UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTER

En Informatique
Spécialité : Systèmes d'Informations et de Connaissances

FOUILLE DANS LES DOCUMENTS XML

Par

Amina MADANI

Devant le jury composé de :

M. Hadj-SADOK	Maître de conférences, U. Blida	Président
K. BENATCHBA	Maître de conférences, E.S.I, Alger	Examinateur
D. BENNOUAR	Maître de conférences, U. Blida	Examinateur
H. BOUARFA ABED	Professeur, U. Blida	Promoteur
O. BOUSSAID	Professeur, U. Lyon 2, France	Co-Promoteur

Blida, Mai 2010

RESUME

L'objectif de notre travail est de faire une étude bibliographique exhaustive sur la fouille dans les documents XML. Il s'agit d'effectuer un bilan de l'ensemble des travaux réalisés dans ce domaine et d'en faire une étude approfondie des différentes approches afin de comprendre les objectifs visés et les mécanismes utilisés et comparer les différents travaux en se basant sur quelques critères de comparaison.

De plus, il s'agit de proposer une approche de fouille combinant les deux catégories : la fouille dans la structure et la fouille dans le contenu des documents XML. Il s'agit de formaliser l'approche et de développer un prototype permettant l'implémentation de cette approche. Une dernière phase porterait sur la validation de l'approche proposée. C'est une évaluation expérimentale à réaliser sur différentes collections de documents XML.

Mots clés : fouille dans les documents XML, fouille de données, extraction de connaissances, *text mining*, *clustering*, classification, thésaurus, sac de mots.

ABSTRACT

The objective of our work is to browse many works achieved in the literature concerning the XML mining. We are going to study the different approaches to understand the aims and the mechanisms used and compare them using some criterias of comparison.

After, we propose a new approach of XML mining combining the two categories: XML structure mining and XML content mining. We formalize the approach proposed and we develop a prototype allowing the implementation of this approach.

The last step would focus on the validation of the approach suggested. It is an experimental evaluation to realize on different XML documents collections.

Key words : XML mining, data mining, text mining, clustering, classification, thesaurus, bag of words.

الملخص

فـي الـبداية، فإن الهدف من عملنا هو القيام بدراسة مجال البحث في وثائق إكس أم أل، استعراض جميع الأعمال في هذا المجال وإجراء دراسة شاملة للمناهج المختلفة لفهم الأهداف والآليات المستخدمة و مقارنة هذه الأعمال بناء على بعض المعايير.

والخطوة التالية هي اقتراح نهج جديد للبحث في وثائق إكس أم أل، مع الجمع بين هاتين الفئتين : البحث في هيكل الوثائق و البحث في محتوى الوثائق، ثم وضع نموذج أولي لتنفيذ هذا النهج.

المرحلة الأخيرة تركز على التحقق من صحـة النهج المقترح و تقييمه بإجراء تجارب على مجموعات مختلفة من وثائق أكس أم أل.

الكلمـات الـرئيسـية : البحث في وثائـق إكس أم أل، البحث في المعطـيات، البحث في النصوص، استخراج المعلومات، التصنيف، حقيبة الكلمات.

REMERCIEMENTS

En tout premier lieu, je remercie **Allah** le tout puissant à la sagesse, la volonté et au savoir infinis, " **Gloire à Toi ! Nous n'avons de savoir que ce que Tu nous as appris. Certes c'est Toi l'Omniscient, le Sage.** " (Sourate al-Baqarah, verset 32).

Je tiens à remercier mon promoteur, **Mr. BOUSSAID Omar**, qui m'a donné l'opportunité de découvrir le domaine de *XML mining* à travers un projet dans son intégralité. J'ai particulièrement apprécié son encadrement privilégié, ses qualités humaines et scientifiques et son soutien m'ont permis de mener à bien et avec grand plaisir et liberté ces travaux et je le remercie très chaleureusement.

Mes sincères remerciements à **Mme. ABED Hafida**, d'avoir accepté de rapporter ce mémoire, pour sa disponibilité, ses commentaires, ses conseils et pour la liberté de recherche qu'elle a bien voulu me laisser.

Je remercie vivement les membres du Jury qui ont accepté d'honorer ma soutenance.

Et surtout

Merci à mes parents (**Ma Mère, Mon Père**). Je leur serais éternellement reconnaissante de m'avoir assuré de leurs confiances, de leurs encouragements et de leurs soutiens sans limite. Merci aussi à mes frères. Merci à toute ma famille. L'amour bienveillant de ma famille, représente une force précieuse qui m'a toujours permis d'affronter les aléas de la vie.

Je ne pourrais assez remercier Mr. SAADI pour son aide inestimable.

Bien d'autres personnes ont contribué à rendre possible ce travail, j'exprime ma gratitude à tous les chercheurs, consultants et internautes rencontrés lors des recherches effectuées et qui ont accepté de répondre à mes questions avec gentillesse.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis, qui m'ont toujours soutenue et encouragée au cours de la réalisation de ce mémoire.

Merci à tous et à toutes.

