolution of $D(p_1,h_1)$ with one of $D(p_2,h_2)$.

Furthermore, to each cell in $D(p,h)$ (or $\widetilde{D}(p,h)$), we add a dictionary with entries of the type $frontier : number\_of\_associated\_ \sim_2 \_eq\_classes$ with the double purpose of lifting the information on the frontiers and, at the same time, computing the number at (p,h) of $\sim_2$-subsolutions ($eqClasses$ in the pseudo-code) with the given frontier (see item 4. of Fact 8).

Observe now that, while the verification of item 1. and 2. in Definition 28 can be done while computing the current values for the cell $D(p,h)$, item 3. would require at least $O(height(P_p))$ time to recompute the values for $D(p',h').eqClasses$ with $p' \prec_P p$ and $h' \in V(H)$ in the case p is a candidate head node for p'.

For this reason, we decide to lift the information of both the scenarios of the current mapping: for the *non-bundle* scenario, $eqClasses$ and $frontier$ store the values under the hypothesis that p is not a bundle node (*i.e.* we will not find an head node for it). For the *bundle scenario*, $eqClasses\_b$ and $frontiers\_b$ store the corresponding values in the case we will find an head for p, hence it will be a bundle node.
The two alternative scenarios and their values will be "lifted" and carried on together (in the filling $D(p,h)$ by post order transversals of the nodes of P and H) up to the time we find a candidate head fulfilling item 3.c and Lemma 14. In that case we fall in the "non-bundle" scenario finding that we lack item 3.a and so re-initializing the values of $eqClasses\_b$ and $frontiers\_b$ to the same values of $eqClasses$ and $frontier$ respectively for initialization purposes (see Algorithm 6). Notice that, at the end, if we do not confirm that we are in a bundle node scenario we keep the non bundle one as we have failed to find an head (we are in the case where all the ancestors are comparable nodes associated to host switch).

Finally we extend $D(p,h)$ (and $\widetilde{D}(p,h)$) by adding another dictionary $seenMappings$ whose key set are still frontiers and whose values set are list of (h',p') pairs where $p' \in V(P)$ and $h' \in V(H)$. This dictionary is useful only when $D(p,h)$ (or $\widetilde{D}(p,h)$) is a multiple solution to take notes of all the necessary information on the simple solutions which lift a frontier to do not count them many times. In particular we are interested on the simple solution not associated to an host-switch event and in particular to the node h' on which the parasite p is mapped and the non jumping child $njc(D(p,h')) = p'$ in that simple solution (see the proof of Lemma 16 to justify this choice).

Lifting the Frontier $\Phi(T,\varrho)$ and Identifying the Head $\chi(T)$

In the following we discuss only the filling of matrix D as the filling of matrix $\widetilde{D}$ depends on the ones of D (see Section 2.4) and it undergoes the same computations of multiple solutions with just a wider list of optimal candidates (the ones coming from $D(p,h_i)$ for $h_i \preceq_H h$) than its corresponding $D(p,h)$.

The only remark is on the lifting of the frontiers on a multiple solution where underline that in D p is always mapped to the same node h in all the simple solutions while in $\widetilde{D}$ do not. We will recall this fact when it is worthy.

At first we show how to compute the values $eqClass$, $eqClasses\_b$, $frontier$ and $frontiers\_b$ and for $D(p,h)$ by the function $ComputeSimpleFrontiers(D(p,h))$ in the case $D(p,h)$ is a simple solution. This function call the function $ComputeEqClasses(D(p,h))$ (Algorithm 5) which compute the initialization values for $eqClasses$ and $eqClasses\_b$, in a simple solution, by multiplying the corresponding values for the left and right children mapping and, in a multiple solution, summing up all the corresponding values of all the pointed simple solutions.

Lemma 15. *The function* `computeSimpleFrontiers`$(D(p,h))$*, computes correctly the values* $D(p,h).eqClasses$*,* $D(p,h).frontiers$ *and the corresponding values of the "bundle scenario"* $D(p,h).eqClasses\_b$*,* $D(p,h).frontiers\_b$ *for* p,h *leaves of* P *and* H *respectively.*
Assume furthermore that the values in $D(p',h')$ *are correctly computed in the case* $D(p',h')$ *is a multiple solution. Then the function* `computeSimpleFrontiers`$(D(p,h))$*, computes correctly the value* $D(p,h).eqClasses$ *and the* $D(p,h).frontierstogether$ *with their corresponding "bundle scenario" values* $D(p,h).eqClasses\_b$*,* $D(p,h).frontiers\_b$ *when* $D(p,h)$ *is a simple solution.*

Proof. Let us analyze case by case the switch statement assuming inductively that are well computed the values $eqClasses, frontiers, eqClasses\_b, frontiers\_b$ for $D(p_1,\cdot)$ and $D(p_2,\cdot)$ where p_1,p_2 of p; the base of the induction is when $D(p,h)$ with p,h are leaves, but this is trivial as there is only one subsolution hence one $\sim_2$-subsolution (lines 6,7 and lines 17-23). The $frontier$ and $frontier\_b$ dictionaries are set for initialization

Algorithm 5: Compute Equivalence Classes

```
1  Function computeEqClasses(D(p,h))
2      eqClasses ← 0;
3      eqClasses_b ← 0;
4      switch D(p.h).getType() do
5          case MULTIPLE_SUBSOLUTIONS:
6              for simple_solution in D(p,h).getChildrenOptimalMappings() do
7                  eqClasses += simple_solution.getEqClasses();
8                  eqClasses_b += simple_solution.getEqClassesB();
9          case SIMPLE_SUBSOLUTION:
10             if D(p,h).getEvent() != LEAF then
                   // p_1, p_2 are the children of p in P
11                 {(p_1,h_1),(p_2,h_2)} ← D(p,h).getChildrenOptimalMappings();
12                 eqClasses ← D(p_1,h_1).eqClasses * D(p_2,h_2).eqClasses;
13                 if D(p,h).getEvent() == HOSTSWITCH then
14                     eqClasses_b ← D(p_1,h_1).eqClasses_b * D(p_2,h_2).eqClasses_b;
15                 else
16                     eqClasses_b = eqClasses
17     D(p,h).eqClasses ← eqClasses;
18     D(p,h).eqClasses_b ← eqClasses_b;
19     return D(p,h).eqClasses;
```

Algorithm 6: Compute Simple Frontiers

```
1  Function computeSimpleFrontiers (D(p,h))
2      f ← ∅;
3      f_b ← ∅;
4      switch D(p,h).getEvent() do
5          case LEAF:
6              f.add({(p,h) : 1});
7              f_b.add({(p,h) : 1});
8          case HOSTSWITCH:
               // Let p_1 be the jumping child
9              Let h_1 ≇ h;
10             for frontier f_1 of D(p_1,h_1) do
11                 for frontier f_2 of D(p_2,h_2) do
12                     f.add({{f_1 ∪ f_2} : D(p_1,h_1).val_b(f_1) * D(p_2,h_2).val(f_2)});
13                     f_b.add({{f_1 ∪ f_2} : D(p_1,h_1).val_b(f_1) * D(p_2,h_2).val_b(f_2)});
14         case DUPLICATION,COSPECIATION:
15             f.add({(p,h) : ComputeEqClasses(D(p,h))});
16             f_b.add({(p,h) : ComputeEqClasses(D(p,h))});
17     D(p,h).setFrontiers(f);
18     D(p,h).setFrontiersB(f_b);
19     D(p,h).eqClasses ← 0;
20     D(p,h).eqClasses_b ← 0;
21     for frontier x in f do
           /* frontiers and frontiers_b contain the same set of keys but possibly
              different values if D(p,h).getEvent() == HOSTSWITCH          */
22         D(p,h).eqClasses += f.val(x);
23         D(p,h).eqClasses_b += f_b.val(x);
```

purposes to $\{(p,h):1\}$ which are required for the HOSTSWITCH case.

The HOSTSWITCH case: by inductive hypothesis the dictionary of frontiers and relative subsolutions are well compute in $D(p_i,h_i)$ for $i=1,2$ and for p_1,p_2 children of p. The dictionary in $D(p,h)$ will have a key for each possible unions of a key in $D(p_1,h_2).frontiers$ with a key in $D(p_2,h_2).frontiers$ (recall that the keys are frontiers, *i.e.* sets and so we can take their set union) and as value the product of left and right values for the respective keys At this step of the computation there is no need to drop frontiers or risk to double count them as all the frontiers arising from the cartesian product are different by induction and by virtue of having stored also the parasite node in the frontier pairs.

Furthermore, if p_1 is the jumping child of p, now we know that p is a candidate for $\chi(p_1)$ (and perhaps some other descendants of p), *i.e.* the head of p_1, if it is the $\succeq_P$ greater ancestor. This suffice to grant to p_1 (and perhaps other descendants of p) the status of bundle nodes. For this reason in lines 15-13 we use for p_1 the bundle scenario values (which coincide with the non-bundle scenario values in case of no doubt).

On the other hand, for the non jumping child p_2, we are not sure yet if we are dealing with a bundle node or not (or better we may know it if $h\neq h_2$) but we postpone this to the multiple solution function in case of multiple alternatives subsolution of p_2 stored in $\widetilde{D}(p_2,h)$ and in case of single solution we also postpone up to the time we find an head for p. In both the cases by induction as we assume correct the values for both the children and in both the scenario we correctly compute the values related to both the scenario for $D(p,h)$ (lines 12,13).

For the case COSPECIATION,DUPLICATION, it is the function `computeEqClasses` to compute the value $eqClasses$ and since we are dealing with a simple solution it will be $D(p_1,h_1).eqClasses * D(p_2,h_2).eqClasses$ i.e. all the possible combinations of picking a $\sim_2$ subsolutions equivalence classes for p_1 and one for p_2.We also assign this value to the key $\{(p,h)\}$ of the $frontier$ dictionary at line 15 for initialization purposes as this is the number of $\sim_2$-subsolution which will be the left or right children $\sim_2$-subsolutions combined in the HOSTSWITCH case with the right or left children ones respectively. The values $eqClasses\_b$ and $frontiers\_b$ coincides with the non-bundle scenario values ($eqClasses$ and $frontiers$) as at this point it is clear that what is underneath is not a bundle node failing item 1. of Definition 28.

For all the three events the correct values of $eqClasses$ is computed and set in lines 19-23. □

Lemma 15 grant that `computeSimpleFrontiers` computes correctly the number of $\sim_2$-subsolutions (stored in $eqClasses\_b$ and $eqClasses$ for both the bundle and non-bundle scenario respectively) for a simple solution in $D(p,h)$ if p,h are leaves or if it is correct the computation of this values for multiple solutions.

Observe and recall that in function `computeSimpleFrontiers` we simply lift the information on the frontiers by looking at the event assign to it and we identify some bundle nodes when we are dealing with jumping children in an host-switch event.

