Order number: 7515

PhD

of

Paris–South University

to obtain a

Paris–South University PhD degree

Speciality: Computer Science – Artificial Intelligence

By

Alexandre Termier

Extraction of frequent trees in an heterogeneous corpus of semi-structured data: application to XML documents mining

Defended on 2004 the 28th of April with the jury:

Mr.	Serge Abiteboul	Director of Research	Jury president
Mrs	Marie-Christine Rousset	Professor	Advisor
Mrs	Michele Sebag	Director of Research	Advisor
Mr.	Marc Sebban	Professor	Reviewer
Mr.	Hannu Toivonen	Professor	Reviewer
Mr.	Jean-Daniel Zucker	Professor	Reviewer

Laboratoire de Recherche en Informatique, U.M.R. CNRS 8623,
Université Paris-Sud, 91405 Orsay Cedex, France

To Sawako

Contents

Chapter I

Introduction *(shortened)*

An important Data Mining task is the discovery of frequent itemsets. It consists in finding, in a propositional database, sets of items that appear in more than ε transactions. A typical application is the market basket analysis: each transaction is a sale report, and the items are the products sold. Finding the products that are often sold together is a valuable information for sales managers. A lot of works have addressed this problem in the last ten years ([AS94], [SON95], [Toi96], [LK98], [PBTL99] among others).

Nowadays, the structure of data is changing. More and more available data from telecommunications, chemistry, genomics are *structured* as trees or graphs. Finding frequent structures is still important, but the previous algorithms can no longer be used.

Two solutions can be devised:

- Either use a general approach, that find frequent structures in a language expressive enough to represend all kinds of structured data. It's the idea of the Warmr system ([Deh98], [DT99]).

- Or use an approach specific to one data structure, hence optimised for this structure. This was done for graphs by [IWM00], and for trees by [AAK$^+$02] and [Zak02] (restricted to find frequent trees whose siblings order is the same as in the data).

As expected, the specific approaches are more efficient than Warmr on the data structure they are specific of.

Our motivation is to be able to find efficiently frequent structures in collections of XML documents. XML documents can be modeled as labelled ordered trees. The labels are the tags of the nodes, and the tree structure comes from the XML tag nesting.

So we are interested in approaches specific to trees. The approach of [AAK$^+$02] finds frequent structures that have always exactly the same shape in the data, like shown in figure I.1.

The approach of [Zak02] allows some changes in the label nesting, handling some heterogeneity in the data (see figure I.2).

However, when looking at figures I.1 and I.2, we can notice that the node *editor* is never taken into account in the frequent trees. It's because for the algorithms of [AAK$^+$02] and [Zak02], the order of the siblings in the frequent trees and in the documents must be the same. And the *editor* node is once on the rightmost position (tree A_1), once on the leftmost position (tree A_2). So it cannot be seen as frequent by these algorithms.

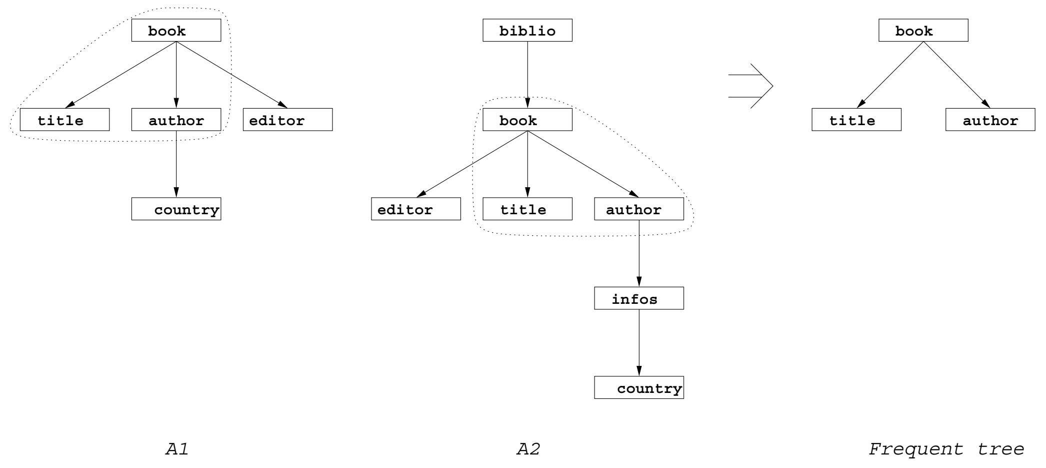

Figure I.1: Biggest tree found by Find-Freq-Trees [AAK+02] from A_1 and A_2 ($\varepsilon = 2$)

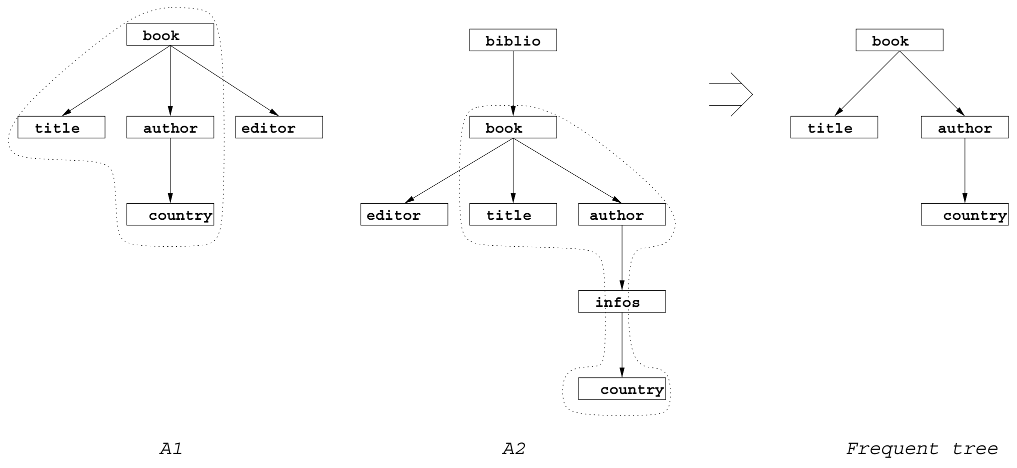

Figure I.2: Biggest tree found by TreeMiner [Zak02] from A_1 and A_2 ($\varepsilon = 2$)

But the information that a *book* have an *editor* is always present. We think that this is an important information, that must be reflected in the frequent trees our approach will find (see figure I.3).

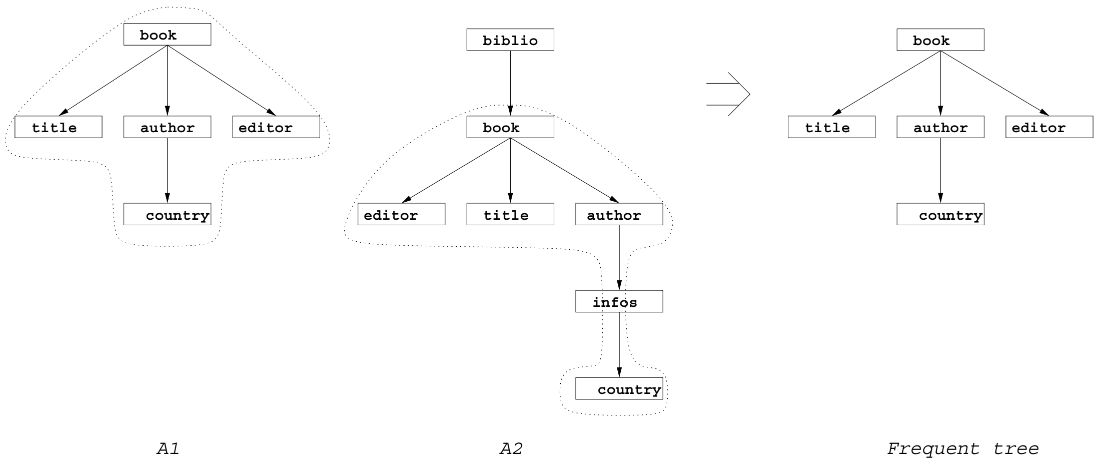

Figure I.3: Biggest frequent tree found by our algorithms with A_1 and A_2 ($\varepsilon = 2$)

Contributions of this PhD

Our goal is to find frequent trees in labelled ordered trees, extending the previous works from Asai and Zaki by allowing a greater flexibility in the frequent structures to find.

The algorithms we have designed allow differences in tag nesting (like [Zak02]), and do not take into account the order of the siblings in the documents. Hence our algorithms are adapted to find frequent trees in heterogeneous collections of XML documents.

In such collections, several tags can refer to the same concept. For example, a car can be addressed with tags such as: *car, automobile, motor* (synonyms). For our frequent tree discovery algorithm to be as efficient as possible, ideally the problems of *synonymy* and *polysemy* should be solved. However, this represents a different research topic, and we have decided not to focus on this task. For some preliminary ideas, the interested reader can read [TRS01].

For the rest of this thesis, we suppose we have a preprocessor unifying the tags refering to the same concept, so we have neither synonymy nor polysemy problems. The processing chain is shown in figure I.4.

The frequent trees we consider are more general than those considered in previous works. This comes with an higher algorithmic complexity: hence Kilpelinen shown in [Kil92] that if the problem of testing tree inclusion in the ordered case was polynomial, it was NP-complete in the unordered case that we address.

In order to be efficient despite this bigger difficulty, we have explored two different approaches: the first one resulted in the developement of the TreeFinder algorithm ([TRS02]), the second one lead to the creation of the algorithms Dryal and Dryade.

TreeFinder: approximation of the set of frequent trees

Because of the generality of the researched structures, our algorithms can be confronted with huge search spaces. If the data are too numerous or have too complex frequent trees, this space can become so huge that finding frequent trees in a reasonable amount of time will become impossible. Then, a *complete* approach, i.e. finding all the frequent trees, can be too expensive. Hence we

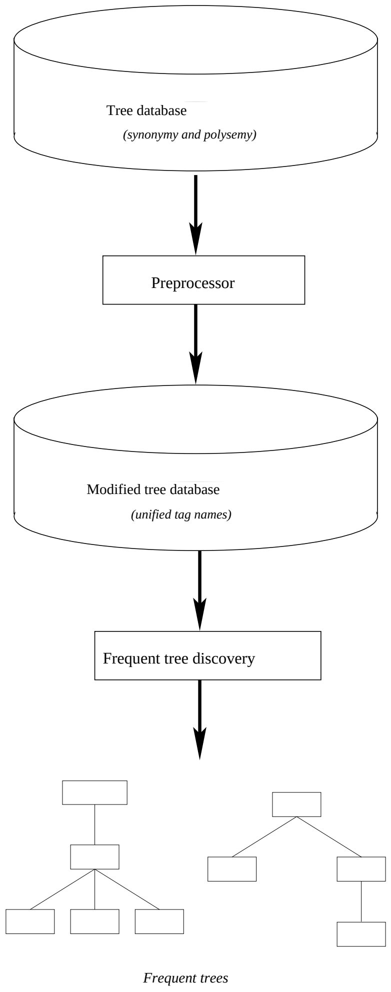

Figure I.4: Processing chain to find frequent trees from an heterogeneous collection of XML documents

developped an *approximate* algorithm, TreeFinder, that finds only one part of all the frequent trees that are in the data. For this algorithm, we made an qualitative and quantitative study of the frequent trees found, and also of those not found. This allows to estimate the quality of our approximation.

Dryal and Dryade: a new complete approach for searching frequent trees

The Dryal and Dryade algorithms make use of the specific features of the tree structure to be efficient, and are based on the two following principles.

Constructive approach:

Our approach is a levelwise approach, that first creates frequent trees of depth 1, then assembles them to get frequent trees of depth 2, and so on until finding all the frequent trees. We were inspired by the recent *vertical* methods (see [ZPOL97]), which are often more efficient than the classical Apriori-like approaches (called *horizontal*). Vertical approaches are usually more adapted to the discovery of complex frequent itemsets.

Divide to reformulate:

An usual Computer Science way of solving a complex problem is called *"divide and conquer"*. The problem is split into simpler subproblems, and from the solutions of the subproblems the solution of the general problem is found. We were inspired by this method to solve our problem in a way that the subproblems come, after a *reformulation*, to the computation of **frequent itemsets in transactional data**. Thanks to these reformulations, our approach benefits from the considerable amount of work that was devoted to the problem of finding frequent itemsets in transactional data. Any improvement on these techniques can be used to improve our approach. We also present in this thesis the concept of *closed frequent tree*, which was first presented in the CMTreeMiner system [CYXM04], an algorithm developped concurrently to Dryade and addressing a simpler frequent tree discovery problem. The use of closed trees allows exponential performance improvement without loss of information.

Restriction

To be able to cope with a huge search space, we impose for our algorithms Dryal and Dryade a restriction on the frequent trees that we can discover: these are the trees where a node cannot have two children with same label. To discriminate them, we call such frequent trees *frequent patterns*. They correspond to XML query patterns proposed in [DRR$^+$03].

This restriction avoids a too big combinatorial explosion. In chapter III, we will outline when the restriction is used in the algorithms, and the complexity gain.

The figure I.5 illustrates the kind of frequent trees we cannot find.

The frequent tree *a*) corresponds to a journal with an article having a picture and text and an article with a drawing and text. With the restriction we impose, we cannot find tree *a*), but we find the two frequent trees of *b*). The frequent tree *a*) is more precise and corresponds to a fine-grained analysis of the XML documents : we can differentiate several kinds of journals for example. The two frequent trees of *b*) correspond to a coarser-grained analysis, representing a little information loss compared to *a*). But they are far less time-expensive to obtain, which can be crucial for some applications.

Outline

To conclude this introduction, we give the outline of this thesis:

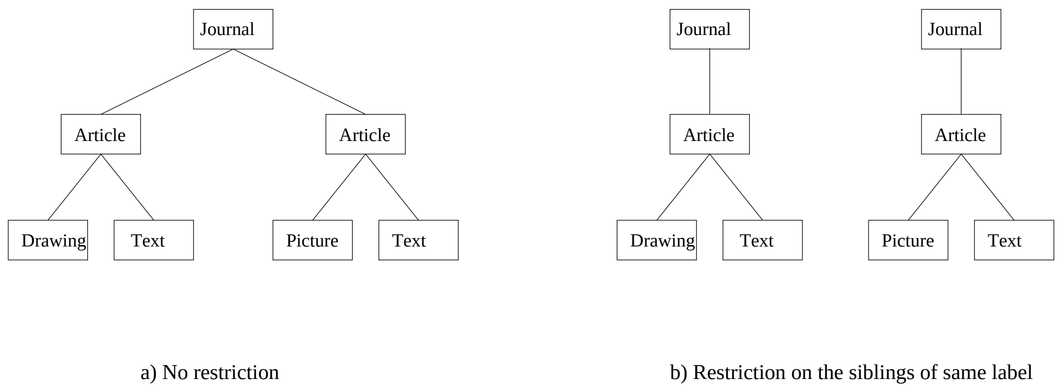

Figure I.5: Frequent trees discovered: a) without restriction b) with the restriction on siblings labels that we impose

- The **Chapter II** presents formally our problem.
- *Not translated: state of the art, with two parts, the first on on frequent itemset discovery, and the second one about related works on frequent trees by [AAK$^+$02] and [Zak02]*
- *Not translated:* TreeFinder*. It is a french translation of our ICDM'02 article [TRS02]*
- The **Chapter III** is about the algorithms Dryal and Dryade, which are sound and complete algorithm for finding frequent patterns. The algorithms are detailed, soundness and completeness proofs are given.
- The **Chapter IV** presents an experimental study of Dryade's performances. The study was realised with artifical and real data, and Dryade was compared to Warmr.
- The **Chapter V** concludes the thesis and gives some perspectives.

Chapter II

Definitions

In this chapter, we will define formally the data we are working on, as well as the problem we address in this thesis.

1 Labelled ordered trees: definitions

The data type we will nearly always deal with in this thesis is the *tree*. Hence, we will define in this section a theoretical framework adapted to trees, and adapted to the discovery of frequent patterns into trees. The theoretical framework we define is very common, and takes as a starting point the framework defined into Pekka Kilpelinen's PhD [Kil92].

Definition 1 (Binary relation) *Let D be a set. A* binary relation *R over D is a subset of the Cartesian product $D \times D$.*

Definition 2 (Transitive closure) *The* transitive closure *of R, written as R^+, is defined by:*

$$\begin{aligned}
R^1 &= \{(x,y) \mid (x,y) \in D^2\} \\
R^{n+1} &= \{(x,y) \mid \exists z \in D \ st \ (x,z) \in R \ and \ (z,y) \in R^n\}, \quad n > 0 \\
R^+ &= \bigcup_{n>0} R^n
\end{aligned}$$

Definition 3 (Tree) *A tree is defined by the 3-tuple $T = (N_T, A_T, root(T))$ where N_T is a set of nodes, $A_T \subseteq N_T^2$ is a binary relation over N_T, and $root(T) \in N_T$ is a specific node called the* ***root****. For any pair $(u,v) \in A_T$, we say that (u,v) is an* arc *of the tree, and that u is the* parent *of v, written as $u = parent(v)$. A_T must satisfy:*

- *$root(T)$ has no parent.*
- *Every node of the tree (except the root) has exactly one parent.*
- *From the root, it is possible to reach any node in the tree by following arcs in A_T, i.e. for all node $v \in N_T$ except the root, $(root(T), v) \in A_T^+$.*

Notations : Let T be a tree. Let $u \in N_T$ be a node of T. All the nodes of T having u as parent will be called the *children* of u :

$$children(u) = \{v \in N_T \ | \ (u, v) \in A_T\}$$

We define the *descendants* of u by :

$$descendants(u) = \{v \in N_T \ | \ (u, v) \in A_T^+\}$$

ande the *ancestors* of u by :

$$ancestors(u) = \{v \in N_T \ | \ (v, u) \in A_T^+\}$$

An arc sequence in A_T $(u_1, u_2), ..., (u_{n-1}, u_n)$ from u_1 to u_n is called a *path* between u_1 and u_n. This path length is $n - 1$.

The nodes of N_T having no children are called *leaves*. The other nodes of N_T are called *internal nodes*.

We will call *depth* of the tree T, written as $depth(T)$, 1 plus the length of the longest path from the root of T to a leaf of T.

Definition 4 (Subtree) *Let T be a tree. Let $u \in N_T$ be a node of T. The subtree of T having u as root, written as $T[u]$, is the tree $T[u] = (N_{T[u]}, A_{T[u]}, u)$ such as:*

- $N_{T[u]} = \{u\} \cup descendants(u)$
- $A_{T[u]} = A_T \cap (N_{T[u]} \times N_{T[u]})$.

Definition 5 (Labelled tree) *Let E be a label set. A tree labelled by E is defined by $(N, A, root(T), \varphi)$ such as:*

- *$(N, A, root(T))$ is a tree.*
- *$\varphi : N \mapsto E$ is an application mapping each node of N to a label in E.*

Definition 6 (Ordered tree) *An ordered tree is defined by $(N_T, A_T, root(T), \preceq)$ such as:*

- *$(N_T, A_T, root(T))$ is a tree.*
- *$\prec \subseteq N^2$ is the* sibling *relation, it's a binary relation defining a partial order on N_T^2 and a total order on the children of any node of N_T. For an internal node u, its children $u_1, ..., u_n$ $(n \geq 0)$ are ordered from left to right such as $u_1 \prec u_2 \prec ... \prec u_n$.*

  According to this order, u_i is called u's i-th child.

Notations : Let T be an ordered tree, and let u be an internal node of T having n children $u_1, ..., u_n$. The children have the order: $u_1 \prec u_2 \prec ... \prec u_n$. Then for all $i \in [1, n]$ we write:

- $\forall j < i$ u_j is a *left sibling* of u_i $(i > 1)$.
- u_{i-1} is a *immediate left sibling* of u_i $(i > 1)$.
- $\forall j > i$ u_j is a *right sibling* of u_i $(i < n)$.

- u_{i+1} is a *immediate right sibling* of u_i $(i < n)$.

In the following, all the trees considered are, unless otherwise stated, labelled ordered trees. We then use the word **tree** instead of "labelled ordered trees".

Node identification: The nodes of a tree can be identified by assigning them numbers, each node having a unique number determined by its position in the tree.

The two main ways to assign numbers to nodes in a tree are the *prefix order* and the *postfix order.*

Definition 7 (Prefix order) *Let T be a tree. To a node $u \in N_T$ we can assign a number in the **prefix** order $pre(u)$ with the following rules:*

- $pre(root(T)) = 1$
- *The leftmost child of u has the prefix number of u more 1 : $pre(u_1) = pre(u) + 1$*
- *Let v be the immediate left sibling of u. Let p be the biggest prefix number assigned in $T[v]$. Then $pre(u) = p + 1$.*

Definition 8 (Postfix order) *Let T be a tree. To a node $u \in N_T$ we can assign a number in the **postfix** order $post(u)$ with the following rules:*

- *Let u be the leftmost leaf of T. Then $post(u) = 1$.*
- *Let u be an internal node of T, and let p be the biggest postfix number assigned to a descendant of u. Then $post(u) = p + 1$.*
- *Let v be the immediate right sibling of u. Then the postfix number of the leftmost leaf of $T[v]$ is $post(u) + 1$.*

One can notice that the prefix (postfix) order comes to number the tree nodes during a prefix (postfix) traversal.

2 Tree inclusion

There many ways to define the inclusion between two trees T_1 and T_2. For example, in his PhD [Kil92], Kilpelinen defines 10 different tree inclusion relations.

All these definitions are based on the existence of a *mapping* from the nodes of T_1 towards the nodes of T_2. This mappings strongly or weakly preserves the labels of the nodes and the tree structure.

Differences come from the preservation properties that are considered, and from the injectivity or non-injectivity of the mapping.

In the following definition, we give all the inclusion definitions that we will use in this thesis. We will then show the differences between these inclusion definitions on examples.

Definition 9 ((strict)(exact)(ordered)(weak) tree inclusion) *Let T_1 and T_2 be two labelled ordered trees, T_1 is included in T_2, written as $T_1 \sqsubseteq T_2$, if there exists a mapping $\mu : N_{T_1} \mapsto N_{T_2}$ such as:*

1. *μ preserves labels, strictly (1.a) or according to an order $\leq_E$ on labels (1.b)*

1.a) strict preservation: $\forall u \in N_{T_1}\ \ \varphi_{T_1}(u) = \varphi_{T_2}(\mu(u))$

1.b) order-bound prservation : $\forall u \in N_{T_1}\ \ \varphi_{T_1}(u) \leq_E \varphi_{T_2}(\mu(u))$

If the preservation is strict (1.a)), we say that the inclusion is ***strict****.*

2. *μ preserves the parent relationship (2.a) or the ancestor relationship (2.b).*

2.a) parent relationship preservation : $\forall u, v \in N_{T_1}$ *if* $(u, v) \in A_{T_1}$ *then* $(\mu(u), \mu(v)) \in A_{T_2}$

2.b) ancestor relationship preservation : $\forall u, v \in N_{T_1}$ *if* $(u, v) \in A_{T_1}$ *then* $(\mu(u), \mu(v)) \in A^+_{T_2}$

If the parent relationship is preserved (2.a), we say that the inclusion is ***exact****.*

Optionnaly:

3. *μ preserves children order :* $\forall u, v \in N_{T_1}$ *if* $u \preceq_{T_1} v$ *then* $\mu(u) \preceq_{T_2} \mu(v)$*. If the children order is preserved, we say that the inclusion is* ***ordered****.*

4. *μ is injective :* $\forall u, v \in N_{T_1}$ *if* $u \neq v$ *then* $\mu(u) \neq \mu(v)$*. If injectivity isn't satisfied, we say that the inclusion is* ***weak****.*

This general definition allows us to build many variants. The most constrained inclusion definition is the **strict exact ordered inclusion**.

The figure II.1 gives the example of a tree T_1 included in a tree T_2 by this definition. But the trees T_3 to T_7 from the figures II.2 to II.6 are not included in T_2 by this definition.

However, there is a **strict exact inclusion** between T_3 and T_2 (fig. II.2), and a **strict ordered inclusion** between T_4 and T_2 (fig. II.3).

Between T_5 and T_2, there is a **strict inclusion** (fig. II.4).

And between T_6 and T_2, we can fing a **strict weak inclusion** (fig. II.5).

Last, the figure II.6 is an example of an **exact ordered inclusion** using a label order $\leq_E$ such as *in-folio* $\leq_E$ *livre*.

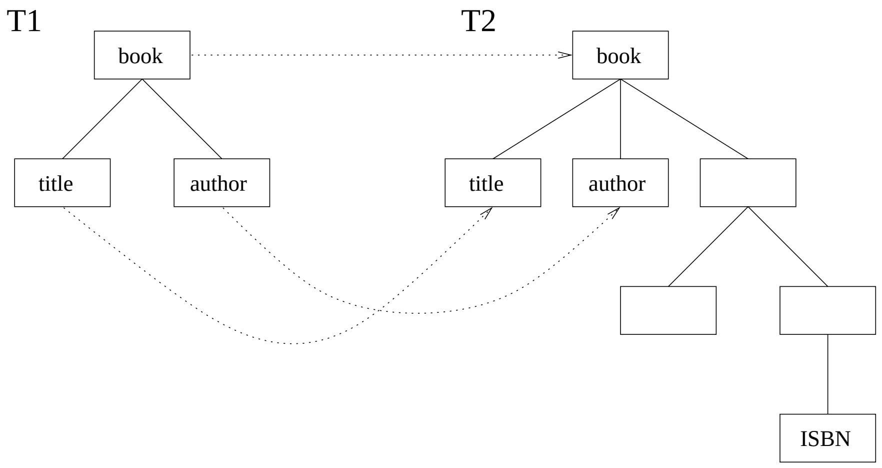

Figure II.1: Strict exact ordered inclusion

Note that for sake of legibility, we didn't show node identifiers in the figures II.1 to II.6. We'll introduce node identifiers only when they are really necessary.

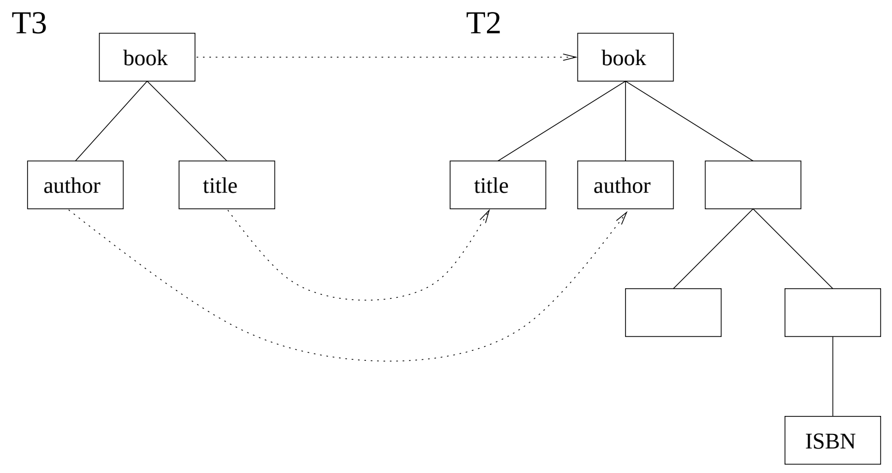

Figure II.2: Strict exact inclusion (not ordered)

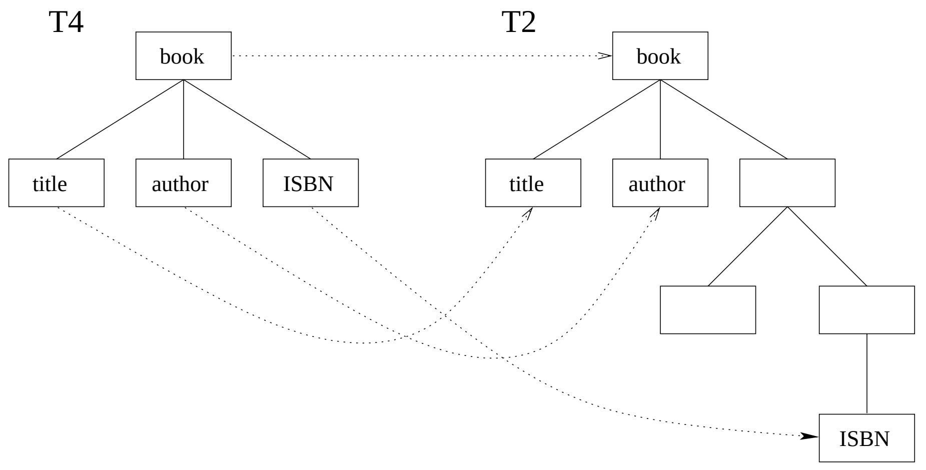

Figure II.3: Strict ordered inclusion (not exact)

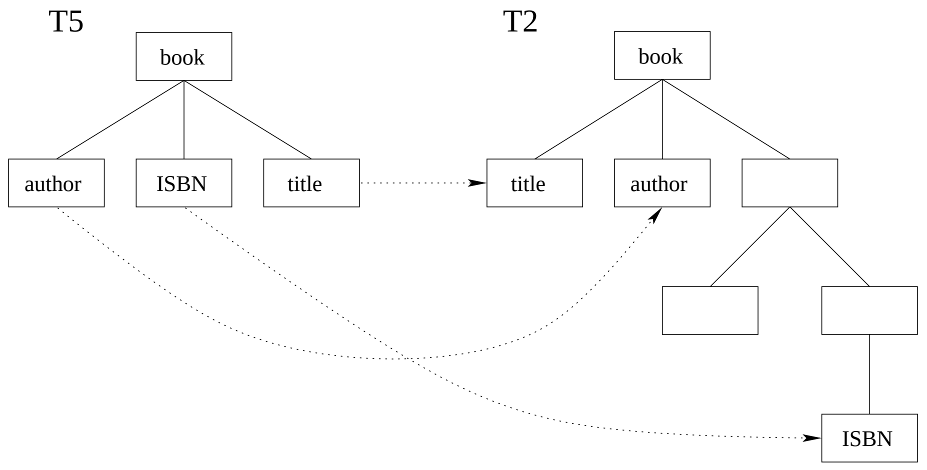

Figure II.4: Stricte inclusion (not exact, not ordered)

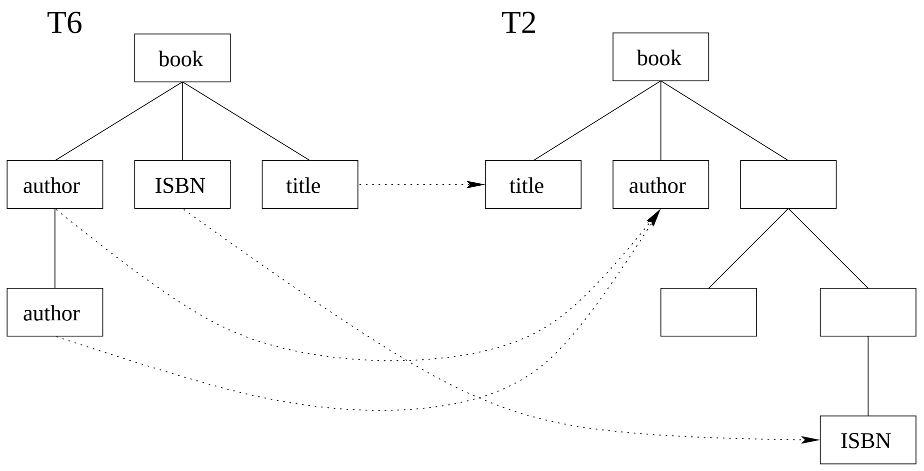

Figure II.5: strict weak inclusion (not ordered, not exact)

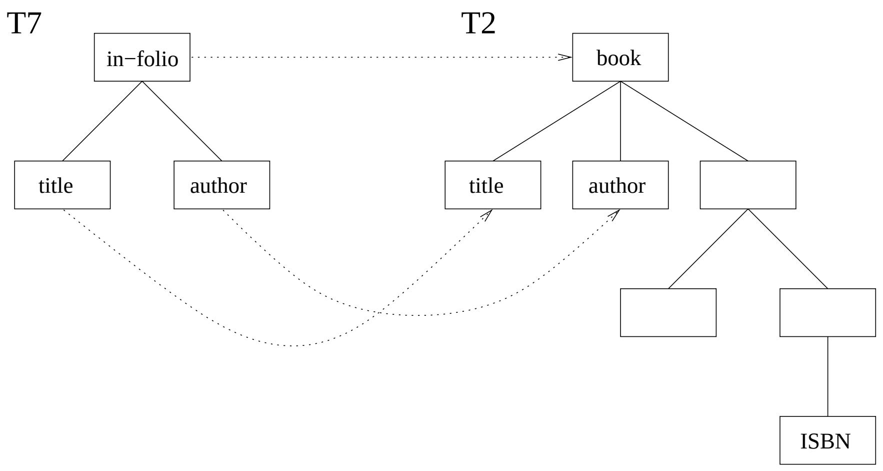

Figure II.6: Not strict but exact and ordered inclusion

The probleme of the tree inclusion test is a difficult problem. Determining if a tree is included in another tree is either polynomial (ordered inclusions) or NP-complete (other inclusions). A detailled complexity reckoning, as well as tree inclusion algorithms, can be found in [Kil92].

Notation : For a given tree inclusion $\sqsubseteq$, the set of mappings between T_1 and T_2 will be written $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$.

The general problem of tree inclusion is to find, given two trees T_1 and T_2, all the subtrees of T_2 that are occurrences of T_1. We then say that the root of T_2 subtree that is an occurrence of T_1 is a *root occurrence* of T_1. The general problem of tree inclusion is more difficult than the tree inclusion test between two trees. All the occurrences of tree T_1 in tree T_2 have to be found, whereas doing the test simply means proving the existence of one of them.

We focused on a restriction of this problem, which is finding all the subtrees of T_2 that are occurrences of T_1 **with a different root**.

This is the same as finding all the root occurrences of T_1 in T_2.

Definition 10 (Root occurrence) *Let $\sqsubseteq$ be an inclusion definition between trees, and let T_1 and T_2 be two trees such as $T_1 \sqsubseteq T_2$. Let μ be a mapping from T_1 to T_2. We call* ***root occurrence*** *of T_1 for μ the node u of T_2 such that $u = \mu(root(T_1))$.*

The set of root occurrences of T_1 in T_2 is called $Locc_{\sqsubseteq}(T_1, T_2)$ and is defined by:

$$Locc_{\sqsubseteq}(T_1, T_2) = \{u \in N_{T_2} \mid u = \mu(root(T_1)) \text{ with } \mu \in \mathcal{EM}_{\sqsubseteq}(T_1, T_2)\}$$

Note that for a root occurrence u there can be several mappings μ satisfying the previous conditions. The set of these mappings is written $\mathcal{EM}_u(T_1, T_2)$.

Property 1 *We have:*

$$\mathcal{EM}_{\sqsubseteq}(T_1, T_2) = \bigcup_{u \in Locc_{\sqsubseteq}(T_1, T_2)} \mathcal{EM}_u(T_1, T_2)$$

Proof : Straightforward because $Locc(T_1, T_2)$ gives us all the roots of mappings, for each root $u \in Locc(T_1, T_2)$, $\mathcal{EM}_u(T_1, T_2)$ represents all the mappings associated to this root. So the union of the $\mathcal{EM}_u(T_1, T_2)$ gives $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$.

Remark 1 *In the following, when no confusion is possible we will use the word* occurrence *of a tree T for a root occurrence of T.*

3 Frequent trees

We have seen in the introduction of this thesis that semi-structured documents can be represented as trees. Hence a collection of such documents can be represented by a set of trees, that we will regroup in a structure called *tree database.*

Definition 11 (Tree database) *Let $F = \{T_1, ..., T_m\}$ be a set of trees, and let $\Delta = \{id_1, ..., id_m\}$ be a set of identifiers such as forall $i \in [1, m]$ we associate the tree T_i to the identifier id_i.*

A ***tree database*** *$\mathcal{BA}$ is defined by the triple (D, Δ, δ), with:*

- *D is a labelled ordered tree with labels in E, called* ***data tree****. The root r of D is a specific node, whose label e_0 is not in E. The children of D are the trees $T_1, ..., T_m$: $\forall i \in [1, m]$ $D[r_i] = T_i$ (where r_i is the root of T_i).*

- *$\delta : N_T \backslash r \mapsto \Delta$ is a function for identifying tree nodes. Let $D[r_i]$ be a subtree whose root is the i-th child of r. Then δ is the function such as for all $v \in N_{D[r_i]}$ we have $\delta(v) = id_i$.*

The figure II.7 presents a set of trees and the corresponding tree database.