DEDICACE

Je dédie ce mémoire à

Mes chers parents,

Mes deux adorables frères,

Ma famille,

Mes défunts grands-pères,

Mon défunt oncle Messaoud,

Et à toute l’humanité.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES EQUATIONS

INTRODUCTION

Le développement des documents électroniques et du Web ont permis l'émergence de formats semi-structurés permettant la représentation et le stockage de documents textuels ou multimédias. Différents formats comme le XML sont aujourd'hui très populaires et sont en train de s'imposer. Ils permettent de représenter l'information sous une forme enrichie et adaptée à des besoins spécifiques. Ces types de formats permettent de représenter conjointement les données et une information sur leur structure.

En outre, Les techniques d'analyse traditionnelles ne permettent pas une exploration adaptée à ce genre de documents. En effet, la nature des données semi-structurées présente des informations diverses sur les données. Les techniques de fouille de données (*data mining*) s'appliquant sur les documents textuels, plus particulièrement le *text mining*, ne tiennent pas compte des aspects spécifiques des documents XML.

Différents travaux sur le *XML mining* sont alors proposés. D'une manière générale, la fouille de documents XML (*XML mining*) s'appuie sur les techniques classiques de fouille et notamment le classement (*classification*) et la classification (*clustering*).

La fouille dans les documents XML comprend deux catégories :

- La fouille dans la structure des documents XML (*XML Structure Mining)* qui s'intéresse à l'extraction d'informations à partir de la structure des documents XML.

- La fouille dans le contenu des éléments XML (*XML Content Mining*) qui regroupe les méthodes de fouille sur le contenu des documents. La fouille sur le contenu utilise habituellement les techniques de *text mining* pour extraire l'information contenue dans le document.

Quelques tentatives ont été réalisées dans le domaine de *XML mining*. Dans la première étape de notre travail, nous avons exposé l'évolution de ces tentatives en faisant une analyse approfondie de l'état de l'art et en parcourant un nombre

important des travaux réalisés dans ce domaine. Aussi, nous avons arrêté un certain nombre de critères pour pouvoir faire une comparaison entre ces travaux.

L'étape suivante a permis de proposer une démarche de fouille combinant les deux approches *XML Structure Mining* et *XML Content Mining* pour extraire des connaissances à partir des documents XML. En se basant sur d'autres travaux, notre approche permet de faire le *clustering* des documents XML. Pour cela, nous avons défini tout d'abord un thésaurus contenant deux sacs de mots. L'un représente le contenu structurel et l'autre comporte le contenu textuel de toute la collection des documents XML. Les sacs recensent tous les éléments présents dans la collection tels que chaque élément est composé de plusieurs termes. Le thésaurus regroupe les éléments qui ont une même signification en faisant appel soit à une ontologie du domaine soit à l'ensemble des experts du domaine. Chaque document XML est ensuite transformé en un arbre réduit et un ensemble de chemins est généré. Les ensembles sont alors mappés dans une matrice binaire sur laquelle des méthodes standard de *clustering* peuvent être appliquées. Nous avons opté pour le *k-means* vu sa complexité linéaire et la simplicité de son algorithme.

La dernière étape est la validation de la démarche proposée. C'est une évaluation expérimentale appliquée sur deux collections de documents XML.

Le présent mémoire est structuré comme suit :

Le chapitre 1 présente les notions essentielles du langage de balisage extensible XML, sa structure de base, ses principaux composants, les techniques de description des documents telles que les DTD et les schémas XML et les conditions de validation d'un document XML.

Le chapitre 2 aborde les concepts de base du *data mining*, son processus, ses tâches et ses différents algorithmes et techniques ainsi que le *text mining* et l'application du *data mining* sur les données multimédia.

Le chapitre 3 est consacré à l'état de l'art sur les approches de fouille dans les documents XML. Un survol des approches existantes dans ce domaine permettra de voir différentes démarches de mise en œuvre. Aussi, un certain nombre de critères sont définis pour pouvoir établir une comparaison entre les approches.

Dans le chapitre 4, nous présentons notre approche de fouille des documents XML qui permet de prendre en compte la structure et le contenu. Elle consiste à

appliquer le *clustering* sur les documents représentés par un ensemble des chemins. Ces derniers sont mappés dans une matrice de chemins sur laquelle le *clustering* est appliqué. L'approche utilise un thésaurus créé au préalable pour gérer l'aspect sémantique des mots.

Le chapitre 5 présente la validation de notre approche. Elle est effectuée à travers un prototype que nous avons développé qui permet de réaliser des expérimentations sur les collections de tests. Les collections XML de tests et les résultats de l'évaluation expérimentale sont aussi présentés dans ce dernier chapitre.

La conclusion dresse quelques points importants sur nos contributions dans cette recherche et enfin les perspectives envisagées pour complémenter ce travail.

CHAPITRE 1
LA TECHNOLOGIE XML

1. Introduction

Dans ce chapitre, nous présentons les grandes lignes du langage XML. Nous allons voir comment écrire un document XML, étudier sa structure de base, ses principaux composants. Nous allons aussi aborder les techniques de description de documents telles que les DTD et les schémas XML et les conditions de la validation d'un document XML. La syntaxe de toutes les déclarations des balises, des éléments composant un document XML, des DTD et des schémas XML utilisée dans ce chapitre est définie dans la référence **[133]**.