The identification of the heads and bundle nodes in the remaining cases (*i.e.* when these are non jumping children) and the pruning of the non correct scenario,is due and postponed to the function `computeMultipleFrontiers`.

In the next lemma we show the correctness of the function `computeMultipleFrontiers` which computes the values $eqClasses$, $frontiers$ and the bundle scenario values $eqClasses\_b$ and $frontiers\_b$, in the case $D(p,h)$ it is a multiple solution and solve the remaining of the bundle scenario alternatives and the identification of the heads.

Lemma 16. *Let $D(p,h)$ be a multiple solution and let $c_l > 0$. Then the function* `computeMultipleFrontiers` *(Algorithm 7) computes correctly the number of $\sim_2$ sub-equivalence classes and the frontiers underneath p.*

Proof. Primarily observe that by $c_l > 0$ and Lemma 14 to dermine if a node u associated to an host-switch event is a bundle node, we need only to check that its head $\chi(u)$ exists. Let us recall that Lemma 15 grants, for solution $D(p',h')$ whose computation not depends on multiple solutions, the correct computation of the dictionary of frontiers and the value for $eqClasses$, also the corresponding bundle scenario, and in the case of p', h' being leaves.

So we can assume, by induction, them to be correct also for the simple solution $D(p,h_i)$ stored in $D(p,h)$.

The function `computeMultipleFrontiers` consists mainly of a loop which pass through all the frontiers of all the simple solutions and count each one just once as this is an invariant of reconciliations in the same $\sim_2$-equivalence class (item 4. of Fact 8).

Let us analyze first the case where $D(p,h_i)$ is not associated to an host-switch event: in this case there is only one frontier consisting of the pair (p,h_i) and its values in the dictionary is the number $D(p,h_i).getEqClasses$ of $\sim_2$-equivalence classes in $\underline{\mathcal{R}}(H,P_p,\varphi,C,h_i)$ (which coincide with the bundle scenario value as clearly a non host-switch node is not a bundle node). It is not difficult to see in the pseudo-code that either if the frontier $\{(p,h_i)\}$ has already been seen or not, at the end of the loop we have the number of $\sim_2$ subsolution beared by $D(p,h_i)$ added to the number of all the possible $\sim_2$ sub-solutions with frontier $\{(p,h_i)\}$. It is correct to sum up in the $EqClass$ and $EqClass_b$ values all the values related to the key $\{(p,h_i)\}$ as we do not drop any of the partial solutions having the mapping $\varrho(p) = h_i$ associated to a non host-switch event as each of them imply a different mapping for the children of p and which are surely not bundle nodes by item 1. of the definition.

We assume now then, for sake of simplicity, that all the $D(p,h_i)$ (in the list of simple solutions in $D(p,h)$) are associated to host-switch events as we have just seen that the other case is correctly counted and summed up in the final computation at lines 22-27.

Algorithm 7: Compute Multiple Frontiers

```
1  Function computeMultipleFrontiers (D(p,h))
2      f ← ∅;
3      f_b ← ∅;
       // seenMapping is a dictionary with values lists of (h',p') nodes.
4       // It helps identifying heads of bundles nodes and
       // hence also the roots of trees in Φ(ϱ)
5      seenMappings ← ∅;
       // Run on all frontiers of all simple solutions and count just a
          candidate for each of them
6      for simple_solution D(p,h') in D(p,h) do
7          for frontier f' in D(p,h') do
8              if not f.containsKey(f') then
9                  f.add({f' : D(p,h').val(f')});
10                 f_b.add({f' : D(p,h').val(f')});
11                 if D(p,h').event() == HOSTSWITCH then we keep trace of the
                   mapping of p in h' as it may be an head and we do it frontier-wise as we
                   do not want
12                     Let njc(D(p,h')) ∈ V(P) be the non jumping child of D(p,h');
13                     seenMappings.add(f', newList(h', njc(D(p,h'))));
14             else we have already seen this frontier
                   // we may have to update the frontier count if we are dealing
                      with a non host switch or a different head
15                 if D(p,h').event()! = HOSTSWITCH or not
                   seenMapping.val(f').contains(h', njc(D(p,h'))) then
16                     f.val(f')+= D(p,h').val(f');
                       // We count this frontier also for the bundle scenario
                       // only minding at the mapping of p
                       // and not who is njc(D(p,h'))
17                     if not seenMappings.val(f').getListHostTreeNodes().contains(h')
                       then
18                         f_b.val(f')+= D(p,h').val_b(f');
                   // We update seenMappings if we are dealing with an host
                      switch
19                 if D(p,h').event() == HOSTSWITCH then
20                     seenMappings.get(f').add_to_list(h', njc(D(p,h')));
21     D(p,h).setFrontiers(f);
22     D(p,h).setFrontiersB(f_b);
23     D(p,h).eqClasses ← 0;
24     D(p,h).eqClasses_b ← 0;
25     for frontier x in f do
26         D(p,h).eqClasses += f.val(x);
27         D(p,h).eqClasses_b += f_b.val(x);
```

Let $q, s(q)$ be the children of p in P and let us assume without lack of generality that q is the jumping child.

If one of the frontiers listed in $D(p, h_i)$ is not already seen, then we add it to the keys of the dictionary of $D(p, h)$ with its value, finally we took notes that this frontier as been already seen in a solution which map p to h_i and its non jumping child is $njc(D(p, h_i))$ (lines 8-13).

These information are necessary to distinguish between frontiers and eventually exclude already seen.

Indeed, if a frontier has already been seen, then another simple solution $D(p, h_j)$ has the same frontier where recall that $h \succeq_P h_i, h_j$. Now, the differences between $D(p, h_i)$ and $D(p, h_i)$ which lead to lift the same frontier are in one or more of the followings points considering that they may differ only on the mapping of p, q or $s(q)$:

1. $h_j = h_i$ and q is the jumping child for both the solutions but its mapping differs,
2. $h_i \neq h_j$, *i.e.* the mapping of p in H for these subsolutions differ,
3. For both the solutions $s(q)$ is the non-jumping child, its mapping differs in these solutions but being associated to an host-switch event it lifts the same frontier,
4. q is not the jumping child for both the solutions, although in these solutions the pointer in D of the optimal mapping of q and $s(q)$ are to sub-solutions which associate them to host-switch event (because they lift the same frontier).

The first case, together with the fact of having the same frontier, imply that q and possibly its descendant are bundle nodes and p may be the head of a tree in $\Gamma_P(\cdot)$ (if it is the oldest ancestor with this property); having seen already one such frontier we do not add its count to the dictionary count as they will alternative mappign that will fall in the same equivalence class.

In the second case, the node p will not be associated to a bundle node as observe that for at least one of the two nodes h_i, h_j, say h_j it holds $h \succ_H h_j$ (this case holds only in the $\widetilde{D}$ matrix), we could then improve the cost of the solution mapping p (and all the bundle nodes between him and its head) to h_j (preserving their state of being associated to host-switch event) sparing the loss cost as in Lemma 14.
So p is not a bundle nodes, but it can still be an head, this is the reason to update the count for this frontier and the number of subsolutions with this frontier saving also in $seenMapping$ the mapping of p to h_i (if it was not present already) and the non jumping child mapping, line 13 (a case where this information is required is when p has a slide, see Figure 2.23).

The third case is solved by inductive hypothesis as the optimal mapping of the non jumping child are saved in another solution in the matrix $\widetilde{D}$, indeed: $D(p, h_i)$ is a simple

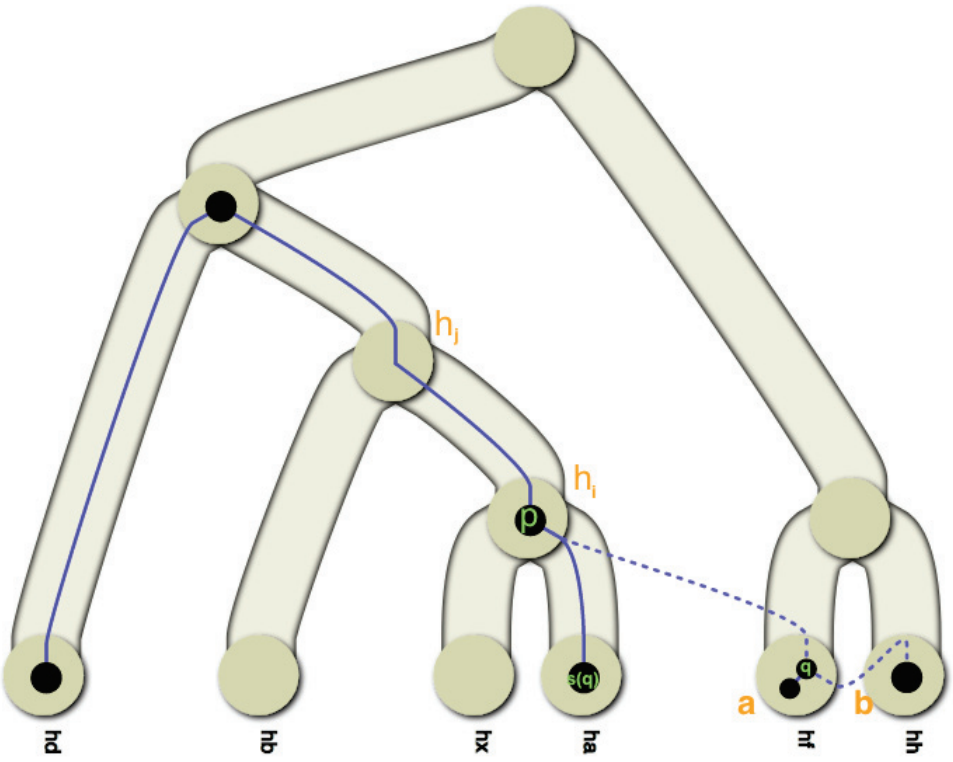

Figure 2.23: Case 2 in the proof of Lemma 16: *orange* nodes are labels for the thick tree, the *green* ones for the thinner black tree. The node p is not a bundle node as $\alpha(p)$ exists. p can be mapped both to h_i and h_j without changing the cost of the reconciliation leaving the other mappings unchanged. We have then in Algorithm 7 at line 13 save in $seenMappings$, together with p, the different mappings to h_i and h_j as they will lead to $\sim_2$-different reconciliations. On the other hand if it is only the mapping of q to change from a to b we obtain $\sim_2$-equivalent reconciliations, hence we should count only one of them.

solution associated to an host-switch whose mapping for the children are given by $D(q, h^*)$ for the jumping child and $\widetilde{D}(s(q), h')$ for the non-jumping child; the third case is actually then treated in the second case when we call the function `computeMultipleFrontiers` on $\widetilde{D}(s(q), h')$.