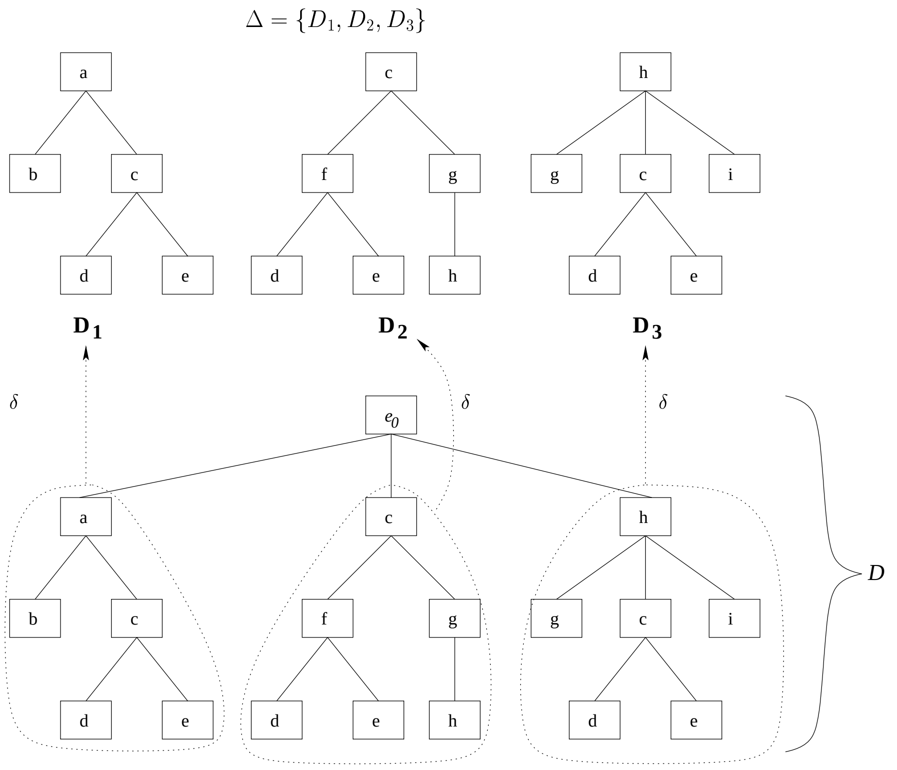

Figure II.7: A tree database example

Definition 12 (Projected of a tree in a datatree) *Let $\mathcal{BA} = (D, \Delta, \delta)$ be a tree database, $\sqsubseteq$ an inclusion definition and P a tree such as $P \sqsubseteq D$.*

*The **projected** of P in D, named $projected(P, D)$, is the datatree whose root is the root of D, and in which each node n is a node of D which is the image of a node of P by a mapping of $\mathcal{EM}_{\sqsubseteq}(P, D)$. In $projected(P, D)$, we have $n_1 = parent(n_2)$ if in D, $n_1 = ancestor(n_2)$ and if there is no node of $projected(P, D)$ in the path from n_1 to n_2 in D.*

Property 2 *By construction, this projected is unique.*

The projected of a tree P in the datatree of figure II.7 is shown in figure II.8.

We will see in chapter III that for two trees P and P' such as $P' \sqsubseteq P$, we may have to compute the occurrences of P' in the projected of P, in order to compare them to the occurrences of P' in D. In the specific case where all the mappings between P' and P send the root of P' on the root of P (i.e. $Locc_{\sqsubseteq}(P', P) = \{racine(P)\}$), the set of occurrences of P' in the projected of P in D is the same as the set of occurrences of P in D. This property is formalised below and will be used in the algorithms of chapter III.

Property 3 *Let $\mathcal{BA} = (D, \Delta, \delta)$ be a tree database, $\sqsubseteq$ an inclusion definition, P and P' two trees such as $P' \sqsubseteq P \sqsubseteq D$.*

If $Locc_{\sqsubseteq}(P', P) = \{root(P)\}$ then $Locc_{\sqsubseteq}(P', projected(P, D)) = Locc_{\sqsubseteq}(P, D)$.

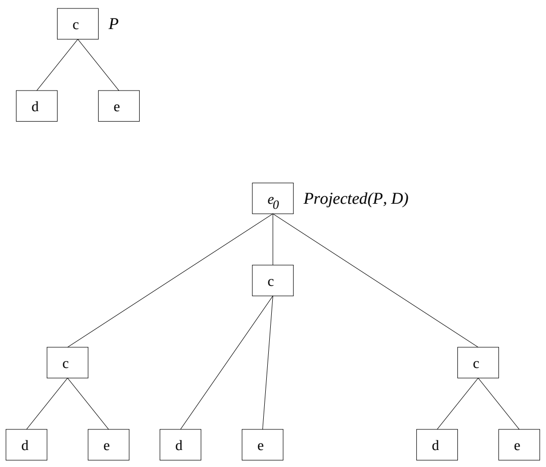

Figure II.8: Projected of a tree P in the datatree D of figure II.7

The problem we are interested in is to find frequent tree patterns in a tree database. A frequent tree pattern (called *frequent tree* in the following), is a tree that has for a given inclusion a number of occurrences greater than a threshold defined at the start.

For a given inclusion definition, there are several ways to define tree frequency. One can define:

- the occurrence frequency, based on the number of tree occurrences in the datatree.
- the identifier frequency, counting the number of distinct trees (having different identifiers) of the datatree in which in tree is included.

The difference between these two frequencies is that the first one is only based on the number of occurrences of the tree in the datatree, whereas the second imposes that the occurrences appear in distinct trees.

Definition 13 (Occurrences-frequent tree) *Let $\mathcal{BA} = (D, \Delta, \delta)$ be a tree database, $\sqsubseteq$ an inclusion definition, and ε a frequency threshold. Then P is an **occurrences-frequent tree** of $\mathcal{BA}$ if P has at least ε occurrences in D, i.e. $\mid Locc_{\sqsubseteq}(P, D) \mid \geq \varepsilon$*

*The **occurrences support** of P, written as $S_o(P)$, is defined by:*

$$S_o(P) = \mid Locc_{\sqsubseteq}(P, D) \mid$$

Definition 14 (Identifier-frequent tree) *Let $\mathcal{BA} = (D, \Delta, \delta)$ be a tree database, $\sqsubseteq$ an inclusion definition, and ε a frequency threshold. Let P be a tree included in D, we call* **lid** *the list of the identifiers of the trees of D in which P is included. This list is the image by δ of the elements of $Locc_{\sqsubseteq}(P, D)$, so $lid(P, D) = \{d_i \mid d_i = \delta(u) \text{ and } u \in Locc_{\sqsubseteq}(P, D)\}$.*

*Then P is an **identifier-frequent tree** of $\mathcal{BA}$ if P is included in at least ε distinct trees of $\mathcal{BA}$: $\mid lid(P, D) \geq \varepsilon \mid$*

*The **identifier support** of P, written as $S_D(P)$, is defined by:*

$$S_D(P) = \mid lid(P, D) \mid$$

Remark 2 *If a tree P est identifier-frequent, then it is also occurrences-frequent.*

The figure II.9 shows that with a threshold $\varepsilon_a = 2$, the tree of root A having as children B and C is occurrences-frequent (the tree has 2 occurrences), but not identifier-frequent (the tree is included is only one tree of the datatree). However, the tree of root D having as only child F is included in the two trees identified by T_1 and T_2 of the datatree: so it is identifier-frequent.

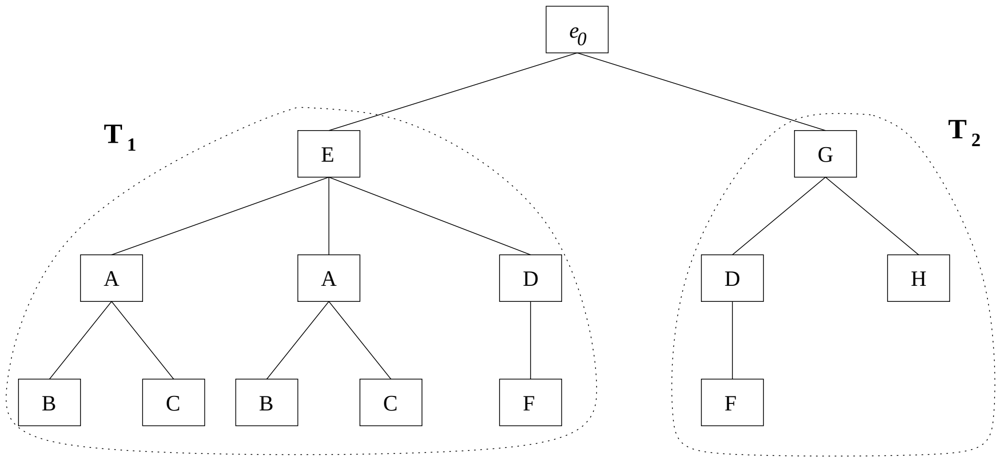

Figure II.9: Example of the differences between the frequency definitions

It is important to notice that an occurrence or an identifier can be seen as a clustering of several mappings from P to D having a property in common. For an occurrence, this property is to give the same image of the root; for an identifier, it is to give the same tree identifier for the different images of the root.

4 Summary example

Consider the tree database $\mathcal{BA}$ whose datatree D_X is given in figure II.10, and occurrence frequencies only.

With a frequency threshold of 2 and an inclusion with strict preservation of labels, the biggest frequent tree is the tree AF represented in figure II.11.

We have : $Locc(AF, D) = \{n_1, n_6\}$.

So $\mathcal{EM}_{\sqsubseteq}(AF, D) = \mathcal{EM}_{n_1}(AF, D) \cup \mathcal{EM}_{n_6}(AF, D)$. These sets are given in the array II.1.

	$\mathcal{EM}_{n_1}(AF, D)$		$\mathcal{EM}_{n_6}(AF, D)$
μ :	$A \to n_1$	μ' :	$A \to n_6$
	$B \to n_2$		$B \to n_{10}$
	$C \to n_4$		$C \to n_8$
	$D \to n_3$		$D \to n_{12}$
		μ'' :	$A \to n_6$
			$B \to n_{10}$
			$C \to n_8$
			$D \to n_{13}$

Table II.1: $\mathcal{EM}_{\sqsubseteq}(AF, D)$

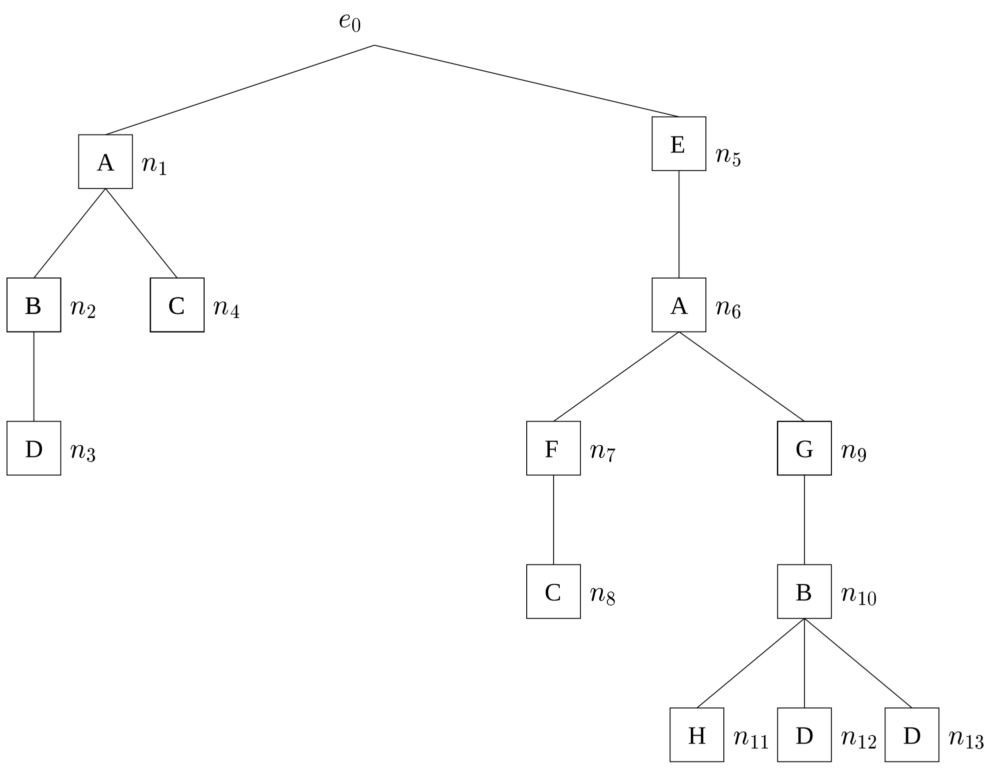

Figure II.10: D_X

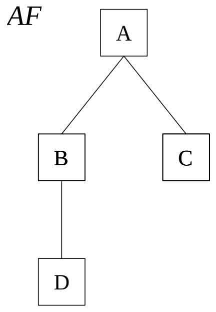

Figure II.11: Biggest frequent tree of figure II.10, threshold $\varepsilon = 2$

Chapter III

The Dryal and Dryade algorithms

1 Goal

We recall that the goal of the algorithms presented in this thesis is to discover frequent trees in a tree database, as illustrated in figure III.1.

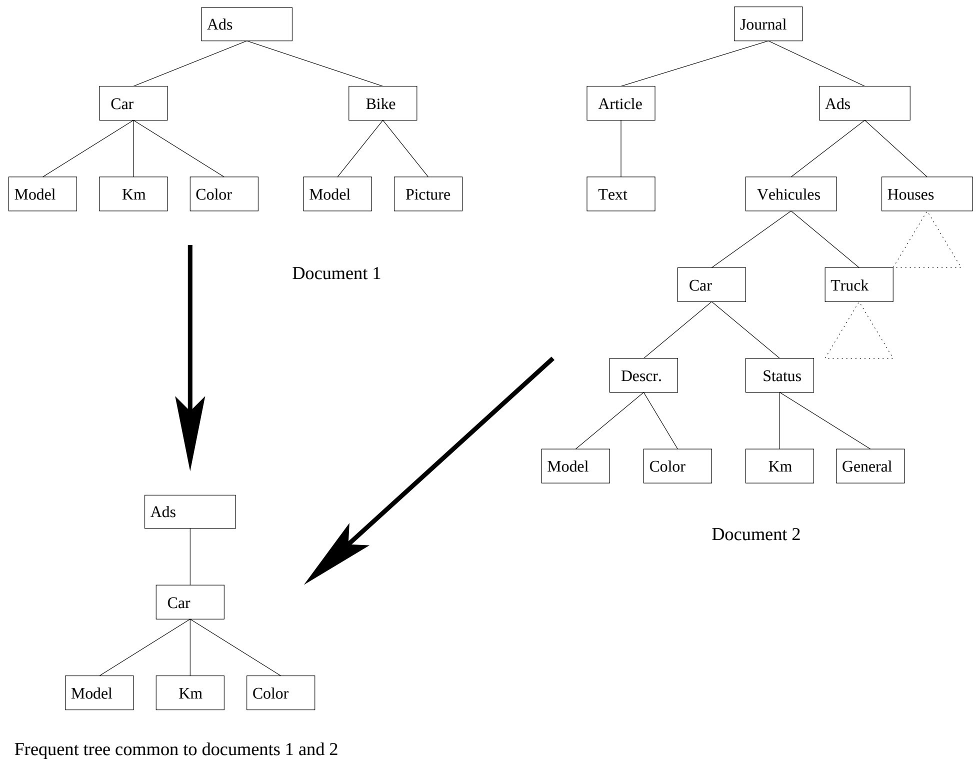

Figure III.1: An example of frequent tree discovery

As previously mentioned, in this chapter we restrict the frequent trees we want to discover to frequent **patterns**, with a pattern being a tree where no node has two children with the same label.

We will go back to this restriction when we will describe the algorithms which make use of it, in order to show the complexity benefits it allows.

This chapter is organised as follows :

- We will present a first frequent tree discovery algorithm, the Dryal algorithm. This algorithm represents the essence of our approach.
- Then we will present some ways of parameterizing Dryal, allowing it to use different tree inclusion definitions.
- The Dryade[1] algorithm extends de Dryal approach, and makes use of an algebraic property (closed sets) to allow an efficient resolution beyond the limits of Dryal.

For these two algorithms, we will give soundness and completeness proofs.

- The *soundness* proof ensures that all the results given by our algorithms are frequent patterns.
- The *completeness* proof ensures that our algorithm find **all** the frequent patterns that are in the datatree.

We sum up in table 1 the notations given in chapter II.

$\sqsubseteq$	Tree inclusion (can be strict, exact, ordered, weak)	Definition 9 page 11
$\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$	Set of mappings between the trees T_1 and T_2 such that $T_1 \sqsubseteq T_2$	
$Locc_{\sqsubseteq}(T_1, T_2)$	Set of T_2 nodes where maps the root of T_1 by a mapping from $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$	Definition 10 page 15
$\mathcal{EM}_u(T_1, T_2)$	For a node $u \in Locc_{\sqsubseteq}(T_1, T_2)$, set of mappings from $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$ mapping the root of T_1 on u	Definition 10 page 15

Table III.1: Notations summary

2 The Dryal algorithm

For all this section, our tree inclusion definition will be the **strict weak tree inclusion**, noted $\sqsubset_f$; the mappings supporting the inclusion strictly preserve the labels, preserve the ancestor relationship (but not necessarily the parent relationship) et are not necessarily injective.

To simplify reading, we will note $Locc(X, Y)$ instead of $Locc_{\sqsubset_f}(X, Y)$.

2.1 Specifications

2.1.1 Input data

- $\mathcal{BA} = (D, \Delta, \delta)$ a tree database
- L a set of labels
- ε a frequency threshold

[1]Dryads are magical creatures living in trees, and for trees. An old legend says that they all come from the blood of Dryal, a young mortal girl having entered and perished in a magical forest.

2.1.2 Expected results

- All the frequent patterns present in $\mathcal{BA}$ with a minimal threshold of ε.
- For each frequent pattern PF, the algorithm also gives $Locc(PF, D)$.

Notation : For a tree database $\mathcal{BA}$ and a frequency threshold ε, we will note $\mathcal{P}$ the set of all frequent patterns from $\mathcal{BA}$ having at least the frequency threshold ε.

2.2 Essence of the algorithm

All existing algorithms for frequent tree discovery are "Generate and Test" algorithms ([AS94], [Toi96], [BMS97], [Jr.98] among many others). It means that they create many candidate trees, then test if these candidate trees are frequent. New candidate trees are created by adding one edge to the detected frequent trees. This continues until no new frequent tree can be found.

This method is costly, because it explores a large part of the search space (frequent trees + many infrequent candidates). So it can be slow, especially if the frequent trees to find are large.

Most frequent itemset discovery algorithms use the monotonicity property on frequency to prune the search space. When searching for frequent structures, the structure itself can be used for further pruning.

These heuristics will be the strong point of our approach.

We propose a constructive algorithm, using the constraints on frequency and on data structure to genereate a candidate pattern set as small as possible, while remaining complete.

The algorithm we present thus computes all frequent patterns occurring in the data in a levelwise manner, gradually adding structure to candidate patterns.

Our method is the following: we start by finding all frequent patterns of depth 1. Then we hook together some of these patterns, in order to obtain new and more complex frequent patterns. By repeated hookings, our algorithm computes all the frequent patterns occurring in the data.

The following paragraphs detail this approach and start by some definitions.

2.3 Definitions

Our starting point is the set PFA_1 of frequent patterns of depth 1, which will be the basis for all further hookings.

Definition 15 (PFA_1) *We call PFA_1 the set of frequent patterns of depth 1.*
$PFA_1 = \{P \mid |Locc(P, D)| \geq \varepsilon \text{ and } depth(P) = 1\}$

Example: Lets go back to the example of chapter II, given again in figure III.2.

With the frequency threshold $\varepsilon = 2$, the set $PFA_1 = \{P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{16}, P_{17}, P_{18}\}$ extracted from this datatree is represented in figure III.3.

We now define the hooking operator between patterns.

Definition 16 (Hooking) *Let P be a frequent pattern, and $P_1 \in PFA_1$. We say that P and P_1 are* hookable*, noted $hookable(P, P_1)$, if the label of the root of P_1 is the same as the label of at least one of the leaves of P.*

The set of nodes of N_P where P_1 can hook is $hook\_nodes(P, P_1)$.

The set of patterns resulting from the hooking of P and P_1 is noted $P \oplus P_1$.

We have $|hook\_nodes(P, P_1)| = |P \oplus P_1|$.

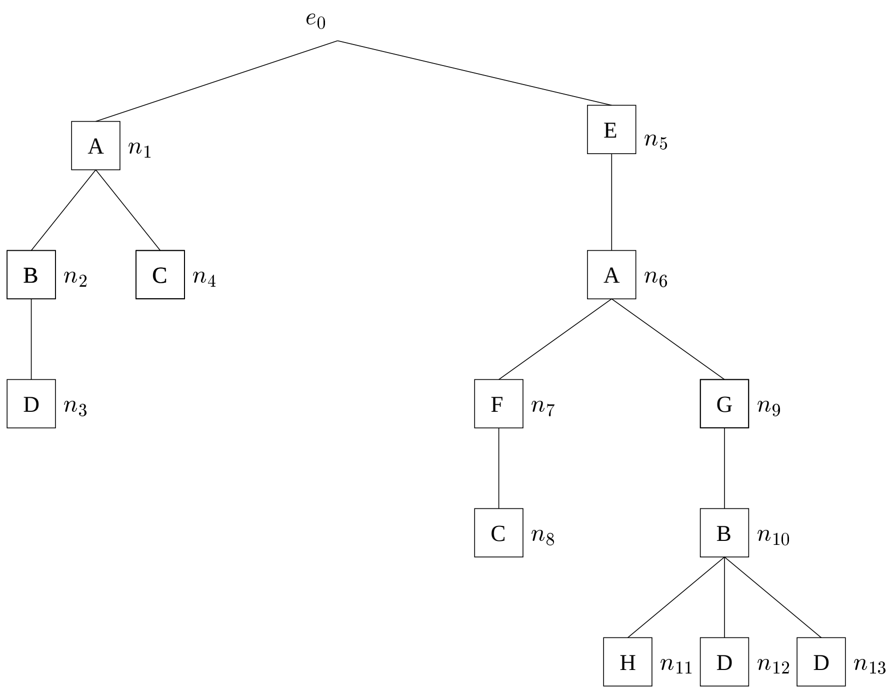

Figure III.2: Datatree example.

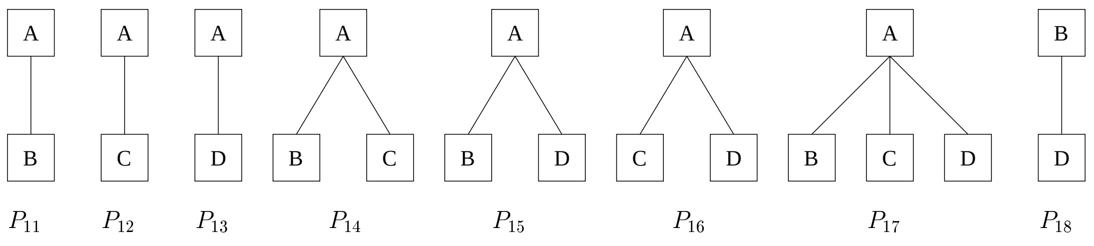

Figure III.3: PFA_1 for D_X

Let $hn \in hook\_nodes(P, P_1)$ be a hook node. The unique pattern resulting from the hooking of P and P_1 on hn is noted: $P \oplus_{hn} P_1$.

Notice that $\oplus$ is neither commutative nor associative.

Notation: Let $\{P'_1, ..., P'_n\}$ be a set of patterns, and let $P_1 \in PFA_1$ be a frequent pattern of depth 1. Then $\{P'_1, ..., P'_n\} \oplus P_1 = \bigcup_{i=1}^{n} P'_i \oplus P_1$.

The figure III.4 shows two hooking examples, the first one between two depth 1 patterns, and the second one between an unspecified pattern and a depth 1 pattern.

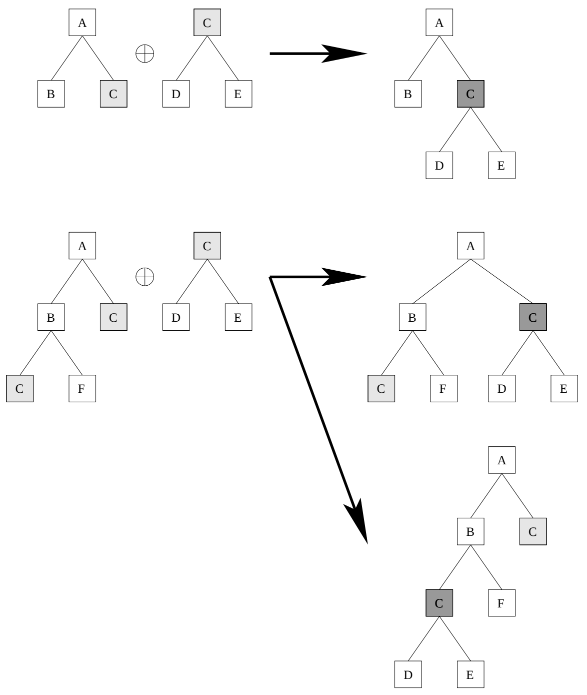

Figure III.4: Two hooking examples

Notice that the hooking of two frequent patterns PF_1 and PF_2 does not necesserarily gives a frequent pattern. The following definition allows to select **frequent** patterns resulting from successive hookings of frequent patterns.

Definition 17 **(PFA_i)** *Let $i \geq 2$. PFA_i is the set of all frequent patterns resulting from the successive hookings of i frequent patterns of depth 1. It is recursively defined as follows: for all pattern $P_i \in PFA_i$, there exists a pattern $P_{i-1} \in PFA_{i-1}$ and a pattern $P_1 \in PFA_1$ such that $P_i = P_{i-1} \oplus P_1$.*

Example: The figure III.5 shows the set $PFA_2 = \{P_{21}, P_{22}, P_{23}, P_{24}\}$ for D_X.

Let $P_i \in PFA_i$ and $P_1 \in PFA_1$ be two frequent patterns. Let $P \in P_i \oplus P_1$. To compute it's frequency and thus test it's belonging to PFA_{i+1}, we need to compute $Locc(P, D)$. The important point is to be able to determine $Locc(P, D)$ from $Locc(P_i, D)$ and $Locc(P_1, D)$, hence avoiding costly tree inclusion tests.

For this, we need to define the notion of hooking anchor in the data, that is for the hooking of P_1 on P_i the set of occurrences pairs of P_i and P_1 characterizing the subtrees from data to which the hooking of P_1 on P_i can be associated by mapping.

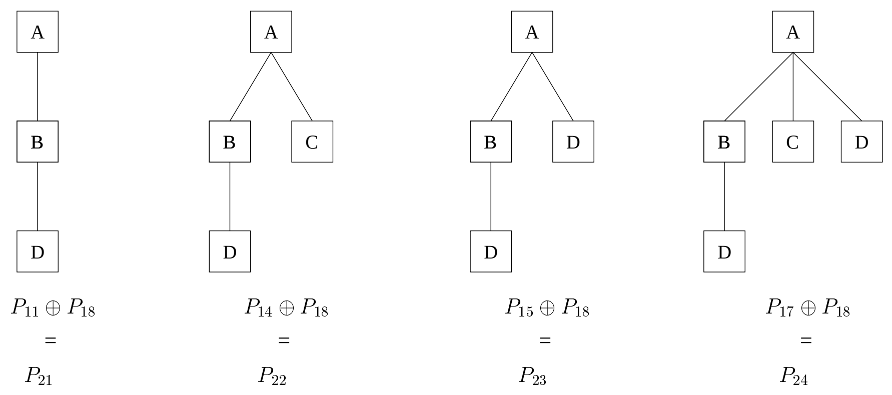

Figure III.5: PFA_2 for D_X

Definition 18 (Hooking anchored in data) *Let $P_i \in PFA_i$ and $P_1 \in PFA_1$ be two hookable patterns. Let $hn \in N_{P_i}$ such that $hn \in hook\_nodes(P_i, P_1)$.*

The hooking of P_1 on P_i by hn is anchored in data if there exists $n_i \in Locc(P_i, D)$ and $n_1 \in Locc(P_1, D)$ such that there exists $\mu_i \in \mathcal{EM}_{n_i}(P_i, D)$ and $\mu_1 \in \mathcal{EM}_{n_1}(P_1)$ such that $\mu_i(hn) = \mu_1(root(P_1))$.

*The set of suchs pairs (n_i, n_1) is called the **data anchor** of $P_i \oplus_{hn} P_1$, and is noted $Anchor_{hn}(P_i, P_1)$.*

By extension $Anchor(P_i, P_1) = \bigcup_{hn \in hook\_nodes(P_i, P_1)} Anchor_{hn}(P_i, P_1)$.

This definition is illustrated on figure III.6.

For a more concrete example, in the case of D_X, by examining figure III.2 one can see that $Anchor_B(P_{14}, P_{18}) = \{(n1, n2), (n6, n10)\}$.

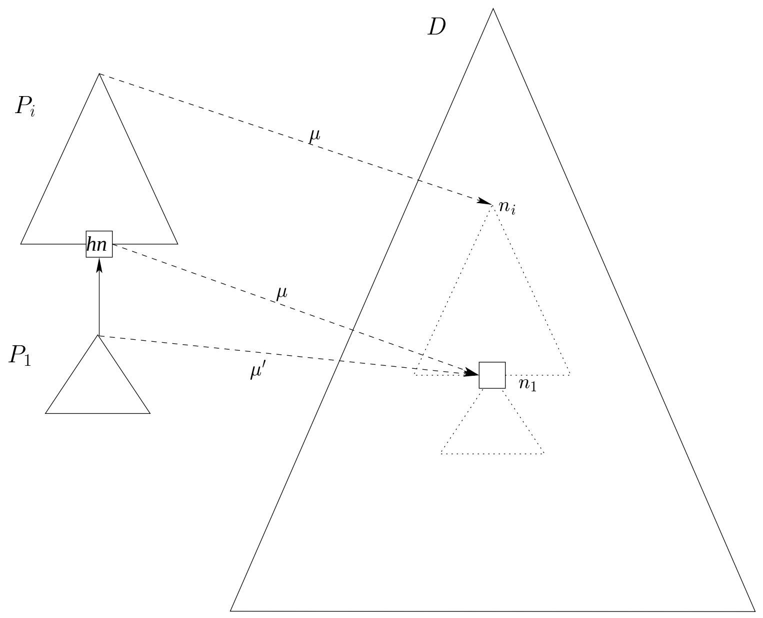

Figure III.6: Illustration of the anchoring in data of a hooking

We can deduce how to compute the set of occurrences of $P_i \oplus P_1$ from the set of occurrences of P_i and the set of occurrences of P_1:

Property 4 *Let $P_i \in PFA_i$ and $P_1 \in PFA_1$ be two hookable patterns. Let $hn \in N_{P_i}$ be such that $hn \in hook\_nodes(P_i, P_1)$. Let $n \in Locc(P_i \oplus_{hn} P_1, D)$. We have:*

$$\mathcal{EM}_n(P_i \oplus_{hn} P_1, D) = \{\mu_i \circ \mu_1 \ \mid\ \mu_i \in \mathcal{EM}_n(P_i, D) \text{ and } \mu_1 \in \mathcal{EM}_{\mu_i(hn)}(P_1, D)\}$$

Hence:

$$Locc(P_i \oplus_{hn} P_1, D) = \{n \in Locc(P_i, D) \ \mid\ \exists m \in Locc(P_1, D) \text{ such that } (n, m) \in Anchor_{hn}(P_i, P_1)\}$$

Proof: Let $P_{i+1} = P_i \oplus_{hn} P_1$. Lets show that:

- $\mathcal{EM}_n(P_{i+1}, D) \subseteq \{\mu_i \circ \mu_1 \mid \mu_i \in \mathcal{EM}_n(P_i, D)$ and $\mu_1 \in \mathcal{EM}_{\mu_i(hn)}(P_1, D)\}$: straightforward, as $P_i \sqsubseteq P_{i+1}$ and $P_1 \sqsubseteq P_{i+1}$, we can decompose each mapping μ_i of P_{i+1} in a mapping of P_i and a mapping of P_1.

- $\mathcal{EM}_n(P_{i+1}, D) \supseteq \{\mu_i \circ \mu_1 \mid \mu_i \in \mathcal{EM}_n(P_i, D)$ and $\mu_1 \in \mathcal{EM}_{\mu_i(hn)}(P_1, D)\}$: As P_i and P_1 are hookable by hn, then $(n, \mu_i(hn)) \in Anchor(P_i, P_1)$. Hence $\mu_i \circ \mu_1$ is in $\mathcal{EM}_n(P_{i+1}, D)$.

□

2.4 Algorithm

The algorithm that we will present in this section builds all the sets PFA_i previously defined.

Our algorithm is described in algorithm 1. It takes as input a tree database $\mathcal{BA}$ and an a frequency threshold ε

The result of the algorithm is:

- a set of frequent patterns, which is the unions of sets $ER_1, ..., ER_k$. For all $i \in [1, k]$, ER_i is the set of frequent patterns resulting from successive hookings of i frequent patterns of depth 1 found by the algorithm.

- for all frequent pattern P discovered, the algorithm also provides $Locc(P, D)$, and for all $n \in Locc(P, D)$, the algorithm produces $\mathcal{EM}_n(P, D)$.

  In the algorithms, to represent the set of occurrences of a frequent pattern P, we will use a data structure grouping occurrences and mappings. For P, this structure is noted *P.locc*. We have: $P.loc = \{Occ_1, ..., Occ_n\}$. The objects Occ_i contains a root $Occ_i.root$ which is a node from D, and a set of mappings $Occ_i.mapings$. We have

$$\bigcup_i Occ_i.root = Locc(P, D)$$

  and

$$\forall i \in [1, n] \ \ Occ_i.mappings = \mathcal{EM}_{Occ_i.root}(P, D)$$

The first step of the algorithm is to find the set ER_1. The procedure FindPFA_1 is detailed in algorithm 2. The principle is simple: for each label l we want to find all the depth 1 patterns having a root with this label (main loop). Hence we create a matrix where every line (transaction) corresponds to a node n of label l in the data. The columns (items) correspond to the labels of nodes that are descendant from n. Then we apply the Eclat algorithm ([ZPOL97]) to this matrix. This algorithm gives us the frequent sets of labels of nodes descending from a node of label l. So we can, for this frequent label sets, create a frequent pattern of depth 1, which root label will be l and which leaves label will be the frequent labels of the frequent label set (line 15).

Thanks to the tid-lists given by ECLAT, we can then easily find the nodes which are root occurrences (line 18), and build the mappings between the frequent pattern of depth 1 found and its instances in the data (line 19).

The discovery of set ER_1 is the starting point of our algorithm. Two important characteristics must be noticed:

- The procedure FindPFA_1 uses a APRIORI-like algorithm. We have chosen ECLAT because its vertical nature allows us to have the tid-lists that we need, while benefitting from a good efficiency. Using a APRIORI-like algorithm allows us to access all the many improvements made to this kind of algorithms, present and future.

- We can see here the effect of the restriction that we imposed on the frequent trees we discover. We recall that this restriction is to find frequent *patterns*, these being labels trees where no nodes can have two children with the same label. With this restriction, the APRIORI-like algorithm used would have to be able to take as input a matrix of integer values (to handle label duplications), and to find frequent itemsets with item duplications. To evaluate the complexity increase, one can imagine the use of integer values in the matrix as a boolean matrix with many more columns: the label l would have as many columns as it can have duplications. The APRIORI algorithm is exponential in the number of items. Hence any increase in the number of columns induces a big efficiency loss.

Algorithm 1 DRYAL

```
Require: $\mathcal{BA}$ a tree database, $\varepsilon$ a threshold, $L$ a set of labels
Ensure: $\bigcup ER_i$ set of resulting frequent patterns, and for each discovered frequent pattern $P$,
    $P.loc$.
 1: $ER_1 \leftarrow$ Find$PFA_1$ // see algorithm 2
 2: $i \leftarrow 1$
 3: while $ER_i \neq \emptyset$ do
 4:    $ER_{i+1} \leftarrow \emptyset$
 5:    for all $P \in ER_i$ do
 6:       for all $P_1 \in ER_1$ do
 7:          if $hookable(P, P_1)$ then
 8:             $ER_{i+1} \leftarrow ER_{i+1} \cup Hooking(P, P_1)$ // see algorithm 3
 9:          end if
10:       end for
11:    end for
12:    $i \leftarrow i + 1$
13: end while
14: Return $\bigcup_{j=1}^{i} ER_j$
```

From the discovery of the set ER_1 of frequent patterns of depth 1, the sets $ER_{i+1} (i \geq 1)$ are discovered iteratively by hooking at each step i the patterns of ER_i and the patterns of ER_1. This is realised by the main algorithm in the double loop of lines 5 to 11. For each element pair $(P_i, P_1) \in ER_i \times ER_1$, we determine if they can be hooked, that is if P_i has at least one leaf of same label as the root of P_1. If so, we use the hooking procedure described in algorithm 3.

Algorithm 2 FindPFA_1

```
**Require:** $\mathcal{BA}$ a tree database, $\varepsilon$ a threshold, $L$ a set of labels
**Ensure:** $ER_1$ the set of frequent patterns of depth 1, and for each frequent pattern $P$, $P.loc$
 1: $ER_1 \leftarrow \emptyset$
 2: for all label $l \in L$ do
 3:   // Matrix building
 4:   $M_l \leftarrow$ bidimensionnal matrix with $|\{n \in N_D | label(n) = l\}|$ lines and $|L|$ columns
 5:   for all $n \in N_D$ tq $label(n) = l$ do
 6:     for all $l' \in L$ do
 7:       if $\exists m \in N_D$ tq $ancestor(n, m)$ et $label(m) = l'$ then
 8:         $M_l[n, l'] = 1$
 9:       end if
10:     end for
11:   end for
12:   $EEL \leftarrow Eclat(M_l, \varepsilon)$ // EEL is a set of label sets given by the algorithm
                                  // ECLAT. Each label set is associated to its tid-list.
13:   // Frequent pattern
14:   for all $EL \in EEL$ do
15:     $P_{new} \leftarrow$ depth 1 pattern, root label = $l$, leaves labels = $EL.itemset$
16:     $P_{new}.loc \leftarrow \emptyset$
17:     for all $n \in EL.tidList$ do
18:       $NewOcc.root = n$
19:       $NewOcc.mappings = \{\mu \mid \mu(root(P_{new})) = n$ and
                    $\forall m \in leaves(P_{new})\ label(m) = label(\mu(m))$ and $ancestor(n, \mu(m))\}$
20:       $P_{new}.loc \leftarrow P_{new}.loc \cup NewOcc$
21:     end for
22:     $ER_1 \leftarrow ER_1 \cup P_{new}$
23:   end for
24: end for
25: Return $ER_1$
```

For each potential hook node n, the procedure $ComputeLocAccr$ described in algorithm 4 checks that there exists enough occurrences in the trees supporting the hooking on that node. If there is, a new frequent pattern is created, resulting from the hooking of P_1 on P_i by node n. The occurrences of this new pattern are computed by keeping the roots of P_i occurrences supporting the hooking between P_i and P_1 by n, and by composing the mappings associated to these roots for P_i with the mappings of corresponding occurences of P_1.