2. XML

XML (*eXtensible Markup Language*), langage de balisage extensible standardisé par le W3C [1] (*World Wide Web Consortium*), est un langage de description et d'échange de documents structurés **[84]**.

XML est un méta-langage de description qui permet de créer des langages de description pour différents types de documents et différents secteurs d'activités **[60]**. Il pourrait être davantage perçu comme une partie d'un système d'information, car il contribue fortement à l'échange de l'information à un moment donné **[14]**.

Nous pouvons déduire, qu'il s'agit d'un format de documents permettant de représenter des données et leurs sens sous forme de texte, tout en conservant une certaine organisation. Il permet de définir d'autres langages de description structurés spécifiques à certains domaines.

3. Origine

L'objectif d'un grand nombre d'entreprises et de chercheurs partenaires du W3C était de définir un formalisme permettant d'échanger facilement des documents complexes sur le web, en dépassant les limites imposées par HTML (*Hyper-Text*

[1] W3C : *World Wide Web Consortium* : association ayant pour charge de développer les standards web, http://www.w3c.org/.

Markup Language) tout en offrant la richesse sémantique de SGML (*Standard Generalized Markup Language*).

- HTML : c'est un langage de balisage avec des liens hypertextes parfaitement adapté au web mais dont les applications sont limitées par un ensemble de balises prédéfini. HTML qui devait décrire le contenu du document s'est orienté vers la présentation du contenu.
- SGML : c'est un langage normalisé de balisage généralisé, très utilisé dans l'industrie pour créer de grandes documentations techniques (systèmes d'armes, véhicules automobiles, avions, etc.) ou lexicographiques (dictionnaires, encyclopédies) **[84]**, riche en sémantique mais relativement lourd et complexe à mettre en œuvre et inadapté au web.

XML est le nouveau langage d'échange basé sur le balisage, plus simple que SGML et plus ouvert que HTML **[84]**.

4. Les avantages de XML

Les principaux avantages qui peuvent motiver le choix de XML sont :

- Lisibilité **[84]** : Les balises peuvent être définies librement pour marquer les éléments préservant la sémantique des composants d'un document en utilisant des termes assez intuitifs.
- Séparation de la structure et de la présentation **[73]** : Dans un document XML, l'information de formatage est écrite dans un langage de style et est stockée séparément. Une balise indique ce que l'information signifie, non pas comment l'afficher.
- Modularité et réutilisabilité **[84]** : Tout utilisateur de XML peut définir librement sa propre structure de document XML (DTD ou Schéma XML). XML offre un mécanisme simple et puissant pour partager et réutiliser des parties de structures types.
- Accès à des sources d'information hétérogènes **[84]** : XML propose un format d'échange de données normalisé, général, indépendant de toute plate-forme et de tout SGBD, et suffisamment puissant pour que l'immense majorité des données puissent être efficacement représentées et manipulées.
- Contrôle de validité **[84]** : La validation d'un document XML garantit que la structure de données utilisée respecte sa DTD ou son schéma XML.

5. Structure d'un document XML

Tout document XML se compose **[84]** :

- D'un prologue : C'est l'entête du document dont la présence est facultative mais conseillée.
- D'un arbre d'éléments : C'est le corps du document XML.
- De commentaires et d'instructions de traitements, dont la présence est facultative, et qui pourront moyennant certaines restrictions, apparaitre aussi bien dans le prologue que dans l'arbre d'éléments.

Nous illustrons à travers un exemple les différents composants d'un document XML :