The forth case implies that both q and $s(q)$ are associated to an host-switch event contemporary in the two different simple solutions, and these two alternate the non jumping child (see Figure 2.24 for an example). If $q, s(q)$ and p are bundle nodes, then we have to drop the second time we see the frontier while. If, on the contrary, p is not a bundle nodes (*i.e.* we will end up with no heads for p), we cannot discard (or not count) the solutions beared by these frontiers and the alternating jumping child as p would be an head for the jumping child but not for the non-jumping child. This mean that the two alternatives will not have the same set of bundle nodes. We have to compute both the scenarios and postpone the decision at the time we find an head for p (or keep the non bundle scenario).

To conclude, only in the first case or in the forth case when we confirm the bundle scenario we drop equivalent $\sim_2$-subsolutions, *i.e.* when we have confirmed the nature of

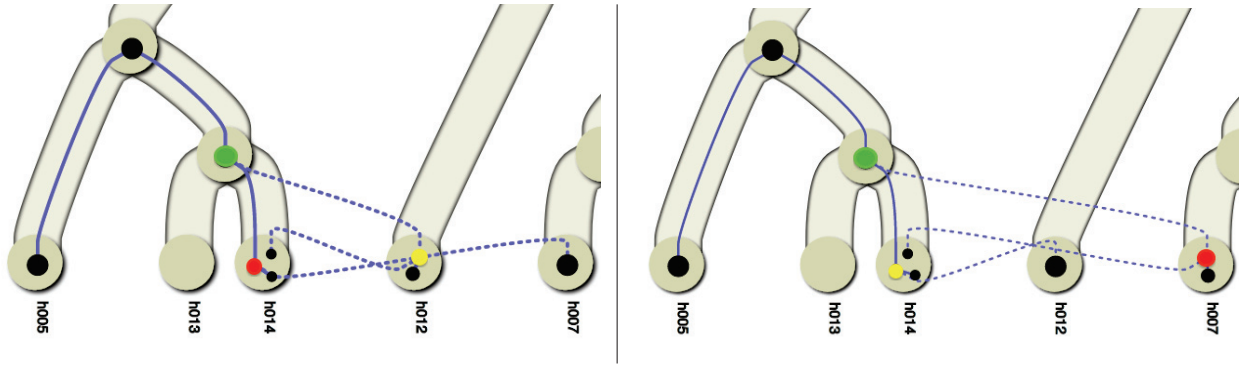

Figure 2.24: A typical situation related to the fourth case of the proof of Lemma 16: The *yellow* and *red* nodes plays alternatively the role of the jumping child of the *green* node. These reconciliations are not $\sim_2$-equivalent, hence the number of sub-equivalence classes stored in $\widetilde{D}(green\_node, h)$ is 2 for the non bundle scenario of the $green_node$ (stored in $\widetilde{D}(green\_node, h).eqClasses$), while it is 1 for the bundle scenarion (stored in $\widetilde{D}(green\_node, h).eqClasses\_b$).

bundle nodes for q or for the non-jumping child respectively. This is done in lines 15-18 where in particular lines 15,16 cover the cases of non-bundle scenario and lines 17-18 the bundle one where it should do not mind the non jumping child when we are in the fourth case, but still consider the mapping of h to avoid redundancy in the first case.

After the loop the correct values for $eqClasses$ and the frontiers are set in $D(p, h)$ together with the bundle scenario ones. □

From the previous two lemmas we derive directly the following result:

Theorem 11. *Let $c_l > 0$ and let $D(root(P), h_i)$ for $1 \leq i \leq k$ be the cells with lower cost among all the costs for $D(root(P), h)$ with $h \in V(H)$. Then the number of number of $\sim_2$-equivalence relations in $\mathcal{R}(P, H, \varphi, c)$ is:*

$$\sum_{i=1}^{k} ComputeEqClasses(D(root(P), h_i))$$

Proof. The values returned by $ComputeEqClasses(D(root(P), h_i))$ is the value $D(root(P), h_i).eqClasses$ (see Algorithm 5) which contains the correct number of $\sim_2$-equivalence classes of reconciliations in $\underline{\mathcal{R}}(H, P, \varphi, C, h_i)$ in virtue of Lemmas 16 and 15 and the fact that, reached $root(P)$, we are in a non-bundle scenario by definition of head of a bundle node.
The results follows as $D(root(P), h_i)$ has the lowest cost among $D(root(P), h)$ for $h \in V(H)$ hence $\mathcal{R}(H, P, \varphi, c) = \cup_{i=1}^{k} \underline{\mathcal{R}}(H, P, \varphi, C, h_i)$. □

2.8 A New Distance Measure between Reconciliations

In this section we introduce a new distance on the space of optimal reconciliations $\mathcal{R}(H, P, \varphi, c)$. This distance measures the similarities between reconciliations on the ba-

sis of the host-switch edge set $\widetilde{\Theta}$ considering that when we restrict the model to only co-speciation, duplication and loss events, there is only one optimal reconciliation give by the *lca* mapping [46, 48, 63, 71]. We prove in Theorem 12 in this section that the knowledge of $\widetilde{\Theta}$ is sufficient to reconstruct a reconciliation.

Another motivation to introduce this host-switch-based distance is that even if working with the equivalence classes introduced in Section 2.6, we drastically reduce the number of objects to consider (see Section 2.9 for a discussion and experimental results), although in some cases this number remains outside the possibilities of a direct inspection.

On the other hand, reconciliations in the same equivalence class differ only on host switches hence, if we restrict our domain to only canonical representatives of equivalence classes, intuitively we ends up to work in a smaller metric space.

To the best of our knowledge, only few measures of similarity between reconciliations have been defined. One of them is based on the comparison of the *event vector*, *i.e.* a four dimensional vector which collects the numbers of occurred events for each one of the four allowed events. Then, a hierarchical cluster tree is constructed based on the event vectors. Nevertheless, it is not difficult to find examples of very different reconciliations having the same event vector. One of them is in Figure 2.18.

In the special case of a model where no host switches are allowed (*i.e.* only cospeciation, duplication and loss events are permitted) the authors of [26] define two operators which turn one reconciliation to another, and then they define a distance measure between two reconciliations as the smallest number of operations needed to pass from one reconciliation to the other one. In [92] the definition of operators is extended to a more general model in which host switches can appear and new operators are introduced.
With this definition, there are reconciliations that appear very similar and nevertheless have a rather high distance, as shown for example in Figure 2.25.

Indeed, in the specific case of this figure, if we call u the only host-switch node and ϱ_1, ϱ_2 the reconciliations, the distance coincide with the length of $path_H(\varrho_1(u), \varrho_2(u))$.
However, we can make this distance arbitrarily large by extending this path (which in Figure 2.25 has length 4) branching the host tree along this path to have an arbitrary number of intermediate nodes between $\varrho_1(u)$ and $\varrho_2(u)$.

In this section, we try to overcome all the problems highlighted in the other known metrics and we present a distance based only on host switches.

Given a reconciliation $\varrho \in \mathcal{R}(H, P, \varphi, C)$, preliminarily note that each one of its host switches is univocally determined by an edge $e = (u, v)$ of P and by its mapping on a non-edge of H $(\varrho(u), \varrho(v))$, where $\varrho(u)$ and $\varrho(v)$ are incomparable. We introduce here an host-switch set which collect all these informations:

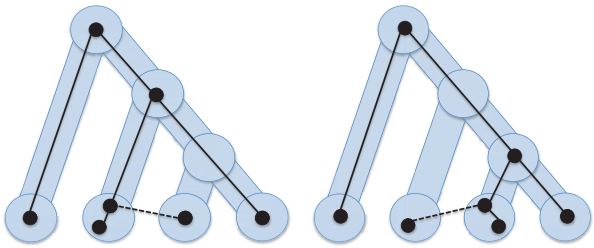

Figure 2.25: Two reconciliations very similar with a possibly high distance based on operators (by arbitrarily adding nodes on the right path from the root.

Definition 31. *Let $\varrho : V(P) \to V(H)$ be a reconciliation. The* host-switch set of ϱ *is defined as the set of quartuple :*

$$\Theta(\varrho) := \left\{ (u, \varrho(u), v, \varrho(v)) \subseteq V_P \times V_H \times V_P \times V_H : (u, v) \in \widetilde{\Theta}(\varrho) \right\}.$$

The next theorem shows that the information in $\Theta(\varrho)$ (together with the scenario (H, P, φ)) is enough to reconstruct the reconciliation ϱ.

Theorem 12. *Under the hypothesis that $c_l > 0$ and $c_d \geq c_c$, let $\varrho, \sigma \in \mathcal{R}(H, P, \varphi, C)$. Then $\Theta(\varrho) = \Theta(\sigma)$ if and only if $\varrho = \sigma$.*

Proof. if part: This part is trivial, indeed if $\varrho = \sigma$, then their host-switch sets must be the same.

only if part: Let us prove the statement by induction on the number of host switches of ϱ.
If $|\Theta(\varrho)| = |\Theta(\sigma)| = 0$ then from Theorem 10, we have that the only optimal reconciliation is given by the *lca* mapping. Now, let $|\Theta(\varrho)| = |\Theta(\sigma)| = t > 0$ and, by inductive hypothesis, assume the lemma holds for any value strictly lower than t. Consider a lowest host switch $(u, \varrho(u), v, \varrho(v)) \in \Theta(\varrho) = \Theta(\sigma)$ and the two trees P_v and P^u. Reconciliations ϱ_v and σ_v, ϱ^u and σ^u have all less than t switches (because at least $(u, \varrho(u), v, \varrho(v))$ is not included) and they are optimal by Lemma 12, so $\varrho_v = \sigma_v$ and $\varrho^u = \sigma^u$ by inductive hypothesis and both ϱ and σ include the switch $(u, \varrho(u), v, \varrho(v))$.

It follows that $\varrho = \sigma$. □

Observe that we can put in relation the set of all possible host switches $\Theta(\mathcal{R}) := \cup_{\varrho \in \mathcal{R}(H,P,\varphi,c)} \Theta(\varrho)$, with the first k integer numbers (for an opportune value of k). In this way, each reconciliation ϱ can be put in relation with the k long characteristic vector representing $\Theta(\varrho)$ in which a 1 appears in position i if host switch i belongs to $\Theta(\varrho)$ while all the other positions are set to 0. Theorem 12 guarantees that this relation is a one to one function and in particular we can embed $\mathcal{R}(H, P, \varphi, c)$ into the k dimensional discrete hypercube $\mathcal{H}^k := \{0, 1\}^k$. Let us denote with $\Theta_k : \mathcal{R}(H, P, \varphi, c) \to \mathcal{H}^k$ this embedding.

We can then define the distance between two reconciliations as:

Definition 32. *Let $\varrho, \sigma \in \mathcal{R}(H, P, \varphi, C)$. Then the* Host-switch Distance *between ϱ and σ is defined as:*

$$\mathsf{DH}(\varrho, \sigma) := \big|\Theta(\varrho) \ominus \Theta(\sigma)\big| \qquad \Big(= \big|\big(\Theta(\varrho) \setminus \Theta(\sigma)\big) \cup \big(\Theta(\sigma) \setminus \Theta(\varrho)\big)\big|\Big),$$

where $\ominus$ denote the symmetric difference of two sets.