Algorithm 3 Hooking

```
Require: $P_i$ and $P_1$ are hookable
Ensure: a set of frequent patterns of depth $i+1$ resulting from the hooking of $P_1$ on $P_i$
 1: for all $hn \in hook\_nodes(P_i, P_1)$ do
 2:   $(LOC, Anchor_{hn}(P_i, P_1)) \leftarrow ComputeLocAccr(P_i, P_1, hn)$
 3:   if $|LOC| \geq \varepsilon$ then
 4:     // The hooking on node hn is frequent: we process it
 5:     $P_{new} \leftarrow P_i \oplus_{hn} P_1$
 6:     $P_{new}.loc \leftarrow \emptyset$
 7:     for all $(Occ_i, Occ_1) \in P_i.loc \times P_1.loc$ st $(Occ_i.root, Occ_1.root) \in Anchor_{hn}(P_i, P_1)$ do
 8:       $NewOcc.root \leftarrow Occ_i.root$
 9:       $NewOcc.mappings \leftarrow \emptyset$
10:       for all $(\mu, \mu') \in Occ_i.mappings \times Occ_1.mappings$ st $\mu(hn) = Occ_1.root$ do
11:         $NewOcc.mappings \leftarrow NewOcc.mappings \cup \mu \circ \mu'$
12:       end for
13:       $P_{new}.loc \leftarrow P_{new}.loc \cup NewOcc$
14:     end for
15:     $Result \leftarrow Result \cup P_{new}$
16:   end if
17: end for
18: Return $Result$
```

2.5 Dryal properties

THEOREM 1 *The* Dryal *algorithm is sound and complete, i.e. the set of patterns returned by* Dryal *for a tree database $\mathcal{BA}$ and a threshold ε is exactly the set $\mathcal{P}$ of all the frequent patterns of $\mathcal{BA}$ for the threshold ε.*

Proof: We will first show soundness, then we will show completeness.

Soundness:

Let's show by induction on i that all $P \in ER_i$ are frequent, and that $\forall P \in ER_i$ the stucture $P.loc$ represents the set of occurrences and mappings of P in the data.

- case $i = 1$: Given that the procedure FindPFA_1 is based on the Eclat algorithm, that this algorithm must find all the patterns of depth 1 having a root of a given label, and that Eclat is correct and complete, then for each label we are assured to have all the patterns of depth 1 whose root label is this label. Hence $ER_1 = PFA_1$.

  For each frequent pattern discovered P_{new}, the data structure $P_{new}.loc$ built at lines 17 to 19 of procedure FindPFA_1 regroups correctly the set of occurrences (line 18) and mappings (line 19) of P_{new} in the data.

Algorithm 4 ComputeLocAccr

```
Require: $P_i$ and $P_1$ are hookable, $hn \in hook\_nodes(P_i, P_1)$
Ensure: $LOC$ the set of occurrences in $N_D$ of $P_i \oplus_{hn} P_1$, and $Anchor_{hn}(P_i, P_1)$
 1: $LOC \leftarrow \emptyset$
 2: $Anchor_{hn}(P_i, P_1) \leftarrow \emptyset$
 3: for all $Occ_i \in P_i.loc$ do
 4:   $OccMappingOK \leftarrow FALSE$
 5:   for all $Occ_1 \in P_1.loc$ do
 6:     if $\exists \mu \in Occ_i.mappings$ st $\mu(hn) = Occ_1.root$ then
 7:       $OccMappingOK \leftarrow TRUE$
 8:       $Anchor_{hn}(P_i, P_1) \leftarrow Anchor_{hn}(P_i, P_1) \cup \{(Occ_i.root, Occ_1.root)\}$
 9:     end if
10:   end for
11:   if $OccMappingOK$ then
12:     $LOC \leftarrow LOC \cup \{Occ_i.root\}$
13:   end if
14: end for
15: Return $LOC$ and $Anchor_{hn}(P_i, P_1)$
```

- case $i+1$: Suppose the property is true for i, we will prove it is true for $i+1$.

  Let $P' \in ER_{i+1}$. According to our algorithm, there exists $P \in ER_i$ and $P_1 \in ER_1$, P and P_1 hookable, such that $P' \in Hookings(P, P_1)$, moreover P and P_1 are frequent by the induction hypothesis.

  Still according to our algorithm, there exists at least ε occurrences of P and P_1 where the hooking of these two in the data is frequent. So, by definition of a frequent tree, P' is frequent.

  Let's prove now that $\cup_{Occ \in P.loc} Occ.root = Locc(P', D)$, and $\cup_{Occ \in P.loc} Occ.mappings = \mathcal{EM}(P', D)$. Proof by negation: lets suppose that the previous property isn't true

  - 1st case: there exists an occurrence which isn't discovered. But all occurrences of P' come from occurrences of P, and by the induction hypothesis all the occurrences of P are correctly discovered. So this case is impossible.
  - 2nd case: for an occurrence r, there exists a mapping $\mu \in \mathcal{EM}_r(P, D)$ that isn't discovered. We can restrict this mapping μ on the nodes of P_i to μ_i, and on the nodes of P_1 to μ_1, with $\mu = \mu_i \circ \mu_1$. If μ isn't discovered, then μ_i and μ_1 as well aren't discovered. This contradicts the induction hypothesis.

Completeness:
This proof comes in two steps:

- (i) We first prove that the union of the sets of frequent patterns PFA_i gives all the frequent patterns (i.e. any frequent pattern can be written as a series of hookings of frequent patterns from PFA_1).

- (ii) We then prove that $\forall i \;\; ER_i = PFA_i$ (recall that PFA_i is the set of frequent patterns resulting from the hooking of i patterns of depth 1 from PFA_1).

The point (i) is straightforward: any pattern can be written as a series of hookings of patterns of depth 1. If the pattern is frequent, then all the depth 1 patterns composing it are also frequent, so they are all in PFA_1.

For the point (ii), the correction proved that $\forall i \quad ER_i \subseteq PFA_i$. We only have to prove the converse.

By induction:

- Case $i = 1$: we have seen before that $ER_1 = PFA_1$
- General case: suppose that for i we have $PFA_i \subseteq ER_i$. Let's show that this still holds for $i + 1$.

  Let $P_{i+1} \in E_{i+1}$ $(i \neq 0)$. There exists $P_i \in PFA_i$, $P_1 \in PFA_1$ such that $P_{i+1} = P_i \oplus_{hn} P_1$, with $hn \in N_{P_i}$.

  By induction hypothesis, $P_i \in ER_i$ and $P_1 \in ER_1$. They are hookable by hn so the algorithm goes into procedure *Hooking*.

  As $P_{i+1} \in PFA_{i+1}$, it is frequent, so $|LOC| \geq \varepsilon$.

  Hence $P_{i+1} \in ER_{i+1}$.

□

3 Dryal parameterising

The Dryal algorithm can be easily modified to handle tree inclusion definitions other than the strict weak inclusion used before. The following subsections show the modifications to do in order to handle some other inclusion definitions.

3.1 Dryal$_{child}$

Preserving the ancestor relationship in the strict weak inclusion definition exponentially increases the number of results to find. Hence the cost of the alorithm with this definition of weakness is high.

We can easily restrict Dryal to a more constrained inclusion definition, for example the strict exact weak inclusion (definition 9 of chapter II, page 11), which only preserves parent relationship.

For this, the only changes to do are in the procedure FindPFA_1. The patterns of depth 1 must be built only with the parent relationship. After, the hooking mechanism is the same.

The change in the procedure FindPFA_1 occurs at line 7: $ancestor(n, m)$ must be replaced by $parent(n, m)$.

We will call the algorithm Dryal parameterised this way Dryal$_{child}$.

3.2 Dryal$_{childN}$

Considering the previous parameterising, one can want a compromise between the flexibility of the inclusion definition used and the program performances.

Given an integer N, we can then choose an inclusion definition considering at most N ancestorship levels. For example, $N = 1$ corresponds to parent relationship, and $N = 2$ considers parents and grandparents. Here again, the change is in line 7 of FindPFA_1: $ancestor(n, m)$ must be replaced by $ancestor_{\leq N}(n, m)$, where $ancestor_{\leq N}$ considers all ancestor relations of at most N levels.

3.3 Dryal$_{injective}$

We may also want to add another constraint on the inclusion definition that we use, namely the injectivity constraint. The inclusion definitions used then are no longer "weak" definitions, according to definition 9 from chapter II.

The interest of considering inclusion definitions based on injective mappings is double:

- The given patterns will have exactly the same form as they have in the documents. Without injectivity, we authorise some pattern forms that can go against the user's intuition, as long as inclusion definition is not exact (i.e. uses ancestor relationship).

  In our example, let's consider the frequent patterns from PFA_2 shown in figure III.5. With a weak inclusion definition, and a threshold $\varepsilon = 2$, the pattern P_{24} is frequent. However, this implies that for the occurrence of P_{24} of root n_1 in the data, the two nodes of P_{24} of label D are mapped onto the node n_3 from data (see figure III.2).

  This is not necessarily very intuitive, and such a pattern wouldn't be considered as frequent with an inclusion definition taking into account injectivity.

- Taking mapping injectivity into account increases some costs in the program, but also allows to reduce some other costs, by diminishing the number of duplicate frequent patterns generated. Consider example of figure III.7. We have on top of the figure a tree database, and we have shown some frequent patterns of depth 1, for the threshold $\varepsilon = 2$. The frequency of pattern P_{21}, built by hooking of P_{12} on P_{11}, lies on the mapping of two nodes labelled by D of P_{21} on the same node of label D in the data. The fact that P_{21} have two nodes of label D will allow the hooking on these two nodes of pattern P_{13} and thus generate the two frequent patterns P_{31} and P_{32}. Each of these patterns having to be considered in the further computations, the cost overrun is important. If we impose mapping injectivity, it suppresses one of the occurrences of a node of label D in P_21, hence only one hooking of P_{13} is possible.

The duplicate nodes inside patterns come from the use of ancestrality relationship. For example, in figure III.8, we can see a tree database and all the associated frequent patterns of depth 1, for $\varepsilon = 1$ (we use here a the version of Dryal using ancestrality relationship). If we hook pattern 9 on pattern 6, then pattern 11 on the resulting pattern, we obtain P_C, which is very close to the data. But we can also hook pattern 9 on pattern 3. Then the node of label C is duplicated: there is one version coming from pattern 9 and correctly inserted at depth 2, and one (duplicate) version coming from pattern 3 and which represents an ancestrality relation in the pattern.

In order for Dryal to limit itself to do hookings satisfying the injectivity of corresponding mappings, hookings producing duplicate nodes must be avoided.

For this, we introduce the function *CheckInjectivity*, described in algorithm 5. Given two hookable patterns P_i and P_1, a hook node hn and the sets $Anchor_{hn}(P_i, P_1)$ and LOC computed by $ComputeLocAccr$, this function returns the set $NewLOC$ of the occurrences of LOC satisfying injectivity constraint for the hooking $P_i \oplus_{hn} P_1$.

The essence is the following: for each occurrence of P_i where P_1 hooks on hn, we look for a mapping of $P_i \oplus_{hn} P_1$ in the data satisfying the injectivity constraint. As at this state of the algorithm we don't know the mappings of $P_i \oplus_{hn} P_1$, our analysis uses the mappings of P_i and those of P_1. For a given leaf of P_i, if in $P_i \oplus_{hn} P_1$ this leaf only reflects an ancestrality relation between it's parent and a leaf coming from P_1 (injectivity constraint not satisfied), this comes in the data to the equality between the set of nodes of D that can be reached by mapping from this leaf of P_i, and the set of nodes of D that can be reached by mapping from the corresponding leaf of P_1.

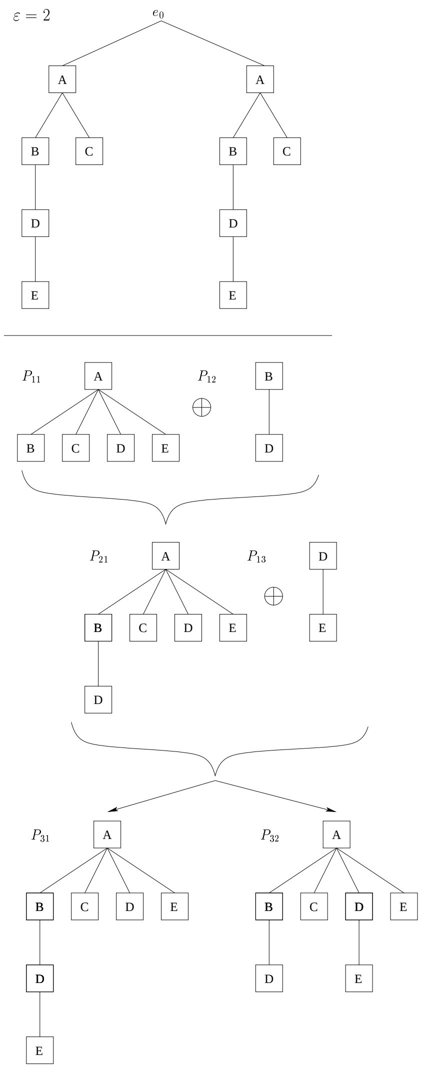

Figure III.7: P_{31} are P_{32} are duplicates

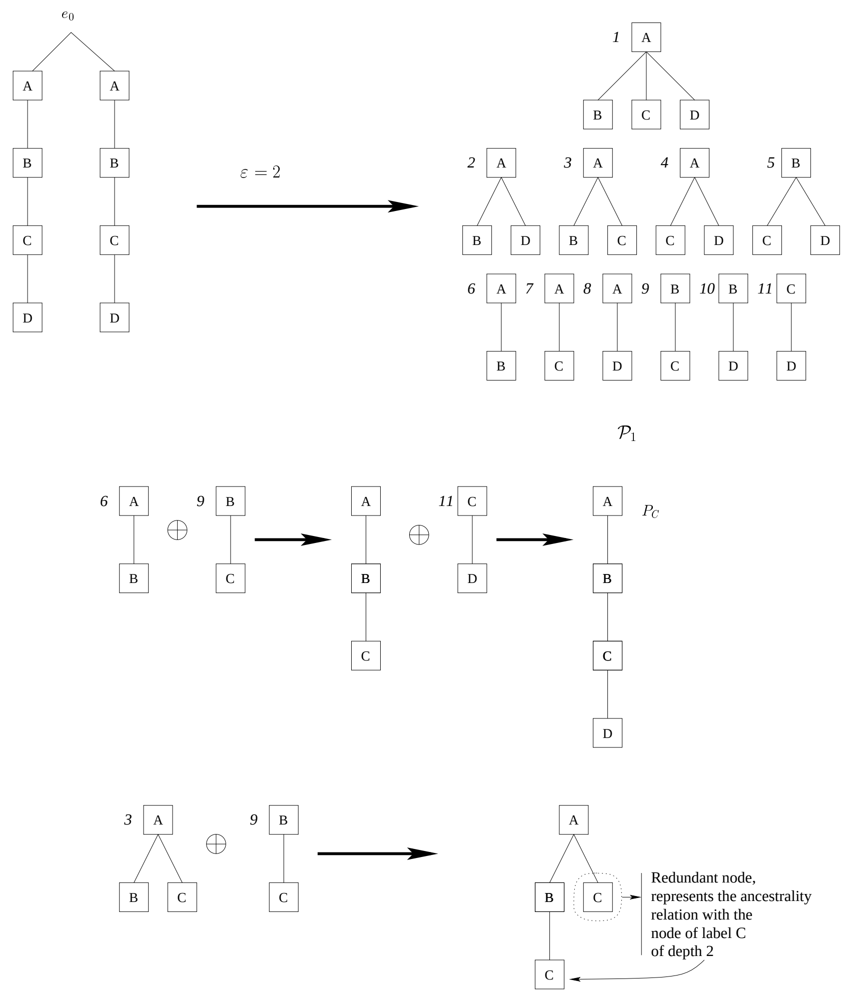

Figure III.8: A tree database and all the frequent patterns of depth 1 associated

Algorithm 5 *CheckInjectiviy*

```
Require: $P_i$ and $P_1$ hookable, $hn$ a hook node, $Anchor_{hn}(P_i, P_1)$ , $LOC$ set of occurrences from
    $P_i$ where $P_1$ hooks.
Ensure: $NewLoc$ the list of occurrences fo $LOC$ satisfying injectivity constraint.
 1: $NewLOC \leftarrow \emptyset$
 2: for all $n_i \in LOC$ do
 3:   $AllLeavesOK \leftarrow TRUE$
 4:   for all leaf $f \in N_{P_i}$ do
 5:     $f' \leftarrow$ leaf from $P_1$ having same label as $f$
 6:     $ThisLeafOK \leftarrow FALSE$
 7:     for all $\mu \in \mathcal{EM}_{n_i}(P_i, D)$ do
 8:       for all $\mu' \in \mathcal{EM}_{n_1}(P_1, D)$ with $(n_i, n_1) \in Anchor_{hn}(P_i, P_1)$ do
 9:         if $\mu'(f') \subset \mu(f)$ then
10:           $ThisLeafOK \leftarrow TRUE$
11:         end if
12:       end for
13:     end for
14:     $AllLeavesOK \leftarrow AllLeavesOK\ AND\ ThisLeafOK$
15:   end for
16:   if $AllLeavesOK$ then
17:     $NewLOC \leftarrow NewLOC \cup \{n_i\}$
18:   end if
19: end for
20: Return $NewLoc$
```

Then the procedure *Hooking* (algorithm 3) must be modified to use *CheckInjectivity*.

We add line 3': $LOC \leftarrow CheckInjectivity(P_i, P_1, hn, Anchor_{hn}(P_i, P_1), LOC)$.

This line ensures that only the hookings satisfying the injectivity condition for at least ε of their occurrences will be performed.

4 Dryade

4.1 Motivations

The previous algorithm, Dryal, doesn't make the most of the fact that if a pattern P is frequent, then necessarily all the patterns included in A will be frequent (as shown by figure III.9). Why then consume computing time to generate all the frequent patterns included in P ?

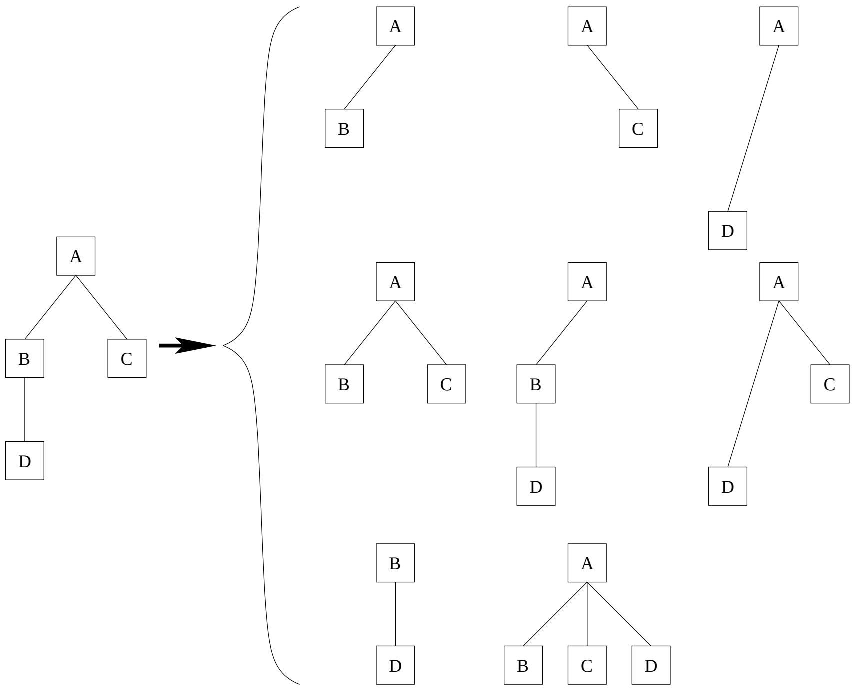

Figure III.9: If the pattern on the left is frequent, then all the patterns on the right are also frequent

This flaw is common to many algorithms discovering frequent structures or itemsets. It is particularly important in our problem, because the inclusion definition based upon an ancestrality relationship exponentially increases the number of frequent patterns that we can find in the data.

In the classical case of transactionnal databases, there exists an efficient solution to skip many redundancies existing in the results. This solution in to compute **closed** frequent itemsets ([PBTL99], [PHM00], [ZH02]).

Hence we will modify our algorithm to no longer compute frequent patterns, but **closed** frequent patterns, which we will define formally in next section. This solution has also been applied for the discovery of frequent trees in [CYXM04]. However, their system consider the more constrained inclusion definition of strict exact inclusion, preserving the parent relationship, and CMTreeMiner discovers closed identifier-frequent trees. This comes with a different closure definition, based on identifiers and not on occurrences.

In the rest of this chapter, we use the injective inclusion definition, only preserving ancestor relationship. We have presented in the previous section the algorithm DRYAL$_{injective}$, computing the set of all frequent patterns corresponding to this inclusion definition, as well as the interest of using this definition. In this section, we present DRYADE, which computes the set of **closed** frequent patterns corresponding to this definition.

4.2 New definitions

We define formally the notion of closed set of frequent patterns. The intuition behind our definition is that for a frequent pattern P' to be closed, it's occurrences in the datatree mustn't be the same as it's occurrences in the projected of a pattern P in which it is included. In fact in this case the pattern P' would be redundant with respect to P: it is included in P and has no occurrence in the data other than those coming from its inclusion in P.

Definition 19 (Closed set of frequent patterns) *Let E be a set of frequent patterns with threshold ε. We say that E is closed if for all frequent pattern $P \in E$, there is no frequent pattern $P' \in E$ such that:*

i. $P' \sqsubset P$

ii. $Locc(P', D) = Locc(P', projected(P, D))$

Notation: By extension, we say of a frequent pattern from a closed set of frequent patterns that it's a **closed** frequent pattern.

Example: For our example datatree D_X (figure III.2), let's consider PFA_1 the set of frequent patterns of depth 1. The figure III.10 shows the set of occurrences of these patterns, and the set $Closed(PFA_1)$ that we can deduce.

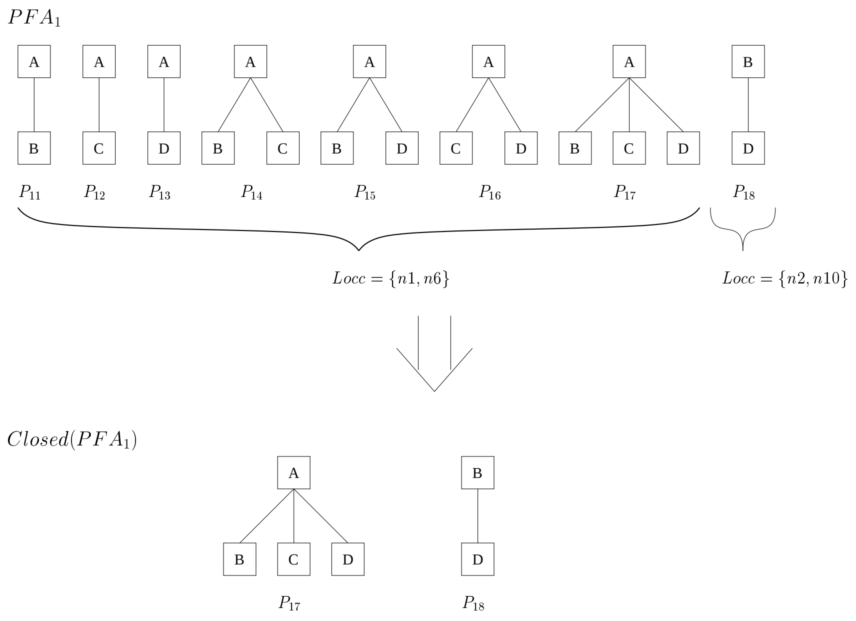

Figure III.10: For D_X, PFA_1 and $Closed(PFA_1)$

As we can see, because of condition **i**, when a frequent pattern is included into another one it is often eliminated from the closed set.

We only keep a frequent pattern P' included in a frequent pattern P when the condition **ii** isn't satisfied. This case is shown in figure III.11: the frequent pattern P_2 is included in the frequent pattern P_1, but it is closed because it's occurrence n_2 doesn't appear in the projected of P_1 in the data, which is the tree of root e_0 without the subtree of root n_2.

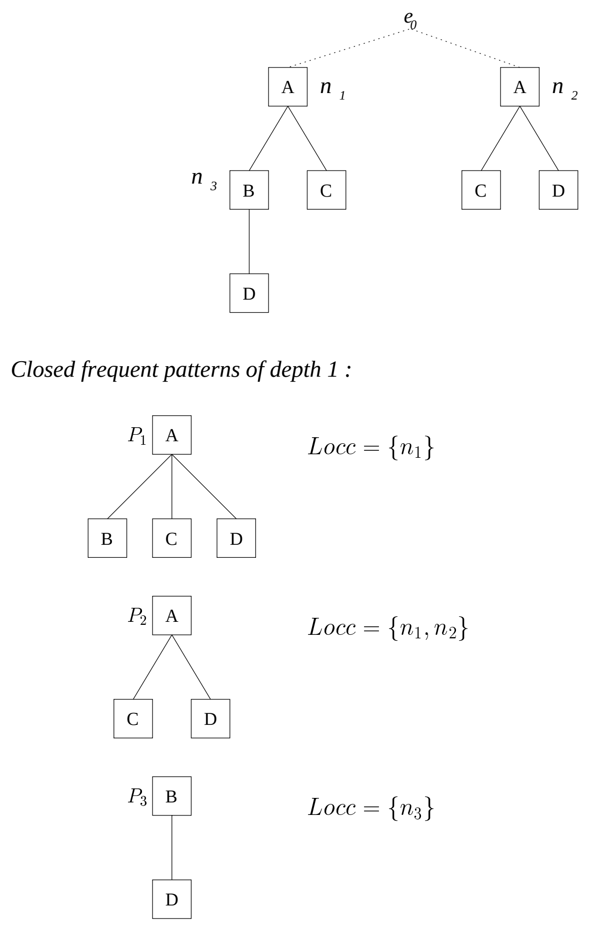

Figure III.11: $\varepsilon = 1$

Definition 20 (Closure of a frequent pattern set) *Let E be a set of frequent patterns. We call $Closed(E)$ the biggest set (according to set inclusion) included in E such that $Closed(E)$ is a closed set of frequent patterns.*

From now on, we no longer seek the set of all frequent patterns $\mathcal{P}$, but the biggest closed set of frequent patterns included in this set, i.e. we are looking for $Closed(\mathcal{P})$. We will sometimes call **target** patterns the patterns in $Closed(\mathcal{P})$.

An "easy" solution to find $Closed(\mathcal{P})$ is to compute $\mathcal{P}$, then to deduce $Closed(\mathcal{P})$. This solution is counterproductive: the very long computation time needed for the discovery of $\mathcal{P}$ must be paid, then an additional time must be paid to keep only closed patterns.

The main idea of optimisation by closed patterns is to avoid all duplicate computations. So we will directly compute $Closed(\mathcal{P})$. For this, we will continue to apply our hooking strategy, but with much smaller frequent pattern sets. It means that we will impose that the set of frequent patterns handled by the algorithm is **always a closed set**.

To illustrate the benefit of this method, let's consider the example of figure III.12.

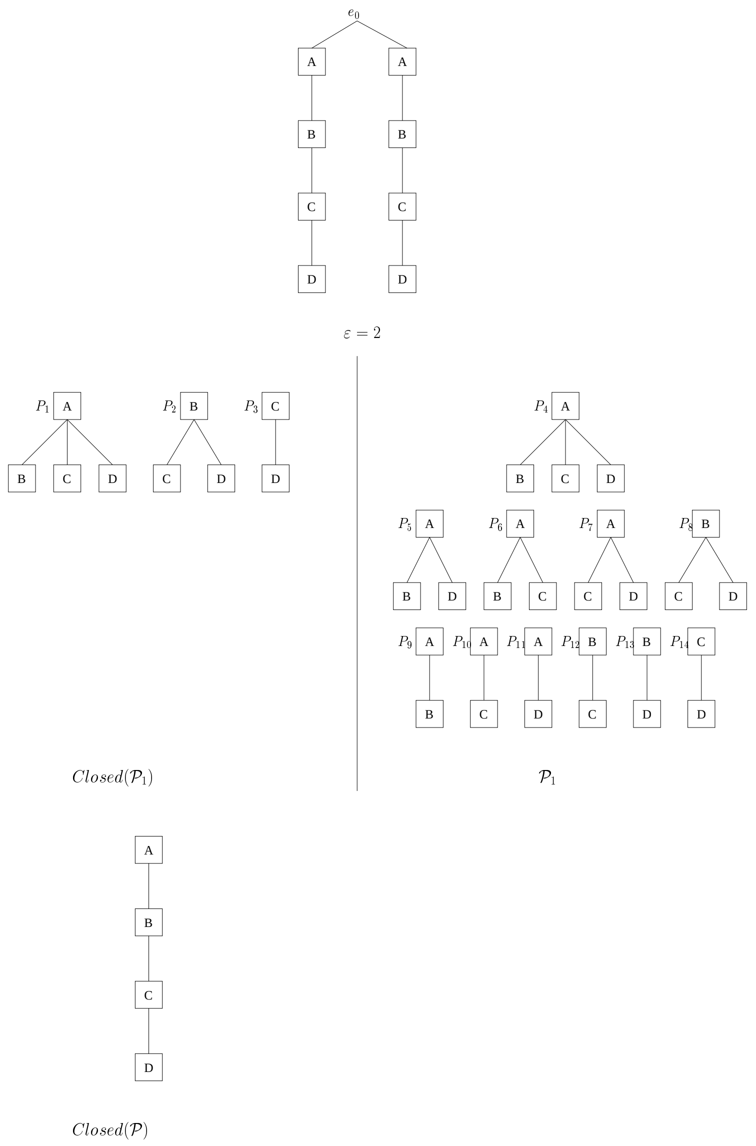

Figure III.12: Comparison of $Closed(\mathcal{P}_1)$ and $\mathcal{P}_1$

For the tree database on top of this figure, we have shown on the right the set $\mathcal{P}_1$ of frequent patterns of depth 1 it contains, and on the left the set $Closed(\mathcal{P}_1)$. All the patterns are numbered in the figure.

On the left, the number of possible hookings is very limited:

- P_2 and P_3 are hookable on P_1
- P_3 is hookable on P_2

To hook only depth 1 patterns on patterns of any depth, we can only hook P_2 on P_1, then P_3 on $P_1 \oplus P_2$. We thus generate a minimal number of frequent patterns. However, to get the unique pattern in $Closed(\mathcal{P})$, we have to suppress, from the frequent patterns produced, some "excess" leaves that do not satisfy the injectivity constraint (see figure III.13)

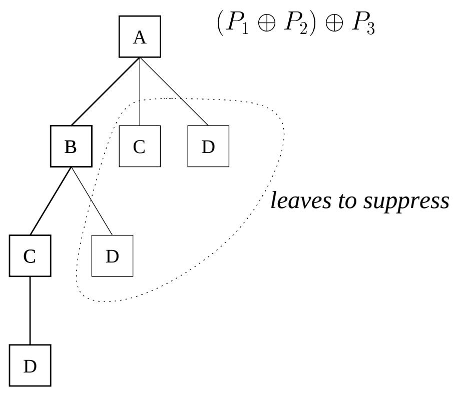

Figure III.13: Pattern produced by hooking of P_2 on P_1 then P_3 on $P_1 \oplus P_2$.

On the other hand, on the right in figure III.12, many hookings can be done:

- P_{12} can hook on P_5
- P_{14} can hook on P_6
- P_8, P_{12}, P_{13} can hook on P_9
- P_{14} can hook on P_{10} and P_{12}

For all potential hookings, frequent patterns will be produced. These frequent patterns are correct, but are all included in the unique pattern of $\mathit{Closed}(\mathcal{P})$, which simply results from the hooking of P_{12} on P_9, then of P_{14} on the previous pattern. Hence generating all of them is useless.

We can see that as the closed set of frequent patterns of depth 1 is very small, it saves us from having to generate "useless" patterns, and so avoids duplicate computations. This can be generalised to the closed sets of frequent patterns of any depth.

However, the hookings on closed frequent patterns induce a non-trivial problem to respect injectivity constraint. For a better understanding of this problem and it's origin, and of the way we will solve it, the following section gives detailed explanations about the closed set of frequent patterns of depth 1.

4.2.1 Closed set of frequent patterns of depth 1

Let's consider again the example of figure III.12. Let P_C be the unique pattern in $\mathit{Closed}(\mathcal{P})$.

More specifically, let's focus on the frequent patterns of $\mathcal{P}_1$ whose root has label A. These are the frequent patterns P_4, P_5, P_6, P_7, P_9, P_{10} and P_{11}. All these patterns have the same occurrence set. The patterns P_5, P_6, P_7, P_9, P_{10} and P_{11} represent all the possible ways to take away one then two nodes from the pattern P_4. So they are redundant, because from pattern P_4 it's very easy to find them.

As expected, the only pattern whose root has label A in $\mathit{Closed}(\mathcal{P}_1)$ is pattern P_1, which corresponds to pattern P_4 from $\mathcal{P}_1$.

The leaves of this pattern correspond to all the ancestor relationships between the root of P_C and the other nodes in P_C.

Understanding this is very important to understand the DRYADE algorithm. We can formalise it as follows:

Definition 21 (Flattening) *Let P be a pattern. We call **flattening** of P, noted $flat(P)$, the unique pattern of depth 1 such that there exists a surjective mapping μ between the nodes of P and the nodes of $flat(P)$ verifying:*

- $\mu(root(P)) = root(flat(P))$
- $\forall n \in N_P \;\; label(n) = label(\mu(n))$
- $\forall n \in N_P$ *such that* $ancestor(root(P), n) : parent(root(flat(P)), \mu(n))$

Property 5 *For all frequent pattern $P \in Closed(\mathcal{P})$, $flat(P) \in Closed(\mathcal{P}_1)$*

Proof: By definition of frequent patterns of depth 1, the pattern $flat(P)$ exists in $\mathcal{P}_1$: it's the pattern of depth 1 made with the root of P and having as leaves exactly all the labels of the nodes of P, without duplicates. This pattern is in $Closed(\mathcal{P})$ because all the patterns with the same root, the same occurrences set, but less nodes will be eliminated by $flat(P)$ when computing closure. On the other hand $flat(P)$ cannot be eliminated, because it is maximal for pattern inclusion. □

The reason why satisfying injectivity constraint is difficult can be understood thanks to property 5. Given that a frequent pattern of depth 1 contains all ancestrality relations of the target patterns it allows to build, when another frequent pattern of depth 1 is hooked on it, the relations with the nodes of this other pattern are found twice: in the nodes of the hooked pattern and in the nodes of the pattern supporting the hooking.

We already encountered this problem in section 3.3. To solve it, we detected illegal hookings from the point of view of injectivity constraint, in order to avoid them.

But here we cannot do that: we have seen that the frequent patterns of depth 1 that we have **always** lead to a non-satisfaction of injectivity constraint in the hookings.

So we must perform hooking, and then eliminate redundant leaves in the produced frequent pattern, in order to have closed frequent patterns.

The redundant leaves are defined as follows:

Definition 22 (Redundant leaf) *Let P_i and P_1 be two closed hookable frequent patterns, and let $hn \in hook\_nodes(P_i, P_1)$. Let $P_{i+1} = P_i \oplus_{hn} P_1$.*

*We say that a leaf F of P_{i+1}, coming from P_i, is **redundant** if for all occurrence $o \in Locc(P_{i+1}, D)$ and for all mapping $\mu \in \mathcal{EM}_o(P_{i+1}, D)$, we have $\mu(F) = \mu(F_1)$, where F_1 is the leaf of P_{i+1} coming from P_1 having the same label as F.*

From the previous definition, we deduce that to obtain closed frequent patterns from the hooking of two closed frequent patterns, we must first perform the hooking of the closed frequent patterns, and then suppress the redundant leaves in the resulting pattern, the leaves to suppress varying according to occurrences.

For example in figure III.13, the leaves to suppress are enclosed in the dotted line.

4.3 Essence of the algorithm

4.3.1 General essence

The idea on which Dryade is based is to create the closed set of all frequent patterns, starting with the closed set of all frequent patterns of depth 1 ($Closed(\mathcal{P}_1)$), then to compute by successive hookings the closed frequent patterns of higher depth.

As soon as we obtain $Closed(\mathcal{P}_1)$, we have a very basic approximation of the final result, $Closed(\mathcal{P})$. Actually, in the patterns of $Closed(\mathcal{P}_1)$ are the flattenings of the patterns of $Closed(\mathcal{P})$. The structural informations will be progressively earned by hooking patterns of depth 1, and suppression of redundant leaves.