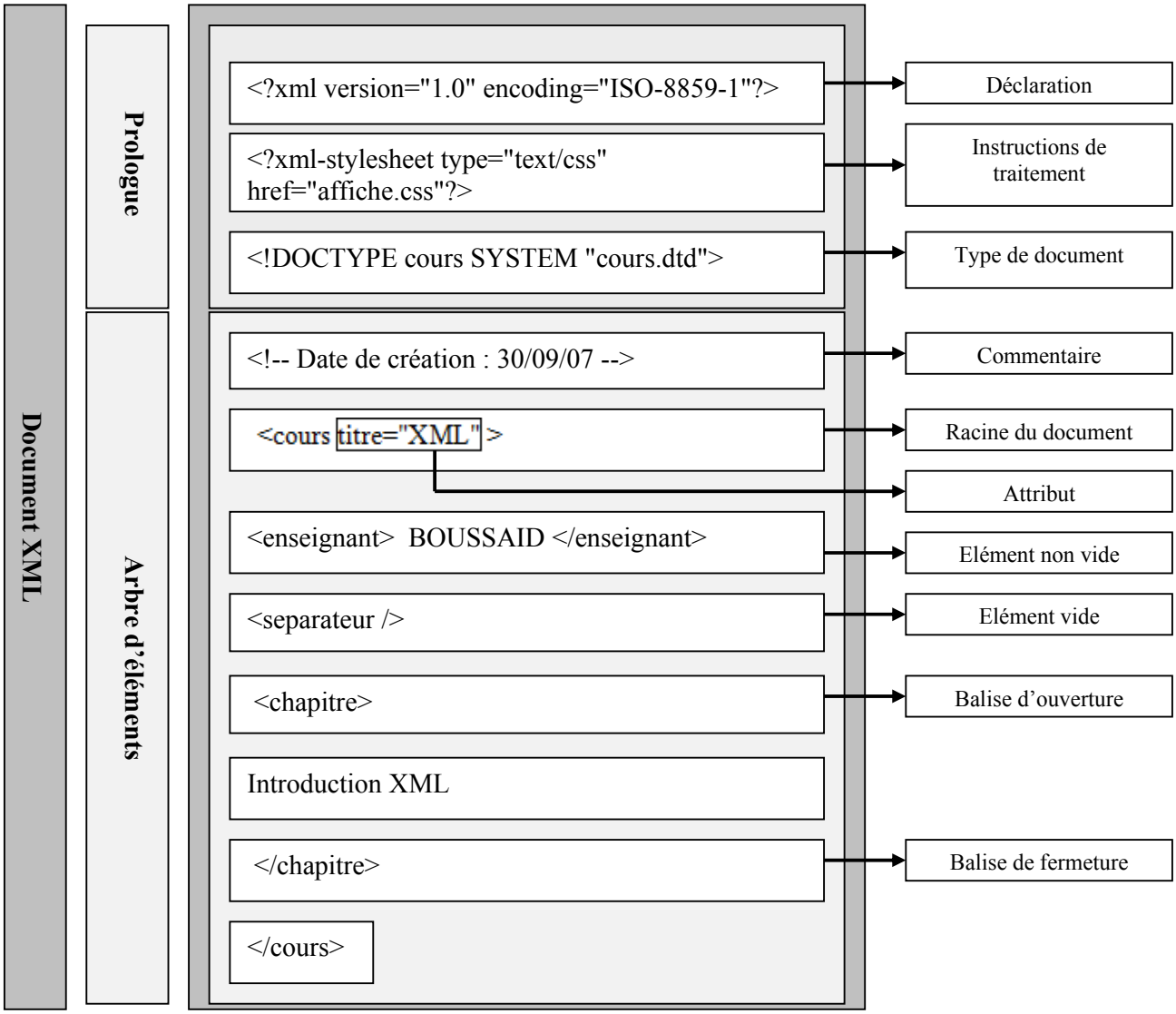


Figure 1.1 : Structure d'un document XML.

5.1. Le prologue

Le prologue XML est composé d'une déclaration XML, d'instructions de traitement et éventuellement d'une ou plusieurs déclarations de type de documents **[84]**.

5.1.1. La déclaration XML

La déclaration XML est la première ligne d'un document XML. Elle permet de donner les caractéristiques globales du document **[84]**. Parmi ces caractéristiques, nous citons:

- La version du langage XML (1.0 ou 1.1).
- Le jeu de caractères d'encodage du document XML « *encoding »* (UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 à ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP), lorsque l'encodage n'est pas précisé, la valeur UTF- 8 est utilisée par défaut.
- La déclaration de document autonome : (*standalone="yes"*) spécifie si le document est autonome ou s'il dépend, pour son fonctionnement, d'autres fichiers ou de toutes autres ressources externes (*standalone="no"*).

Une déclaration XML est encapsulée par <*?* et *?*> et a la forme suivante :

<?xml version="1.0" encoding=" UTF-8"?>

5.1.2. Les instructions de traitement

Les instructions de traitement (*processing instruction* ou PI) servent à donner à l'application qui utilise le document XML des informations. Ces dernières sont totalement libres et dépendent avant tout du concepteur de l'application de traitement **[14]**.

Un cas typique est d'attacher une feuille de style « affiche.css » à un document XML pour afficher le contenu de ce dernier sur l'un des navigateurs Mozilla, Firefox ou Internet Explorer avec l'instruction :

<?xml-stylesheet type="text/css" href="affiche.css"?>

5.1.3. Les déclarations de type de document

La déclaration de type de document annonce la nature du document **[73]**, cette déclaration optionnelle sert à attacher une grammaire de type DTD (*Document Type Definition*) à un document XML **[14]**. Elle est introduite sous cette forme :

<!DOCTYPE nom-DTD SYSTEM "Lien de l'emplacement de la DTD">

Le mot-clé SYSTEM indique qu'il s'agit d'une DTD privée, utilisée par un groupe. L'alternative est le mot-clé PUBLIC qui est utilisé pour les DTD conçues pour être exploitées de manière publique.

Dans l'exemple suivant, la DTD « cours.dtd » est attachée au document XML :

<!DOCTYPE cours SYSTEM "cours.dtd">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Strict//EN" >

Ici, l'URI[1] est remplacée par un identifiant public composé de **[14]** :

- - : indique qu'il s'agit d'un format non compatible ISO[2] (sinon on utilise +).
- IETF : organisme gérant le format.
- DTD : indique la nature du document.
- HTML 2.0 Strict : description du document.
- EN : un code langue (ISO 639).