We can easily see that DH is a distance by observing that, for a sufficiently large integer k, $\mathsf{DH}(\varrho, \sigma) = Hamming\_distance(\Theta_k(\varrho), \Theta_k(\sigma))$.

We want to conclude this section by commenting that the definition of the distance DH, based on the host-switch set $\Theta(\varrho)$ which characterize a reconciliation ϱ (Theorem 12), may be improved by some extra biological knowledge, for instance: by introducing a weight function $w : \Theta(\mathcal{R}) \to \mathbb{R}$ so has to have:

$$\widetilde{\mathsf{DH}}(\varrho, \sigma) := \sum_{s \in \Theta(\varrho) \ominus \Theta(\sigma)} w(s).$$

Then we can weight the host switch $s = (u, \varrho(u), v, \varrho(v))$ according to some properties that we judge important as for instance the *length of the jump* so:

$$w(s) := dist_H(\varrho(u), \varrho(v)),$$

or if we consider that a lowest host switch s has a weaker impact on a reconciliation structure compare to other host switches (as in a lowest host switch the mapping of the nodes in P_u, P_v are determined by the *lca* mapping) we can try to catch this property defining:

$$w(s) := 2 * height(H) - height(\varrho(u)) - height(\varrho(v)).$$

We ends up leaving these two examples as proof of concepts together with the visual analysis made in Section 2.9.2 in which we present and comment some histograms of DH distances based on classical datasets [25] together with some statistics on the number of host switches and dimensions of the space $\mathcal{H}^k$.

2.9 Experimental Results

In this section we show the results of some experiments performed on some real data sets.

We selected 13 datasets which correspond to those also used in [25] and that are indicated in that paper as GL, RH, FD, COG2085, COG3715, COG4964, COG4965, PP, SFC, EC, PMP, PML, and *Wolbachia*. The latter is a dataset of our own which corresponds to arthropod hosts and a parasitic bacterium, *Wolbachia*, living inside the

cells of their hosts (for further details on *Wolbachia* see the introductory section of Chapter 1.

These datasets are listed in Table 2.2 where we report also the following information:

dataset	$\|L(P)\|$	$\|\varphi(L(P))\|$	$\|V(P)\|$	$\|\varrho(V(P))\| \leq$
GL	10	8	19	15
RH	42	33	83	65
FD	51	20	101	39
COG2085	43	25	85	49
COG3715	50	15	99	29
COG4964	27	17	53	33
COG4965	30	19	59	37
PP	41	36	81	71
SFC	16	15	31	29
EC	10	7	19	13
PMP	18	18	35	35
PML	18	18	35	35
Wolbachia	387	387	773	773

Table 2.2: General information on datasets employed.

- the number of leaves of P (column $|L(P)|$),
- the number of leaves of H on which some leaf of P is mapped through φ (column $|\varphi(L(P))|$),
- the number of nodes of P (column $|V(P)|$),
- the maximum number of nodes of H that can be the mapping of some parasite node through a reconciliation; this value is derived by $|\varphi(L(P))|$ by observing that the largest number of interested nodes in H is obtained when they are organized as a complete binary tree and it is hence $\min\{|V(P)|, 2|\varphi(L(P))| - 1\}$ (column $|\varrho(V(P))| \leq$)

2.9.1 Equivalence Classes

We present here the experimental results that we obtained applying the equivalence classes described in Section 2.6 to the datasets of Table 2.2 taken from [25].
There are five tables, one for each of the cost vectors: (-1, 1, 1, 1), (0, 1, 1, 1), (0, 1, 2, 1), (0, 2, 3, 1) and (1, 1, 3, 1), which are some of the most common [25].

In all the tables, *# solutions* indicates the number of all optimal reconciliations, while # $\sim_1$, # $\sim_2$ and # $\sim_2$ + $\sim_1$ indicate the number of equivalence classes when relations $\sim_1$, $\sim_2$ or both are applied; the last column, called *NMR*, indicates the value of the Normalized Magnitude Reduction, rounded to two digits after the decimal point, which is given by $\frac{\log(\#sol)-\log(\#\sim_1+\sim_2))}{\log(\#sol)}$. Such value is 1 when all optimal solutions are reduced to a single parsimonious reconciliation

when applying the two equivalences. Inversely, the closer this value is to 0, the less the two equivalences were able to reduce by similarity the number of solutions.

Observe that for *Wolbachia*, the number of solutions is so huge that, for space reason, we rounded the number to fit the table.

Regarding alternative choices to NMR for evaluating our results could have been a simple proportion $\frac{\sim_2+\sim_1}{\#sol}$ but just obtaining, for instance, 0.001% of (equivalence classes of) reconciliations it could be meaning less when facing huge numbers of solutions as in the *Wolbachia* case. Indeed, we would have obtained, in this way, the score of score 0.001, where the optimum would be reached at $1/\#sol$, passing from $\sim 10^{48}$ to $\sim 10^{47}$ objects which are actually. Hence we decide to measure the exponents of the base 10 so in this way in this example it would be $47/48 \sim 0.97$. Now halving the order of *magnitude* it could obtained both passing from 100 to 10 solutions and passing from $\sim 10^{48}$ to $\sim 10^{24}$. Although having to treat 10 solutions and $\sim 10^{24}$ could not be evaluated at the same way. We decide then to rapport the reduction of the magnitude w.r.t. the overall magnitude (See Figure 2.26):

$$NMR = 1 - \frac{\log(a \cdot 10^x)}{\log(b \cdot 10^y)} = \frac{y + \log(b) - x - \log(a)}{y + \log(b)} \underset{\text{ROUGH}}{\simeq} \frac{y - x}{y}$$

When $y \geq x \gg 1$ and assuming $0 \leq a, b < 10 \Longrightarrow 0 \leq \log(a), \log(b) < 1$.

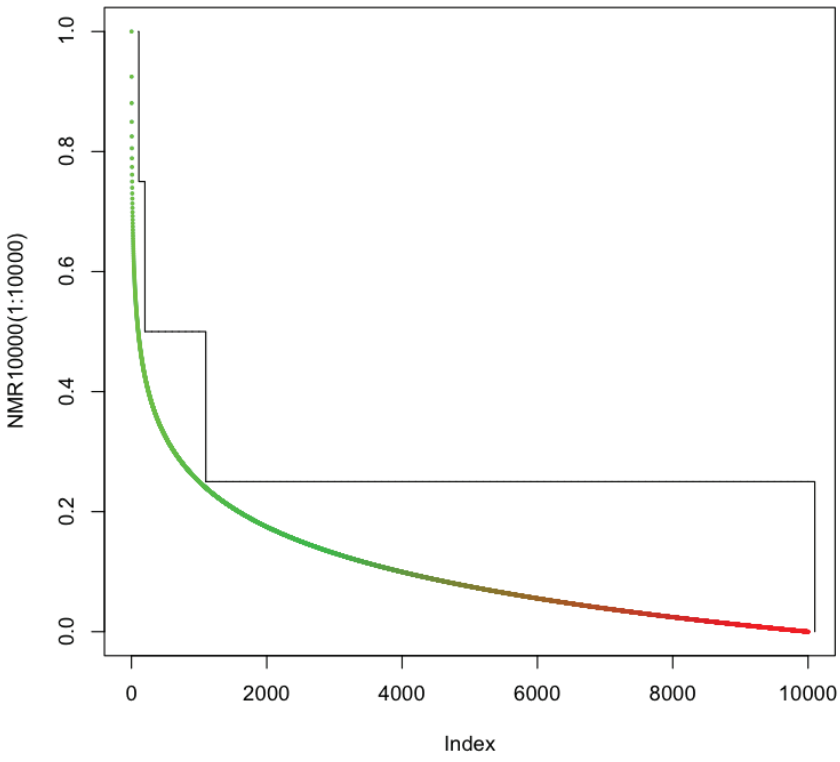

Figure 2.26: green-to-red points: NMR($x = Index$, $y = 10000$). Black line: rough super-extimation of NMR obtained by approximating numbers involved in the computation as $a \cdot 10^x \sim x$; in particular the function is constant in the intervals of the form $[10^x, 10^{x+1})$.

From the previous tables we can deduce a number of observations.

Dataset	# solutions	# $\sim_1$	# $\sim_2$	# $\sim_2 + \sim_1$	NMR
GL	2	2	2	2	0
RH	1056	176	528	88	0,36
FD	944	368	50	18	0,58
COG2085	109056	7360	171	6	0,85
COG3715	63360	2520	1408	32	0,69
COG4964	36	4	9	1	1
COG4965	44800	23456	121	13	0,76
PP	144	144	72	72	0,14
SFC	40	16	10	4	0,62
EC	2	2	2	2	0
PMP	2	2	1	1	1
PML	2	2	1	1	1
Wolbachia	$\sim 1.01 \cdot 10^{47}$	$\sim 3.77 \cdot 10^{44}$	$\sim 2.92 \cdot 10^{8}$	$\sim 2.42 \cdot 10^{4}$	0,91

Table 2.3: Results for cost vector $(-1, 1, 1, 1)$.

Dataset	# solutions	# $\sim_1$	# $\sim_2$	# $\sim_2 + \sim_1$	NMR
GL	2	2	2	2	0
RH	42	42	8	8	0,44
FD	25184	22752	256	206	0,47
COG2085	44544	36224	11	4	0,87
COG3715	1172598	777030	2256	1112	0,50
COG4964	224	224	2	2	0,87
COG4965	17408	17408	4	4	0,86
PP	5120	4480	344	280	0,34
SFC	184	160	16	10	0,56
EC	16	16	13	13	0,07
PMP	2	2	1	1	1
PML	180	160	33	21	0,41
Wolbachia	$\sim 3.19 \cdot 10^{48}$	$\sim 5.72 \cdot 10^{47}$	$\sim 9.33 \cdot 10^{5}$	$\sim 7.68 \cdot 10^{4}$	0,90

Table 2.4: Results for cost vector $(0, 1, 1, 1)$.

Firstly, note that it is not surprising that in the case of the cost vector $(0, 1, 1, 1)$, there are on average more optimal solutions than with the other cost vectors. Since events different from cospeciation are indistinguishable in terms of cost, there is a freedom in terms of choices of the events keeping the overall cost constant.

Secondly, given that both equivalence relations are primarily based on host-switch mappings, we would then expect that the higher is the number of host switches, the greater would be the chance of having a lower number of equivalence classes w.r.t. the total number of solutions. For both the equivalence class relations it is anyway different how these host switches will be present inside the reconciliation depending on the existence of not of the $\alpha(\cdot)$ ancestor for them. For both of the relations, to benefit of an huger number of reconciliations in the same class, hence a lower number of different equivalence classes, it is desirable to have many adjacent nodes in P associated to host-switch event.