4.3.2 Differences from the essence of DRYAL

As we are looking for a **closed** set of frequent patterns, the hooking strategy that we used in DRYAL will have to be changed in DRYADE, for three reasons:

Leaves suppression: Each hooking done will have to be followed by a step of redundant leaves suppression, to obtain closed frequent patterns. We will describe later how this step is performed.

Multiple hookings: In DRYAL, we were hooking patterns of depth 1 one by one. This is no longer possible for computing closed patterns. For example, lets consider the new example datatree D_Y, shown in figure III.14. For a threshold $\varepsilon = 2$, we have represented on the figure III.15 the set $\mathcal{F}_1 = Closed(\mathcal{P}_1)$.

If we apply the strategy of DRYAL, hooking patterns of depth 1 one by one, we have the hookings of figure III.16.

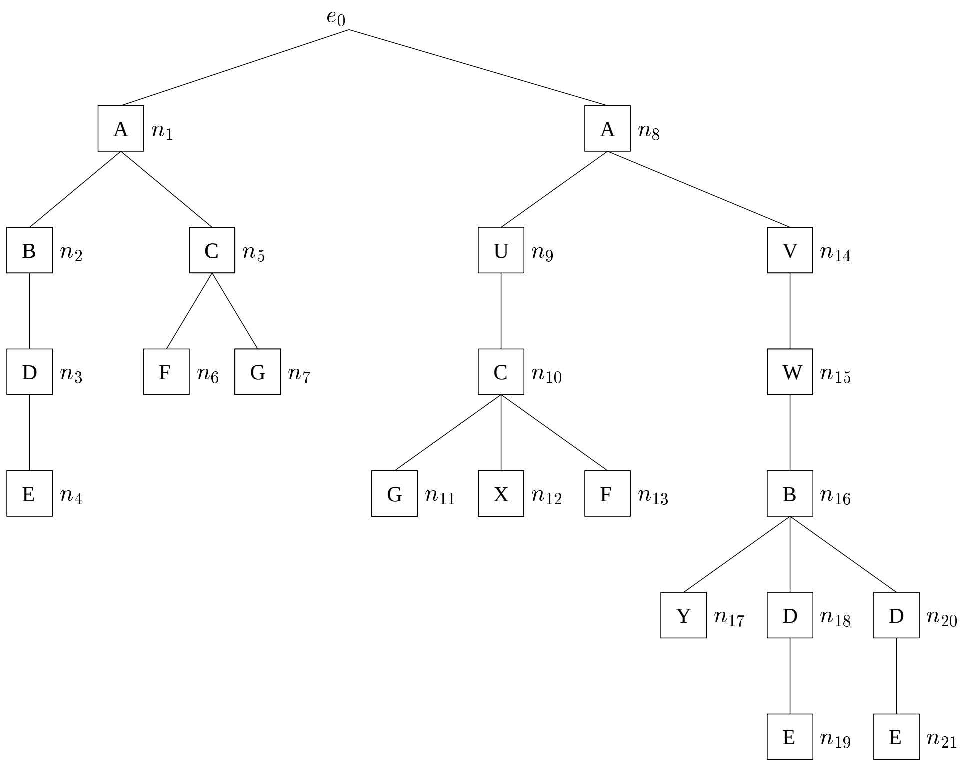

Figure III.14: D_Y

Two problems may arise. The first one is that we generate twice the pattern P_2. This is unefficent. The second problem is even worse: we generate the patterns P_{2a} and P_{2b}, which do not satisfy the closure condition (both are strictly included in pattern P_2, with the same occurrences set).

To solve efficiently this problem, we will have to, for a frequent pattern P, hook on it simultaneously several frequent patterns of depth 1. In the case of figure III.16, we have to hook

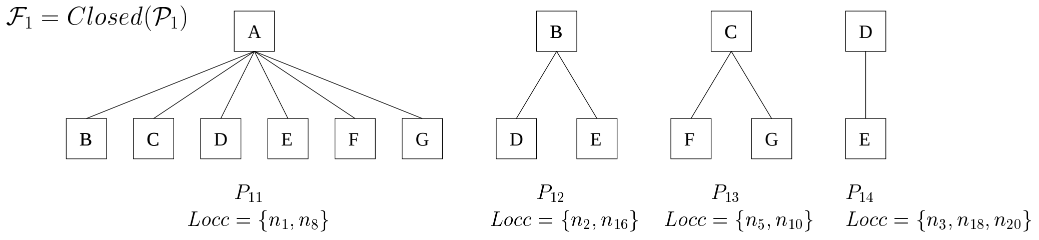

Figure III.15: For $\varepsilon = 2$, closed set of frequent patterns of depth 1 of D_Y

simultaneously P_{12} and P_{13} on P_{11}.

Hookings order: The satisfaction of closure condition imposes a third difference with DRYAL. Let's come once again to the example of figure III.12. We could hook P_2 to P_1, and also P_3 to P_2 in order to make another pattern. On $P_1 \oplus P_2$, we could then hook P_3. After the suppression of the redundant leaves, we would have two patterns, the pattern from $Closure(\mathcal{P})$ $(P_1 \oplus P_2) \oplus P_3$: $A - B - C - D$, and also the patern $P_2 \oplus P_3$: $B - C - D$. This last pattern is frequent but not closed.

To avoid this kind of problems, we have to respect a certain order to hook the patterns, with the rule of retarding as far as possible the hookings, while they are still possible.

At each DRYADE iteration, the patterns on which we must hook other patterns will be called **root** patterns. To define them, we make explicit the hooking graph structure existing between the patterns, which allows a "stratification" between patterns to appear.

Definition 23 (Hooking graph) *Let EP be a set of patterns. We call **hooking graph** the labelled directed graph $\mathcal{G}_{EP} = (EP, \mathcal{A})$ such that:*

- *the set of nodes of the graph is EP*
- *there exists an arc from P_1 to P_2 (i.e. $(P_1, P_2) \in \mathcal{A}$) if $Anchor(P_2, P_1) \neq \emptyset$.*

  In this case, the arc (P_1, P_2) is labelled by the set $Anchor(P_2, P_1)$.

Example: With the datatree D_Y, the hooking graph of closed frequent patterns of $\mathcal{F}_1$ (figure III.15) is shown on figure III.17.

Definition 24 (Predecessor and successor) *Let EP be a set of patterns. Let $\mathcal{G}_{EP} = (EP, \mathcal{A})$ be the hooking graph of EP. For all pattern $P \in EP$, we note:*

- *$pred(P) = \{P' \mid (P', P) \in \mathcal{A}\}$, these are the patterns hooking on P.*
- *$succ(P) = \{P' \mid (P, P') \in \mathcal{A}\}$, these are the patterns on which P hooks.*

Definition 25 (Cycle equivalence class) *Let EP be a set of patterns, and $\mathcal{G}_{EP}$ the corresponding hooking graph. The **cycle equivalence classes** of EP are the equivalence classes of the equivalence relation $\equiv$, defined on the nodes of $\mathcal{G}_{EP}$ by:*

$N_1 \equiv N_2$ iff there exists a cycle in $\mathcal{G}_{EP}$ going through N_1 and N_2.

The set of cycle equivalence classes of EP is noted $\mathcal{CE}(EP)$.

Definition 26 (Hookings quotient graph) *Let EP be a set of patterns. We call **hookings quotient graph** of EP, noted $\mathcal{GQ}_{EP} = (\mathcal{CE}(EP), \mathcal{A}')$, the graph satisfying:*

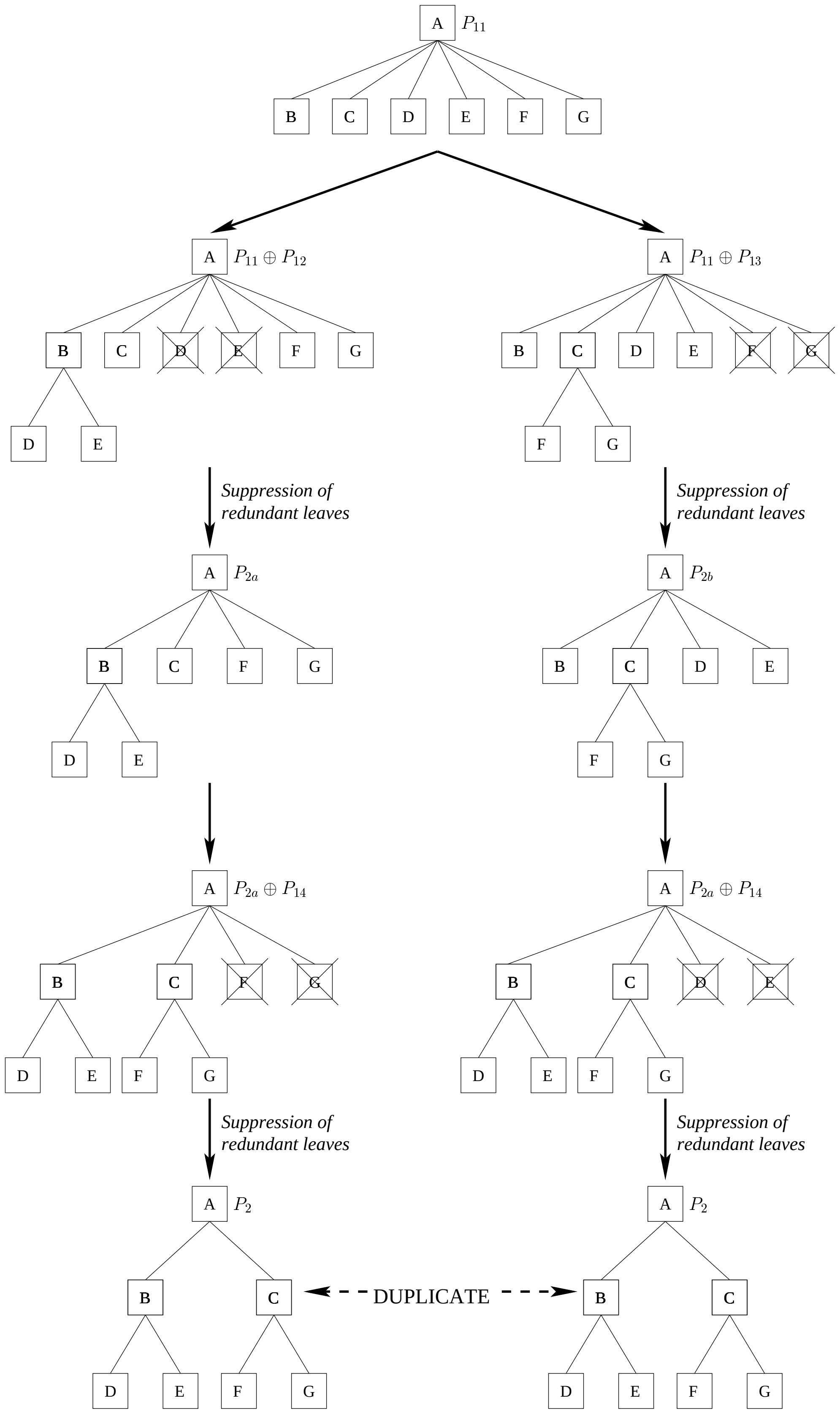

Figure III.16: Hookings with the DRYAL strategy

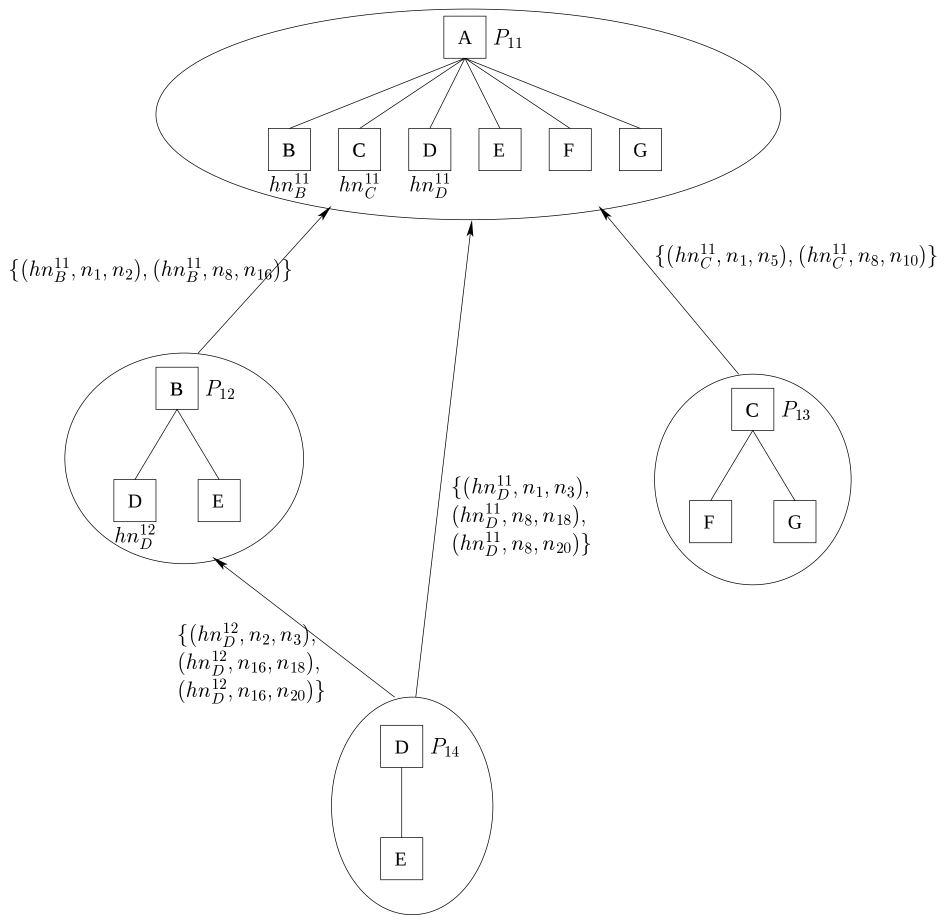
A
P_{11}
B
C
D
E
F
G
hn_B^{11}
hn_C^{11}
hn_D^{11}
$\{(hn_B^{11}, n_1, n_2), (hn_B^{11}, n_8, n_{16})\}$
$\{(hn_C^{11}, n_1, n_5), (hn_C^{11}, n_8, n_{10})\}$
B
P_{12}
D
E
hn_D^{12}
C
P_{13}
F
G
$\{(hn_D^{11}, n_1, n_3),$
$(hn_D^{11}, n_8, n_{18}),$
$(hn_D^{11}, n_8, n_{20})\}$
$\{(hn_D^{12}, n_2, n_3),$
$(hn_D^{12}, n_{16}, n_{18}),$
$(hn_D^{12}, n_{16}, n_{20})\}$
D
P_{14}
E

Figure III.17: The hooking graph of $\mathcal{F}_1$ ($\mathcal{F}_1$ is shown on figure III.15)

- *the set of nodes of the graph is $\mathcal{CE}(EP)$.*
- *there exists an arc between two nodes CE_1 and CE_2 of $\mathcal{CE}(EP)$ if and only if there exists an arc in $\mathcal{G}_{EP}$ between a pattern element of CE_1 and a pattern element of CE_2.*

Property 6 *For a set of patterns EP, it's hookings quotient graph is a directed acyclic graph. Hence there exists nodes in $\mathcal{GQ}_{EP}$ that have no father, and which will be called the roots of the hookings quotient graph.*

Definition 27 (Root pattern) *Let EP be a set of patterns, let $\mathcal{GQ}_{EP}$ be its hookings quotient graph. A* ***root pattern*** *of EP is a pattern that:*

- *belongs to a root equivalence class of the hookings quotient graph.*
- *has at least ε* ***root occurrences*** *in the data.*

The set of ***root occurrences*** *of a pattern $P \in EP$ is the set $OR(P)$ defined by:*

$$RO(P) = \{o \in Locc(P, D) \mid \forall P' \in succ(P) \ \ \nexists o' \in Locc(P', D) \ st \ (o', o) \in Anchor(P', P)\}$$

Remark 3 *If the hooking graph of a set of patterns hasn't any cycle, then the root patterns are the root nodes of the hooking graph.*

4.3.3 Pseudo-code and details of the algorithm Dryade

Dryade is shown in algorithm 6. The notations used are summed up in table III.2.

$\mathcal{F}_D^i$	Closed set of frequent patterns being computed at iteration i
$\mathcal{G}$	Hooking graph $\mathcal{F}_D^i$
RPS	Set of root patterns of $\mathcal{F}_D^i$
$IHCS$	Set of immediate hooking contexts on a root pattern
$NewPatternsInfos$	New frequent patterns of depth $i + 1$ and the patterns they come from by hooking.

Table III.2: Notations for algorithm Dryade

In the following paragraphs, we will explain thoroughly how Dryade works.

Computing $Closed(\mathcal{P}_1)$ and $\mathcal{G}$

We have seen in algorithm 2 (FindPFA_1) a method to determine the set $\mathcal{P}_1$ of the frequent patterns of depth 1. For this algorithm to compute $Closed(\mathcal{P}_1)$, only a minor modification is necessary. To compute the set of sets of frequent leaves of depth 1, FindPFA_1 uses an exhaustive Apriori-like algorithm, namely Eclat. As summed up in table III.3, the transactions are all the node n of data of label l, for a given l, and the items are all the labels of the nodes descendant of n. In the Dryade framework, we are interested in the **closed** set of sets of leaves of depth 1. So Eclat must be replaced by an algorithm computing closed frequent itemsets. To have an easy access to tid-lists, the new algorithm must be vertical: the Charm algorithm [ZH02] hence is a good solution.

Property 7 *Let $\mathcal{F}_D^1$ be the result of algorithm FindPFA_1 where* Eclat *is replaced by* Charm. *Then $\mathcal{F}_D^1 = Closed(\mathcal{P}_1)$.*

Algorithm 6 DRYADE

```
1: Compute $\mathcal{F}_D^1 = Closed(\mathcal{P}_1)$
2: $\mathcal{G} \leftarrow ComputeGraph(\mathcal{F}_D^1)$ // *See algorithm 7*
3: $Continue \leftarrow TRUE$
4: $i \leftarrow 1$
5: while $Continue$ do
6:   $NewPatternsInfos \leftarrow \emptyset$
7:   $Continue \leftarrow FALSE$
8:   $RPS \leftarrow GiveRoots(\mathcal{G})$ // *See algorithm 8*
9:   $\mathcal{F}_D^{i+1} \leftarrow \mathcal{F}_D^i$
10:  for all $RP \in RPS$ do
11:    $IHCS \leftarrow PatternsToHook(RP)$ // *See algorithm 10*
12:    if $IHCS \neq \emptyset$ then
13:      $Continue \leftarrow TRUE$
14:    end if
15:    $(NewP, NewPInfos) \leftarrow HookingSuppr(RP, IHCS)$ // *See algorithm 14*
16:    $\mathcal{F}_D^{i+1} \leftarrow \mathcal{F}_D^{i+1} \cup NewP$
17:    $NewPatternsInfos \leftarrow NewPatternsInfos \cup NewPInfos$
18:  end for
19:  $Update(\mathcal{F}_D^{i+1}, \mathcal{G}, NewPatternsInfos)$ // *See algorithm 16*
20:  $i \leftarrow i + 1$
21: end while
22: Return $\mathcal{F}_D^i$
```

Transactions	Items
nodes n of $\mathcal{BA}$ of label l	labels of nodes descendant of n

Table III.3: Frequent Item Sets for the detection of frequent patterns of depth 1 whose root is labelled by l, for a tree database $\mathcal{BA}$

Preuve: We have seen in the previous section that FindPFA_1 computes correctly $\mathcal{P}_1$ with the algorithm Eclat. The closure property of $Closed(\mathcal{P}_1)$ depends on the inclusion between elements of $\mathcal{P}_1$. There cannot be any inclusion between two patterns of $\mathcal{P}_1$ of different roots. A pattern of $\mathcal{P}_1$ having same root as another pattern of $\mathcal{P}_1$ will be included into it (by tree inclusion) if their sets of leaves are included by set inclusion. Hence in $\mathcal{F}_1 = Closed(\mathcal{P}_1)$ the sets of leaves are closed for the patterns having the same root labels.

And for a root of a given label, the sets of leaves are the results of the algorithms Eclat or Charm. As the algorithm Charm produces closed sets of leaves, we deduce that we have $\mathcal{F}_1 = Closed(\mathcal{P}_1)$ as an output of FindPFA_1 where Eclat has been replaced by Charm. □

We get a set $\mathcal{F}_D^1 = Closed(\mathcal{P}_1)$. By then we already know the nodes of the hooking graph of $\mathcal{F}_D^1$: these are the patterns of $\mathcal{F}_D^1$. To find the arcs of the graph and their correct labels, it is necessary for all couples $(P_1, P_2) \in \mathcal{F}_D^1 \times \mathcal{F}_D^1$ of hookable patterns to compute $Anchor(P_1, P_2)$. An implementation of this computation is given in the procedure *ComputeGraph*, shown in algorithm 7.

Algorithm 7 ComputeGraph

```
Require: $\mathcal{F}_D^1$ a closed set of frequent patterns of depth 1
Ensure: $\mathcal{G}$ the hooking graph of $\mathcal{F}_D^1$
 1: $\mathcal{A} \leftarrow \emptyset$
 2: for all $P_1 \in \mathcal{F}_D^1$ do
 3:   for all $P_2 \in \mathcal{F}_D^1$ do
 4:     if $hookable(P_1, P_2)$ then
 5:       $AnchorArc \leftarrow \emptyset$
 6:       $\{hn\} \leftarrow hook\_nodes(P_1, P_2)$ // only one hook node because depth 1 patterns
 7:       for all $occ_1 \in P_1.loc$ do
 8:         for all $occ_2 \in P_2.loc$ do
 9:           if $\exists \mu \in occ_1.mappings$ st $\mu(hn) = occ_2.root$ then
10:             $AnchorArc \leftarrow AnchorArc \cup \{(hn, occ_1.root, occ_2.root)\}$
11:           end if
12:         end for
13:       end for
14:       $\mathcal{A} \leftarrow \mathcal{A} \cup \{(P_2, P_1)$ of label $AnchorArc\}$
15:     end if
16:   end for
17: end for
18: Return $\mathcal{G} = (\mathcal{F}, \mathcal{A})$
```

Root patterns

The first step of a Dryade iteration is to find the root patterns of $\mathcal{F}_D$. This is done in the procedure *GiveRoots*, shown in algorithm 8.

Property 8 *From a closed set of frequent patterns $\mathcal{F}_D$ and its hooking graph $(\mathcal{F}_D, \mathcal{A})$, the procedure GiveRoots computes the set RPS of pairs (RP, RO) such that the union of all RP is the set of root patterns in $\mathcal{F}_D$, and for all pairs $(RP, RO) \in EPR$, RO is exactly the set of root occurrences of RP.*

Algorithm 8 *GiveRoots*

Require: The hooking graph $\mathcal{G} = (\mathcal{F}_D^i, \mathcal{A})$, for a given iteration i

Ensure: The set RPS of all the pairs (RP, RO) where RP is a root pattern and RO are its root occurrences.

```
$RPS_{tmp} \leftarrow \emptyset$
for all $P \in \mathcal{F}_D^i$ do
    $RO \leftarrow P.loc$
    for all $P' \in succ(P)$ do
        for all $(o', o) \in Anchor(P', P)$ do
            // $Anchor(P', P)$ is the label of arc $(P', P)$ computed by the procedure $ComputeGraph$
            $RO \leftarrow RO - o$
        end for
    end for
    if $\mid RO \mid \geq \varepsilon$ then
        $RPS_{tmp} \leftarrow RPS_{tmp} \cup \{(P, RO)\}$
    end if
end for
// We are looking for the root equivalence classes of the hookings quotient graph
$RPS \leftarrow \emptyset$
for all $(P, RO) \in RPS_{tmp}$ do
    if $P.loc = RO$ then
        // Equivalence class reduced to a singleton and root in the quotient graph
        $RPS \leftarrow RPS \cup (P, RO)$
    else
        // We determine if equivalence class of $P$ is a root cycle in the quotient graph.
        if $isInRootCycle(P, RO, \mathcal{G})$ then
            $RPS \leftarrow RPS \cup (P, RO)$
        end if
    end if
end for
Return $RPS$
```

Algorithm 9 *isInRootCycle*

Require: A pattern P, it's root occurrences RO, the hooking graph $\mathcal{G} = (\mathcal{F}_D^i, \mathcal{A})$

Ensure: Returns TRUE if P is in a root cycle of the quotient graph associated to $\mathcal{G}$, and FALSE otherwise.

// The algorithm of this function isn't given here. It is based on a classical cycle detection algorithm, enhenced to find the root cycles of the quotient graph associated to $\mathcal{G}$.

Proof: The algorithm iterates on all the patterns P of $\mathcal{F}_D$, so all potential root patterns are analyzed.

Let's show first that the root occurrences of patterns are correctly computed: We see at line 7 that to compute the root occurences of a pattern P, we remove from the set of the occurrences of P all the occurrences that do not satisfy the condition given in definition 27. At the end of the loop the set RO which as then been found is $RO(P)$. The condition on line 10 ensures that only the root patterns verifying $\mid RO(P) \mid \geq \varepsilon$ can be detected as root patterns.

Let's show now that the patterns resulting from *GiveRoots* are the patterns of the root equivalences classes from the hookings quotient graph: The are two kinds of root equivalences classes in the hookings quotient graph:

- First case, an equivalence class that does not correspond to a cycle. Hence this equivalence class contains only one pattern P. In the hookings quotient graph, this class has no father, it means that the pattern P doesn't hook on any other pattern, for any of it's occurrences. Hence $Locc(P, D) = RO(P)$. This case is detected by line 17 of the procedure *GiveRoots*.

- Second case, the equivalence class corresponds to a cycle, containing several patterns $P_1, ..., P_n$ $(n > 1)$. As this class is root for the hookings quotient graph, it has no father. This means that the patterns $P_1, ..., P_n$ doesn't hook on any pattern other than a pattern from the cycle. This is detected by the function *isInRootCycle*. In order to avoid overloading this chapter we haven't given it's code.

□

Patterns to hook under root patterns

For the rest, we suppose that we are at iteration $i > 1$ of Dryade.

When iteration i starts, we have the result of previous iteration $\mathcal{F}_D^{i-1}$, and after the execution of *GiveRoots* we have RPS the set of root patterns in $\mathcal{F}_D^{i-1}$. For each root pattern of RPS, the algorithm must discover the patterns of depth 1 to hook simultaneously on this pattern, and the occurrences for which this multiple hooking happens.

For this, we first extend the notion of hooking anchored in the data. Until now, we had restricted our hook anchor definition to the hooking of a frequent pattern of depth 1 on a frequent pattern having any depth. We now give the definition of the anchor for the simultaneous hooking of several patterns of depth 1 on a root pattern of any depth.

Definition 28 (Extension of anchor definition) *Let $\mathcal{F}_D^i$ be a closed set of frequent patterns, and RP a root pattern in $\mathcal{F}_D^i$. Let RO be the set of root occurrences of RP.*

Let $PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\} \subseteq N_{RP} \times \mathcal{F}_D^i$ be a set of pairs, each pair having an hook node on RP and a depth 1 pattern hooking on RP by this node.

The anchor of the set of pairs PS_1 on RP is defined by:

$Anchor(RP, PS_1) = \{(o_R, o_1, ..., o_n) \mid o_R \in RO, o_1 \in Locc(P_1, D), ..., o_n \in Locc(P_n, D)$ st $(o_R, o_1) \in Anchor_{hn_1}(RP, P_1) \wedge ... \wedge (o_R, o_n) \in Anchor_{hn_n}(RP, P_n)\}$.

We can thus define the notion of **multiple hooking set**, i.e. all the patterns of depth 1 that hook on a root pattern, with their anchors and hooking occurrences.

Definition 29 (Multiple hooking set and occurrence) *Let $\mathcal{F}_D^i$ be a closed set of frequent patterns, and RP a root pattern of $\mathcal{F}_D^i$. Let RO be the root occurrences of RP.*

Let $PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\} \subseteq N_{RP} \times \mathcal{F}_D^i$ be a set of pairs hook node on RP / frequent pattern of depth 1 hooking on RP by this node.

*We note $MHOS(PS_1)$ the set of **multiple hooking occurrences** of PS_1 defined by:*

$$MHOS(PS_1) = \{o \in RO \mid \exists (o, o_1, ..., o_n) \in Anchor(RP, PS_1)\}$$

*PS_1 is a **multiple hooking set** iff:*

- $\mid MHOS(PS_1) \mid \geq \varepsilon$
- *there are no pairs (hn_j, P_j) and (hn_k, P_k) in PS_1 satisfying $hn_j = hn_k$ and $P_j \sqsubset P_k$*

Definition 30 (Closed set of multiple hooking sets) *For a root pattern RP, the set $MHS(PR)$ is the **closed** set of all the multiple hooking sets on RP, i.e. it's the biggest set (according to set inclusion) of multiple hooking sets on RP such that: for all set $PS_1 \in MHS(RP)$, there is no set $PS'_1 \in MHS(RP)$ such that:*

- $PS'_1 \subset PS_1$
- $MHOS(PS'_1) = MHOS(PS_1)$

Let's consider a root pattern RP, and a multiple hooking set $PS_1 \in MHS(RP)$. In order to select the patterns from PS_1 corresponding to the depth level to build at iteration i, Dryade looks for the patterns that are root patterns **inside** PS_1 (i.e. we perform a new "stratification" of the patterns, limited here to the patterns of PS_1).

Definition 31 (Immediate multiple hooking set) *Let RP be a root pattern, and $PS_1 \in MHS(RP)$ be a multiple hooking set of RP.*

*The set $PS_{1R} = \{(hn'_1, P'_1), ..., (hn'_m, P'_m)\} \subseteq PS_1$ is an **immediate multiple hooking set** if each pattern P in PS_{1R} is a root pattern of the set of patterns of PS_1.*

*The **immediate hooking anchor** of PS_{1R} is the set $IHAS(PS_{1R}) \subseteq Anchor(RP, PS_{1R})$ such that: $\forall (o_R, o_1, ..., o_m) \in IHAS(PS_{1R})$ and $\forall o_i \in \{o_1, ..., o_m\}$ o_i is a root occurrence for P'_i when it is hooked by node hn'_i.*

*The set of **immediate hooking occurrences** of PS_{1R} is the set $IHOS(PS_{1R}) = \{o \in IHOS(PS_1) \mid \exists (o, o_1, ..., o_m) \in IHAS\}$, with $|IHOS| \geq \varepsilon$*

Notation : In the algorithms, we will call **immediate hooking context** the triple $(PS_{1R}, IHAS(PS_{1R}), IHOS(PS_{1R}))$.

For each root pattern RP, Dryade computes the closed set of all its immediate multiple hooking sets.

Definition 32 (Closed set of immediate multiple hooking sets) *Let RP be a root pattern of $\mathcal{F}_D$.*

*The **closed** set of all the immediate multiple hooking sets on RP, noted $IHCS(RP)$, is the biggest set (according to set inclusion) of immediate multiple hooking sets on RP such that for all set $PS_1 \in IHCS(RP)$, there is no set $PS'_1 \in IHCS(RP)$ satisfying:*

- $PS_1 \subset PS'_1$
- $IHOS(PS'_1) = IHOS(PS_1)$

Example: Let's consider the datatree D_Y. By analysing it's hooking graph, figure III.17, we can deduce that P_{11} is the only root pattern, and the candidates for hooking on P_{11} are the patterns P_{12}, P_{13} and P_{14}.

In fact, the closed set of multiple hooking sets on P_{11} contains only one element: the set $PS_1 = \{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13}), (hn_D^{11}, P_{14})\}$, for the occurrences $MHOS(PS_1) = \{n_1, n_8\}$.

Inside PS_1, the root patterns are P_{12} and P_{13} (to find them, one only has to look at the graph in figure III.17, and hide P_{11}). We then get the only immediate hooking context:

$IHC = (\{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}, \{(n_1, n_2, n_5), (n_8, n_{16}, n_{10})\}, \{n_1, n_8\})$

The set $IHCS(P_{11})$ is then reduced to the singleton $\{IHC\}$.

For a root pattern RP, whose set of root occurrences is RO, the procedure $PatternsToHook$, shown in algorithm 10, computes $IHCS(RP)$.

We will now explain how $PatternsToHook$ works, with an example. For this, let's consider the datatree D_Z shown in figure III.18. The frequency threshold is set to $\varepsilon = 1$. The closed set $\mathcal{F}_1$ of the frequent patterns of depth 1 is represented on figure III.19. The hooking graph for these patterns is shown in figure III.20. Obviously, the only root pattern is P_{11}, whose root occurrences set is $RO = \{n_1, n_8\}$.

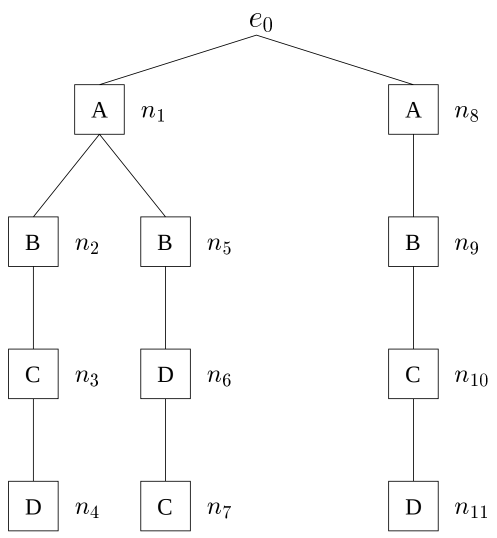

Figure III.18: D_Z

The procedure $PatternsToHook$ starts by building a transaction matrix M for the patterns hookable on P_{11}, see table III.4.

Transaction identifiers(occurrences)	(hn_B^{11}, P_{12})	(hn_C^{11}, P_{13})	(hn_D^{11}, P_{14})
n_1	$\{n_2, n_5\}$	$\{n_3\}$	$\{n_6\}$
n_8	$\{n_9\}$	$\{n_{10}\}$	

Table III.4: Transaction matrix M of the patterns to hook on P_{11}

This matrix contains for each occurrence $o \in RO$ and each pattern P_X hookable on P_{11} by hook node hn all the occurrences o' of P_X such that $(o, o') \in Anchor_{hn}(P_{11}, P_X)$. The attributes are not binary: so we apply CHARM to $boolean(M)$, which is a transformtion of M into binary attributes. The transformation rule is simple: an empty cell receives 0, a non-empty cell receives 1.