5.2. Les commentaires

Les commentaires sont utilisés pour ajouter du texte explicatif à un document XML. Ils peuvent inclure n'importe quel caractère sauf le « -- » **[84]**, ils prennent la forme suivante : <!-- Ceci est un commentaire -->

5.3. L'arbre d'éléments

Les documents écrits avec XML ont une structure arborescente. Un arbre est composé d'un élément racine qui est à son tour composé de sous éléments.

5.3.1. Les éléments

Les éléments gèrent la structuration des données d'un document XML. Un élément se compose d'une balise ouvrante, d'un contenu et d'une balise fermante

[1] URI : *Uniform Ressource Identifier*, chaine de caractères identifiant le type et l'emplacement d'une ressource Internet.
[2] ISO : *International Organisation for Standardization*, définit des normes industrielles pour toutes sortes de choses, le mesurage, les jeux de caractères, les containers d'expédition, les procédés qualité, etc.

[84]. Le contenu, peut être composé de sous éléments. Un élément prend la forme suivante : <nom> contenu </nom>

Le nom d'un élément figurant dans les balises ouvrante et fermante doit respecter les règles suivantes **[84]**, **[95]** :

- Il doit être formé de caractères alphanumériques, du tiret-souligné (_), du signe moins (-) ou du point (.). Le caractère deux points (:) est utilisable mais il a un sens particulier qui sera décrit plus tard.
- Il ne doit pas contenir de caractère d'espacement ou de fin de ligne.
- Son premier caractère doit être alphabétique ou un tiret-souligné (_).
- Il ne peut pas commencer par la chaine « *xml* » qu'elle soit en minuscules, majuscules, ou un mélange des deux.
- Les majuscules et minuscules sont différenciées.

Un élément peut contenir d'autres éléments, des données, des références à des entités **[84]**. Il peut être vide c'est à dire il n'y a pas de contenu. Il peut contenir des attributs **[14]**. Le document XML ci-dessous contient :

- *cours* : élément racine contenant les éléments fils : *intervenant*, *separateur* et *chapitre* ;
- *enseignant* : élément contenant des données ;
- *separateur* : élément vide sans contenu ;
- *chapitre* : élément contenant des données et un élément fils *para* ;
- *para* : élément contenant des données.
- *renvoi* : élément vide avec attribut *cible* qu'on peut l'écrire aussi sous cette forme : <renvoi cible="I15.1"/>.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
   <enseignant> BOUSSAID </enseignant >
   <separateur/>
   <chapitre>
      Introduction XML
      <para>Un paragraphe</para>
   </chapitre>
   <renvoi cible="I15.1"></renvoi>
</cours>
```

5.3.2. Les attributs

Un attribut est un couple (nom, valeur) associé à un élément. Il est localisé dans la balise ouvrante de l'élément. Un élément peut donc avoir de 0 à n attributs **[14]**. Le nom d'un attribut obéit aux mêmes règles que les noms d'éléments **[84]**. Les valeurs d'attributs doivent toujours être présentes et entre guillemets.

Dans le document XML ci-dessous *nom* et *prenom* sont des attributs de l'élément *Enseignant* et *numero* est un attribut de l'élément *chapitre* :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
   <Enseignant nom="MADANI" prenom="Amina"/>
   <introduction/>
   <chapitre numero="1">
      Formation XML
      <paragraphe> Détails du format  </paragraphe>
   </chapitre>
</cours>
```

Un document XML peut être découpé en entités enregistrées dans un ou plusieurs fichiers. Une entité est un fragment nommé de document **[133]**. Dans la section suivante, nous allons voir plus de détails sur les entités.

6. Les entités

Les entités permettent aux documents XML de tirer parti de nombreuses ressources **[114]**. Elles permettent d'insérer du texte ou d'autres données de manière dynamique dans le document.

Les entités caractères servent à donner un nom facilement lisible à des caractères qui ne sont peut être pas représentables dans l'alphabet utilisé, ou qui ne sont pas disponibles au clavier.

Les entités non caractères peuvent appeler d'autres entités et peuvent provoquer leur inclusion dans le document XML.

Il existe trois classifications principales pour les entités : internes/externes, générales/paramètres et analysables/non analysables **[60]**.

6.1. Les entités caractères

Comme en HTML, il existe des entités prédéfinies pour certains caractères spéciaux par exemple **[84]** :

- < pour le caractère « < »
- > pour le caractère « > »
- & pour le caractère « & »
- " pour le caractère « " »
- ' pour le caractère « ' »

Un caractère non disponible sur une station de travail peut être représenté par son code Unicode[1] en décimal ou en hexadécimal, sous la forme d'une référence a une entité telle que « &# » est utilisée pour un code décimal et « &#x » pour un code hexadécimal. Par exemple :

- &#38 : référence au caractère *&*.
- &#x03A6 : référence à la lettre grecque Φ.

6.2. Les entités internes

Les entités internes sont définies dans le document XML et sont utilisées à l'intérieur du même document (entités générales). Elles permettent la définition d'un texte sous un nom. Leur définition se fait dans la déclaration de type de document **[84]**. Elle prend la forme suivante : <!ENTITY nom-de-l'entité "définition">.