Dataset	# solutions	# $\sim_1$	# $\sim_2$	# $\sim_2 + \sim_1$	NMR
GL	2	2	2	2	0
RH	2208	368	1608	268	0,27
FD	408	180	48	20	0,50
COG2085	37568	3200	226	14	0,75
COG3715	9	7	4	2	0,68
COG4964	36	4	9	1	1
COG4965	640	576	4	3	0,83
PP	72	72	36	36	0,16
SFC	40	16	10	4	0,62
EC	18	18	18	18	0
PMP	2	2	1	1	1
PML	2	2	1	1	1
Wolbachia	$\sim 1.01 \cdot 10^{47}$	$\sim 3.77 \cdot 10^{44}$	$\sim 2.92 \cdot 10^{8}$	$\sim 2.42 \cdot 10^{4}$	0,91

Table 2.5: Results for cost vector $(0, 1, 2, 1)$.

Dataset	# solutions	# $\sim_1$	# $\sim_2$	# $\sim_2 + \sim_1$	NMR
GL	2	2	2	2	0
RH	288	48	288	48	0,32
FD	80	16	10	2	0,84
COG2085	46656	1344	540	10	0,79
COG3715	33	2	33	2	0,80
COG4964	54	6	18	2	0,83
COG4965	6528	448	94	5	0,82
PP	72	72	36	36	0,16
SFC	40	16	10	4	0,62
EC	16	16	16	16	0
PMP	18	18	10	10	0,20
PML	11	6	7	4	0,42
Wolbachia	$\sim 4.08 \cdot 10^{42}$	$\sim 1.33 \cdot 10^{36}$	$\sim 4.18 \cdot 10^{10}$	$\sim 1.15 \cdot 10^{3}$	0,93

Table 2.6: Results for cost vector $(0, 2, 3, 1)$.

In the case of $\sim_1$, letting n be this adjacent nodes and letting l be the length of the path in H between the associated $\alpha(\cdot)$ and $\omega(\cdot)$, we would obtain an higher value for $\binom{n-l-1}{l-1}$ *i.e.* a lower bound on the number of reconciliations in the same $\sim_1$-class which map these n nodes into the l nodes of the path.

For $\sim_2$ an higher number of adjacent nodes associated to host-switch events leads to a bigger tree T in $\Gamma_P(\cdot)$, let say of size n, hence it is possible to roughly extimate with $\log(2^n - 1/2)^n$ the number of possible mappings of these n nodes on the frontier of T (See Fact 7).

Thirdly, the presence of many host switches clearly deeply depends on the geometry of the host tree H and the leaf mapping which, if intuitively is *spread*, *i.e.* it map nodes that are closer in P to nodes that are far in H, then clearly promotes the existence of host switches.

Dataset	# solutions	# $\sim_1$	# $\sim_2$	# $\sim_2 + \sim_1$	NMR
GL	9	9	9	9	0
RH	648	108	504	84	0,32
FD	36	12	14	5	0,55
COG2085	6208	1296	120	10	0,74
COG3715	36	28	16	8	0,42
COG4964	1056	72	264	18	0,59
COG4965	832	736	7	5	0,76
PP	84	72	54	45	0,14
SFC	42	18	11	5	0,57
EC	3	3	3	3	0
PMP	2	2	1	1	1
PML	2	2	1	1	1
Wolbachia	$\sim 1.51 \cdot 10^{47}$	$\sim 5.66 \cdot 10^{44}$	$\sim 5.68 \cdot 10^{8}$	$\sim 4.32 \cdot 10^{4}$	0,90

Table 2.7: Results for cost vector $(1, 1, 3, 1)$.

Fourtly, the cost vector C plays a role as well. We present here after an heuristic related to C which has different consequences for $\sim_1$ and $\sim_2$.
Indeed observe that: when a long slide between $\alpha(s)$ and $\omega(s)$ occurs then there is a parasite non-leaf node $\alpha(s)$ for which is convenient to speciate for example; while its child assigned to host switch s, *the sliding node*, can scroll through the slide paying a fix cost which is at least the cost of the length of the slide $c_l \cdot slide\_length$. If it is parsimonious this reconciliation it should be more convenient in term of cost w.r.t. the alternative solution of mapping s to $\varrho(q)$ where q is the jumping child of s. In this second case, we introduce two host-switch edge $(\varrho(\alpha(s)), \varrho(q))$ and $(\varrho(q), \varrho(\omega(s)))$ (assuming that $\varrho(\alpha(s)) \ncong \varrho(q)$) and removed the host-switch edge $(\varrho(s), \varrho(q))$.

Then we pass from a cospeciation of $\alpha(s)$ to an host-switch and we have to pay also the loss for having $\varrho(s(s)) \neq \varrho(\alpha(s))$.
Comparing the differences in both the scenario,if it is preferable the slide scenario then:

$$c_l \cdot slide\_length + c_s \leq c_h + c_l,$$

From this intuitive reasoning follows:

$$slide\_length \leq 1 + \frac{c_h - c_s}{c_l},$$

that the greater the difference in cost of host switch and loss events the greater the length l of the slide, hence the probably greater the reduction of the number of object to treat for $\sim_1$. On the contrary, if the second scenario is preferred, we would have that q is a bundle node on behalf of $\sim_2$.

With this heuristic in mind, let us focus on the columns $\sim_1$ and $\#solutions$. It is not surprising that the worst results are obtained by the cost vectors $(0, 1, 1, 1)$ where,

intuitively, there cannot be slides of length more than 2, when in the other cases the length can reach $3, 4$.

2.9.2 Metric Space

From a pure computer science point of view and without any biological specific analysis we do not know how to evaluate the goodness of the distance DH (defined in Section 2.8) if not the fact that treats directly the minimal information necessary to reconstruct an optimal reconciliation without knowing the original cost vector.

To supply to this lack of a formal evaluation and inspired by the paper of Robinson and Foulds [78], we present in this section some histograms and statistics as a sound form to evaluate this distance. In other words, the histogram somehow shows the empirical probability for two different reconciliations to be at most a certain distance apart. For space matters we will not display all the histograms for all the datasets and cost vectors but just the ones we consider more meaningful or which show some interesting pattern.

On the other hand we present in tables 2.8-2.12 all the datasets and cost vectors the maximum and minimum distances found for each dataset, together with a theoretical upper bound.

We use the same datasets summerized in Table 2.2 to collect some statistics on the distance DH between optimal reconciliations for the cost vectors: (-1, 1, 1, 1), (0, 1, 1, 1), (0, 1, 2, 1), (0, 2, 3, 1) and (1, 1, 3, 1).

In the tables 2.8-2.12 (one for each cost vector) we present together with the number of optimal reconciliations (column # solutions), the number of all different host switches found among all the optimal reconciliations, *i.e.* the minimum dimension k to embed the optimal reconciliation space into $\mathcal{H}^k$ (column $dim(\mathcal{H}^k)$). We then present the maximum number (column s_M) and minimum number (column s_m) of host switches found in a optimal reconciliation. The column s_a contain the average number of host switches among all the optimal reconciliations.

In column DH $\leq$, we give an empirical overbound on the maximum distance assuming that there exists at least 2 reconciliations with the maximum number of switches s_M and at least a reconciliation with x switches for $s_m \leq x \leq s_M$. Then either the maximum theoretic distance is $2s_M$ if $2s_M \leq dim(\mathcal{H}^k)$ or it is $dim(\mathcal{H}^k)$ if $s_m + s_M \leq dim(\mathcal{H}^k)$ (supposing than that there exist two reconciliations complementary in terms of switches) or the maximum distance is $dim(\mathcal{H}^k) - s_M - s_m$ (supposing that we can minimize the overlaps of the vector representing the two reconciliations in $\mathcal{H}^k$).

In tables 2.27-2.30, we present some histograms which show the empirical distribution of the values $\mathsf{DH}(\varrho, \sigma)$ for any pair of distinct reconciliations $\varrho, \sigma \in \mathcal{R}(H, P, \varphi, c)$.

The area of the columns with respect to the all area of these histograms represent the empirical probability of a randomly selected pair of reconciliations to be at that distance. The area at the left of the column x (plus the column itself) with respect to the whole

dataset	# solutions	$dim(\mathcal{H}^k)$	s_m	s_M	s_a	DH $\leq$
GL	2	4	3	3	3.0	2
RH	1056	42	13	17	15.09	34
FD	944	73	20	24	21.95	48
COG2085	109056	145	25	27	26.01	54
COG3715	63360	91	25	28	27.16	56
COG4964	36	21	13	13	13.0	16
COG4965	44800	216	17	19	18.23	38
PP	144	25	12	13	12.66	25
SFC	40	27	10	10	10.0	20
EC	2	5	4	4	4.0	2
PMP	2	8	6	6	6.0	4
PML	2	10	8	8	8.0	4

Table 2.8: Vector $(-1, 1, 1, 1)$.

dataset	# solutions	$dim(\mathcal{H}^k)$	s_m	s_M	s_a	DH $\leq$
GL	2	4	3	3	3.0	2
RH	42	41	18	20	18.90	40
FD	25184	132	23	27	25.74	54
COG2085	44544	186	27	28	27.71	56
COG3715	1172598	231	20	31	27.98	62
COG4964	224	43	15	16	15.85	32
COG4965	17408	157	18	19	18.97	38
PP	5120	134	17	20	18.5	40
SFC	184	55	10	11	10.78	22
EC	16	15	4	6	5.18	12
PMP	2	8	6	6	6.0	4
PML	180	60	8	11	10.54	22

Table 2.9: Vector $(0, 1, 1, 1)$.

are, represent the probability that a randomly picked pair of reconciliations has distance at most x.

In general almost all histograms (generate for all the dataset and vector costs as for the tables) present a bell-shaped-like form as the ones of Figure. So intuitively we can say that it is rare to find two reconciliations which are extremely far or extremely close and to pass from a reconciliation to another on average a certain number of host switches has to change.

From the histograms we can also deduce other characteristics of the reconciliations in $\mathcal{R}(H, P, \varphi, c)$; indeed, the histograms in figure 2.27 and 2.28 have none or very few pairs of reconciliations at odd distance. We may than think that there are no or almost none alternative solutions to an host switch, *i.e.* either you took a subsolution with an host switch or the alternative is taking another solution with another host switch which is then at distance 2.