Algorithm 10 *PatternsToHook*

```
Require: A root pattern $RP$, its root occurrences set $RO$, $\mathcal{F}_D^i$ the frequent patterns
Ensure: $IHCS(RP)$ all the immediate hooking contexts on $RP$.
 1: $IHCS \leftarrow \emptyset$
 2: $MHS \leftarrow \emptyset$
 3: // Compute all multiple hooking sets
 4: Create the thansaction matrix $M$ with transactions = occurrences in $RO$, items = patterns
    of depth 1 in $\mathcal{F}_D^i$ hookable on $RP$ (for a pattern, as many items as hook nodes). We have:
    $M[o \in RO, P_1 \text{ on } hn \in hook\_nodes(RP, P_1)]$ = set of nodes of $P_1$ that hook on occurrence $o$ of
    $RP$.
 5: $MHS \leftarrow CHARM(boolean(M), \varepsilon)$
 6: $M' \leftarrow$ empty matrix
 7: for all $(PS_1, MHOS(PS_1)) \in MHS$ do
 8:   for all $(hn, P) \in EP_1$ do
 9:     // Inclusion test on depth 1 patterns, so can be performed easily
10:     if $\exists (hn', P') \in PS_1$ st $hn = hn'$ AND $P \sqsubset P'$ then
11:       $PS_1 \leftarrow PS_1 - \{(hn, P)\}$
12:     end if
13:   end for
14:   for all $occRP \in MHOS(PS_1)$ do
15:     $AnchorTransactions \leftarrow AllAnchors(occRP, M, 0)$ // See algorithm 11
16:     $AnchorTransactions \leftarrow RootInAnchor(AnchorTransactions, occRP, RP, PS_1)$ // See al-
        gorithm 12
17:     Add $AnchorTransactions$ to $M'$
18:     $transacMapping(occRP) \leftarrow$ all the transaction identifiers in $AnchorTransactions$
19:   end for
20: end for
21: Suppress from $M'$ duplicate transactions (same root patterns, same anchors for these root
    patterns)
22: $PS_{1R} \leftarrow CHARM(boolean(M'), \varepsilon)$
23: $IHCS \leftarrow ImmediateContexts(PS_{1R}, transacMapping)$ // See algorithm 13
24: // Guarantee closure on occurrences
25: for all $IHC = (PS_1, IHAS, IHOS) \in IHCS$ do
26:   if $\exists IHC' = (PS'_1, IHAS', IHOS') \in IHCS$ tq $PS_1 \subseteq PS'_1$ et $IHOS = IHOS'$ then
27:     $IHCS \leftarrow IHCS \setminus IHC$
28:   end if
29: end for
30: Return $IHCS$
```

Algorithm 11 *AllAnchors*

Require: M a transaction matrix, $occRP$ the identifier of a transaction in M, col a column index in M

Ensure: All the transactions of anchors corresponding to $occPR$ in M, from column col.

```
1:  $Result \leftarrow \emptyset$
2:  if $col \leq$ number of columns in $M$ then
3:     // Get all combinations that can be built from items in the next columns
4:     $tmpTrans \leftarrow AllAnchors(occRP, M, col + 1)$
5:     if $tmpTrans \neq \emptyset$ then
6:        if $M[occRP][col] \neq \emptyset$ then
7:           for all occurrence $o \in M[occRP][col]$ do
8:              for all $transac \in tmpTrans$ do
9:                 // For each combination of next columns, add current occurrence of column col
10:                $Result \leftarrow Result \cup \{\{o\} \cup transac\}$
11:             end for
12:          end for
13:       else
14:          $Result \leftarrow tmpTrans$
15:       end if
16:    else
17:       $Result \leftarrow M[occRP][col]$
18:    end if
19: end if
20: Return $Result$
```

Algorithm 12 *RootInAnchor*

Require: $AnchorTransactions$ a set of transactions, $occRP$ the occurrence of RP to which these transactions correspond, RP a root pattern, PS_1 a multiple hooking set on RP

Ensure: The transactions of $AnchorTransactions$ where for each transaction (so for each anchor) we only keep the patterns that are root for that anchor.

```
1: for all $T \in AnchorTransactions$ do
2:    for all $itemP_1 \in T$ do
3:       if $\exists itemP'_1 \in T$ st $(itemP'_1, itemP_1) \in Anchor(P'_1, P_1)$ then
4:          $T \leftarrow T - itemP_1$
5:       end if
6:    end for
7: end for
8: Return $AnchorTransactions$
```

Algorithm 13 *ImmediateContexts*

```
Require: a set $FISS$ of frequent itemsets $(PS_1, tidlist)$, where $tidlst$ is a list of anchors,
    $transacMapping$ maps each occurrence of $RP$ to its anchor (here we'll use $transacMapping^{-1}$),
    $M'$
Ensure: Returns the immediate hooking contexts on $RP$
 1: $CS \leftarrow \emptyset$
 2: for all $(PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\}, tidlist) \in FISS$ do
 3:     $Anchor \leftarrow \emptyset$
 4:     $TID\_occRP \leftarrow \emptyset$
 5:     for all $id \in tidlist$ do
 6:         $TID\_occRP \leftarrow TID\_occRP \cup \{transacMapping^{-1}(id)\}$ // union WITHOUT duplicates
 7:         $Anchor \leftarrow Anchor \cup \{(transacMapping^{-1}(id), M'[id, (hn_1, P_1)], ..., M'[id, (hn_n, P_n)])\}$
            // idem
 8:     end for
 9:     if $|TID\_occRP| \geq \varepsilon$ then
10:         $CS \leftarrow CS \cup \{(PS_1, Anchor, TID\_occRP)\}$
11:     end if
12: end for
13: Return $CS$
```

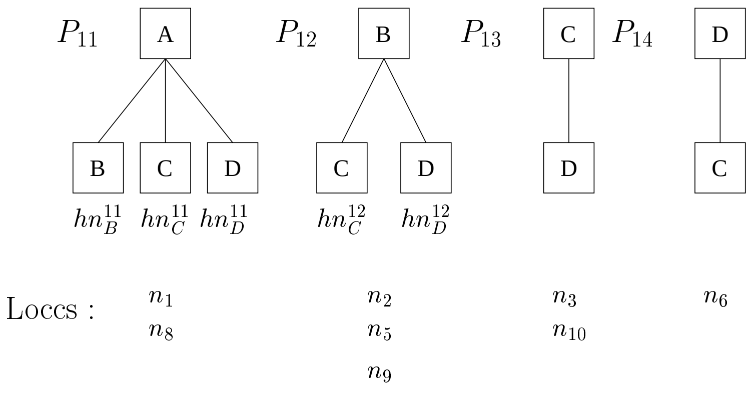

Figure III.19: Closed frequent patterns of depth 1 of D_Z

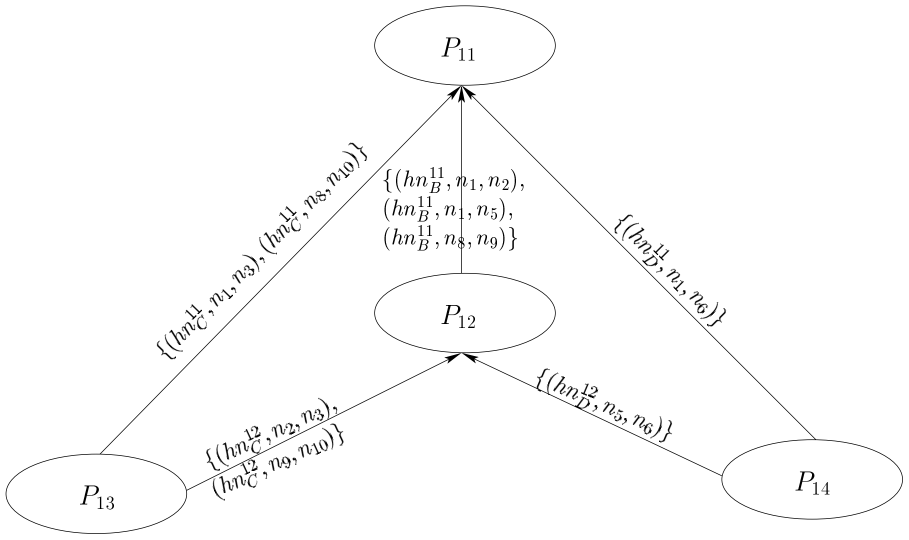
P_{11}
P_{12}
P_{13}
P_{14}
$\{(hn_C^{11}, n_1, n_3), (hn_C^{11}, n_8, n_{10})\}$
$\{(hn_B^{11}, n_1, n_2),$
$(hn_B^{11}, n_1, n_5),$
$(hn_B^{11}, n_8, n_9)\}$
$\{(hn_D^{11}, n_1, n_6)\}$
$\{(hn_C^{12}, n_2, n_3),$
$(hn_C^{12}, n_9, n_{10})\}$
$\{(hn_D^{12}, n_5, n_6)\}$

Figure III.20: Hooking graph of the closed frequent patterns of depth 1 of D_Z

We then get closed frequent itemsets which are multiple hooking sets on P_{11}, as well as their tidlists, which are the occurrences of the root pattern where the depth 1 patterns hook. Here, the frequent itemsets are:

- $PS_1^1 = \{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13}), (hn_D^{11}, P_{14})\}$, $tidlist^1 = \{n_1\}$
- $PS_1^2 = \{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}$, $tidlist^2 = \{n_1, n_8\}$

The next step of the algorithm consists in determining, among the multiple hooking sets discovered, the immediate multiple hooking sets, hence the immediate hooking contexts.

For this, we must use the anchors of the sets of frequent patterns.

Let's consider the first frequent itemset, $PS_1^1 = \{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13}), (hn_D^{11}, P_{14})\}$, $tidlist^1 = \{n_1\}$. By checking M, we can see that P_{12} hooked by hn_B^{11} has two possible occurrences. So there are two possible anchors, it's the goal of the recursive procedure *AllAnchors* to make them explicit. From a transaction of M, whose each item can be a set of occurrences, this procedure builds all the possible transactions, with only one occurrence by item. This is equivalent to building all the combinations of occurrences, by choosing one occurrence in each set of occurrences in an item. The new transactions are shown in the table III.5.

Transaction identifiers (anchors)	(hn_B^{11}, P_{12})	(hn_C^{11}, P_{13})	(hn_D^{11}, P_{14})
n_1^1	n_2	n_3	n_6
n_1^2	n_5	n_3	n_6

Table III.5: *AnchorTransactions*

Then, the procedure *RootInAnchor* computes for each transaction, therefore for each anchor, the patterns that are root for this anchor. The occurrences of these patterns do not hook on the occurrences of the other patterns in the anchor. The modified transactions are given in the table III.6.

These transactions are added to the matrix M'. Let's note that going from the "occurrences" granularity level to the "anchor" granularity level was unavoidable: what was regrouped in M at occurrence n_1 has been separed in M' in two different transactions.

Transaction identifiers (anchors)	(hn_B^{11}, P_{12})	(hn_C^{11}, P_{13})	(hn_D^{11}, P_{14})
n_1^1	n_2		n_6
n_1^2	n_5	n_3	

Table III.6: *AnchorTransactions* after *RootInAnchors*

The same operations on the frequent itemset $PS_1^2 = \{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}$, $tidlist^2 = \{n_1, n_8\}$ give the matrix M' of table III.7.

Transaction identifiers (anchors)	(hn_B^{11}, P_{12})	(hn_C^{11}, P_{13})	(hn_D^{11}, P_{14})
n_1^1	n_2		n_6
n_1^2	n_5	n_3	
n_1^3	n_2		
n_1^4	n_5	n_3	
n_8	n_9		

Table III.7: M' before suppressing duplicates

We then suppress duplicate transactions, here n_1^4 (duplicate of n_1^2), to get the final matrix of table III.8.

Transaction identifiers (anchors)	(hn_B^{11}, P_{12})	(hn_C^{11}, P_{13})	(hn_D^{11}, P_{14})
n_1^1	n_2		n_6
n_1^2	n_5	n_3	
n_1^3	n_2		
n_8	n_9		

Table III.8: final M'

The execution of Charm on M' gives us:

- $PS_1^{1'} = \{(hn_B^{11}, P_{12}), (hn_D^{11}, P_{14})\}$, $tidlist^1 = \{n_1^1\}$
- $PS_1^{2'} = \{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}$, $tidlist^2 = \{n_1^2\}$
- $PS_1^{3'} = \{(hn_B^{11}, P_{12})\}$, $tidlist^3 = \{n_1^1, n_1^2, n_1^3, n_8\}$

The result of *PatternsToHook* is the set of immediate hooking contexts:

- $IHC_1 = (\{(hn_B^{11}, P_{12}), (hn_D^{11}, P_{14})\}, \{(n_1, n_2, n_6)\}, \{n_1\})$
- $IHC_2 = (\{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}, \{(n_1, n_5, n_3)\}, \{n_1\})$
- $IHC_3 = (\{(hn_B^{11}, P_{12})\}, \{(n_1, n_2), (n_1, n_5), (n_8, n_9)\}, \{n_1, n_8\})$

(We can notice that in IHC_3, the anchor (n_1, n_2) has been produced two times, but the union without duplicates in the procedure *ImmediateContexts* has only kept one of them, as expected).

Property 9 *For a root pattern RP, the algorithm PatternsToHook computes correctly $IHCS(RP)$.*

Proof: The transaction matrix M built at line 4 of $PatternsToHook$ has as columns the patterns of depth 1 of $\mathcal{F}_D^i$ hooking on RP with their hook nodes, and as lines the occurrences of RP. So the j-th element of the i-th line indicates the occurrences of the pattern j that hooks on occurrence i of RP for a specific hook node.

The Charm algorithm of line 5 operates on a boolean version of M, for each occurrence of RP (line of M), we know the patterns of depth 1 that hook on this occurrence, as well as their hook node. So here Charm computes exactly the closed set $MHS(RP)$ of definition 30 (taking into account the suppression of patterns of PS_1 included into other patterns made at lines 9 to 13).

Then, for each pair $PS_1 \in MHS(RP)$ associated to its multiple hooking occurrences $MHOS(PS_1)$, the lines 8 to 12 of the algorithm fill the corresponding lines of matrix M'.

The columns of M' are the same as M, that are the patterns that hook on RP and their hook node, but the lines of M' are all the elements of $Anchor(RP, PS_1)$, for all the PS_1 in $MHS(PR)$.

For each $PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\} \in MHS(RP)$, the recursive function $AllAnchors$ fills for each occurrence o_R of $MHOS(PS_1)$ the lines of M' corresponding to the elements of $Anchor(RP, PS_1)$ that have o_R as RP occurrence. This is done by extending the line corresponding to the occurrence o_R in M, that contains for each pattern of PS_1 all its occurrences hooking on RO, by the specified hook node. So an element of $Anchor(RP, PS_1)$ having o_R as RP root corresponds to the choice of an occurrence for each of the $(hn_j, P_j), j \in [1, n]$ (this choice is performed for each (hn_j, P_j) by the loop of line 7 of $AllAnchors$). There are as many elements of $Anchor(RP, PS_1)$ having o_R as RP occurrence as there are possible choices for the occurrences of the $(hn_j, P_j), j \in [1, n]$.

For each line of M' corresponding to an element $(o_R, o_1, ..., o_n)$ of $Anchor(RP, PS_1)$, the procedure $RootInAnchor$ suppresses the occurrences o_j that hook on other occurrences $o_k \in \{o_1, ..., o_n\}$ (lines 3 and 4). This guarantees that the occurrences that stay in a line of M' are root occurrences.

While building M', two different multiple hooking sets PS_1, PS_1', with $PS_1 \subset PS_1'$, can produce the same lines after $RootInAnchor$. The line 21 of $PatternsToHook$ eliminates all duplicates from M'.

Applying Charm on M' at line 23 produces all the closed sets of patterns of depth 1 hooking on RP. The occurrences of these sets of patterns all are root occurrences, so the patterns are root patterns. Each set of pattern is an immediate multiple hooking set.

To ease the task of further procedures, we create structures called immediate hooking contexts. The creation of these structures is made by the function $ImmediateContexts$, which eliminates the immediate multiple hooking sets whose immediate hooking occurrences set $IHOS$ size is lower than ε (lines 9-10), and else sets the result in the expected output structure.

The resulting set $IHCS$ satisfies the closure property, but according to immediate hooking anchors. The definition 32 of $IHCS(RP)$ asks a closure according to immediate hooking occurrences. The lines 25 to 29 guarantee this closure according to immediate hooking occurrences.

Hence the resulting set $IHCS$ is $IHCS(RP)$, satisfying definition 32. □

Hooking and suppression of redundant leaves

The immediate hooking contexts previously computed allow to know which pattern of depth 1 to hook on a root pattern RP of $\mathcal{F}_D^i$, and on which hook node. The next step consits in performing the hookings correctly in order to get the patterns of $\mathcal{F}_D^{i+1}$.

The hooking itself isn't a problem, but once it has been done the redundant leaves coming from the root pattern must be suppressed from the pattern resulting from the hooking. Several combinations of suppressions are possible, and will produce several patterns of $\mathcal{F}_D^{i+1}$.

So for a root pattern RP and an immediate hooking context IHC of this pattern, we must be able to detect the redundant leaves.

Definition 33 (Redundant leaves for an anchor) *Let RP be a root pattern, let $IHC = (PS_1, IHAS, IHOS)$ be an immediate hooking context on RP, with $PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\}$. Let $f_{n_1}, ..., f_k$ $k > n$ be the leaves of the pattern RP where does not hook a pattern of PS_1.*

For an anchor $(o_R, o_1, ..., o_N) \in IHAS$, and for a mapping $\mu \in \mathcal{EM}_{o_R}(RP, D)$ such that $\forall i \in [1, n]$ $\mu(hn_i) = o_i$, a leaf $f_{red} \in \{f_{n+1}, ..., f_k\}$ is redundant if there exists a pattern $P_j \in \{P_1, ..., P_n\}$, a mapping $\mu' \in \mathcal{EM}_{o_j}(P_j)$ and a leaf f' of P_j such that $\mu(f_{red}) = \mu'(f')$.

We introduce the notion of **pattern creation context**, with which it is possible to know exactly what patterns to create. The computing of these pattern creation contexts from the immediate hooking contexts comes before the creation of the patterns by hooking and suppression of the redundant leaves.

Definition 34 (Pattern creation context) *Let RP be a root pattern, let $IHC = (PS_1, IHAS, IHOS)$ be an immediate hooking context on RP, with $PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\}$. Let $f_{n_1}, ..., f_k$ $k > n$ be the leaves of the pattern RP where does not hook a pattern of PS_1.*

A pattern creation context $PCC = (PS_1, FG, PCAS, PCOS)$ is made of:

- *the set PS_1 of the frequent patterns of depth 1 immediately hooking on RP, with their hook nodes ;*
- *the set $FG = \{f_{\neg red.1}, ..., f_{\neg red.m}\} \subseteq \{f_{n+1}, ..., f_k\}$ of the leaves of RP that musn't be suppressed after hooking ;*
- *the set $PCAS$ of the anchors supporting the immediate hooking of PS_1 on RP where the leaves of FG are not redundant. An anchor $PCA \in PCAS$ is $PCA = (o_R, o_1, ..., o_n, o_{\neg red.1}, ..., o_{\neg red.m})$, where $(o_R, o_1, ..., o_n) \in IHAS$ and for all mapping $\mu \in \mathcal{EM}_{o_R}(RP, D)$ such that $\forall i \in [1, n]$ $\mu(hn_i) = o_i$ and $\forall i \in [1, m]$ $\mu(f_{\neg red.i}) = o_{\neg red.i}$, the leaves $f_{\neg red.i}, i \in [1, m]$ are not redundant ;*
- *the set $PCOS$ of the occurrences of RP appearing if $PCAS$, with $|PCOS| \geq \varepsilon$.*

For a root pattern RP and an immediate hooking context IHC, we want to find the **closed** set $PCCS(RP, IHC)$ of all the pattern creation contexts for RP and IHC.

Definition 35 *Let RP be a root pattern and $IHC \in IHCS(RP)$ be an immediate hooking context on RP. $PCCS(RP, IHC)$ is the closed set of the pattern creation contexts on RP, IHC. Hence it is the biggest set such that there are no two pattern creation contexts $PCC = (PS_1, FG, PCAS, PCOS)$ and $PCC' = (PS'_1, FG', PCAS', PCOS')$ in $PCCS(RP, IHC)$ such that:*

- *$FG \subset FG'$*
- *$PCAS = PCAS'_{|FG}$ (we only consider common parts of anchors, i.e. the occurrences of PS_1 and FG)*

Hence for each IHC we get a set of pattern creation contexts $PCCS(RP, IHC)$. The set $\cup_{IHC \in IHCS} PCCS(RP, IHC)$ is the set that will be used to perform hookings. Nothing guarantees that this set is closed according to patterns to hook and RP occurrences, so we define a new set $PCCS$.

Definition 36 *Let RP be a root pattern, and $IHCS(RP)$ be the set of immediate hooking contexts on RP. We define $PCCS = Closed(\cup_{IHC \in IHCS} PCCS(RP, IHC))$, i.e. there are no two pattern creation contexts $PCC = (PS_1, FG, PCAS, PCOS)$ and $PCC' = (PS'_1, FG', PCAS', PCOS')$ in $PCCS(RP, IHC)$ such that:*

- $PS_1 \subset PS'_1$
- $PCOS = PCOS'$

We can at last create frequent patterns by hooking frequent patterns of depth 1 and suppressing the redundant leaves, with the guidance of the pattern creation contexts of $PCCS$.

The procedure $HookingSuppr$, shown in algorithm 14, computes the set $PCCS$ and then builds the patterns associated to it.

Algorithm 14 $HookingSuppr$

```
Require: A root pattern $RP$, its immediate hooking contexts $IHCS$
Ensure: $PS$ the closed set of frequent patterns resulting from the hooking of the patterns in the
    contexts $IHCS$ on $RP$, with suppression of redundant leaves, $NewPattInfos$ which gives for
    each new patterns the patterns it comes from.
 1: $PS \leftarrow \emptyset$
 2: $PCCS \leftarrow \emptyset$
 3: $NewPattInfos \leftarrow \emptyset$
 4: for all $IHC = (PS_1, IHAS, IHOS) \in IHCS$ do
 5:    $PCCS \leftarrow PCCS \cup CreatePCC(RP, IHC)$ // See algorithm 15
 6: end for
 7: // Guarante closure on occurrences
 8: for all $PCC = (PS_1, FG, PCAS, PCOS) \in PCCS$ do
 9:    if $\exists PCC' = (PS'_1, FG', PCAS', PCOS') \in PCCS$ st $PS_1 \subseteq PS'_1$ and $PCOS = PCOS'$
       then
10:       $PCCS \leftarrow PCCS \setminus PCC$
11:    end if
12: end for
13: for all $PCC = (PS_1, FG, PCAS, PCOS) \in PCCS$ do
14:    $P_{new} \leftarrow$ new tree based upon $RP \oplus PS_1$
15:    Suppress from $P_{new}$ the leaves coming from $RP$ that are not in $FG$
16:    $P_{new}.loc \leftarrow \emptyset$
17:    for all $o \in PCOS$ do
18:       $Occ.root \leftarrow o$
19:       $Occ.mappings \leftarrow \{$ mappings of $RP$ corresponding to the anchors in $PCAS\} \circ \{$ mappings
          of $PS_1$ corresponding to the anchors in $PCAS\}$
20:       $P_{new}.loc \leftarrow P_{new}.loc \cup Occ$
21:    end for
22:    $PCAS' \leftarrow PCAS$ without the references to non redundant leaves (same infos as an $IHC$)
23:    $NewPattInfos \leftarrow NewPattInfos \cup \{(P_{new}, RP, (PS_1, PCAS', PCOS))\}$
24:    $PS \leftarrow PS \cup P_{new}$
25: end for
26: Return $(PS, NewPattInfo)$
```

Algorithm 15 *CreatePCC*

```
Require: A root pattern $RP$, an immediate hooking context $IHC = (PS_1, IHAS, IHOS)$ (with
    $PS_1 = \{P_1, ..., P_n\}$). Leaves of $RP = \{f_1, ..., f_n, f_{n+1}, ..., f_k\}$
Ensure: $PCCS$ the set of pattern creation contexts for $RP$ and $IHC$.
 1: $PCCS \leftarrow \emptyset$
 2: // Creation of the matrix to discover the set of non-redundant leaves
 3: $M \leftarrow$ empty transaction matrix, columns = leaves $f_{n+1}, ..., f_k$
 4: for all $A = (o_R, o_1, ..., o_n) \in IHOS$ do
 5:   for all $Occ \in RP.loc$ st $Occ.root \in IHAS$ do
 6:     for all $\mu \in Occ.mappings$ st $\mu(f_1) = o_1$ and ... and $\mu(f_n) = o_n$ do
 7:       Create a transaction $T$, $T.tid = (o_R, o_1, ..., o_n, \mu(f_{n+1}), ..., \mu(f_k))$
 8:       for all leaf $f \in \{f_{n+1}, ..., f_k\}$ do
 9:         if $\nexists P_i \in PS_1, Occ' \in P_i.loc, \mu' \in Occ'.mappings, f' \in leaves(P_i)$ with $\mu'(root(P_i)) =$
            $o_i$ st $\mu(f) = \mu'(f')$ then
10:           $T[f] \leftarrow 1$ // for this occurrence $f$ exists as an exclusive leaf of $o_R$, so it must not be
              suppressed
11:         else
12:           $T[f] \leftarrow 0$
13:         end if
14:       end for
15:       Add $T$ to $M$
16:     end for
17:   end for
18: end for
19: $FISS \leftarrow CHARM(M, \varepsilon)$
20: Suppress from $FISS$ all the itemsets whose anchors of their tidlist correspond to less than $\varepsilon$
    occurrences of $IHOS$
21: // Creation of the patterns creation contexts
22: for all $(fis, tidlist) \in FISS$ do
23:   $PCOS \leftarrow$ set of occurrences of $RP$ appearing in the anchors of the tidlist
24:   $PCCS \leftarrow PCCS \cup \{(PS_1, fis, tidlist, PCOS)\}$
25: end for
26: // Handling the case where all leaves are suppressed
27: if there is no $PCOS$ such that $PCOS = IHOS$ then
28:   $PCCS \leftarrow PCCS \cup \{(PS_1, \emptyset, IHAS, IHOS)\}$
29: end if
30: Return $PCCS$
```

Example: To demonstrate how *HookingSuppr* works, we will perform the hooking of the immediate hooking contexts IHC_1, IHC_2, IHC_3 previously defined on the root pattern P_{11}. We recall the content of these immediate hooking contexts:

$$IHC_1 = (\{(hn_B^{11}, P_{12}), (hn_D^{11}, P_{14})\}, \{(n_1, n_2, n_6)\}, \{n_1\})$$

$$IHC_2 = (\{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}, \{(n_1, n_5, n_3)\}, \{n_1\})$$

$$IHC_3 = (\{(hn_B^{11}, P_{12})\}, \{(n_1, n_2), (n_1, n_5), (n_8, n_9)\}, \{n_1, n_8\})$$

We name f_B, f_C and f_D the leaves of P_{11}.

When computing $PCCS(P_{11}, IHC_1)$, the only leaf that may not be redundant is f_C, because P_{12} hooks on $f_B = hn_B^{11}$ and P_{14} hooks on $f_D = hn_D^{11}$. The corresponding matrix M, built by $CreatePCC$, is then:

Transaction identifier = $(P_{11}, (hn_B^{11}, P_{12}), (hn_D^{11}, P_{14}), f_C)$	f_C
(n_1, n_2, n_6, n_3)	0
(n_1, n_2, n_6, n_7)	0

A 0 in a cell of the matrix indicates that for this transaction, f_C is redundant and must be suppressed. Clearly here, the leaf f_C is redundant, so it must be suppressed. Hence $PCCS(P_{11}, IHC_1) = \{PCC_1\}$, with $PCC_1 = (\{(hn_B^{11}, P_{12}), (hn_D^{11}, P_{14})\}, \emptyset, \{(n_1, n_2, n_6)\}, \{n_1\})$.

The matrix for the hooking of IHC_2 on P_{11} built by $CreatePCC$ is:

Transaction identifier = $(P_{11}, (hn_B^{11}, P_{12}), (hn_C^{11}, P_{13}), f_D)$	f_D
(n_1, n_5, n_3, n_4)	0
(n_1, n_5, n_3, n_6)	0

This case is similar to the previous one, so $PCCS(P_{11}, IHC_2) = \{PCC_2\}$, with $PCC_2 = (\{(hn_B^{11}, P_{12}), (hn_C^{11}, P_{13})\}, \emptyset, \{(n_1, n_5, n_3)\}, \{n_1\})$.

For the hooking of context IHC_3, the matrix M built by $CreatePCC$ is:

Transaction identifier = $(P_{11}, (hn_B^{11}, P_{12}), f_C, f_D)$	f_C	f_D
(n_1, n_2, n_3, n_4)	0	0
(n_1, n_2, n_3, n_6)	0	1
(n_1, n_2, n_7, n_4)	1	0
(n_1, n_2, n_7, n_6)	1	1
(n_1, n_5, n_7, n_6)	0	0
(n_1, n_5, n_7, n_4)	0	1
(n_1, n_5, n_3, n_6)	1	0
(n_1, n_5, n_3, n_4)	1	1
$(n_8, n_9, n_{10}, n_{11})$	0	0

The case where all the leaves are suppressed still happens thanks to occurrence n_8. But this time there are other closed frequent sets, we find in $PCCS(P_{11}, IHC_3)$:

- $PCC_3 = (\{(hn_B^{11}, P_{12})\}, \{f_C\}, \{(n_1, n_2, n_7, n_4), (n_1, n_2, n_7, n_6), (n_1, n_5, n_3, n_6), (n_1, n_5, n_3, n_4)\}, \{n_1\})$
- $PCC_4 = (\{(hn_B^{11}, P_{12})\}, \{f_D\}, \{(n_1, n_2, n_3, n_6), (n_1, n_2, n_7, n_6), (n_1, n_5, n_7, n_4), (n_1, n_5, n_3, n_4)\}, \{n_1\})$
- $PCC_5 = (\{(hn_B^{11}, P_{12})\}, \{f_C, f_D\}, \{(n_1, n_2, n_7, n_6), (n_1, n_5, n_3, n_4)\}, \{n_1\})$

- $PCC_6 = (\{(hn_B^{11}, P_{12})\}, \emptyset, \{(n_1, n_2), (n_1, n_5), (n_8, n_9)\}, \{n_1, n_8\})$

However, when building $PCCS = PCCS(P_{11}, IHC_1) \cup PCCS(P_{11}, IHC_2) \cup PCCS(P_{11}, IHC_3)$, the pattern creation contexts PCC_3, PCC_4 and PCC_5 are eliminated by the closure property. So $PCCS = \{PCC_1, PCC_2, PCC_6\}$.

The figure III.21 shows all the patterns created from $PCCS$.

Definition 37 (Frequent patterns by immediate multiple hookings) *Let RP be a root pattern, and let $PS_1 = \{(hn_1, P_1), ..., (hn_n, P_n)\}$ be an immediate multiple hooking set on RP. The set $SPatt(RP, PS_1)$ of the frequent patterns resulting from the immediate multiple hooking of PS_1 on RP is the closed set of the patterns P satisfying:*

- $P \sqsubseteq ((RP \oplus P_1) \oplus ...) \oplus P_n$
- $\mid Locc(P, D) \mid \geq \varepsilon$
- *P hasn't any redundant leaf*
- *There is no pattern $P' \in SPatt(PR, EP_1)$ such that:*
  - $P' \sqsubset P$
  - $Locc(P', D) = Locc(P', projected(P, D)) (= Locc(P, D)$ *here*$)$

Definition 38 (Closed set of the patterns derived from a root patterns) *Let RP be a root pattern. $SPatt(RP)$ is the* ***closed set of the patterns derived from*** *RP:*

$SPatt(RP)$ is the biggest set such that $SPatt(RP) \subseteq \bigcup_{PS_1 \in IHCS(RP)} SPatt(RP, PS_1)$ and for all pattern $P \in SPatt(PR)$ there is no $P' \in SPatt(RP)$ such that:

- $P' \sqsubset P$
- $Locc(P', D) = Locc(P', projected(P, D)) (= Locc(P, D)$ *here*$)$

Property 10 *From a root pattern RP and the set $IHCS(RP)$ of its immediate hooking contexts, the procedure $HookingSuppr$ computes $SPatt(RP)$.*

Proof: For each $IHC = (PS_1, IHAS, IHOS) \in IHCS(RP)$, $HookingSuppr$ calls the procedure $CreatePCC$. The goal of this procedure is to compute $PCCS(RP, IHC)$, let's show that it actually does it.

From lines 3 to 19, the procedure $CreatePCC$ creates a matrix M. This matrix will be used to compute the combinations of non-redundant leaves (leaves that must not be suppressed). Each transaction of the matrix is built from a mapping respecting an element from $IHAS$ (test of line 6), from which we consider the occurrences for the leaves of RP where no pattern of PS_1 hooks (leaves $\{f_{n+1}, ..., f_k\}$). For each of these occurrences, the test of line 9 checks that it isn't redundant. This tests exactly corresponds to the definition 33.

So we deduce that the closed set $FISS$ produced at line 19 corresponds to the anchors and non redundant leaves of $PCCS(RP, IHC)$, without taking care of the constraint on the number of occurrences of $IHOS$ in the anchors. The line 20 ensures the satisfaction of this constraint, and the lines 22 to 25 ensure correct formatting. To get exactly the set $PCCS(RP, IHC)$, we have to take into account the case when $FG = \emptyset$, never taken into account by the algorithm Charm.

Actually, to compute a closed set like $PCCS(RP, IHC)$, Charm does not takes into account the case when $FG = \emptyset$. But here this case is interesting, because it's the case when all the leaves are suppressed. Hence we also take it into account. The empty set is included into all other sets, but the

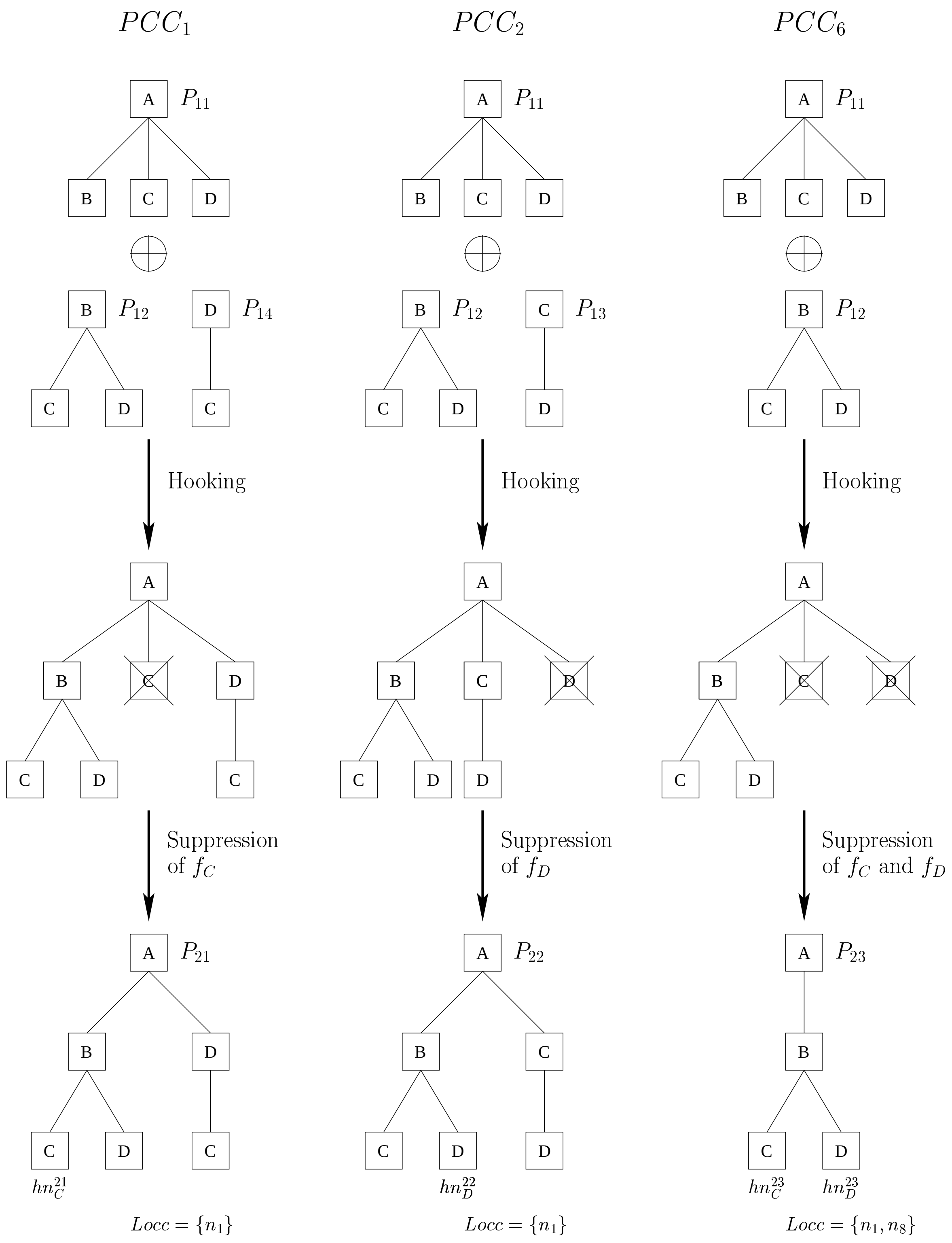

Figure III.21: Patterns created from the pattern creation contexts in $PCCS$

support of the pattern where all the leaves are suppressed is the set $IHAS$: all the anchors support at least that case, precisely because the empty set is minimal for pattern inclusion. Hence the case $FG = \emptyset$ will not be eliminated if there exists at least one anchor $A = (o_R, o_1, ..., o_n) \in IHAS$ that does not appear in any $PCAS$ of the set of pattern creation contexts. For this, we demonstrate that the occurrence o_R of A must not appear in any $PCAS$ of the set of pattern creation contexts. To do this, one have to notice that in the opposite case, if there exists a pattern creation context whose $PCAS$ contains an anchor like $(o_R, o'_1, ..., o'_n)$ having non-redundant leaves, then either its non-redundant leaves are not descendants of $o_1, ..., o_n$, and so the pattern of anchor A can also use them as non-redundant leaves, or its non-redundant leaves are descendants of the nodes $o_1, ..., o_n$, hence symetrically A can use the descendants of $o'_1, ..., o'_n$ as non-redundant leaves.

We deduce that if an occurrence of $IHOS$ does not appear in any $PCAS$, then the pattern creation context $(PS_1, \emptyset, IHAS, IHOS)$ is in $PCCS(RP, IHC)$.

The lines 27 to 29 of $CreatePCC$ handle this case.

So the set $PCCS$ returned correspond exactly to the set $PCCS(RP, IHC)$ of definition 35.

The procedure $HookingSuppr$ does the union of the $PCCS(RP, IHC)$ for all $IHC \in IHCS$, then in lines 7 to 12 guarantees that the set $PCCS$ obtained is closed according to patterns to hook and occurences. Hence the $PCCS$ set after line 12 is the set defined in definition 36.

After, for each pattern creation context $PCC = \{PS_1, FG, PCAS, PCOS\} \in PCCS$, the algorithm build a pattern P such that:

- $P = RP \oplus PS_1$ (line 14)
- P only keeps the leaves of RP that are in FG, so doesn't contain any redundant leaf (line 15)
- The occurrences of P are the occurrences of $PCOS$ (loop of line 17 and line 18)
- The mappings of P correspond to the anchor $PCAS$ (line 19)

Let's show that the set PS of patterns returned is the set $SPatt(RP)$ of definition 38.

For a given immediate multiple hooking set PS_1, and more precisely for its immediate hooking context $IHC = (PS_1, IHAS(PS_1), IHOS(PS_1))$, we have computed the set $PCCS(RP, IHC)$. Each element $PCC \in PCCS(RP, IHC)$ allows to make a pattern P, as previously described. Each pattern P built this way satisfies the conditions of the definition of $SPatt(RP, PS_1)$ (definition 37). The closure of the set of previously built patterns comes from the closure of $PCCS(RP, IHC)$: the only differences between the created patterns are the set of suppressed leaves. As $PCCS(RP, IHC)$ is closed according to the leaves to suppress, then from $PCCS(RP, IHC)$ we correctly built $SPatt(RP, IHC)$

Hence the patterns built from $\cup_{IHC \in IHCS} PCCS(RP, IHC)$ make $\cup_{IHC \in IHCS} SPatt(RP, IHC)$.