Une entité générale est réutilisable dans un document XML par un simple appel à *&nom-de-l'entité* **[60]** comme c'est illustré dans l'exemple suivant :

```
<?xml version="1.0"?>
<!DOCTYPE exemple [
<!ELEMENT exemple (#PCDATA, important)>
<!ELEMENT important (#PCDATA)>
<!ENTITY reu "Une reunion de travail a midi">
<!ENTITY imp "<important>Attention!</important>"> ]>
<exemple> &reu; &imp; </exemple>
```

Les entités internes peuvent être définies dans la DTD et utilisée dans la DTD elle-même (entités paramètres) **[84]** par un simple appel de *%nom-de-l'entité* **[60]**. Sa syntaxe de définition est la suivante : <!ENTITY % nom-de-l'entité "définition">.

Dans l'exemple suivant l'entité paramètre est définie et est appelée dans la même DTD :

[1] La norme ISO 10646 en accord avec l'Unicode définit un jeu de caractères universel : l'UCS (*Universal Character Set*) qui permet de représenter les caractères de toutes les langues. Chaque caractère UCS est identifié par un code qui est un nombre représenté sur 4 octets.

```
<!ENTITY % sexes "(homme | femme)">
<!ATTLIST auteur sexe %sexes; #REQUIRED>
```

6.3. Les entités externes

Les entités externes rapatrient leur contenu à partir d'une autre ressource externe désignée par une URI **[60]**. Elles peuvent être également des entités générales ou des entités paramètres accessibles à partir de fichiers externes. Elles prennent la syntaxe suivante :

```
<!ENTITY nom-de-l'entité SYSTEM "URI">
```

Dans cet exemple, une entité paramètre externe est définie par :

```
<!ENTITY % mpeg SYSTEM "http://www.mpeg.com">
```

6.4. Les entités analysables

Le contenu d'une entité analysable est représenté par le nom de texte de remplacement, lequel fait partie intégrante du document. En conséquence, des données pouvant être correctement analysées par un processeur XML, formant un contenu XML bien formé, contribuent à rendre une entité analysable **[133]**. Un document bien formé est un document qui suit toutes les règles de notation et de structure du langage XML **[60]**.

6.5. Les entités non-analysables

Une entité non-analysable permet de déclarer un contenu non XML dans un document XML. La déclaration des entités nécessite de déclarer au préalable le format de l'entité et d'associer à ce format une application capable de traiter ce type. Une telle déclaration de format de fichier s'appelle une notation et utilise le mot-clef NOTATION **[84]**. Les entités non-analysables sont notamment des fichiers audio, vidéo ou images.

L'importation d'éléments ou d'attributs contenus dans des entités externes dans un document peut entraîner des conflits de noms. Ces conflits peuvent être évités en définissant des espaces de noms identifiés de façon unique et en associant à un nom l'espace dont il provient **[133]**.

7. Les espaces de noms

L'espace de noms est une collection des noms, identifiée par une référence d'URI **[133]**. Un premier usage des espaces de noms dans un document XML

consiste à utiliser simplement l'espace de noms par défaut. Ce dernier est précisé par un pseudo-attribut *xmlns* (*ns : namespace*). L'espace de noms par défaut s'applique à l'élément où se situe sa déclaration et à tout son contenu **[14]**.

Ci-dessous, l'élément *chapitre* est dans l'espace de noms http://www.monouvrage.dz. C'est également le cas de l'élément *paragraphe*, puisqu'il est dans l'élément *chapitre*.

```
<chapitre xmlns="http://www.monouvrage.dz">
  <paragraphe>
  XML
  </paragraphe>
</chapitre>
```

L'espace de noms par défaut peut être changé même dans les éléments enfants en utilisant une règle de priorité.

```
<chapitre xmlns=" http://www.monouvrage.dz ">
  <paragraphe xmlns="http://www.autrelivre.com">
  XML
  </paragraphe>
</chapitre>
```

L'élément *paragraphe* n'appartient pas à l'espace de noms http://www.monouvrage.dz mais uniquement à l'espace de noms http://www.autrelivre.com.

L'espace de noms par défaut présente l'inconvénient d'être peu contrôlable sur un document de taille importante. En effet, tout ajout ou modification d'un tel espace va se répercuter sur la totalité du contenu **[14]**.

Pour disposer de plus de souplesse dans ces espaces et pouvoir également les appliquer aux attributs et aux valeurs d'attributs, la syntaxe introduit la notion de préfixe. Celui-ci est une sorte de raccourci vers l'URI de l'espace de noms **[14]**.

On déclare un préfixe comme un pseudo-attribut commençant par *xmlns:prefixe*. Une fois déclaré, il est employable uniquement dans l'élément le déclarant et dans son contenu. L'emploi consiste à ajouter en tête de l'élément, de l'attribut ou d'une valeur d'attribut, le préfixe suivi des deux points « : » **[14]**.