Typical situation like the described one are the reconciliations in a $\sim_1$ equivalence

dataset	# solutions	$dim(\mathcal{H}^k)$	s_m	s_M	s_a	DH $\leq$
GL	2	4	3	3	3,00	2
RH	2208	41	10	16	13,00	32
FD	408	61	18	21	20,05	42
COG2085	37568	107	23	25	24,49	50
COG3715	9	33	19	20	19,66	27
COG4964	36	21	13	13	13,00	16
COG4965	640	58	17	18	17,80	36
PP	72	22	11	12	11,66	21
SFC	40	27	10	10	10,00	20
EC	18	8	1	4	2,60	8
PMP	2	8	6	6	6,00	4
PML	2	10	8	8	8,00	4

Table 2.10: Vector $(0, 1, 2, 1)$.

dataset	# solutions	$dim(\mathcal{H}^k)$	s_m	s_M	s_a	DH $\leq$
GL	2	4	3	3	3.0	2
RH	288	21	10	13	11.5	19
FD	80	32	19	20	19.5	25
COG2085	46656	82	23	25	24.0	50
COG3715	33	32	19	19	19.0	26
COG4964	54	22	12	13	12.66	19
COG4965	6528	95	16	17	16.98	34
PP	72	22	11	12	11.66	21
SFC	40	27	10	10	10.0	20
EC	16	5	1	4	2.5	5
PMP	18	14	5	6	5.11	12
PML	11	15	7	8	7.18	15

Table 2.11: Vector $(0, 2, 3, 1)$.

class as for instance the ones in Figure 2.19 (which any pair of them has distance 2).

Another situation is represented in Figure 2.20 where the bundle node v (and u in figure b.) change its mapping in the two figures a. and b. creating a distance equal to 2 times the number of host switch involved (2 host switches in figure a. and 3 in figure b.).

The presence in Figure 2.28 of some odd distance pairs may represent a situation where it is alternatively possible having an host switch or a combination of loss and duplications or cospeciations events.

In Figure 2.29 the histogram resemble a mixture of two Gaussian distributions and we may think that the reconciliations space is split in two sets of close distance pairs hence you find with either high probability a close distance pair or with high distance according to the belonging or not of the two reconciliations to the same set.

The histogram of Figure 2.30 presents a situation where the more the distance grows

dataset	# solutions	$dim(\mathcal{H}^k)$	s_m	s_M	s_a	DH $\leq$
GL	9	6	1	3	2.33	6
RH	648	38	8	14	10.90	28
FD	36	31	18	19	18.77	25
COG2085	6208	65	22	25	22.96	50
COG3715	36	44	17	20	18.66	40
COG4964	1056	41	9	13	11.93	26
COG4965	832	77	17	18	17.76	36
PP	84	20	8	11	9.98	20
SFC	42	33	10	10	10.0	20
EC	3	2	0	1	0.66	2
PMP	2	8	6	6	6.0	4
PML	2	10	8	8	8.0	4

Table 2.12: Vector $(1, 1, 3, 1)$.

the higher the number of pairs which have that distance is (distinguishing between odd and even distances) with a drop of the number of pairs at the highest distance. In this example the empirical probability of picking a pair at close distance is very low. We may think that the dataset is sparse.

2.10 Conclusions and Open Problems

In this chapter we have presented and proved various lemmas and propositions (Section 2.5) related to the Reconciliation Model posing some conditions on the cost vector C. Very often these conditions are that the cost of a cospeciation must not exceed the cost of a duplication and that the cost of a loss must be strictly positive.
However it is fairly accepted to promote the coevolution w.r.t. the duplication (as we are looking for this kind of event when it may happen) while assuming $c_l \leq 0$ may lead to parsimonious reconciliations where the nodes of P tends to get closer to the root of H. This is a way to avoid also expensive host switch events; recall our heuristic in Section 2.9:

$$c_l \cdot slide\_length + c_s \leq c_h + c_l.$$

It will be interesting to provide other heuristics or prove some results which connect costs of the events to specific situations that can happen in a parsimonious reconciliation.

Problem 5. *Is it true the conjecture on the length of the sliding path or in which cases is it true? Can we deduce other similar results based on the costs of the events?*

To the best of our knowledge we do not know about any other results or works on this way.

In Theorem 12 we proved that in the host-switch set $\Theta(\varrho)$ of a reconciliation ϱ there is enough information to reconstruct the whole reconciliation ϱ. In Section 2.6 we define

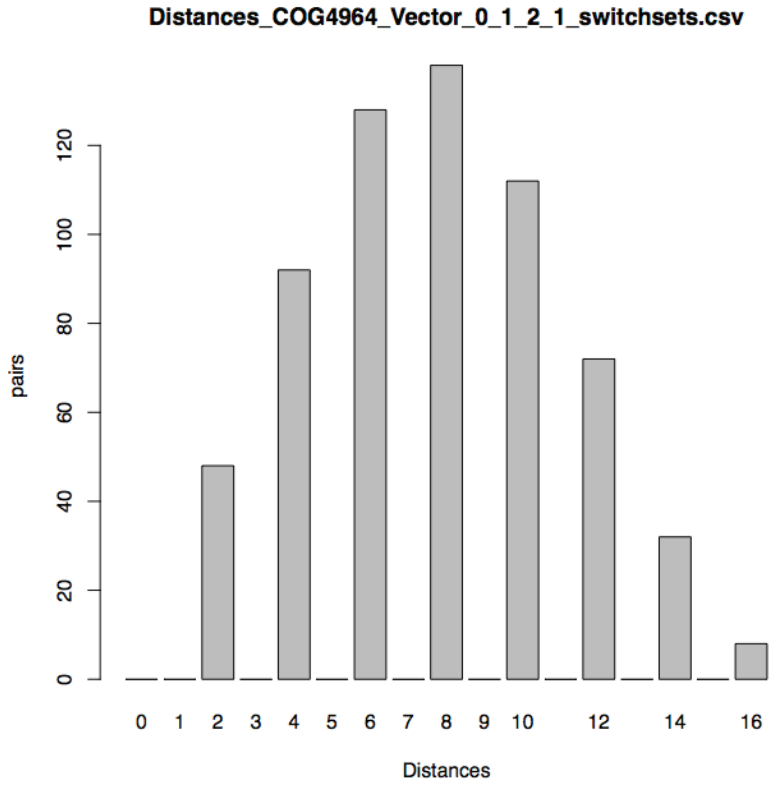

Figure 2.27: Histogram of distances $\mathsf{DH}(\varrho, \sigma)$ for each pair of optimal reconciliations $\varrho \neq \sigma$ of the dataset *COG4964* with cost vector (0,1,2,1).

two equivalence relations $\sim_1$ and $\sim_2$ on $\mathcal{R}(H, P, \varphi, c)$ based on host switch and aimed to reduce the number of reconciliations to treat. In Section 2.9 we showed that this relations reduce drastically the number of object to treat although we can easily retrieve any missing reconciliation. We have also seen that usually $\sim_2$ performs better than $\sim_1$ although in not all the cases and all cost vectors, finally, in some cases both are not performing any substantial reduction. On the comments of the experimental results we propose some explanation of this phenomenon basing on the cost of host switches and the *spread* of the leaf mapping function. From all this the following problems arise:

Problem 6. *Is there another equivalence class $\sim_3$ which gives a better reduction on the number of reconciliation to treat in the cases where $\sim_1$ and $\sim_2$ fail? Applying together $\sim_1 + \sim_2 + \sim_3$ will it be enough to reduce $\mathcal{R}(H, P, \varphi, c)$ to a single reconciliation?*

Problem 7. *Given the leaf mapping φ and calling* spread *a quantity which can measure the expected number of host switches (possibly employing the knowledge of the cost vector) how can we define it? Basing on the intuition that mapping nodes that are closer in P to nodes that are far in H should promote the existence of host switches, is it a good definition the following one:*

$$spread(\varphi) = \sum_{p,q \in L(P)} \frac{dist_H(\varphi(p), \varphi(q))}{dist_P(p, q)}$$

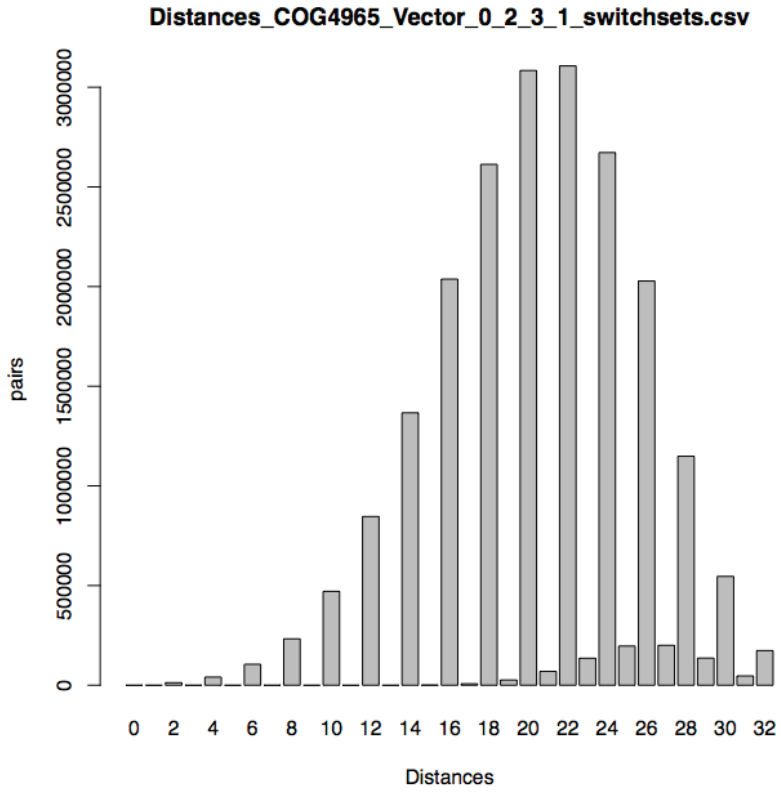

Figure 2.28: Histogram of distances $\mathsf{DH}(\varrho, \sigma)$ for each pair of optimal reconciliations $\varrho \neq \sigma$ of the dataset *COG4965* with cost vector (0,2,3,1).

In section 2.8 we introduce the reconciliation distance DH, also based on host switches. In section 2.9.2 we present some of its characteristics and, inspired by the approach of Robinson and Foulds [78] for the phylogenetic tree distance, we evaluate its goodness by observing its empirical distribution under many datasets and cost vector. We propose also other alternative definitions of this distance. We show and discuss the positive aspects of our distance w.r.t. to other already defined distances. However for a valuable evaluation of all these distances we think that further biological knowledge is require to confirm that the concept of *close by* reconciliations respect the biological perhaps still intuitive meaning of close reconciliation.

Problem 8. *Is* DH *catching the biological meaning of "close reconciliations" and is there a more deterministic way to evaluate its goodness than the comparative and probabilistic approach we used?*

Finally, still in section 2.8, we showed that by a sufficiently large values of k, $\mathcal{R}(H, P, \varphi, c)$ can be embedded into the k dimensional hypercube $\mathcal{H}^k$; it would be interesting to apply some dimension reduction techniques as correspondence analysis or some modification of Principal Component Analysis techniques to actually see the distribution of point on this space, with the hope to have a better insight.