The patterns built by $HookingSuppr$ are created from $PCCS = Closed(\cup_{IHC \in IHCS} PCCS(RP, IHC))$.

So any created pattern $P \in PS$ is in $\cup_{IHC \in IHCS} SPatt(RP, IHC)$.

Let's show that we have the closure. By negation, let's suppose that it isn't the case. Then there exists P_1 and P_2 in EP such that:

- $P_1 \sqsubset P_2$
- $Locc(P_1, D) = Locc(P_2, D)$

P_1 and P_2 cannot come from the hooking of the same immediate multiple hooking set PS_1 on RP, because this would contradict the closure of $SPatt(RP, PS_1)$, which is guaranteed by the closure of $PCCS(RP, PS_1)$ as previously demonstrated.

So P_1 comes from an immediate multiple hooking set PS_1 and P_2 comes from a distinct immediate multiple hooking set PS'_1.

As $P_1 \sqsubset P_2$, we deduce $PS_1 \subsetneq PS'_1$.

P_1 and P_2 come from the pattern creation contexts $PCC = (PS_1, FG, PCAS, PCOS)$ and $PCC = (PS'_1, FG', PCAS', PCOS')$. The occurrences of P_1 and P_2 correspond respectively to $PCOS$ and $PCOS'$, so $PCOS = PCOS'$. We deduce that the set of pattern creation contexts $PCCS$ is not closed, hence a contradiction.

So $PS = SPatt(RP)$. □

Hookings update

$HookingSuppr$ adds patterns to $\mathcal{F}_D^i$, creating $\mathcal{F}_D^{i+1}$. However, no arc is added in the hooking graph towards these patterns. Furthermore, the closure of $\mathcal{F}_D^{i+1}$ can be jeopardized by the adding of the new patterns. The $Update$ procedure, shown in algorithm 16, updates the hooking graph and ensures the closure of $\mathcal{F}_D^{i+1}$, in order to prepare the next iteration.

Property 11 *After the execution of Update, the hooking graph contains all the arcs towards the patterns added by HookingSuppr, correctly labelled.*

Proof: The $Update$ procedures receives as argument a structure NPI that contains the set of all patterns created by $HookingSuppr$, and for each of these patterns P the root pattern RP it comes from, as well as its immediate hooking context $IHC = (PS_1, IHAS, IHOS)$.

And for a pattern P resulting from the hooking of PS_1 on RP, the only patterns of $\mathcal{F}_D^{i+1}$ that can hook on it are the patterns hooking on the patterns of PS_1. The two loops of lines 5 and 6 do analyse those patterns. Let P_X be a pattern hooking on $P_j \in PS_1$. Then: $Anchor(P, P_X) = \{(o, o_x) \mid (o, ..., o_j, ...) \in IHAS$ and $(o_j, o_x) \in Anchor(P_j, P_X)\}$. The building of $NewAnchor$, line 8 to 12, satifies these constraints, so $NewAnchor = Anchor(P, P_X)$. An arc from P_X to P is added only if $Anchor(P, P_X)$ isn't empty. Hence the definition of hooking graph (definition 23) is satisfied, so the property is true. □

Property 12 *After the execution of Update,* $\mathcal{F}_D^{i+1}$ *is closed.*

Proof: This is trivial, the computations made from line 31 to 35 eliminate all the patterns that do not respect the closure property of $\mathcal{F}_D^{i+1}$. □

Consumption of used anchor elements: In order to simplify next iterations, $Update$ consumes all the anchors elements used in current iteration to build frequent patterns, from lines 19 to 27. Actually these elements will never be usefull again, and they musn't be used, else the same patterns would be produced again and again.

Example: Let's continue our example on the datatree D_Z. The patterns P_{11} and P_{12} are both eliminated of $\mathcal{F}_D^{i+1}$, because their occurrences sets on D_Z are included in their occurrences set in $projected(P_{23}, D_Z)$. The new hooking graph is shown in figure III.22.

4.4 Soundness and completeness of Dryade

We will demonstrate in this section that the algorithm Dryade terminates, and that it is sound and complete.

Lemma 1 (Termination) *For all input datatree D and threshold ε, the* Dryade *algorithm terminates. When* Dryade *terminates, there are no arcs left in the hooking graph $\mathcal{G}$.*

Algorithm 16 *Update*

Require: $\mathcal{F}_D^{i+1}$ new set of frequent patterns, $\mathcal{G}$ hooking graph, NPI informations about all patterns created by $HookingSuppr$, with format (created patterns, root pattern, immediate hooking context).

Ensure: G updated hooking graph, $\mathcal{F}_D^{i+1}$ is closed

```
 1: for all $(PS, RP, IHC = \{PS_1, IHAS, IHOS\}) \in NPI$ do
 2:   // All the new patterns of $PS$ come from hookings of patterns in $PS_1$ on $RP$
 3:   for all $P \in PS$ do
 4:     // Add arcs towards the new pattern
 5:     for all $P_1 \in PS_1$ do
 6:       for all $P_X \in pred(P_1)$ do
 7:         $NewAnchor \leftarrow \emptyset$
 8:         for all $(o_1, o_2) \in Anchor(P_1, P_X)$ do
 9:           if $\exists (o_R, o_1) \in IHAS$ then
10:             $NewAnchor \leftarrow NewAnchor \cup \{(o_1, o_2)\}$
11:           end if
12:         end for
13:         if $NewAnchor \neq \emptyset$ then
14:           Create an arc in $\mathcal{G}$ from $P_X$ to $P$, with label $NewAnchor$
15:         end if
16:       end for
17:     end for
18:     // Udate labels of anchors used in this iteration of DRYADE
19:     $PCC = (PS_1, FG, PCAS, PCOS) \leftarrow$ pattern creation context for $P$
20:     for all arc $(RP, P_j)$, $P_j \in EP_1$ do
21:       for all $(o_R, o_1, ..., o_n, o_{n+1}, ..., o_k) \in PCAS$ do
22:         Suppress $(o_R, o_j)$ from the label of $(PR, P_j)$
23:       end for
24:       if the label is empty then
25:         Suppress the arc $(PR, P_j)$
26:       end if
27:     end for
28:   end for
29: end for
30: // Guarantee closure of $\mathcal{F}_D^{i+1}$
31: for all $P \in \mathcal{F}_D^{i+1}$ do
32:   if $\exists P' \in \mathcal{F}_D^{i+1}$ st $P \sqsubset P'$ and $Locc(P, D) = Locc(P, projected(P', D))$ then
33:     $\mathcal{F}_D^{i+1} \leftarrow \mathcal{F}_D^{i+1} - P$
34:   end if
35: end for
```

$$\begin{tikzcd}
P_{21} & P_{22} & P_{23} \\
P_{13} \arrow[u, "\{(hn_C^{21}, n_1, n_3)\}"] \arrow[urr, "{\{(hn_C^{23} n_1, n_3), (hn_C^{23}, n_8, n_{10})\}}"] & & P_{14} \arrow[ul, "\{(hn_D^{22}, n_1, n_6)\}"] \arrow[u, "\{(hn_D^{23}, n_1, n_6)\}"']
\end{tikzcd}$$

Figure III.22: New hooking graph for D_Z

Proof: The Dryade algorithm terminates if there exists an iteration k such that for a root pattern RP of $\mathcal{F}_D^k$, the set $IHCS$ computed from RP is empty. For that set to be empty, there musn't be any immediate multiple hooking possible on RP. And if there exists at least one pattern P hooking on RP, either the hooking is immediate, or it is not immediate and in this case there exists a pattern P' such that P hooks on P' and P' hooks on RP. So P' is in immediate hooking with RP.

So for Dryade to terminate, there mustn't be any arcs left in the graph $\mathcal{G}$ of hookings: no more patterns $\mathcal{F}_D^k$ hook with others patterns of $\mathcal{F}_D^k$. And at each iteration, the *Update* procedure consumes arcs of $\mathcal{G}$.

Let i be an iteration of Dryade. Let P be a pattern of $\mathcal{F}_D^i$. Let's call *root distance* the greatest number of arcs separing P from a root pattern of $\mathcal{F}_D^i$ in $\mathcal{G}$. Then at each iteration, the arcs consumed by *Update* decrease by 1 the root distance of P (if P isn't eliminated from the graph). Actually, let RP be the root pattern for which the root distance of P is maximal. So there exists a maximal chain of patterns in the graph: $P \rightarrow P_1 \rightarrow .... \rightarrow P_n \rightarrow RP$. By definition, in this chain P_n is in immediate hooking on RP. The *HookingSuppr* procedure will then produce a pattern P' resulting from the hooking of P_n on RP (among others). The *Update* procedure will then suppress the hookings between P_n and RP. Some hookings towards P' will also be added by *Update*, the different chains of patterns that can be produced are:

- $P \rightarrow P_1 \rightarrow ... \rightarrow P_n$ (if P_n isn't suppressed)
- $P \rightarrow P_1 \rightarrow ... \rightarrow P_{n-1} \rightarrow RP$ (if RP isn't suppressed)
- $P \rightarrow P_1 \rightarrow ... \rightarrow P_{n-1} \rightarrow P'$

All these chains have a length that are inferior by 1 to the length of the chain $P \rightarrow P_1 \rightarrow .... \rightarrow P_n \rightarrow RP$. Hence the root distance of P decreases by 1 at each iteration.

So for each pattern in $\mathcal{F}_D^i$, there exists an iteration where either it is suppressed, or its root distance is zero, which means it is a root pattern itself.

If P is root, then for its root occurrences it doesn't hook on any pattern. For its other occurrences it hooks on patterns that are in its cycle equivalence class, so these patterns are also root: these hookings too will be suppressed in the next iterations

We deduce that there exists an iteration for which $\mathcal{G}$ has no more arcs, and so DRYADE terminates. □

We will now give a formal characterization of the results of DRYADE.

Property 13 (Frequent occurrences independancy property) *A pattern satisfies the frequent occurrences independancy property if it has at least ε occurrences such as no one is a descendant of another.*

Remark 4 *Every pattern frequent by identifier satisfies the frequent occurrences independancy property.*

Lemma 2 *If all the frequent patterns of $\mathcal{F}$ satisfy the frequent occurrences independancy property, then the closed set $\mathcal{F}_1$ of the frequent patterns of depth 1 is the closed set of the flattenings of the closed set $\mathcal{F}$ of the frequent patterns and of the flattenings of the subtrees of the patterns in $\mathcal{F}$.*

Proof: Assume that all the frequent patterns of $\mathcal{F}$ satisfy the frequent occurrences independancy property. Let's show that the equality holds.

To prove $\supset$: For $Q \in \mathcal{F}$, we know that its flattening is in $\mathcal{F}_1$ (property 5). Let T be a subtree of a pattern $Q \in \mathcal{F}$. As Q verifies the frequent occurrences independancy property, then for the occurrences of Q which are not descendant of one another the corresponding roots of T are all distinct. So T has at least ε occurrences. Hence $P = flat(T)$ is frequent. If $P \notin \mathcal{F}_1$, then it has been suppressed by the closure property. Hence there exists $P' \in \mathcal{F}_1$ such that $P \sqsubset P'$ and $Locc(P, D) = Locc(P', D)$. So $Locc(P', D) = Locc(T, D)$ because P has the same occurrences as T. As $P \sqsubset P'$, there exists at least one label e that is in P' but not in P, and for the occurrences of P', hence also for those of T, this label is always a descendant of the root of P' (hence also the root of T). So we deduce that the tree T' coming from the adding of a son of label e to the root of T is frequent, with the same occurrences as T. As $T \sqsubset T'$, this contradicts the closure of $\mathcal{F}$. So $flat(T) \in \mathcal{F}_1$.

To prove $\subset$: Let $P \in \mathcal{F}_1$ be a frequent pattern of depth 1. Let's show that there exists a pattern $Q \in \mathcal{F}$ such that either $P = flat(Q)$, or there exists a subtree T of Q such that $P = flat(T)$. The first case is a special case of the second, so we will only demonstrate the second case. Let's suppose by negation that there is no pattern $Q \in \mathcal{F}$ such that P is a flattening of a subtree of Q. As P is frequent, its occurrences have to appear in the occurrences of the patterns of $\mathcal{F}$. As P isn't the flattening of a subtree of a pattern in $\mathcal{F}$, then it means that there exists a pattern $Q \in \mathcal{F}$ that has a subtree T' such that $Locc(T', D) = Locc(P, D)$, and T' has at least one more label than P (every subtree from which P could be the flattening is thus included in T'). Hence $P \sqsubset flat(T')$. This contradicts the closure of $\mathcal{F}_1$. So P is the flattening of a subtree of a pattern in $\mathcal{F}$. □

Definition 39 (Truncation of a pattern) *Let P be pattern of depth m, and let m' be a positive integer such that $m' < m$. The truncation at depth m' of P is the pattern $P_{|m'}$ whose nodes are the nodes of depth inferior or equal to m' in P and whose arcs are the arcs between those nodes in P.*

Definition 40 ($m-1$-generator pattern) *Let D be a datatree and $\mathcal{F} = Closed(\mathcal{P})$ be the closed set of frequent patterns of D. Let P be a frequent pattern of D of depth m. Let $\{Q_1, ..., Q_n\}$ be a set of patterns of $\mathcal{F}$.*

*We say that P is $m-1$-**generator** for $\{Q_1, ..., Q_n\}$ if $\forall i \in [1, n]$:*

- $P \sqsubseteq Q_i$

- *$Locc(Q_i, D) \subseteq Locc(P, D)$*
- *the truncations of P and Q_i at depth $m-1$ are isomorph by an isomorphism μ.*
- *any subtree of P whose root r is at depth $m-1$ is a flattening of the subtree of Q_i whose root is $\mu(r)$ (by definition $\mu(r)$ also is at depth $m-1$).*

Example: Suppose that all the patterns of figure III.23 have the same occurrences set. Then the pattern P is 1-generator for the patterns Q_1 and Q_2.

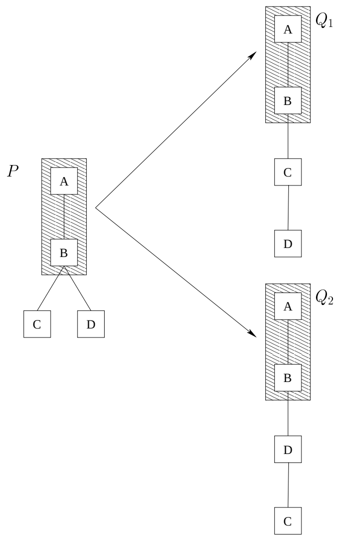

Figure III.23: Example of 1-generator pattern

Lemma 3 (Characterization of Dryade intermediate results) *Suppose that the set of frequent patterns of $\mathcal{F}$ satisfies the frequent occurrences independancy property. A each iteration i, the* Dryade *algorithm produces a closed set $\mathcal{F}_D^i$ of frequent patterns.*

This set can be decomposed in 3 separate subsets: $\mathcal{F}_D^i = F_{\mathcal{F}}^i \cup F_R^i \cup F_1^i$, where:

- *$F_{\mathcal{F}}^i$ is the set of patterns that doesn't take part in any hooking at iteration i.*
- *F_R^i is the set of root patterns of $\mathcal{F}_D^i$ that do not belong to $F_{\mathcal{F}}^i$.*
- *F_1^i is the set of patterns of depth 1 that hook among themselves or on patterns of F_R^i.*

These sets satisfy the following properties:

i. *For all pattern $Q \in \mathcal{F}$, there exists a unique pattern $P \in \mathcal{F}_D^i$ of depth m that is $m-1$-generator for Q, such that there exists no $m'-1$-generator patterns for Q in $\mathcal{F}_D$, with $m' > m$. We say that P is the biggest generator for Q, noted $P = gen_i(Q)$. If $P \in F_{\mathcal{F}}^i$, then $P = Q$ (the patterns of $F_{\mathcal{F}}^i$ are the patterns of $\mathcal{F}$ already discovered by* Dryade *at iteration i).*

ii. *For all root pattern $P \in F_R^i$ there exists at least one pattern $Q \in \mathcal{F}$ such that $P = gen(Q)$, and $Locc(Q, D) \subseteq RO(P)$.*

iii. *For all pattern $Q \in \mathcal{F}$, let m be the depth of its biggest generator. For each subtree T of Q whose root has a depth greater or equal to m, $flat(T) \in F_1^i \cup F_R^i$.*

iv. *For all pattern $P_1 \in F_1^i$ there exist a pattern $Q \in \mathcal{F}$ whose biggest generator is of depth m and which has a subtree T whose root is at a depth greater or equal to m such that $P_1 = flat(T)$.*

Proof:

Let i be an iteration of Dryade. **Let's show that $\mathcal{F}^i_D$ can be split up in $F^i_{\mathcal{F}}$, F^i_R and F^i_1:**

Let's consider $\mathcal{GQ}_{\mathcal{F}^i_D}$ the quotient graph of hookings of $\mathcal{F}^i_D$.

- $F^i_{\mathcal{F}}$ is the set of the patterns of $\mathcal{F}^i_D$ that are not connected to any other pattern by hooking.
- F^i_R is the set of patterns of whose cycle equivalence classes are roots of $\mathcal{GQ}_{\mathcal{F}^i_D}$.
- F^i_F is the set of pattern of $\mathcal{F}^i_D$ that are neither in $F^i_{\mathcal{F}}$ nor in F^i_R.

These sets are clearly separated, and by construction $\mathcal{F}^i_D = F^i_{\mathcal{F}} \cup F^i_R \cup F^i_1$.

Let's show now that the properties i, ii, iii and iv are satisfied for any iteration i of Dryade

We will show by induction on the iterations of Dryade that for each iteration i of Dryade, the properties **i**, **ii**, **iii** and **iv** are satisfied.

Case $i = 1$

We suppose that the first iteration of Dryade is the construction of the set of frequent patterns of depth 1.

From property 7, we have $\mathcal{F}^1_D = \mathcal{F}_1 = Closed(\mathcal{P}_1)$.

Let's show that property i is satisfied:

Let $Q \in \mathcal{F}$. From the property 5, there exists $P \in \mathcal{F}_1$, such that $P = flat(Q)$.

Furthermore, P and Q have the same roots, so P is 0-generator for Q.

From the property 5, P is unique. As all the patterns of $\mathcal{F}^1_D$ are of depth 1, there cannot be any m-generator patterns for Q, with $m > 0$.

Suppose that P is a pattern of $F^1_{\mathcal{F}}$, i.e. that P doesn't hook on any pattern of $\mathcal{F}^1_D$, and no pattern of $\mathcal{F}^1_D$ hooks on P. So Q doesn't have any subtree. Else from the lemma 2 there would exist in $\mathcal{F}^1_D$ patterns being the flattenings of these subtrees, and these patterns would hook on P. And Q is a depth 1 pattern. By construction of P, we deduce that $P = Q$.

So the property **i** is satisfied.

Let's show that property ii is satisfied:

Let $P \in F^1_R$. We must show that P is 0-generator for at least one pattern $Q \in \mathcal{F}$. For this, according to lemma 2, we must show that P isn't the flattening of a subtree of a pattern of $\mathcal{F}$ (it is so specifically the flattening of a pattern $Q \in \mathcal{F}$).

For the set $RO(P)$ of its root occurrences, P does not hook on any pattern. So for these occurrences, P cannot be the flattening of a subtree of a pattern of $\mathcal{F}$, otherwise there would be hookings. Hence for its root occurrences, P is the flattening of a pattern $Q \in \mathcal{F}$. Let's show that $Locc(Q, D) \subseteq RO(P)$. Suppose by negation that this isn't true: then for some of its occurrences, Q is a subtree of a pattern Q'. In this case, there is an arc in the hooking graph between P and $P' = flat(Q')$. In order for P to be anyway a root, P' should be in the same cycle equivalence class as P. The occurrences of P' on which P hooks cannot be extended by hooking to make the cycle and come back to P, otherwise we would have $Q \sqsubset Q' \sqsubset ... \sqsubset Q$, which is impossible. So there exists a pattern Q'' such that P' is generator for Q'', the occurrences of Q'' are different from those of Q', and they hook (passing through the other patterns of the cycle) on the occurrences of P. We deduce that $Q'' \sqsubset Q \sqsubset Q'$. So the occurrences of Q' are also occurrences of Q'', hence a new contradiction. So P isn't root in this case.

Hence the property **ii** is satisfied.

Let's show that properties iii and iv are satisfied

From **i**, for all pattern $Q \in \mathcal{F}$ its biggest generator exists and is 0-generator. The subtrees of Q we are interested in here are all the subtrees of Q except Q itself.

Let's show first that for all pattern $Q \in \mathcal{F}$, the patterns corresponding to the flattening of its subtrees are in $F_1^1 \cup F_R^1$.

- Let $Q \in \mathcal{F}$. From the lemma 2, for all subtree T of Q there exists a pattern $P \in \mathcal{F}_D^1$ which is a flattening of T. As T is a subtree of Q, for its occurrences corresponding to the occurrences of T, P hooks on $flat(Q) \in \mathcal{F}_D^1$. Because P has hookings, then P isn't in $F_{\mathcal{F}}^1$. So $P \in F_R^1 \cup F_1^1$.

Hence the property **iii** is satisfied.

Let's show then that any pattern $P \in F_1^1$ is the flattening of a subtree of a pattern of $\mathcal{F}$.

- Let $P \in F_1^1$. From the lemma 2, P is the flattening of patterns of $\mathcal{F}$ and/or of subtrees of patterns of $\mathcal{F}$. Suppose by negation that there is no T such that $P = flat(T)$ and T is a subtree of a pattern $Q \in \mathcal{F}$. Then P is only the flattening of patterns in $\mathcal{F}$ that never are subtrees of other patterns of $\mathcal{F}$. We deduce that P has no hookings with other patterns of $\mathcal{F}_D^1$. This contradicts $P \in F_1^1$.

Hence the property **iv** is satisfied.

Thus, the case $i = 1$ is satisfied.

General case:

Let's suppose that the property is true at iteration i, i.e. that we have $\mathcal{F}_D^i = F_{\mathcal{F}}^i \cup F_R^i \cup F_1^i$ satisfying **i**, **ii**, **iii** and **iv**. Let's show that the property is still true at iteration $i+1$.

Let's show that property i is satisfied

Let $Q \in \mathcal{F}$. By induction hypothesis there exists $P \in \mathcal{F}_D^i$ such that $P = gen_i(Q)$. Let $m = depth(P)$. Three cases are possible:

- $P \in F_{\mathcal{F}}^i$. In this case, still by induction hypothesis, $P = Q$. So we cannot find a bigger generator for Q than P, and as P doesn't take part in any hooking it stays unchanged at iteration $i+1$. Then $P = gen_{i+1}(Q)$.

- $P \in F_R^i$. Let $P_1, ..., P_n$ be the patterns obtained by hooking and redundant leaves suppression from P. Let's show that there exists $j \in [1, n]$ such that $P_j = gen_{i+1}(Q)$.

  By definition, P is $m-1$-generator for Q. Let $T_1, ..., T_k$ be the subtrees of Q whose root is at a depth greater than $m-1$. By induction hypothesis, property **iii**, the flattenings of all these subtrees are in $F_R^i \cup F_1^i$. Q is frequent, so the multiple hooking of the flattenings of $T_1, ..., T_k$ is also frequent. Hence in $MHS(P)$, there is $PS_1 = \{(hn_1, flat(T_1)), ..., (hn_k, flat(T_k))\}$, and the set of multiple hooking occurrences of PS_1 contains at least all the occurrences of Q. PS_1 is closed because in $MHS(P)$, there is no set $PS_1' = \{(hn_1, flat(T_1)), ..., (hn_k, flat(T_k)), (hn', P')\}$ for the occurrences of Q. Actually in this case because P' cannot be the flattening of a subtree of Q (they already all are represented), it must be included in the flattening of one of these subtrees. This is not possible by construction of $MHS(P)$.

  Let's consider now the set $\{T_1', ..., T_l'\} \subseteq \{T_1, ..., T_k\}$ of the subtrees of Q whose root is exactly at depth m. All the roots of the other subtrees of lower depth are descendants of the roots of $\{T_1', ..., T_l'\}$. We deduce that $flat(T_1'), ..., flat(T_l')$ are the only patterns that does not hook on any pattern of $flat(T_1), ..., flat(T_k)$. Hence they satisfy the definition of immediate hooking.

So there exists in $IHCS(RP)$ an immediate hooking set $\{(hn'_1, flat(T'_1)), ..., (hn'_l, flat(T'_l))\}$ whose occurrences contain the occurrences of Q. The roots of $T'_1, ..., T'_l$ are the internal nodes of depth m of Q.

Let $f_1, ..., f_p$ be the leaves of depth m of Q. By construction, these leaves are also leaves of P. To these leaves correspond nodes in the data, that of course cannot be found in any of the subtrees $T'_1, ..., T'_l$. So these nodes aren't in the nodes corresponding to the leaves of $flat(T'_1), ..., flat(T'_l)$. Hence they satisfy the definition of non-redundant leaf. We deduce that there is in $PCCS(RP)$ a pattern creation context
$PCC = (\{(hn'_1, flat(T'_1)), ..., (hn'_l, flat(T'_l))\}, \{f_1, ..., f_p\}, PCAS, PCOS)$, whose occurrences contain the occurrences of Q.

The pattern P_j built from PCC will then be a m-generator pattern for Q.

To have $P_j = gen_{i+1}(Q)$, we must show that:

- P_j isn't eliminated during update.

  * Suppose by negation that P_j is eliminated during update. It means that P isn't closed, so there exists $P' \in \mathcal{F}_D$ such that $P_j \sqsubset P'$ and $Locc(P_j, D) = Locc(P_j, projected(P, D))$. And we know that the occurrences of Q are included in those of P_j, as P_j is m-generator for Q.

    Suppose first that $root(P_j) \neq root(P')$. Then for all the occurrences of Q, the root of Q has as ancestor the root of P'. This contradicts the closure of $\mathcal{F}$.

    Suppose now that $root(P_j) = root(P')$. P' cannot come from the pattern P which has generated P_j, because for P we have build $SPatt(P)$, which is closed. So either P' has nodes of depth lower than m additional to the nodes of P, or P' has nodes of depth greater or equal to m additional to the nodes of P, and which have labels that are not in the labels of P. In both cases, these additional nodes exist for all the occurrences of Q but cannot be in Q, which contradicts the closure of $\mathcal{F}$.

    Hence P_j isn't eliminated by closure.

- P_j is the unique biggest m-generator for Q.

  * We know by induction hypothesis that P is the unique biggest $m-1$-generator of Q. There exists by construction a unique multiple hooking set of MHS that contains exactly the flattenings of the subtrees of Q. As well, by construction there exists a unique immediate multiple hooking set in $IHCS(P)$ which contains the flattenings of the subtrees of depth m of Q. And by construction, there exists a unique pattern creation context of $PCCS(P)$ that contains the flattenings of the subtrees of depth m of Q and the leaves of depth m of Q. Hence the unicity of P_j.

    P_j is maximal: otherwise, it would mean that there exists another root pattern P' that has produced a pattern at least $m+1$-generator for Q. And as by hooking and suppression of redundant leaves from a k-generator pattern, we can at most get a $k+1$-generator pattern, then P' itself should be m-generator for Q, contradicting $P = gen_i(Q)$.

- $P \in F_1^i$. We must show that P isn't eliminated by *Update*, and that it stays the biggest generator of Q at iteration $i+1$. If P isn't in immediate hooking with a root pattern $RP \in F_R^i$, then it isn't analysed by *Update*, and so stays unchanged. Else, P is suppressed by *Update* if there exists a pattern P' produced from RP and P, that is not eliminated, such as all the occurrences of P can be deduced from the occurrences of P'. As P' isn't eliminated, from previous demonstration there exists $Q' \in \mathcal{F}$ such that $P' = gen_{i+1}(Q')$. We have necessarily $Q \subset Q'$, so by closure of $\mathcal{F}$ we have: $Locc(Q, D) \supseteq Locc(Q, projected(Q', D))$, i.e. there exists occurrecnse of Q that do not come from the inclusion of Q in Q'. So the corresponding occurrences of P aren't included in those of P', so P isn't eliminated

We deduce that P stays the biggest generator for Q: $P = gen_{i+1}(Q)$.

Hence the property **i** is satisfied in the general case.

Let's show that property ii is satisfied

We must demonstrate that for all root pattern $P \in F_R^{i+1}$, there exists a pattern $Q \in \mathcal{F}$ such that $P = gen_{i+1}(Q)$. The patterns come in F_R^{i+1} from two different ways:

1. Either P has been built by hooking and suppression of the redundant leaves from a pattern $RP \in F_R^i$.

2. Or P is a pattern of F_1^i that became root after suppression of some of its hookings by *Update*.

We will show that in these two cases there exists $Q \in \mathcal{F}$ such as $P = gen_{i+1}(Q)$.

Let $P \in F_R^{i+1}$.

1. P has been built by hooking and suppression of the redundant leaves from $RP \in F_R^i$. So we can associate to P a pattern creation context $PCC \in PCCS(RP)$, to which we can associate an immediate multiple hooking set $PS_1 \in IHCS(RP)$, to which we can associate a multiple hooking set $MH \in MHS(RP)$. Let $P_1, ..., P_n$ be the patterns of MH, let's show that there exists $Q \in \mathcal{F}$ containing RP such that $P_1, ..., P_n$ all are flattenings of subtrees of Q at the level of the leaves of RP.

Suppose by negation that there exist no Q containing RP whose flattening of the subtrees at the level of the leaves of RP are $P_1, ..., P_n$.

However, the hooking of the patterns $P_1, ..., P_n$ on RP is frequent and closed, so the corresponding ancestor relations must appear in at least one pattern $Q \in \mathcal{F}$.

- Let $Q \in \mathcal{F}$ whose flattenings of the subtrees hooking on the leaves of RP are $P_1, ., P_i', .., P_n$, with $P_i \sqsubset P_i'$ and P_i, P_i' hook on the same leaf of RP. This case cannot happen, because by construction the patterns $P_1, ..., P_n$ are maximal for tree inclusion for the hooking on a given leaf of RP.

- Let $Q \in \mathcal{F}$ whose flattenings of the subtrees hooking on the leaves of RP are $\{P_1', ..., P_{n'}'\} \supset \{P_1, ..., P_n\}$. As previously, this case cannot happen: $P_1', ..., P_{n'}'$ hook on the hook nodes of RP, this would imply that the computation of the closed set $MHS(RP)$ is uncorrect, whereas we have proved before it's correct.

- Let $Q \in \mathcal{F}$ whose flattenings of the subtrees hooking on the leaves of RP are $P_1, ., P_i', .., P_n$, with $P_i' \sqsubset P_i$.

  This case cannot happen, because here P_i has at least one more node than P_i', and this node is frequent. As Q does not have this node, it contradicts the closure of $\mathcal{F}$.

- Let $Q \in \mathcal{F}$ whose flattenings of the subtrees hooking on the leaves of RP are $\{P'_1, ..., P'_{n'}\} \subset \{P_1, ..., P_n\}$. This case also cannot happen, for the same reason as before: the hooking of any $P_i \in \{P_1, ..., P_n\} \backslash \{P'_1, ..., P'_{n'}\}$ is frequent, but not in Q, so there is a contradiction with the closure of $\mathcal{F}$.

So there exists at least one pattern $Q \in \mathcal{F}$ containing RP whose flattenings of the subtrees at the level of the leaves of RP are the patterns $P_1, ..., P_n$ of the set MH. Two cases are possible: either $RP = gen_i(Q)$, or there exists $RP' \in F^i_R \cup F^i_{\mathcal{F}}$ such that $RP \sqsubset RP' = gen_i(Q)$.

We now have to demonstrate that there exists at least one pattern $Q \in \mathcal{F}$ whose flattenings of the subtrees hooking immediately under the leaves of RP, and which are not included in any other subtree of $P_1, ..., P_n$, are the patterns $P'_1, ..., P'_{n'}$ of $PS_1 \in IHCS(RP)$. Then we have to demonstrate that there exists a pattern $Q \in \mathcal{F}$ for which the leaves of RP that remain leaves in Q are the non-redundant leaves found in the pattern creation context $PCC \in PCCS(RP)$.

We do not detail the demonstrations of these two properties because they follow exactly the same reasoning as the previous one, focusing no longer on $P_1, ..., P_n$ but on $P'_1, ..., P'_{n'}$ and then on the non-redundant leaves.

We deduce that there exists at least one pattern $Q \in \mathcal{F}$ containing RP whose flattenings of the subtrees hooking immediately on the leaves of RP are the patterns of PS_1, and whose leafs common to the leaves of RP are the non-redundant leaves found in the pattern creation contexts PCC.

If $RP = gen_i(Q)$, then P is m-generator for Q by construction. Using the proof of property **i**, we get $P = gen_{i+1}(Q)$.

Else, there exists $RP' \in F^i_R \cup F^i_{\mathcal{F}}$ such that $RP \sqsubset RP' = gen_i(Q)$, and the occurrences of P are induced by the occurrences of RP'. In this case P must be eliminated by the update step, but as P hasn't been eliminated we are in the previous situation: $RP = gen_i(Q)$.

2. P is a pattern of F^i_1 which became root after suppress of some of its hookings by *Update*. We have seen in the demonstration of property **i** that if there exists $Q \in \mathcal{F}$ such that $P = gen_i(Q)$, then P isn't eliminated. Let's show that in the opposite case, P is eliminated by *Update*. As the hookings of P have been suppressed by *Update*, then P was in immediate hooking with a root pattern $RP \in F^i_R$. Suppose by negation that P isn't eliminated by *Update*. It means that for any pattern $P' \in \mathcal{F}^{i+1}_D$ such that $P \sqsubset P'$, some occurrences of P are not induced by its inclusion in P'. As the patterns of $\mathcal{F}$ are included in the patterns of $\mathcal{F}^{i+1}_D$, then as well for all pattern $Q \in \mathcal{F}$ such that $P \sqsubset Q$, some occurrences of P cannot be deduced of its inclusion in Q. This means that P itself is a pattern of $\mathcal{F}$. This isn't possible because we have supposed here that P wasn't generator of any pattern of $\mathcal{P}$. So if P isn't the biggest generator of any pattern of $\mathcal{F}$, then P is eliminated.

As P isn't eliminated, then there exists $Q \in \mathcal{F}$ such that $P = gen_i(Q)$. This doesn't change during the DRYADE iteration so $P = gen_{i+1}(Q)$.

Hence the property **ii** is satisfied.

Let's show that the properties iii and iv are satisfied

The properties **iii** and **iv** are true at iteration i by induction hypothesis. So for all $Q \in calF$, with $P = gen_i(Q)$ of depth m, the flattenings of the subtrees of Q whose root is at a depth greater or equal to m are in $F^i_R \cup F^i_1$. And for all $P \in F^i_1$, we can show that it is the flattening of a subtree whose root is at a depth greater or equal to m of a pattern $Q \in \mathcal{F}$ whose biggest generator is of depth m.

Let's show first that for all pattern $Q \in \mathcal{F}$ whose biggest generator at iteration $i+1$ is of depth $m+1$, the flattenings of its subtrees whose root has a depth superior or equal to $m+1$ are in $F_1^{i+1} \cup F_R^{i+1}$.

For the patterns $Q \in \mathcal{F}$ whose biggest generator doesn't change at iteration $i+1$, we must show that no flattening of the subtrees of Q is eliminated. Suppose by negation that this is wrong: let P be the flattening of a subtree of Q whose root in Q is at a depth greater or equal to m, suppose that P is eliminated. It means that P was in immediate hooking with a pattern RP, generator for the patterns of $\mathcal{F}$ other than Q, and that all the occurrences of P can be deduced from the occurrences of a pattern P' resulting of the hooking and of the suppression of the redundant leaves of P on RP. It would mean that all the occurrences of Q (included in those of P) can be deduced from the occurrences of P', hence a contradiction with the closure of $\mathcal{F}$.

For the patterns $Q \in \mathcal{F}$ whose biggest generator changes at iteration $i+1$, it means that the new generator P has been built by hooking and suppression of the redundant leaves from the old generator RP, as seen in the previous demonstrations. In this case, the only flattenings of subtrees of Q that can be suppressed by *Update* are the patterns in immediate hooking on RP, corresponding to subtrees whose root is at depth m in Q. The flattenings of these subtrees whose root is at depth superior or equal to $m+1$ are not eliminated.

Hence the property **iii** is satisfied.

Let's show now that for all pattern $P \in F_1^{i+1}$, we can find a pattern $Q \in \mathcal{F}$ whose biggest generator is of depth $m+1$, and for which P is the flattening of a subtree whose root is at a depth superior or equal to $m+1$ in Q.

By construction, if $P \in F_1^{i+1}$ then $P \in F_1^i$. So by induction hypothesis, there exists $Q \in \mathcal{F}$ whose biggest generator RP is of depth m, and for which P is the flattening of a subtree whose root has a depth superior or equal to m in Q.

If RP isn't root, the biggest generator of Q doesn't change from one iteration to another. Actually, we cannot have $RP \in F_{\mathcal{F}}^i$, because in this case we would have $RP = Q$, and P would be reduced to a single node, which isn't possible in our context. So $RP \in F_1^i$. As it isn't root, there are no hookings made, so Q doesn't change of biggest generator. So P is still the flattening of a subtree of Q whose root is at a depth superior or equal to m, the depth of the biggest generator of Q.