Voici un élément *resultat* qui est dans l'espace de noms http://www.masociete.com grâce au préfixe *p*. <p:resultat xmlns:p="http://www.masociete.com"> </p:resultat>

La notion de préfixe n'a de sens que par rapport à l'URI associée. Dans la pratique, l'application ne voit pas ce préfixe mais uniquement l'URI associée à telle ou telle partie du document. C'est comme si un élément était un couple (nom de l'élément, espace de noms), de façon analogue à un attribut et sa valeur **[14]**.

Même qu'avec des préfixes différents, Les documents 1 et 2 sont strictement identiques ; *resultat* étant dans tous les cas un élément appartenant à l'espace de noms http://www.masociete.com.

Document 1 : <p:resultat xmlns:p="http://www.masociete.com"> </p:resultat>
Document 2 : <zz:resultat xmlns:zz="http://www.masociete.com"> </zz:resultat>

Grâce aux préfixes, on peut déclarer et utiliser plusieurs espaces de noms comme c'est le cas pour l'exemple suivant :

```
<p:res xmlns:p="http://www.masociete.com"
xmlns:p2="http://www.autresociete.com">
<p2:res>
</p2:res>
</p:res>
```

Le premier élément *res* est dans l'espace de noms http://www.masociete.com alors que l'élément *res* à l'intérieur est dans l'espace de noms http://www.autresociete.com.

Les espaces de nom peuvent s'appliquer via un préfixe sur un attribut ou une valeur d'attribut **[14]**.

```
<livre xmlns:p="http://www.imprimeur.com" p:quantite="p:50lots">
<papier type="p:A4"/>  </livre>
```

Aucun espace de noms n'est utilisé lorsqu'il n'y a pas d'espace de noms par défaut ni de préfixe.

Tout document XML doit être basé sur une DTD ou un schéma XML qui permettent de définir son propre langage. Dans ce qui suit, nous allons étudier les DTD et les schémas XML.

8. DTD

DTD (*Document Type Definition*), définition de type de document qui fournit la liste des éléments, des attributs, des notations et des entités que contient le document XML **[60]**.

Les DTD peuvent apparaitre dans le document qu'elles décrivent (DTD interne) ou bien dans un fichier séparé (DTD externe). Il est préférable de conserver la DTD dans une entité séparée pour qu'elle soit réutilisable et pourra être partagée par plusieurs documents XML différents **[60]**.

Dans un document XML, la DTD interne se déclare entre crochets dans le DOCTYPE après la déclaration XML comme c'est le cas dans l'exemple suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ouvrage [
<!ELEMENT ouvrage (titre, auteur)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (nom, prenom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)> ] >
```

La DTD externe est créée dans un fichier séparé comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ouvrage (titre, auteur)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (nom, prenom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
```

La DTD externe est appelée par le document XML en utilisant une déclaration de type de document (vue précédemment).

En plus des commentaires, une DTD peut contenir des déclarations d'entités générales, des déclarations d'entités paramètres, des déclarations de notations, des déclarations d'éléments et des déclarations de liste d'attributs **[84]**.

8.1. Déclaration d'élément

Une déclaration d'élément est de la forme suivante :

<!ELEMENT nom-de-l'élément modèle-de-contenu >

Elle contient le mot-clé ELEMENT qui s'écrit impérativement en majuscules, suivi du nom de l'élément et d'un modèle de contenu. Le nom d'élément obéit aux mêmes règles de déclaration du nom d'élément dans un arbre d'éléments. Le modèle de contenu peut contenir un élément vide, des éléments fils, un élément mixte ou un élément quelconque **[84]**.

8.1.1. Un élément vide

Il ne peut contenir que des attributs. Il peut être spécifié en utilisant le mot-clé EMPTY : <!ELEMENT Wilaya EMPTY >

8.1.2. Des éléments fils

Les éléments fils peuvent être un seul élément, une séquence de noms d'éléments séparés par des virgules dont l'ordre est important ou une liste de noms séparés par une barre verticale (|) représentant des choix mutuellement exclusifs.

<!ELEMENT Formation (Etudiant)>
<!ELEMENT Etudiant (code, nom, prenom, sexe, tél)>
<!ELEMENT Sexe (Homme | Femme)>

Le nombre d'occurrences autorisées (cardinalités) pour chaque élément peut être fixé par un indicateur d'occurrence, représenté par l'un des caractères suivants **[84]** :

- « ? » : indique que l'élément peut apparaitre zéro ou une fois dans le contenu d'une instance.
- « * » : indique que l'élément peut apparaitre zéro, une ou plusieurs fois dans le contenu d'une instance.
- « + » : indique que l'élément peut apparaitre une ou plusieurs fois dans le contenu d'une instance.

L'absence de ces caractères signifie que l'élément doit apparaitre une et une seule fois **[84]**.

L'élément *Formation* doit contenir un ou plusieurs éléments *Etudiant* et l'élément *Etudiant* doit avoir un élément *code*, un élément *nom*, un élément *sexe*, ainsi qu'un élément facultatif *tel* :

<!ELEMENT Formation (Etudiant+)>
<!ELEMENT Etudiant (code, nom, sexe, tel?)>

8.1.3. Un élément mixte

La forme la plus simple d'un élément mixte est un élément de type donnée. Elle est indiquée par le mot-clé #PCDATA (*Parsed Character Data*). Dans ce cas, l'élément ne peut contenir que des données et pas d'éléments fils. Les données sont constituées par un flot de caractères **[84]** : <!ELEMENT Enseignant (#PCDATA)>.