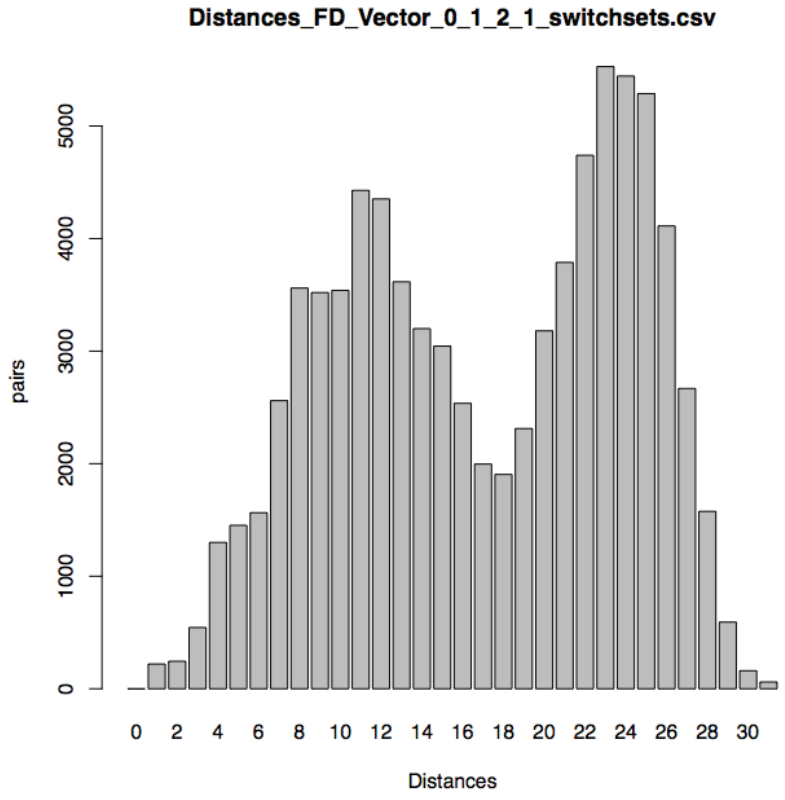

Figure 2.29: Histogram of distances $\mathsf{DH}(\varrho, \sigma)$ for each pair of optimal reconciliations $\varrho \neq \sigma$ of the dataset *FD* with cost vector (0,1,2,1).

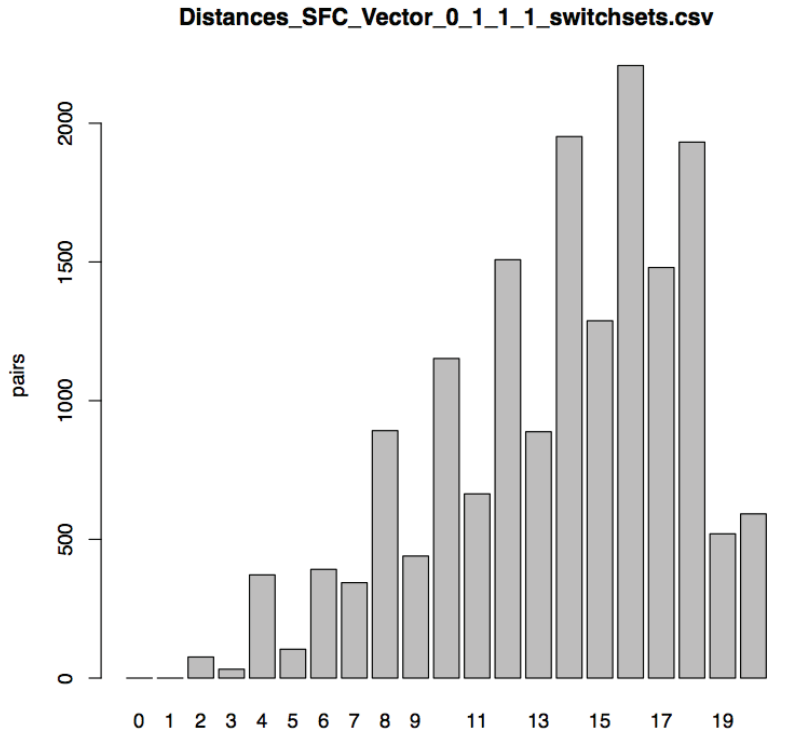

Figure 2.30: Histogram of distances $\mathsf{DH}(\varrho, \sigma)$ for each pair of optimal reconciliations $\varrho \neq \sigma$ of the dataset *SFC* with cost vector (0,1,1,1).

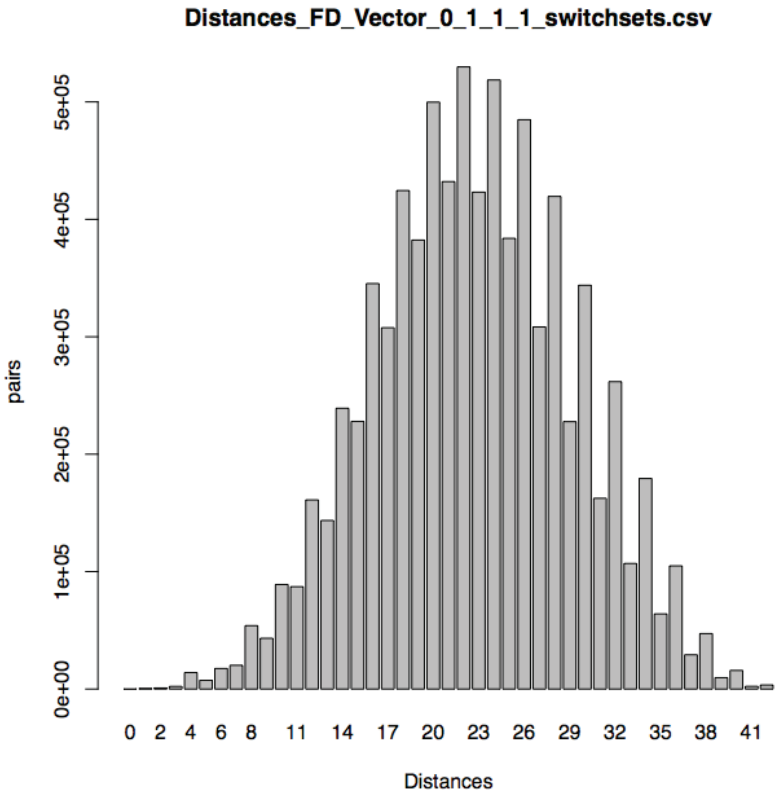

Figure 2.31: Histogram of distances $\mathsf{DH}(\varrho, \sigma)$ for each pair of optimal reconciliations $\varrho \neq \sigma$ of the dataset *FD* with cost vector (0,1,1,1).

Bibliography

[1] Amilhastre, J., Vilarem, M., and Janssen, P. (1998). Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete Applied Mathematics*, 86(2):125 – 144.

[2] Avis, D. and Fukuda, K. (1996). Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46. First International Colloquium on Graphs and Optimization.

[3] Bandi, C., Dunn, A. M., Hurst, G. D., and Rigaud, T. (2001). Inherited microorganisms, sex-specific virulence and reproductive parasitism. *Trends in Parasitology*, 17(2):88 – 94.

[4] Bansal, M., Alm, E., and Kellis, M. (2012). Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28.

[5] Baudet, C., Donati, B., Sinaimeri, B., Crescenzi, P., Gautier, C., Matias, C., and Sagot, M.-F. (2014). Cophylogeny reconstruction via an approximate bayesian computation. 64.

[6] Birmelé, E., Crescenzi, P., Ferreira, R., Grossi, R., Lacroix, V., Marino, A., Pisanti, N., Sacomoto, G., and Sagot, M.-F. (2012). *Efficient Bubble Enumeration in Directed Graphs*.

[7] Björklund, A., Husfeldt, T., and Koivisto, M. (2009). Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563.

[8] Bollobás, B. (1998). *Modern graph theory*, volume 184 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin.

[9] Borassi, M., Crescenzi, P., Lacroix, V., Marino, A., Sagot, M.-F., and Milreu, P. V. (2013). *Telling Stories Fast*, pages 200–211. Springer Berlin Heidelberg, Berlin, Heidelberg.

[10] Boros, E., Elbassioni, K., Gurvich, V., and Khachiyan, L. (2004). *Generating Maximal Independent Sets for Hypergraphs with Bounded Edge-Intersections*, pages 488–498. Springer Berlin Heidelberg, Berlin, Heidelberg.

[11] Bossan, B., Koehncke, A., and Hammerstein, P. (2011). A new model and method for understanding wolbachia-induced cytoplasmic incompatibility. *PLoS ONE*, 6.

[12] Brandstädt, A., Eschen, E. M., and Sritharan, R. (2007). The induced matching and chain subgraph cover problems for convex bipartite graphs. *Theoretical computer science*, 381(1):260–265.

[13] Breeuwer, J. (1997). Wolbachia and cytoplasmic incompatibility in the spider mites tetranychus urticae and t. turkestani. *Heredity*, 79.

[14] Breeuwer, J. and H. Werren, J. (1990). Microorganisms associated with chromosome destruction and reproductive isolation between two insect species. 346:558–60.

[15] Brightwell, G. and Winkler, P. (1991). Counting linear extensions. *Order*, 8(3):225–242.

[16] Calamoneri, T., Gastaldello, M., Mary, A., Sagot, M.-F., and Sinaimeri, B. (2016a). Finding and enumerating chain subgraphs and covers of bipartite graphs. *Theoretical Computer Science (manuscript under review)*.

[17] Calamoneri, T., Gastaldello, M., Mary, A., Sagot, M.-F., and Sinaimeri, B. (2016b). *On Maximal Chain Subgraphs and Covers of Bipartite Graphs*, pages 137–150. Springer International Publishing, Cham.

[18] Callaini, G., Dallai, R., and Riparbelli, M. (1997). Wolbachia-induced delay of paternal chromatin condensation does not prevent maternal chromosomes from entering anaphase in incompatible crosses of drosophila simulans. *Journal of Cell Science*, 110(2):271–280.

[19] Chang-Wu, Y., Gen-Huey, C., and Tze-Heng, M. (1998). On the complexity of the k-chain subgraph cover problem. *Theoretical computer science*, 205(1):85–98.

[20] Charlat, S., Calmet, C., Andrieu, O., and Merçot, H. (2005). Exploring the evolution of wolbachia compatibility types: a simulation approach. *Genetics*, 170.

[21] Charlat, S., Calmet, C., and Merçot, H. (2001). On the mod resc model and the evolution of wolbachia compatibility types. *Genetics*, 159.

[22] Charleston, M. (1998). Jungles: A new solution to the host/parasite phylogeny reconciliation problem. 149:191–223.

[23] Dias, V., M. Herrera de Figueiredo, C., and Szwarcfiter, J. (2005). Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337(1-3):240 – 248.

[24] Dias, V. M., de Figueiredo, C. M., and Szwarcfiter, J. L. (2007). On the generation of bicliques of a graph. *Discrete Applied Mathematics*, 155(14):1826 – 1832.

[25] Donati, B., Baudet, C., Sinaimeri, B., Crescenzi, P., and Sagot, M.-F. (2015). Eucalypt: efficient tree reconciliation enumerator. *Algorithms for Molecular Biology*, 10(1):3.