If RP is root, then we have seen in the proofs of the properties **i** and **ii** that at iteration $i+1$, we build from RP a pattern $P' = gen_{i+1}(Q)$, P' of depth $m+1$. If P was only the flattening of a subtree whose root is of depth m of Q, then P would have been used to build P', and it would have been eliminated because its occurrences could have been deduced from those of P'. Otherwise, it would have became root (see proof of lemma 1). Because P hasn't been eliminated and isn't root, P is the flattening of a subtree of depth greater of equal to $m+1$ of Q.

Hence the property **iv** is satisfied.

So the general property is true by induction. $\square$

Lemma 4 *Let $Q \in \mathcal{F}$. If there exists an iteration i of* Dryade *for which $gen_i(Q)$ is a root pattern, then there exists an iteration of* Dryade *$i' > i$ where Q is discovered by the algorithm, i.e. $Q \in F_{\mathcal{F}}^{i'}$.*

Proof: Let $P = gen_i(Q)$, of depth m, P is then $m-1$-generator for Q. Let's show that $i' = i + depth(Q) - m$.

We know that P is such that Q and P have the same truncation at depth m. We have seen in the proof of the previous lemma that at iteration $i+1$, $SPatt(P)$ will contain a pattern P' which will be m-generator for Q. By applying recursively this method, from iteration i, $depth(Q) - m$ iterations are necessary to discover the pattern Q. $\square$

Lemma 5 *Let $P \in \mathcal{F}_1 = \mathcal{F}_D^1$. If there exists $Q \in \mathcal{F}$ such that $P = gen_1(Q)$, then there exists an iteration i of* Dryade *where the patterns P becomes root, i.e. $P \in F_R^i$. Otherwise, there exists an iteration i' where P will be eliminated from $\mathcal{F}_D^{i'}$.*

Proof: We have seen in the proof of lemma 1 that for any pattern of $\mathcal{F}_D^1$, after some iterations of Dryade, either it becomes a root pattern, or it is eliminated.

Furthermore, we have seen in the proof of the lemma 3 (general case, properties **i** and **ii**), that any eliminated pattern of F_1^1 wasn't a biggest generator for a pattern of $\mathcal{F}$.

Hence the patterns which become root are those which are the biggest generators of a pattern in $\mathcal{F}$. □

THEOREM 2 *If all the frequent patterns of $\mathcal{F}$ satisfy the frequent occurrences independancy property, then the* Dryade *algorithm is sound and complete.*

Proof:

Suppose that all the frequent patterns of $\mathcal{F}$ satisfy the frequent occurrences independancy property. Showing that Dryade is sound and complete comes to show that there exists an iteration k for which $F_D^k = \mathcal{F}$.

Completeness

Let's show that there exists an iterAation k for which $F_{\mathcal{F}}^k = \mathcal{F}$.

Let $Q \in \mathcal{F}$, consider $P \in \mathcal{F}_D^1$ it's biggest generator.

- if $P \in F_{\mathcal{F}}^1$, then Q has already been discovered by Dryade.
- if $P \in F_R^1$, then from the lemma 4, there exists an iteration at which Q will be discovered by Dryade.
- if $P \in F_1^1$, then from the lemma 5, there exists an iteration at which P will become root, so from the previous case Q will be discovered by Dryade at a later iteration.

So for all $Q \in \mathcal{F}$ there exists an iteration k_Q where Q is discovered by Dryade, hence the completeness.

Soundness

Let's consider the iteration k where Dryade terminates, we have $F_{\mathcal{F}}^k = \mathcal{F}$. Let's show that $F_R^k = F_1^k = \emptyset$.

As Dryade has terminated there are no hookings left in the graph, from lemma 1. So if there are any patterns left, these are either patterns of $F_{\mathcal{F}}^k$, or patterns of F_R^k: $F_1^k = \emptyset$.

Thanks to property **ii** of lemma 3, if there is a root patterns left, it is generator for a pattern in $\mathcal{F}$. At iteration k this isn't possible, so $F_R^k = \emptyset$. Hence the soundness.

□

Remark 5 *Another formulation is that* Dryade *discovers all the frequent patterns of $\mathcal{F}$ satisfying the frequent occurrences independancy property.*

The interested reader is refered to annex A for a solution making Dryade sound and complete in the general case.

Chapter IV

Experimental study

The goal of this chapter is to empirically study the performances of the Dryade algorithm. For this study, we have realised an implementation of Dryade. We will first describe the specifications of this implementation, then we will detail the experimental settings. After, we will give experimental results, and we will discuss them.

1 Implementation

From the Dryade algorithm shown in the second part of the chapter III, we realised a C++ implementation. This implementation is 15000 code lines long, and intensively uses the data structures of the *Standard Templates Library (STL)*.

During the realisation of our implementation, due to time constraint we have had to make choices that slightly differ from the algorithm presented in chapter III. We will explain these choices and the corresponding restrictions.

1.1 Eclat instead of Charm

We have seen in chapter III that for Dryade, we need a vertical algorithm of closed frequent itemset discovery, like Charm. M. Zaki kindly gave us his implementation of the Charm algorithm. For a maximal efficiency, the idea was to be able to integrate directly the code of M. Zaki in our program. However, the code of Charm is very complex and relies on several programs preparing the data. Due to lack of time, we couldn't analyse all these programs and regroup them in a C++ class easy to use in our implementation.

Hence, we have used as vertical algorithm of closed frequent itemset discovery the Eclat algorithm, whose C implementation has been given to us by C. Borgelt, whom we thank here. It was very easy to integrate the Eclat code in a C++ class compatible with our program. As Eclat computes **all** the frequent itemsets, we have added a postprocessing filter that returns only the closed frequent itemsets.

The only consequence of this choice is to increase the running time of our program. We do not measure the execution time of the filter, but using Eclat instead of Charm makes our running time loose an order of magnitude, from the studies of M. Zaki ([ZPOL97], [ZG03]). The times we will give for our algorithm will then separate the time relative to the computation of closed frequent itemsets, for which Eclat gives an upper bound, from the Dryade execution time.

1.2 No mapping storage

Our DRYADE implementation is based on an early implementation of DRYAL, which allowed us to test our method.

The data structures used in the DRYAL implementation do not keep, for a frequent pattern, all its mappings in the data. This is unecessary for DRYAL: to do the hookings, the only elements to keep are the elements of the mappings corresponding to the root and the leaves of the pattern.

To speed up the development of our DRYADE implementation, we have kept the same data structures as in the DRYAL implementation, i.e. the mapping of a pattern is represented only by the image of the root and the image of the leaves.

So our code have two limitations compared to the DRYADE algorithm described in the chapter III.

1.Closure test Because we don't know precisely the mappings, the test guaranteeing closure at the end of the *Update* procedure (algorithm 16) cannot be done. It means that some duplicate patterns can be generated. The influence of these duplications upon the running time of our algorithm can be discussed as follows. In one hand, the test guaranteeing closure at the end of procedure *Update* isn't performed, which allows to save some time. On the other hand, the duplicate patterns that can be generated will have to be processed in the next iterations. Clearly, the overcost of the duplications is higher than the cost of the closure test when the size and the number of solutions increases, this corresponding to an exponential increase of the number of duplicate patterns.

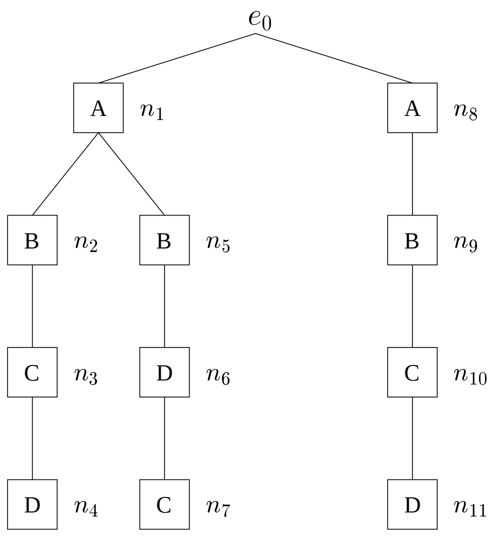

Figure IV.1: D_Z

Even if the closure test at the end of procedure *Update* isn't performed, most of the patterns found by DRYADE are closed patterns, because the results of the algorithm of frequent itemset discovery are closed. This is guaranteed by the postprocessing of the itemsets given by ECLAT.

2. Trees shape In some specific cases, having a data structure that doesn't represents the mappings in full prevents a correct processing of the hookings. These cases correspond to the trees that have at least two nodes u and v of same label, where one isn't descendant of the other, and such as there are no descendants of u and v sharing a common label.

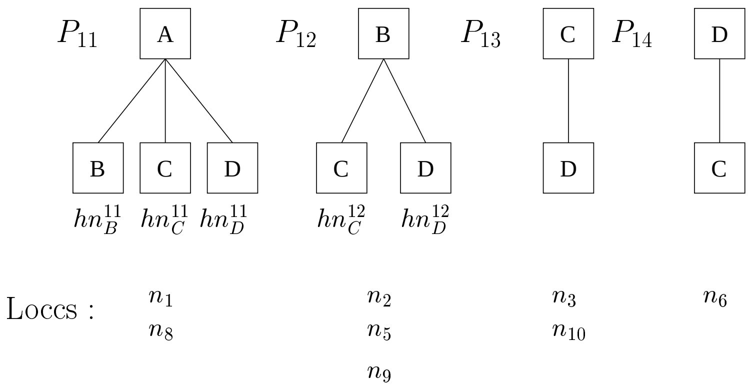

Figure IV.2: Frequent patterns of depth 1 of D_Z

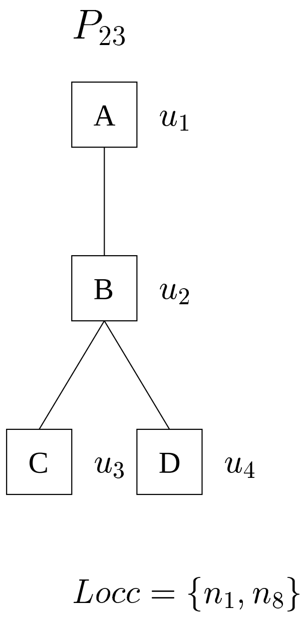

Figure IV.3: Pattern P_{23} obtained by hooking and suppression of redundant leaves

The figure IV.1 shows a datatree D_Z that contains such a tree, it's the tree whose root node is n_1. Actually, the nodes n_2 and n_5 of these tree are both labelled by B, and both have descendants labelled by C and D.

The problem comes when hooking and suppressing redundants leaves of P_{12} on P_{11} (figure IV.2), which gives the pattern P_{23} (figure IV.3). There are three mappings between the nodes of P_{23} and the nodes of D_Z, represented in the next table:

P_{23} node	Mapping 1	Mapping 2	Mapping 3
u_1	n_1	n_1	n_8
u_2	n_2	n_5	n_9
u_3	n_3	n_7	n_{10}
u_4	n_4	n_6	n_{11}

In the code, the mappings 1 and 2 cannot be discriminated: they correspond to a unique data structure that associates to u_1 the node n_1, to u_3 the nodes n_3 and n_7, and to u_4 the nodes n_4 and n_6. This representation implicitly supposes that in D_Z the nodes n_2 and n_5 are the same, which isn't true. This will lead to uncorrect hookings: for exemple, the simultaneous hooking of P_{13} and P_{14} on P_{23}.

Hence our experiments only consider cases where a correct processing of the hookings is guaranteed. This means that in our experiments, the input trees always satisfy the following constraint, that we call **CNTR**.

Constraint: A tree satifies constraint **CNTR** if for all couple of nodes u and v of same label, either u is ancestor of v or no descendant of u has the same label as a descendant of v.

So our implementation differs from the algorithms of chapter III in two ways: in one hand, the update test is incomplete, which penalizes the running time when the problem has big solutions. On the other hand, we suppose that the input trees satisfy the constraint **CNTR**.

2 Experimental settings

The goal of our experiments is to investigate the performances and the behavior of Dryade on tree data. We will also compare Dryade to the only algorithm that can address the same problem, namely Warmr. Of course this comparison isn't fair, because Warmr is an algorithm that can find solutions to problems far more general than those addressed by Dryade, and it doesn't use the closure property. However, it's the only algorithm with which we can compare, hence we will compare the performance increase that our specific approach can bring, compared to a general approach.

Today, to the best of our knowledge, there are very few freely available tree databases. And in XML documents, there are often synonymy and polysemy problems with the tags, problems that we only touched in the introduction, and that come before Dryade.

So the experimental validation will use, in one hand a set of randomized problems (sections 3.1 and 3.2), and in the other hand a real application (section 3.3).

The framework considered for the validation of our approach is first detailed.

The randomized data are generated with an uniform distribution, for which we have defined 4 order parameters:

- N : the number of trees in the corpus
- E : the set of labels (without loss of generality, the labels are integers from 0 to $|E| - 1$)
- B : the average branch factor for a tree of the corpus
- P : the average depth of leaves for a tree of the corpus

The generation of the corpus is done by the procedure *GenerateCorpus*, shown in algorithm 17.

Algorithm 17 *GenerateCorpus*

```
Require: $E$ the set of labels, $B$ the average branch factor, $P$ the average depth, $N$ number of trees
Ensure: A corpus of $N$ trees with labels in $E$ satisfying the constraint CNTR, such that each tree has an average branch factor $B$ and an average depth $P$.
 1: $Corpus \leftarrow \emptyset$
 2: for $i = 1$ to $N$ do
 3:   $nbTrials \leftarrow 0$
 4:   repeat
 5:     repeat
 6:       $tree \leftarrow GenerateTree(P, E, 2B, P)$ // see algorithm 18
 7:     until $averageDepth(tree) = P$ and $averageBranch(tree) = B$
 8:     $nbTrials \leftarrow nbTrials + 1$
 9:   until ($tree$ satisfies CNTR) or $nbTrials > MAX\_TRIALS$
10:   if $nbTrials > MAX\_TRIALS$ then
11:     Exit with an error: the generation of this corpus is overconstrained
12:   else
13:     $Corpus \leftarrow Corpus \cup \{tree\}$
14:   end if
15: end for
16: Return $Corpus$
```

The essence is to generate randomized trees with the procedure *GenerateTree* (algorithm 18), then to keep in the corpus only those that satisfy the constraints of average depth/branch factor, and that satisfy the **CNTR** constraint. This method is expensive in running time, but it avoids to introduce an important bias while generating a tree, as one can see when looking at procedure *GenerateTree*.

The only bias compared to an uniform generation (line 1 of algorithm 18) is motivated by our desire to get non-trivial solutions, i.e. solutions of depth strictly greater than 1, for small values of P. Actually, preliminary investigations showed that for small values of P (lower than 3 or 4), there are few frequent patterns of depth greater than 1 in the data. But increasing P is extremely costly, because the number of ancestor relations to consider increases exponentially. The bias introduced to improve this is to partition E in $P + 1$ separate subsets of same size $E[0], ..., E[P]$. If E cannot be divided by $P + 1$, all the subsets receive $round(|E|/(P + 1))$ labels, except $E[0]$ which receives the remaining labels.

For each depth level p the labels of the nodes of this level are drawn evenly in the subset $E[p]$.

This bias allows to find frequent patterns of depth greater or equal to 2 even if P value is 3 or 4.

Algorithm 18 *GenerateTree*

```
Require: $p$ the current depth, $E$ the set of labels, $bmax$ the maximal branch factor, $pmax$ the
    maximal depth
Ensure: A random tree with labels in $E$ of maximal depth $pmax$, with maximal branch factor
    $bmax$
 1: $e \leftarrow$ random drawing of a label in $E[p]$
 2: $n \leftarrow$ new node of label $e$
 3: if $p > 0$ then
 4:    $b \leftarrow$ random drawing of a branch factor between 0 and $bmax$
 5:    if $(b = 0)$ and $(p = pmax)$ then
 6:       $b \leftarrow 1$ // Avoids to have trees reduced to a single node
 7:    end if
 8:    for $i = 0$ to $b$ do
 9:       $n_i \leftarrow GenerateTree(p - 1, E, bmax, pmax)$
10:       Add $n_i$ as son of $n$
11:    end for
12: end if
13: Return $n$
```

Most of the experiments have been realised with Pentium IV 2.8 Ghz with 521 Mb RAM, running Linux. Some experiments have been realised with an AMD Athlon 1 Ghz with 1 Gb RAM (we will state when it's so).

3 Results

In the first section, we will compare Warmr and Dryade. Then we will study the effective complexity of Dryade in the second section, before finishing by a study with real data.

3.1 Comparison with Warmr

To compare Dryade with Warmr, we traduce each randomly generated tree corpus in a format that Warmr can use. For this, each tree is replaced with its relational encoding. We use the relational encoding Rel^+ described in the paper [TRS02]. Warmr needs to now the maximal number of atoms of the solutions that it is looking for: we have set this limit to the average number of father-son edges of a tree in the corpus.

We have produced three corpuses for the test:

Corpus 1	1000 trees, $P = 3$, $B = 1$, $\lvert E\rvert \in [4..56]$
Corpus 2	1000 trees, $P = 3$, $B = 2$, $\lvert E\rvert \in [16..56]$
Corpus 3	1000 trees, $P = 3$, $B = 3$, $\lvert E\rvert \in [24..56]$

Notice that the values of $|E|$ do not start at 1, because when there are too few labels it's impossible to generate trees satisfying the constraint **CNTR**.

The frequency threshold is 15. For each corpus characterised by P, B and a number of labels $|E|$, we make 8 different random runs. The running time of the programs Warmr and Dryade is limited to 1 hour by run.

On the corpuses 2 and 3, we cannot give any result for WARMR, because depending on the number of labels, either WARMR takes longer than 1 hour for its execution, or it consumes all the memory and is stopped by the OS. In the same conditions, DRYADE always finished its execution correctly.

The results for the corpus 1 are presented in figure IV.4.

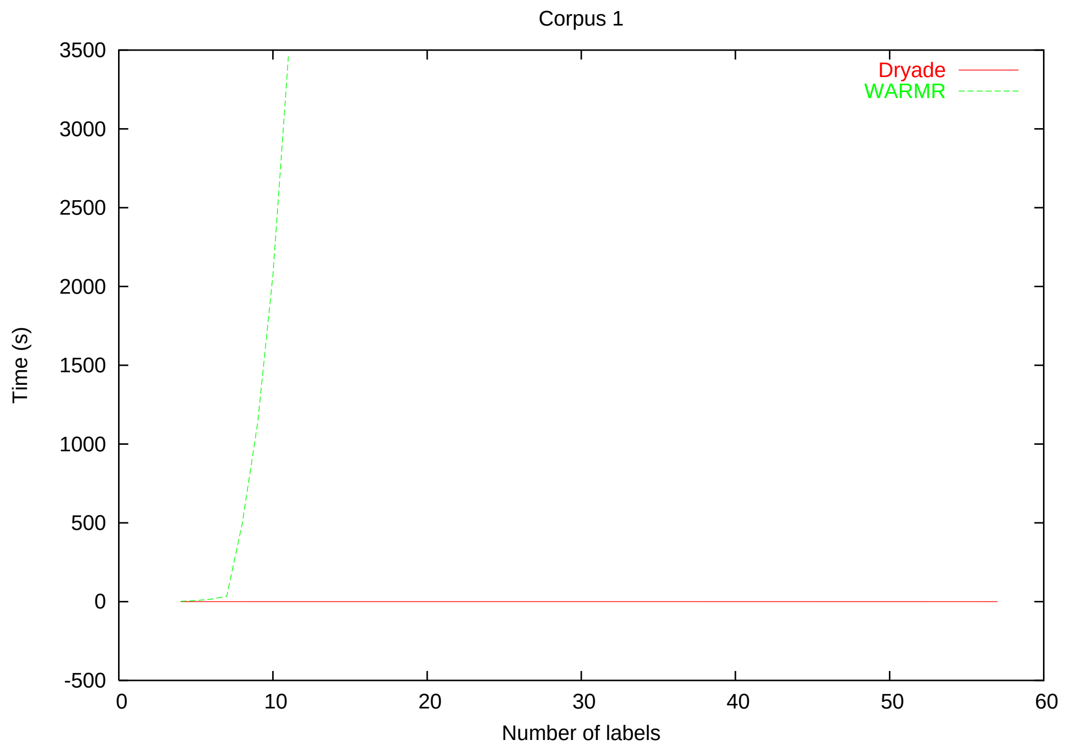

Figure IV.4: Difference of running time for WARMR and DRYADE

We can notice that the run time of DRYADE is lower to WARMR's one by several orders of magnitude. Actually, the maximal time used by DRYADE to find frequent patterns is 1.14s, when there are 8 labels in E. In the same conditions, WARMR needs 493.19s to find the frequent trees. As soon as the number of labels in E is higher than 12, WARMR cannot answer within the given time and memory limits.

As we have seen before, this comparison isn't fair. Whereas DRYADE is computing only the closed frequent patterns, WARMR is computing **all** the frequent patterns. This needs a lot more computation time, and once again we can see the efficiency of an approach using the closure property.

Furthermore, even simple frequent patterns can be represented with logical formulas that can be rather long. Let's consider the tree A of figure IV.5.

Let's consider now a very simple corpus with only this tree, and a threshold of 1. Hence the only tree to find is A itself. On the 1 Ghz Athlon, DRYADE finds A in less than 0.01 second. On the same machine, WARMR needs 1508.6 s to find A.

So WARMR is limited by the fact that it doesn't looks for closed frequent patterns, but also because the formula encoding the parent relation of A is 6 atoms long (number of arcs in A). Finding long frequent formulas with WARMR is difficult, as [IWM00] already showed. In the specific case of data structured as graphs (moleculas) of this paper, the graph-specific approach developped can

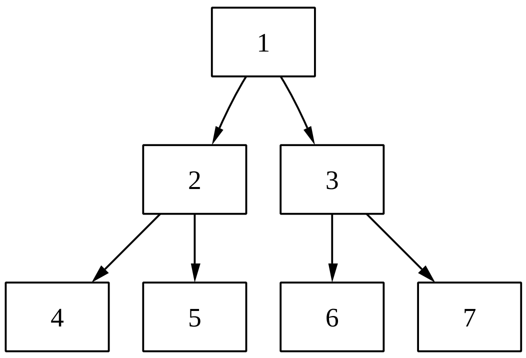

Figure IV.5: Perfect binary tree of depth 2

find frequent structures more complex than those found by Warmr.

In the same way, our approach specific to tree data is significatively faster than Warmr on this kind of data.

3.2 Performance study

3.2.1 Scaleup

We will first study the behavior of Dryade depending on the number of trees given as input. We consider an instance of corpus 2, i.e. trees of average depth 3 and average branch factor 2. The number of labels is set to 20, because we know it's a case where Dryade must find solutions with a depth greater than 1. The relative threshold is 0.015 (15/1000). The number of trees varies between 1000 and 10000. The figure IV.6 shows the execution time of each step of Dryade (logarithmic curves).

With more than 8000 trees, Dryade couldn't finish its execution due to memory saturation.

The lower curve corresponds to the time for discovering closed frequent patterns of depth 1 by Eclat. This curve with logarithmic scale is clearly sublinear, as expected. It is known that Eclat is linear on the number of transactions, i.e. the number of nodes in the data. And here only the number of trees change, there are trees having on average the same number of nodes, and the frequency threshold is relative. So Eclat is expected to be linear on the number of input trees.

The next curve corresponds to the time after the discovery of the hooking anchors between the frequent patterns of depth 1, its logarithmic aspect is also sublinear. Hence this step is polynomial on the number of input trees.

The last curve represents the total time of the algorithm, its sublinear logarithmic aspect allows us to state that Dryade is polynomial on the number of input trees.

This results allows us to hope that with a more efficient memory handling (condensed [JB02] or compressed representations for example), Dryade would be able to handle even more data.

3.2.2 Detailed analysis of the 3 steps of Dryade

The Dryade algorithm has 3 steps: first it computes the closed frequent patterns of depth 1 (with Eclat in our code), then it computes the hooking graph between these patterns, then it makes the hookings.

Computation of closed frequent patterns of depth 1:

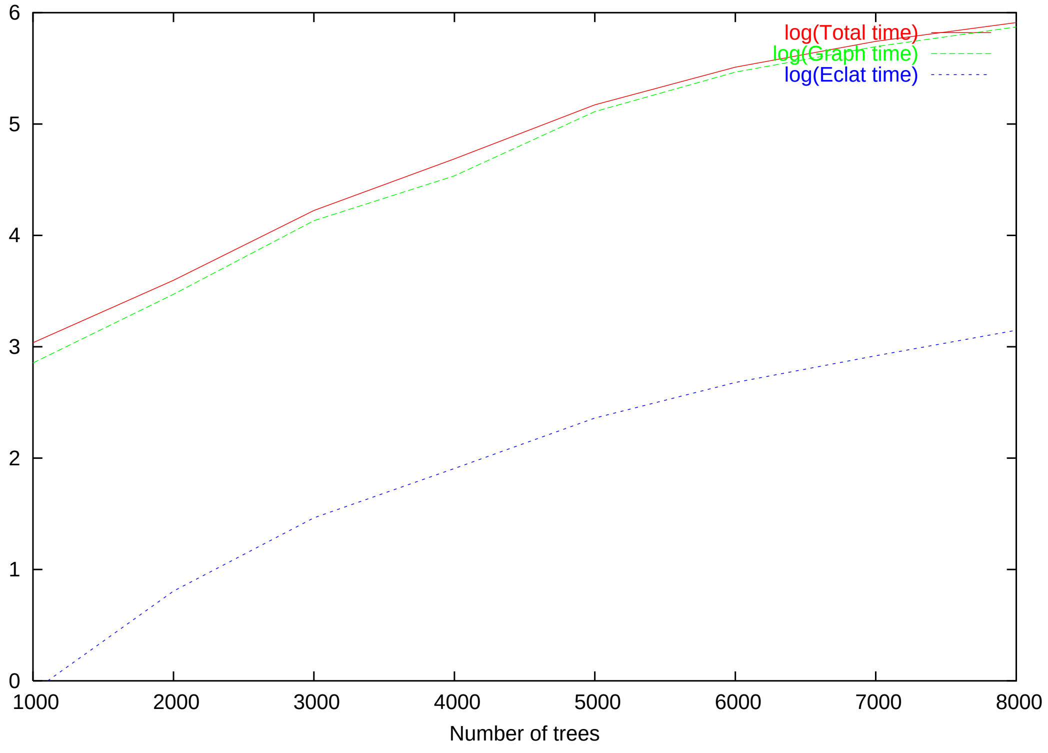

Figure IV.6: Scaleup experiment

This first step relies on well know algorithms, so we do not study it's complexity here.

Computation of the hooking graph:

The time for computing the hooking graph depends on the number of closed frequent patterns of depth 1 discovered at the first step. To show this relation, we give in tables IV.1, IV.2 and IV.3 the curves with on x-axis the number of frequent patterns of depth 1 discovered, and on y-axis the time for computing the hooking graph (left curve), and the logarithm of this time (right curve).

The logarithmic curves for corpus 2 and 3 (average branch factor of 2 and 3 respectively) are sublinear. So the time for computing the hooking graph is in these cases polynomial on the number of patterns of depth 1. By looking at procedure *ComputeGraph* (algorithm 7 page 49), this can be explained: the first test, quadratic on the number of frequent patterns of depth 1, tests the possibility of a hooking, and fails in most cases (hence no further computations are made).

However, the curve for corpus 1 (average branch factor of 1) exhibits a different behavior, that we will analyse.

The curve IV.7 shows the evolution of the number of frequent patterns of depth 1 depending on the number of labels. An analytic modelisation is given in annex at the end of this chapter, we will sum it up here. The key point is that the number of frequent patterns of depth 1 has, depending on the number of labels, two peaks. One corresponds to a majority of “long” patterns (more than 2 sons), the other to a majority of “short” patterns (only one son). To better understand the evolution on hooking graph computation time in this case, we propose another representation in the next paragraph.

We have the following hypothesis: for an average branch factor of 1, there are many frequent structures of depth greater than 1, all the more when there are few labels in the trees of the corpus. So, there are many anchor computations to do in *ComputeGraph*, unlike corpus 2 and 3. To

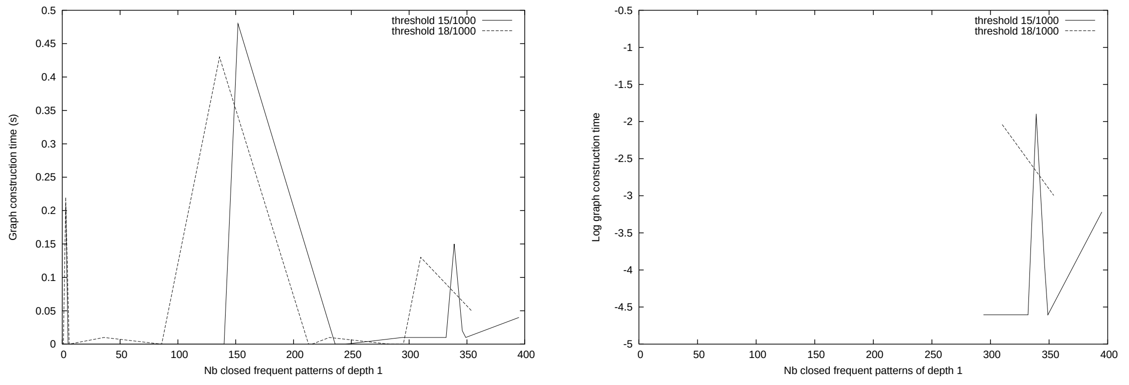

Table IV.1: Corpus 1

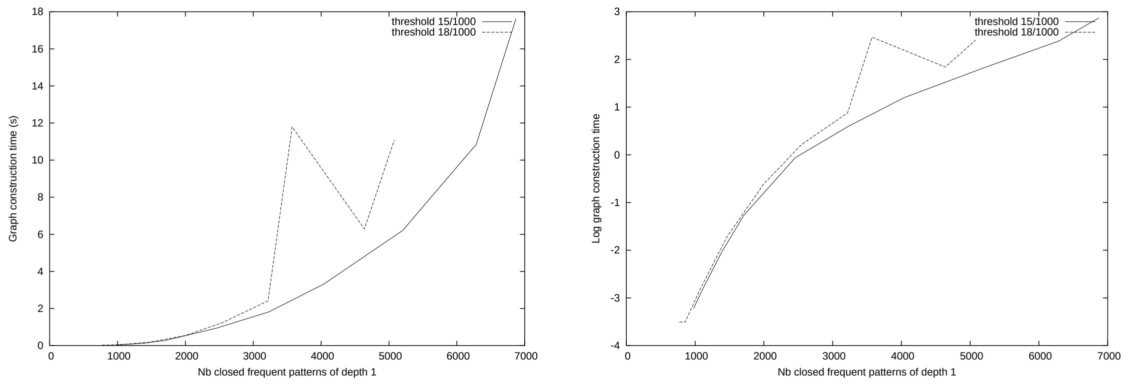

Table IV.2: Corpus 2

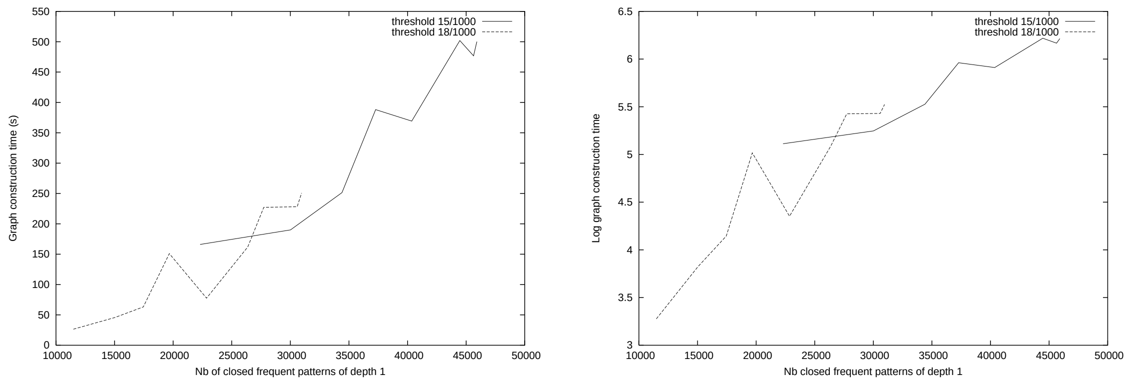

Table IV.3: Corpus 3

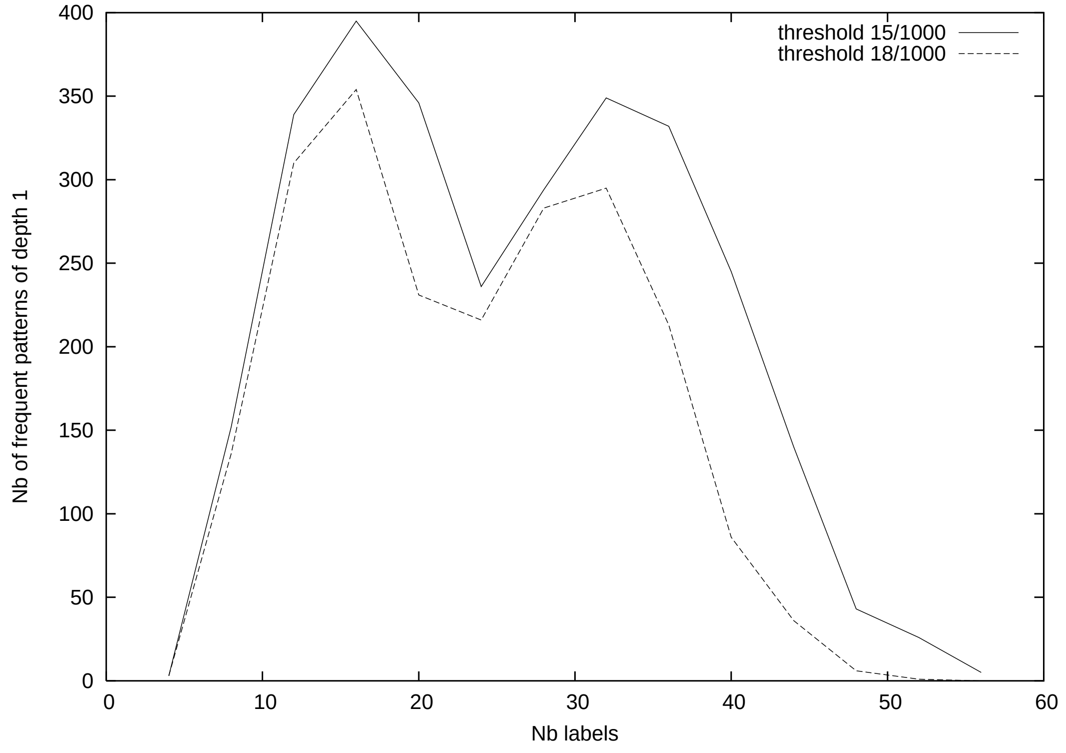

Figure IV.7: Corpus 1

investigate this hypothesis, we will provide a curve with on x-axis the ratio between the number of frequent structures discovered at the end of Dryade execution, and the number of frequent patterns of depth 1 found after first step. The more the mappings, the more the hookings, the higher this ratio. We expect the time used to compute the graph of hookings to increase with this ratio. This is confirmed by figure IV.8. As soon as we are out of the case were all final closed frequent patterns are depth 1 patterns, the hooking graph computation time grows linearly with the ratio of the number of frequent patterns found at the end of Dryade by the number of frequent patterns of depth 1 after first step.

Hookings computation:

As the number of hookings iterations depends on the depth of the closed frequent patterns to find, we have shown of the curves IV.9, IV.10 and IV.11 the time necessary for hookings as a function of the average depth of the frequent patterns in the corpus, for the thresholds 15 and 18. Notice that in most of the curves, the last points for threshold 15 couldn't be computed: as always, for a lower threshold the problem gets more difficult, and solving it can take more than the given time.

All these curves have a sublinear logarithmic aspect, so we can deduce that the complexity of the hooking step of Dryade is polynomial on the average depth of the frequent patterns to discover.

Notice that the lowering that appears for the maximal depth of the curve of figure IV.9 comes from the fact that deep solutions are obtained only with few labels. Hence, the rightmost point of figure IV.9 is computed for $|E| = 4$. With our generation bias, the 1000 trees are identical, so there is only one frequent pattern to discover, which makes the work easier for Dryade, hence lower computation time.

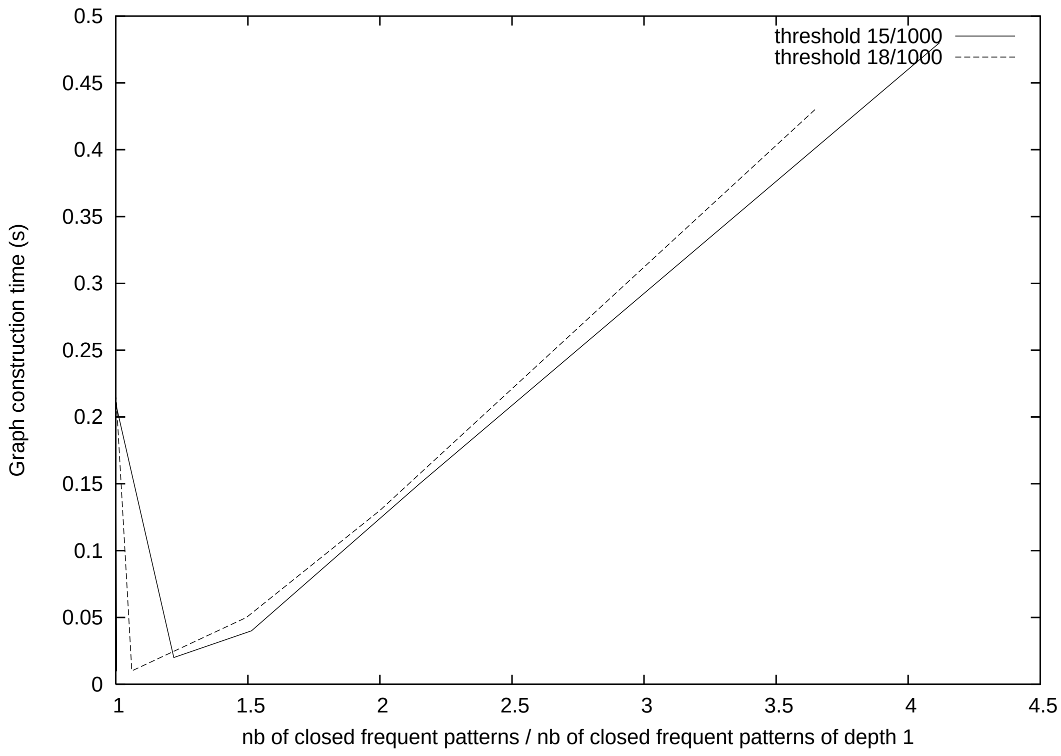

Figure IV.8: Corpus 1

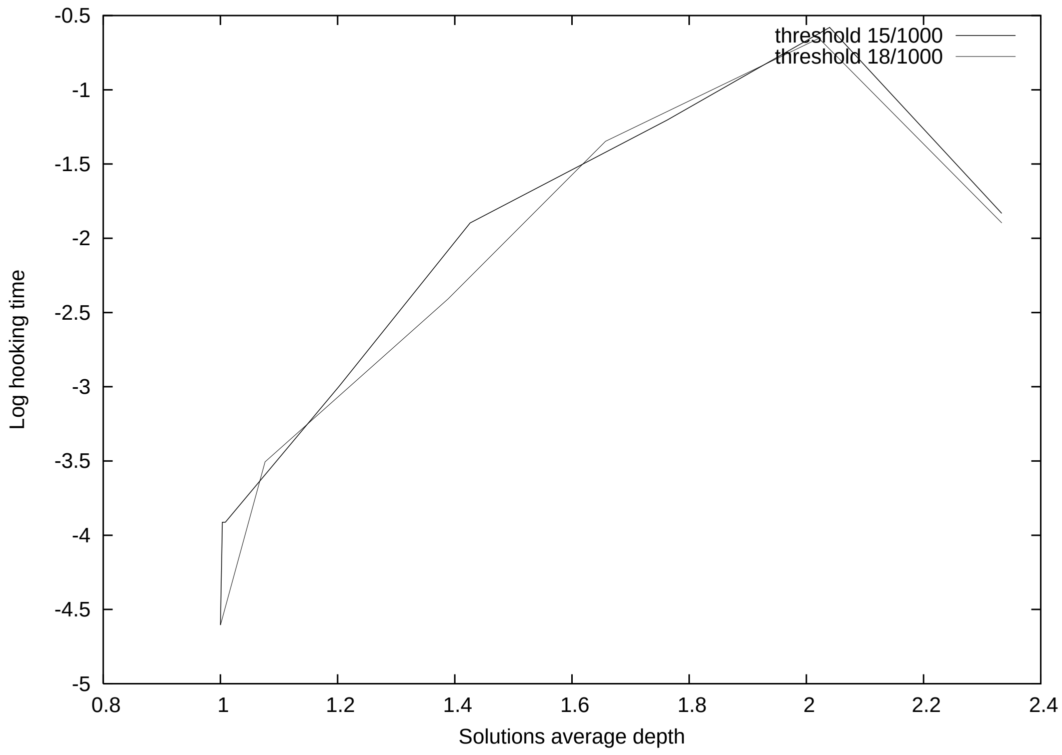

Figure IV.9: Corpus 1

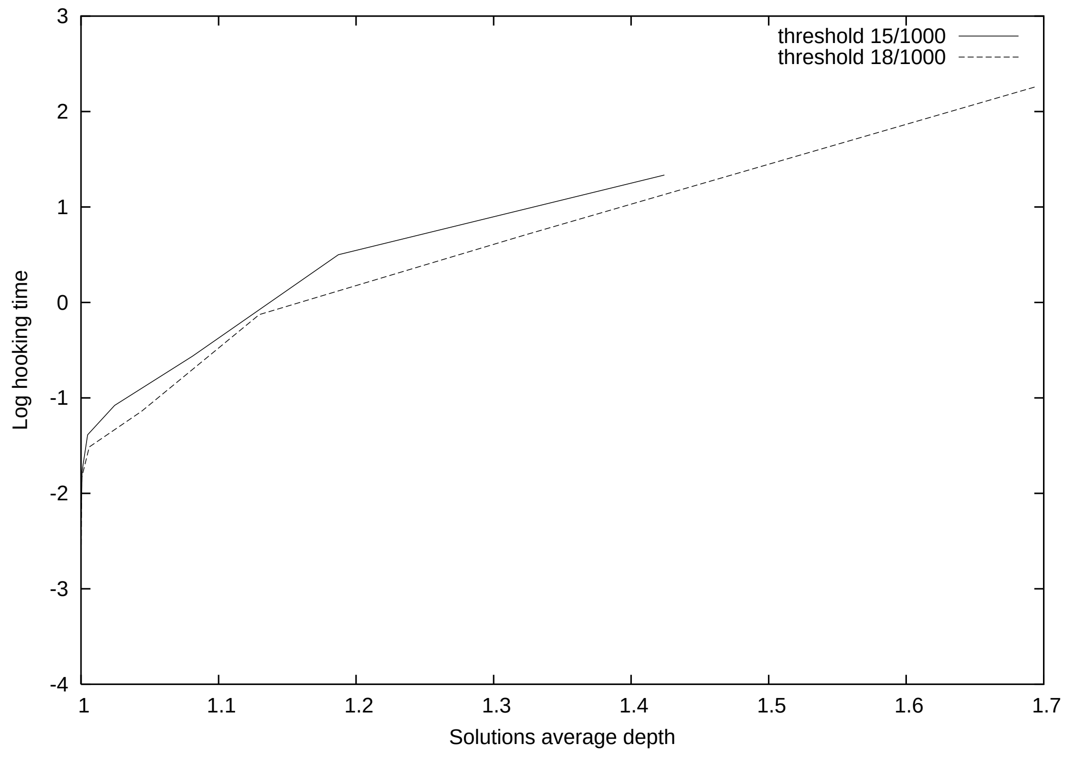

Figure IV.10: Corpus 2

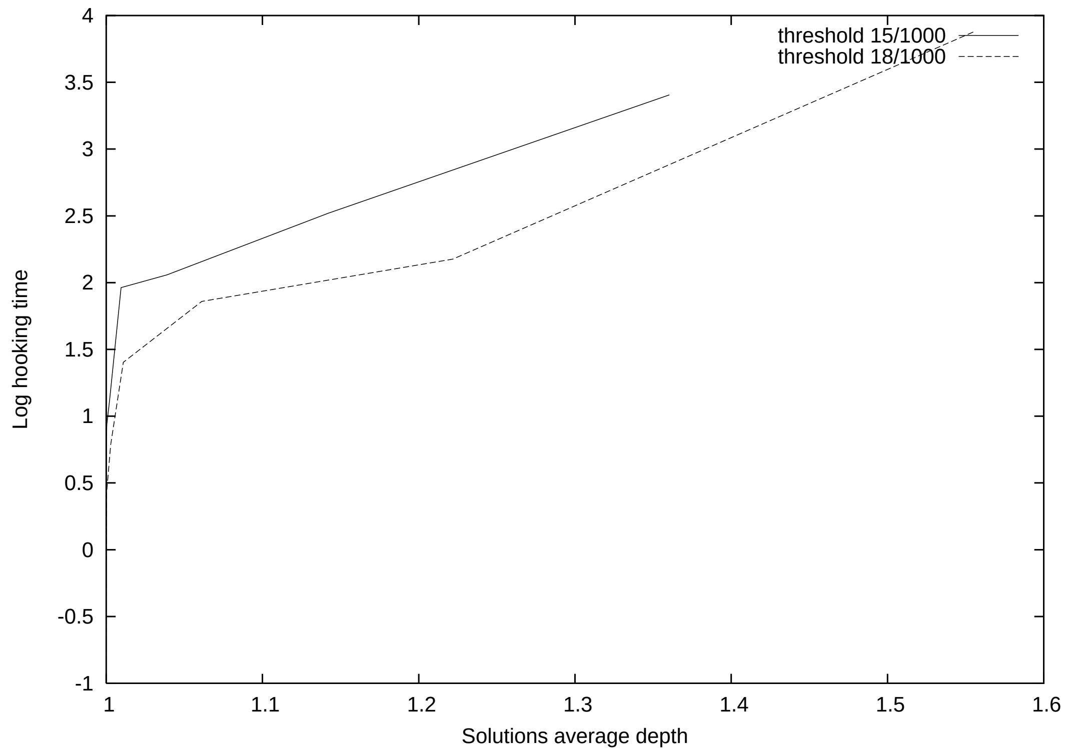

Figure IV.11: Corpus 3

3.3 Real data study

The Xyleme society kindly gave us a corpus of real XML documents. This corpus consists of 3667 XML documents representing sports news from the french press agency,*Agence France Presse (AFP)*. 3396 of these documents (92.6 %) satisfy the constraint **CNTR** and can be used in our experiments. We can here notice that our constraint is satisfied in a real life corpus, which means that in some cases in can be realistic. The trees of the corpus have an average depth between 2 and 3, and the average branch factor is between 4 and 5. The number of different labels is 32. So this experiment is more difficult than our previous experiments, because of the higher branch factor.

The documents follow the same writing guidelines, and 90 % of them have strictly the same structure. Hence the goal of DRYADE was to find this common structure. For this, we set the threshold to 3000.

On a 1 Ghz Athlon, it took 19 hours to DRYADE to find the frequent trees. This time can be decomposed in:

Computation of closed frequent patterns of depth 1	67081 s (18.6 h)
Computation of hooking graph	1592 s (0.4 h)
Hooking iterations	890 s (0.25 h)
Total	69563 s (19.3 h)

So the total time is dominated by the initial step of computation of closed frequent patterns of depth 1, which is as many runs of ECLAT as there are labels. This very high time can be explained by the fact that often, the trees of the corpus have a very high branch factor at the biggest depths. For exemple, some nodes can have more than ten leaves. The saturation by ancestor relationship between these leaves and their ancestors will produce many pairs of ancestor/descendant labels, which are the items for ECLAT. And ECLAT is exponential on the number of items.

Furthermore, our implementation is penalized by using ECLAT instead of CHARM, which would be much faster because CHARM computes directly closed frequent itemsets.

After this step, the hooking graph computation and the frequent pattern building are comparatively very fast, even if these steps are the most complex of DRYADE. So this situations differs from what we have seen in our experiments on randomized data.

We show in figure IV.12 the biggest tree discovered by DRYADE. This tree corresponds to the common structure of the XML documents of the corpus. We can see that this structure is complex, with 28 father/son edges.

4 Conclusion

In this experimental study, we have seen that DRYADE was way more efficient than the generalist approach WARMR. We have studied the effective complexity of DRYADE, which is polynomial on the input trees number, depends polynomialy on the number of patterns of depth 1 for the computation of the hooking graph, and on the depth of the solutions to find for performing the hookings.

In a difficult real-life application, DRYADE could find the structure common to 3000 XML documents, and it's the computation of frequent patterns of depth 1 that took most of the computation time.

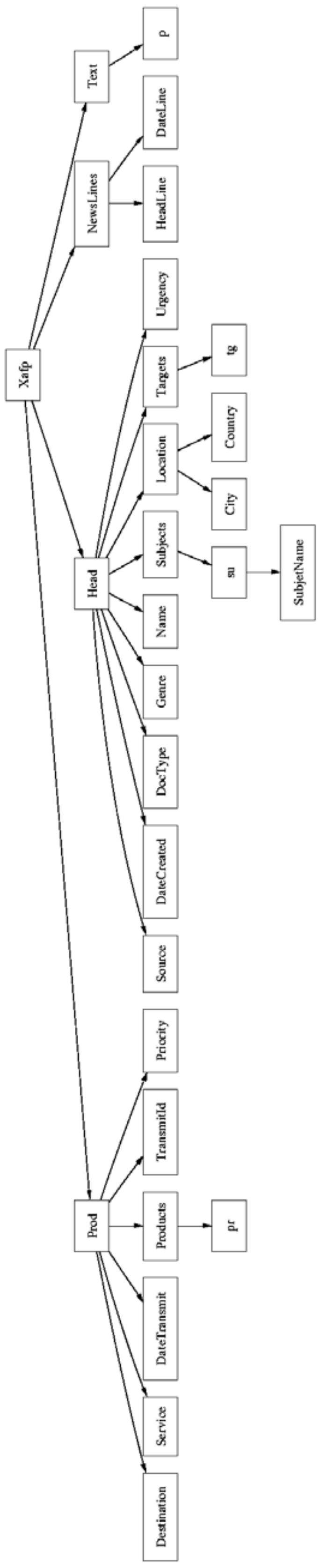

Figure IV.12: Biggest frequent tree

Annex

In this annex, we will explain the behavior of the curve of the number of frequent patterns of depth 1, in a corpus of trees of depth 3 whose average branch factor is 1. To simplify, we will suppose that the trees have a branch factor of 1, so they are like the tree of figure IV.13. We recall that we have a generation bias, so the set E of labels is split in 4 separate subsets of same size, and each node of a tree takes its labels in a different subset of E. The choice of the subset depends on the depth of the node.

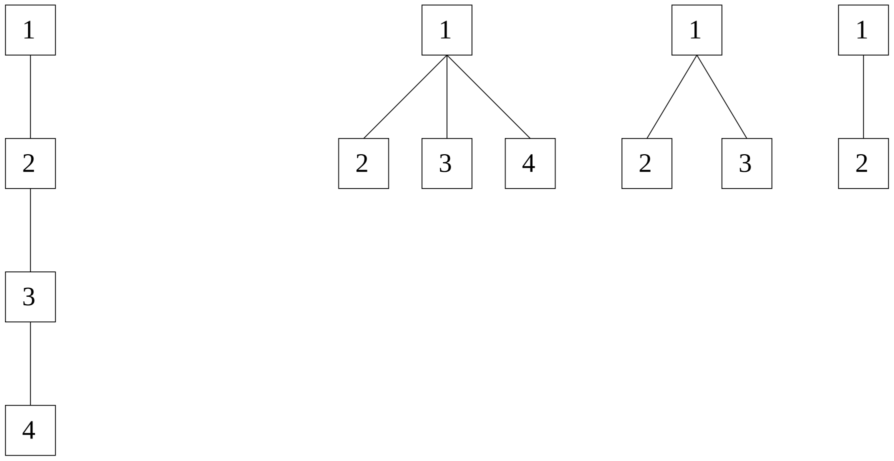

Figure IV.13: On the left, a tree of the corpus, on the right, different kinds of frequent patterns

The frequent patterns of depth 1 have three different kinds in the corpus: with 3 leaves, 2 leaves and 1 leaf, as shown in figure IV.13.

So we will compute the expectation of the frequent patterns of each kind for a given set of labels, the total of the expectations of the three kinds will give the expectation for all the frequent patterns of depth 1. Our variable will always be x, the number of labels.

To get the expectation of the frequent patterns with 3 leaves, we start by computing the probability $p_3(x)$ that a frequent pattern with three leaves is included in a tree A of the corpus.

A frequent pattern P with tree leaves will be included in a tree of the corpus if it's root node has been drawn in the first label subset, and all the leaves labels have been drawn from different subsets. So it's possible to make a mapping between P and a tree A of the corpus. Hence:

$$p_3(x) = (4/x)^4$$

For the pattern P to be frequent, it must appear in at least ε trees of the corpus. The probability that a pattern appears in exactly k trees of the corpus is given by the Bernouilli law:

$$\mathcal{B}(k, p_3(x)) = C_N^k \times p_3(x)^k \times (1 - p_3(x))^{(N-k)}$$

where N is the total number of trees in the corpus. So to have the probability that P is frequent, we have to sum the probability that P appears ε, the probability that P appears $\varepsilon+1$ and so forth.

$$pfrequent_3(x) = \sum_{k=\varepsilon}^{N} \mathcal{B}(k, p_3(x))$$

The expectation of the frequent patterns of depth 1 with three leaves is the product of the previous probability with the number of frequent patterns with 3 leaves, so:

$$E_3(x) = pfrequent_3(x) \times (\frac{|E|}{4})^4$$

The reasoning is the same for frequent patterns of depth 1 with two leaves and one leaf.

Trivilally,

$$p_2(x) = (4/x)^3$$

so:

$$pfrequent_2(x) = \sum_{k=\varepsilon}^{N} \mathcal{B}(k, p_2(x))$$

and

$$p_1(x) = (4/x)^2$$

so:

$$pfrequent_1(x) = \sum_{k=\varepsilon}^{N} \mathcal{B}(k, p_1(x))$$

To find the number of patterns with 2 leaves, notice that we must first choose a node of the target tree on which no mapping will be done (4 choices), then instanciate the labels of the nodes of the pattern. So:

$$E_2(x) = pfrequent_2(x) \times 4 \times (\frac{|E|}{4})^3$$

The same way, for the number of frequent patterns with 1 leaf, we must choose 2 nodes of the tree among 4, then instanciate the labels of the nodes of the pattern:

$$E_1(x) = pfrequent_1(x) \times C_4^2 \times (\frac{|E|}{4})^2$$

The figure IV.14 represents $E_1(x) + E_2(x) + E_3(x)$, we notice that there are the same two peaks that in the experiments, that correspond for the right peak to a majority of frequent patterns with 1 leaf, and for the left peak to a majority of frequent patterns with 2 or 3 leaves.

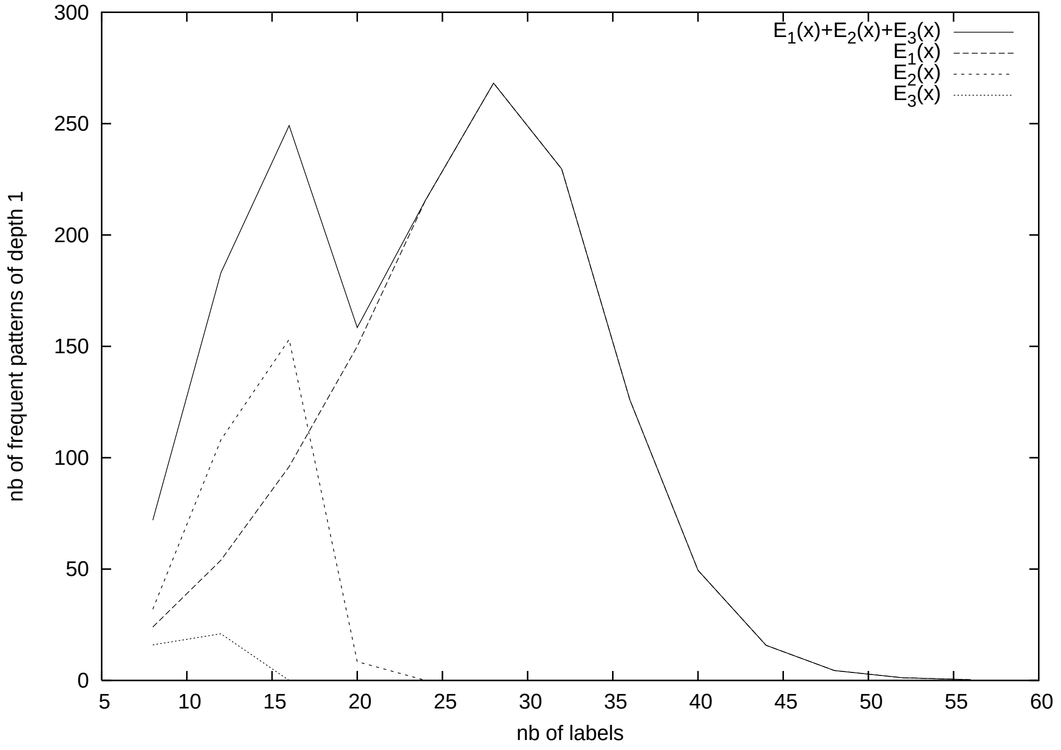

Figure IV.14: $E_1(x) + E_2(x) + E_3(x)$

Chapter V

Conclusion

In this thesis,we were interested in the problem of finding frequent trees. Even if generalist approach like Warmr can be used to solve this problem, we decided to get involved in specialised approaches, for the sake of efficiency. We have seen by analysing related works that existing methods use different tree inclusion definitions. The method proposed by Asai et al. uses a tree inclusion definition preserving the parent relationship, whereas the method proposed by Zaki is more general and only imposes to preserve the ancestor relationship.

However, these two methods impose that in the discovered frequent trees, the children or descendants have the same ordre than in input trees. The problem of finding frequent trees without taking into account siblings order hadn't been addressed so far by a specific approach. This can be explained by the difficulty of the tree inclusion test in this case: we have seen in chapter II that this test is NP-complete in the unordered case.

We have presented several solutions for the problem of frequent tree discovery, in the unordered case. The first of these solutions is the approximate algorithm TreeFinder, presented in [TRS02]. This algorithm uses thanks to adapted recodings some well known algorithms and techniques, the discovery of frequent itemsets in the transactional case, and the *Least General Generalization* computation. Any improvement on these techniques thus directly benefits to TreeFinder. We have seen that the TreeFinder algorithm is sound, but that in the general case it isn't complete. Our experimental study has shown that practically, as long as the overlap rate between the pair or ancestor/descendant labels of the frequent trees to discover was not too high, the completeness of the results given by TreeFinder was reasonably good. It is another important point of TreeFinder and of its experimental study: we have shown that our approximate algorithm could in many cases give satisfactory results. For real applications where speed is more important than approximation quality, an algorithm like TreeFinder can then be successfully used, as long as the input data doesn't lead to a too important completeness loss. It means that the overlap rate between the ancestor/descendant pairs of labels of the frequent trees musn't be too high.

After that, we have presented in the chapter III the algorithms Dryal and Dryade. These algorithms compute frequent patterns, i.e. trees where no nodes has two children with the same label. The paper [DRR+03] has shown that this restriction on the shape of trees didn't prevented to find interesting tree structures to query XML documents. The algorithms Dryal and Dryade make use of the tree structure of data in an innovative approach by building gradually the set of solutions, by successive hookings of the frequent patterns of depth 1. The frequency computations of candidates make an intensive use of the vertical methods for finding frequent itemsets, and thus avoid costly tree inclusion tests. We have detailed the Dryal and Dryade algorithms, and proven their soundness and completeness in the general case. We have then realised an experimental study with our implementation of Dryade, in the chapter IV. This study first shown that Dryade was more efficient than the generalist algorithm Warmr, with a performance gain of several orders of magnitude in execution time. This results justifies *a posteriori* the approached we followed, i.e. making use of the specificities of tree structures. Moreover, the significative performance improvement comes from the use of the closure constraint, inspired from Closed and Charm. An empiric study of the complexity of Dryade has been presented, and showed the good scaleup capacity of Dryade. Last, the experimentation on complex real data shown that Dryade could be used in real applications.

Our main contribution, concretised by the Dryal and Dryade algorithms, can be split in two points.

First, the Dryal and Dryade algorithms are based on the intensive used of vertical approaches for finding closed frequent itemsets. These approaches ensure not only the initialisation step of Dryade, but are also used in the algorithm for the choices of the hookings to do and the choices of the leaves to suppress. The interest of reusing these algorithms is twofold: on one hand, it's a well known framework, which can ease the comprehension and future improvement of our algorithms. On the other hand, these algorithms are the topic of a lot of works, hence any improvement made to them can be directly applied to Dryal and Dryade. So by improving propositional frequent itemset discovery, frequent pattern discovery is automatically improved as well. It seems very important to us.

Second, we have defined for the Dryade algorithm the concept of closed frequent pattern. We have thus extended the concept of closed frequent itemset to the frequent patterns. We have observed in our experiments that the restriction to closed frequent patterns allowed a very important perfomance gain. We hope that the use of the closure constraint will benefit to all the works on structured data mining, our approach illustrating on both theoretical and practical bases the utility of this constraint.

Perspectives

The presented works lead to several research perspectives. In the short-term, an improvement of our current Dryade implementation seems necessary, in order to avoid the restrictions seen in the chapter IV. It will be interesting to measure the performance gain brought by this new implementation. We also wish to study some optimisations for Dryade, especially in the update of the hooking graph. A judicious use of some closure properties could allow an anticipate elimination of some useless arcs, avoiding to generate some patterns that would become redundant later.

In the medium-term, it is clear that Dryade is intended to be integrated in a real data mining process. The kind of application considered would be to start from an heterogeneous collection of XML documents, in such a collection the queries are given with a query language like X-Query. But because of the heterogeneity of the collection, it can be very difficult for the user to know which tags to use for his/her query. Here Dryade can be very usefull: an execution of Dryade on the collection of XML documents will give frequent patterns, and we will be sure that these patterns appear at last ε in the collection. These patterns can be used as typical queries by the user, an inform him/her on what can be found in the collection. A presentation of frequent patterns with forms can be provided to ease their handling by the user (figure V.1).

In this figure, a frequent pattern about used car ads is found. Presented as a form, the user only has to fill the value he/she is interested in, and the query will be issued with X-Query.

The study of a real problem will allow our approach to focus on significative problems, and to

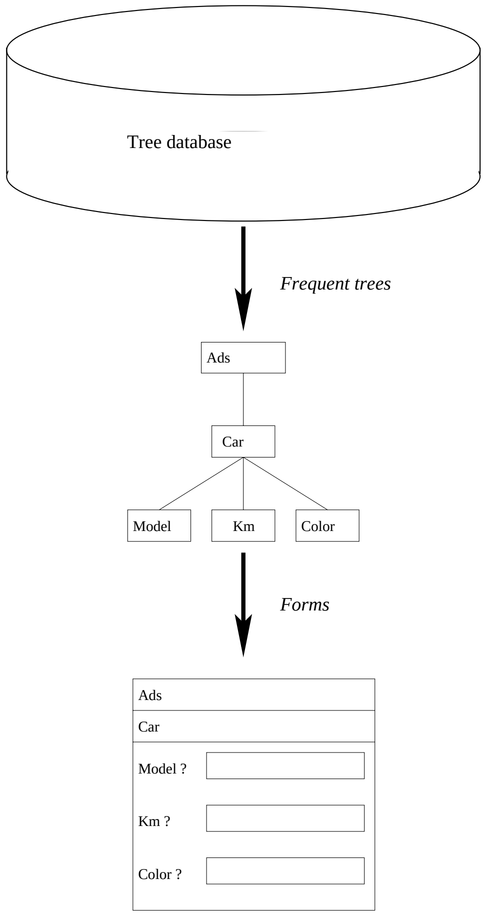

Figure V.1: Form construction example

propose extensions favoring the pertinent compromise ease of use/utility.

In the long run, several perspectives are opened. As we said before, a first perspective is to extend the discovery of closed frequent structures to sequences and graphs. This could allow an efficiency improvement on these problems, and to tackle new kinds of applications. For example for graph data, there are many works with chemical moleculas databases ([IWM00], [MS03]), and analysing them with a closed approach would certainly be very interesting.

Another perspective is to relax the constraint on the shape of trees discovered by Dryade, so it doesn't anymore discovers frequent patterns but frequent trees, that can have for each node several children with the same label. In these conditions the cost of Apriori-like approaches increases a lot: we then have to discover how to stay as efficient as possible despite the combinatorial explosion.

A last, more theoretical perspective is to define, as we have done for closed frequent patterns, the notion of maximal frequent pattern, and to modify Dryade to compute such patterns. An exciting perspective would then be to discover the linking between frequent / frequent closed / frequent maximal itemset and frequent / frequent closed / frequent maximal patterns. This investigation would be interesting to do from a point of view of convergence between algorithms and for theoretical properties. Furthermore, an experimental comparison of the performance differences between algorithms on propositional data (classical, closed, maximal) and the correponding algorithms on structured data would be capital. We could then learn if for frequent structures, it's more interesting to use closed or maximal approaches than for propositional data. An even longer run goal would be to propose a general framework about frequent pattern discovery, whatever there structure. This is an instanciation for structured data (trees and graphs), with closure and maximality constraints, of the general framework defined in [MT97].

CHAPTER A

Annex

In the chapter III, we have restricted the soundness and completeness of DRYADE to the case of frequent patterns verifying the frequent occurrences independancy property, which imposes that each pattern has at least ε occurrences that are not descendant of one another. We will explain in this chapter why we had to use this restriction, and a technique enabling DRYADE to be sound and complete in the general case.

1 Limitations of occurrences support

Lets consider figure A.1.

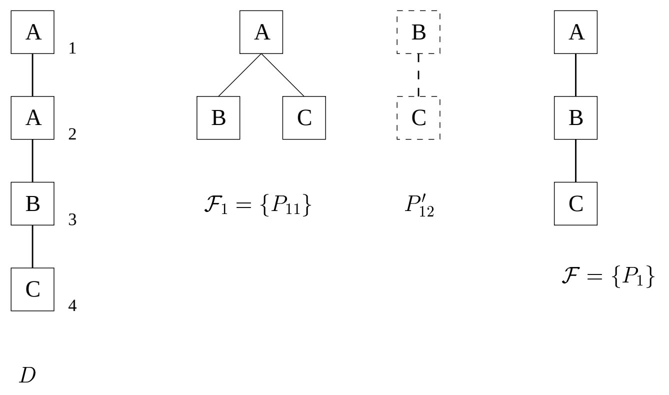

Figure A.1: For $\varepsilon = 2$, D contains the frequent pattern P_1, which does not satisfies the frequent occurrences independancy property

With a frequency threshold of $\varepsilon = 2$, the pattern P_1 has an occurrences support of 2, and so is frequent. This is the unique pattern of $\mathcal{F}$, and it does not satisfies the frequent occurrences independancy property. To find P_1 with DRYADE, first the patterns of depth 1 must be computed. The only closed frequent pattern of depth 1 is P_{11}, which is as expected the flattening of P_1. However, to rebuild the structure of P_1 from P_{11}, we need the flattenings of all the subtrees of P_1. P_1 has only one subtree, already of depth 1 (hence equal to its flattening), represented by P'_{12}. But this subtree is not frequent with the occurrences support. It cannot be discovered by the patterns of depth 1 detection step, hence it is not possible to rebuild P_1 from P_{11}.

2 Weighted support

To solve this problem, i.e. be able to find all the sub-patterns of a frequent pattern even is this pattern does not satisfies the frequent occurrences independancy property, we first give a new definition of support.

Definition 41 (weighted support) *Let P be a pattern, and D a datatree. $Locc(P,D)$ is the set of occurrences of P in D. Each occurrence $o \in Locc(P,D)$ has a weight $weight(o)$ defined as follows : let l be the label of o, and $\{l_1, ..., l_n\}$ the labels of the ancestors of o. The function $\#ancestors\_label(X)$ returns the number of ancestors of o with label X.*

Then

$$weight(o) = \#ancestors\_label(l_1) * ... * \#ancestors\_label(l_n)$$

.

If o has no ancestors, then $weight(o) = 1$.

The support of P in D is the sum of the weights of the occurrences of P :

$$support(P,D) = \sum_{o \in Locc(P,D)} weight(o)$$

Property 14 *A pattern frequent for occurrences support is also frequent by weighted support.*

Proof :

A pattern frequent for occurrences support has at least ε occurrences. The weight of each of these occurrences being at least 1, this pattern is also frequent for weighted support. □

Property 15 (Partial monotony) *If a pattern is frequent for occurrences support, then all its sub-patterns are frequent for weighted support.*

Proof :

Let P be an occurrences-frequent pattern: $support_o(P) \geq \varepsilon$, whose root label is l. Let S be a sub-pattern of P, whose root label is l'. We have $Locc(P,D) = \{o_1, ..., o_n\}$ $(\mid \{o_1, ..., o_n\} \mid \geq \varepsilon)$ and $Locc(S,D) = \{o'_1, ..., o'_m\}$. $S \sqsubset P$ so $\forall o_i \in Locc(P,D)$ $\exists o'_j \in Locc(S,D)$ tq $o'_j \in descendants(o_i)$. Let E be the set of these o'_j.

Then for $o' \in E$, there exists a set $\{o_{i1}, ..., o_{ik}\} \subseteq Locc(P,D)$ of occurrences of P which are ancestors of o'.

Hence $weight(o') = \#ancestor\_label(l) * \alpha_{o'} \geq k$

So $weight(S) = \sum_{o \in Locc(S,D)} weight(o) \geq \sum_{o' \in E} weight(o') \geq \sum_{o' \in E} k_{o'}$

As all the $o' \in E$ have at least one ancestor $o \in Locc(P,D)$, $\sum_{o' \in E} k_{o'} \geq \mid Locc(P,D) \mid \geq \varepsilon$, so S is frequent for occurrences support.

□

With this property, it is possible to modify DRYADE for it to be sound and complete in the general case. The modification is to use weighted support instead of occurrences support for the discovery of patterns of depth 1. Then all the flattenings of the frequent patterns and of their sub-patterns are correctly discovered, and from there the hooking strategy of DRYADE computes all the closed frequent patterns.

This can be proved by noticing that the only place where the frequent occurrences independancy property is used in the soundness and completeness proof of DRYADE is the lemma 2. The property that we just given enables extension of this lemma to the general case.

Note that the frequent patterns that we want as final result must satisfy occurrence frequency. So it is necessary in the rest of the algorithm to use occurrences support as in chapter III, and check that all the selected root patterns at each step truly satisfy occurrences support. This because weighted support can introduce unwanted patterns, as show in figure A.2.

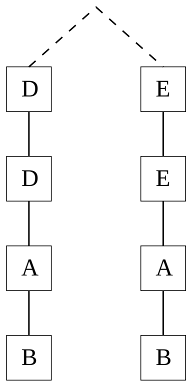

Figure A.2: Datatree example

If the frequency threshold is $\varepsilon = 4$, there are no frequent pattern for occurrences support, but the weighted support will consider that the pattern $A - B$ is frequent. This result is unwanted and would not be understood by the user. A simple check of the number of distinct occurrences of a root pattern allows to filter this kind of cases.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAK$^+$02] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *In Proc. of the Second SIAM International Conference on Data Mining (SDM2002), Arlington, VA*, pages 158–174, Avril 2002.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[BMS97] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: generalizing association rules to correlations. In *In Proc. 1997 Int. Conf. of Management of Data, Tucson, Arizona*, pages 265–276, 1997.

[CYXM04] Yun Chi, Yirong Yang, Yi Xia, and Richard R. Muntz. Cmtreeminer: Mining both closed and maximal frequent subtrees. In *The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, 2004.

[Deh98] Luc Dehaspe. *Frequent pattern discovery in first-order logic.* Phd, Department of Computer Science, K.U.Leuven, Leuven, Belgium, dec 1998. 193+xiii pagesURL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=20990.

[DRR$^+$03] Claude Delobel, Chantal Reynaud, Marie-Christine Rousset, Jean-Pierre Sirot, and Dan Vodislav. Semantic integration in xyleme: a uniform tree-based approach. *Journal on Data and Knowledge Engineering*, 44(2):pp 267–298, 2003.

[DT99] Luc Dehaspe and H. Toivonen. Discovery of frequent Datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999. URL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=20765.

[IWM00] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.

[JB02] Baptiste Jeudy and Jean-Franois Boulicaut. Using condensed representations for interactive association rule mining. In *Proceedings of the 6th European conferences on Principles and practice of Knowledge Discovery in Databases ECML/ PKDD 2002, Helsinki*, pages 225–236, 2002.

[Jr.98] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, Juin 2-4, 1998, Seattle, Washington, USA*, pages 85–93. ACM Press, 1998.

[Kil92] Pekka Kilpelinen. *Tree Matching Problems with Applications to Structured Text Databases.* PhD thesis, Department of Computer Science, University of Helsinki, Novembre 1992. Rapport A-1992-6.

[LK98] Dao-I Lin and Zvi M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent set. *Lecture Notes in Computer Science*, 1377:105–??, 1998.

[MS03] Jrme Maloberti and Einoshin Suzuki. Improving efficiency of frequent query discovery by eliminating non-relevant candidates. In *Discovery Science 2003*, pages 220–232, 2003.

[MT97] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[PBTL99] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, Janvier 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer, 1999.

[PHM00] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[SON95] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *The VLDB Journal*, pages 432–444, 1995.

[Toi96] Hannu Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *In Proc. 1996 Int. Conf. Very Large Data Bases*, pages 134–145. Morgan Kaufman, Septembre 1996.

[TRS01] Alexandre Termier, Marie-Christine Rousset, and Michle Sebag. Combining statistics and semantics for word and document clustering. In *Ontology Learning IJCAI workshop*, 2001.

[TRS02] Alexandre Termier, Marie-Christine Rousset, and Michle Sebag. Treefinder: a first step towards xml data mining. In *International Conference on Data Mining ICDM'02, Maebashi, Japon*, 2002.

[Zak02] Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *In Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Juillet 2002.

[ZG03] Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *In Proc. 9th International Conference on Knowledge Discovery and Data Mining, Washington, DC*, August 2003.

[ZH02] Mohammed J. Zaki and Ching-Jui Hsiao. Charm: An efficient algorithm for closed itemset mining. In *In Proc. 2nd SIAM International Conference on Data Mining, Arlington*, Avril 2002.

[ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *In Proc. of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 283–286. AAAI Press, 1997.