[26] Doyon, J., Chauve, C., and Hamel, S. (2009). Space of gene/species trees reconciliations and parsimonious models. *Journal of Computational Biology*, 16.

[27] Doyon, J.-P., Ranwez, V., Daubin, V., and Berry, V. (2011). Models, algorithms and programs for phylogeny reconciliation. 12:392–400.

[28] Doyon, J.-P., Scornavacca, C., Gorbunov, K., Szollosi, G., Ranwez, V., and Berry, V. (2010). An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications and transfers. 6398:93–108.

[29] Duron, O., Bernard, C., Unal, S., Berthomieu, A., Berticat, C., and Weill, M. (2006). Tracking factors modulating cytoplasmic incompatibilities in the mosquito culex pipiens. 15:3061–71.

[30] Eiter, T. and Gottlob, G. (1995). Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304.

[31] Engelstadter, J. and Telschow, A. (2009). Cytoplasmic incompatibility and host population structure. *Heredity*, 103.

[32] Engelstädter, J., Charlat, S., Pomiankowski, A., and D D Hurst, G. (2006). The evolution of cytoplasmic incompatibility types: integrating segregation, inbreeding and outbreeding. *Genetics*, 172.

[33] Ewacha, K., Rival, I., and Zaguia, N. (1997). Approximating the number of linear extensions. *Theoretical Computer Science*, 175(2):271 – 282.

[34] Felsner, S., Gustedt, J., and Morvan, M. (1998). Interval reductions and extensions of orders: Bijections to chains in lattices. *Order*, 15(3):221–246.

[35] Felsner, S. and Trotter, W. (2000). Dimension, graph and hypergraph coloring. *Order*, 17(2):167–177.

[36] Ferreira, R., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., and Sacomoto, G. (2012). Optimal listing of cycles and st-paths in undirected graphs.

[37] Ferreira, R., Grossi, R., and Rizzi, R. (2011). *Output-Sensitive Listing of Bounded-Size Trees in Undirected Graphs*, pages 275–286. Springer Berlin Heidelberg, Berlin, Heidelberg.

[38] Fomin, F. V. and Kratsch, D. (2010). *Exact Exponential Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA.

[39] Fredman, M. L. and Khachiyan, L. (1996). On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628.

[40] Gastaldello, M., Calamoneri, T., and Sagot, M.-F. (2017a). Characterize reconciliation by host swithes: A new distance and two equivalence relations based on the host-switch set of a reconciliation. *manuscript*.

[41] Gastaldello, M., Calamoneri, T., and Sagot, M.-F. (2017b). Extracting few representative reconciliations with host switches (extended abstract). *14th International Conference on Computational Intelligence Methods for Bioinformatics and Biostatistics*.

[42] Gastaldello, M., Limouzy, V., and Mary, A. (2017c). Computing the poset dimension forgetting about linear extensions. *manuscript*.

[43] Gely, A., Nourine, L., and Sadi, B. (2009). Enumeration aspects of maximal cliques and bicliques. 157:1447–1459.

[44] Ghelelovitch, S. (1952). Sur le determinisme genetique de la sterilite dans les croisements entre differentes souches de culex autogenicus roubaud. *Comptes Rendus Hebdomadaires des Seances de l'Academie des Sciences*, 234.

[45] Golovach, P. A., Heggernes, P., Kratsch, D., and Villanger, Y. (2015). An incremental polynomial time algorithm to enumerate all minimal edge dominating sets. *Algorithmica*, 72(3):836–859.

[46] Goodman, M., Czelusniak, J., William Moore, G., E. Romero-Herrera, A., and Matsuda, G. (1979). Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. 28:132.

[47] Gotoh T., Noda H., I. S. (2007). Cardinium symbionts cause cytoplasmic incompatibility in spider mites. *Heredity*, 98.

[48] Guigo, R., Muchnik, I., and Smith, T. (1996). Reconstruction of ancient molecular phylogeny. 6:189–213.

[49] Hammer, P. L., Peled, U. N., and Sun, X. (1990). Difference graphs. *Discrete Applied Mathematics*, 28(1):35 – 44.

[50] Hilgenboecker, K., Hammerstein, P., Schlattmann, P., Telschow, A., and Werren, J. H. (2008). How many species are infected with wolbachia? a statistical analysis of current data. *FEMS Microbiol. Lett.*, 281.

[51] Hunter, M., Perlman, S., and Kelly, S. (2003). A bacterial symbiont in the bacteroidetes induces cytoplasmic incompatibility in the parasitoid wasp encarsia pergandiella. *Philosophical transactions of the Royal Society of London. Series B: Biological sciences*, 270(1529):2185–2190.

[52] Johnson, D. S., Yannakakis, M., and Papadimitriou, C. H. (1988). On generating all maximal independent sets. *Information Processing Letters*, 27:119–123.

[53] Khachiyan, L., Boros, E., Elbassioni, K., Gurvich, V., and Makino, K. (2005). On the complexity of some enumeration problems for matroids. 19:966–984.

[54] Kiyomi, M. and Uno, T. (2006). Generating chordal graphs included in given graphs. E89D:763–770.

[55] Koda, Y. and Ruskey, F. (1993). A gray code for the ideals of a forest poset. *Journal of Algorithms*, 15(2):324 – 340.

[56] Laven, H. (1967). Eradication of culex pipiens fatigans through cytoplasmic incompatibility [27]. 216:383–4.

[57] Makino, K. and Uno, T. (2004). *SWAT 2004, Lecture Notes in Computer Science*, chapter New Algorithms for Enumerating All Maximal Cliques, pages 260–272. Springer Berlin Heidelberg, Berlin, Heidelberg.

[58] Marino, A. (2015). *Analysis and Enumeration: Algorithms for Biological Graphs*. Atlantis Studies in Computing. Atlantis Press.

[59] Marshall, J. F. (1938). *The British Mosquitoes*.

[60] Mary, A. (2013)). *Énumération des Dominants Minimaux d'un graphe*. PhD thesis.

[61] McMeniman, C. J., Lane, R. V., Cass, B. N., Fong, A. W., Sidhu, M., Wang, Y.-F., and O'Neill, S. L. (2009). Stable introduction of a life-shortening wolbachia infection into the mosquito aedes aegypti. *Science*, 323(5910):141–144.

[62] Merçot, H. and Charlat, S. (2004). Wolbachia infections in drosophila melanogaster and d. simulans: polymorphism and levels of cytoplasmic incompatibility. *Genetica*, 120.

[63] Mirkin, B., B. Muchnik, I., and Smith, T. (1995). A biologically consistent model for comparing molecular phylogenies. 2:493–507.

[64] Moon, J. W. and Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28.

[65] Nor, I., Engelstädter, J., Duron, O., Reuter, M., Sagot, M.-F., and Charlat, S. (2013). On the genetic architecture of cytoplasmic incompatibility: inference from phenotypic data. *The American Naturalist*, 182(1):E15–E24.

[66] Nor, I., Hermelin, D., Charlat, S., Engelstädter, J., Reuter, M., Duron, O., and Sagot, M.-F. (2010). Mod/resc parsimony inference: Theory and application. 213:23–32.

[67] Ono, A. and Nakano, S.-i. (2005). *Constant Time Generation of Linear Extensions*, pages 445–453. Springer Berlin Heidelberg, Berlin, Heidelberg.

[68] Orlin, J. (1977). Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406 – 424.

[69] Osborne, S., San Leong, Y., O'Neill, S., and Johnson, K. (2009). Variation in antiviral protection mediated by different wolbachia strains in drosophila simulans. 5:e1000656.

[70] Ovadia, Y., Fielder, D., Conow, C., and Libeskind-Hadas, R. (2011). The cophylogeny reconstruction problem is np-complete. 18:59–65.

[71] Page, R. (1994a). Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. 43.

[72] Page, R. (1994b). Parallel phylogenies: Reconstructing the history of host-parasite assemblages. *Cladistics*, 10.

[73] Page, R. D. and Charleston, M. A. (1998). Trees within trees: phylogeny and historical associations. *Trends in Ecology & Evolution*, 13(9):356 – 359.

[74] Peled, U. N. and Sun, F. (1995). Enumeration of difference graphs. *Discrete Applied Mathematics*, 60(1):311 – 318.

[75] Poinsot, D., Charlat, S., and Merçot, H. (2003). On the mechanism of wolbachia-induced cytoplasmic incompatibility: Confronting the models with the facts. 25:259–65.

[76] Razgon, I. (2011). *Computing Minimum Directed Feedback Vertex Set in* $O^*(1.9977^n)$, pages 70–81.

[77] Rigaud, T. (1997). Inherited microoraganisms and sex determination of arthropod hosts. in: Influential passengers: Inherited microorgan isms and arthropod reproduction. *Oxfor University Press*.

[78] Robinson, D. and Foulds, L. (1981). Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131 – 147.

[79] S. Hafner, M. and A. Nadler, S. (1988). Phylogenetic trees support the coevolution of parasites and their hosts. 332:258–9.

[80] Stolzer, M., Lai, H., Xu, M., Sathaye, D., Vernot, B., and Durand, D. (2012). Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. 28:i409–i415.

[81] Strozecki, Y. (2010). *Enumeration complexity and matroid decomposition*.

[82] Tofigh, A., Hallett, M., and Lagergren, J. (2011). Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(2):517–535.

[83] Trotter, W. (1981). Stacks and splits of partially ordered sets. 35:229–256.

[84] Trotter, W. (1992). *Combinatorics and Partially Ordered Sets: Dimension Theory*.

[85] Tsukiyama S., Ide M., A. H. and Y., S. (1977). A new algorithm for generating all the maximal independent sets. *SIAM J. on Computing*, (6):505–517.

[86] Uno, T. (2001). A fast algorithm for enumerating bipartite perfect matchings. In *Proceedings of the 12th International Symposium on Algorithms and Computation*, ISAAC '01, pages 367–379, London, UK, UK. Springer-Verlag.

[87] Vavre, F., Fleury, F., Varaldi, J., Fouillet, P., and Bouletreau, M. (2000). Evidence for female mortality in wolbachia-mediated cytoplasmic incompatibility in haplodiploid insects: epidemiologic and evolutionary consequences. *Evolution*, 54.

[88] Werren, J. H. (1997). Biology of *Wolbachia*. *Annual Review of Entomology*, 42.

[89] Wieseke, N., Bernt, M., and Middendorf, M. (2013). Unifying parsimonious tree reconciliation. *Lecture notes in Computer Science*, 8126.

[90] Wieseke, N., Hartmann, T., Bernt, M., and Middendorf, M. (2015). Cophylogenetic reconciliation with ilp. 12:1–1.

[91] Yannakakis, M. (1982). The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358.

[92] Yao-ban, C. Y., Ranwez, V., and Scornavacca, C. (2015). Exploring the space of gene/species reconciliations with transfers. *Journal of Mathematical Biology*, 71.