Un élément mixte peut mêler des données et des éléments **[84]** comme suit :

<!ELEMENT Enseignant (#PCDATA professeur | assistant)*>

8.1.4. Un élément quelconque

Avec ANY aucune contrainte n'est spécifiée sur le contenu de l'élément **[133]**. Tous les éléments déclarés dans la DTD sont utilisés dans n'importe quel ordre et sans limitation du nombre d'occurrences. L'élément quelconque est déclaré comme suit : <!ELEMENT Rapport ANY>

8.2. Déclaration de liste d'attributs

La déclaration d'attributs dans une DTD permet de spécifier les attributs qui devront être associés à une instance d'élément et d'indiquer la valeur par défaut d'un attribut **[84]**. Elle prend la forme suivante :

<!ATTLIST nom-élément nom-attribut type-attribut déclaration-de-défaut>

Le type d'attribut peut être **[60]** :

- CDATA : la valeur de l'attribut sera une chaine de caractères **[84]**.
  <!ATTLIST Etudiant code CDATA #REQUIRED>
- Enumération : offre un choix dans une liste de valeurs autorisées pour un attribut.
  <!ATTLIST Etudiant sexe (Homme | Femme) CDATA #IMPLIED>
- ID et IDREF : pour définir des renvois à l'intérieur des documents **[84]** : renvoi vers une référence bibliographique, vers une note, vers une section, etc. ID identifie de façon unique tous les éléments qui peuvent servir de cible à un renvoi. IDREF permet de créer une référence à un ID.

<!ELEMENT Etudiant (#PCDATA | xref)*>
<!ATTLIST Etudiant source ID #IMPLIED>
<!ELEMENT Enseignant EMPTY>
<!ATTLIST Enseignant ref IDREF #REQUIRED>

L'élément *Etudiant* contient un attribut nommé *source* qui contient une valeur ID. L'attribut *ref* de l'élément *Enseignant* contient une valeur qui pointe sur la valeur ID d'une occurrence de l'élément *Etudiant*.

- ENTITY ou ENTITIES : permettent qu'un attribut prenne comme valeur le nom d'une ou plusieurs entités externes non XML (une image par exemple) **[84]** précédemment déclarées dans la DTD dans une déclaration NOTATION.

<!NOTATION gif SYSTEM “image/gif”>
<!ELEMENT photo EMPTY>
<!ATTLIST photo type ENTITY #REQUIRED>

L'attribut *type* de l'élément *photo* contient un nom d'entité plutôt que du texte.

• NOTATION: l'attribut attend une notation qui apparaît dans la DTD dans une déclaration NOTATION.

<!NOTATION gif SYSTEM “image/gif”>
<!ELEMENT photo ANY>
<!ATTLIST photo type NOTATION (gif) #IMPLIED>

L’attribut *type* de l’élément *photo* contient un nom de notation.

• NMTOKEN ou NMTOKENS (Name token): unité lexicale nominale ou atome nominal qui permet à l’attribut de prendre comme valeur le nom d’un ou plusieurs « token » (noms XML quelconques formés de caractères alphanumériques) **[84]**.

<!ATTLIST Adresse State NMTOKEN # REQUIRED>

L’élément *Adresse* possède un attribut requis nommé *State*. Cet attribut contient un atome nominal.

La déclaration de défaut de liste d’attributs peut prendre l’une des formes suivantes **[60]** :

• ‘valeur’ : il faut indiquer la valeur par défaut de l’attribut.

• #REQUIRED : l’attribut est obligatoire dans chaque instance de l’élément déclaré.

• #IMPLIED : l’attribut est facultatif dans une instance de l’élément déclaré.

• #FIXED ’valeur’ : l’attribut ne peut prendre que la valeur mentionnée.

XML prédéfinit deux attributs qui doivent être déclarés dans la DTD pour chaque élément auquel elle s’applique **[60]** :

• *xml:lang* : il définit la langue de rédaction du contenu d’un élément. La valeur peut être CDATA, NMTOKEN ou une liste d’énumération.

<!ATTLIST UnElement xml:lang NMTOKEN 'en-US'>

• *xml:space* : il définit le traitement de l’espace dans l’élément. Dès qu’un élément contient des espaces qui doivent être préservés, la DTD doit comporter une déclaration pour l’attribut *xml:space* comme c’est mentionné ci-dessous. L’attribut est de type énumération et ne peut prendre que la valeur *default* (l'application traitant le document XML peut traiter les espaces comme elle le désire) ou la valeur *preserve* (l'application traitant le document XML doit tenir compte des espaces).

<!ATTLIST programme xml:space (default | preserve) 'preserve'>

8.3. Exemple d'une DTD

Nous montrons ci-dessous un exemple complet d'une DTD externe et une instance XML valide par rapport à cette DTD. Un document est valide **[84]** s'il est bien formé et s'il est conforme à ce qui est défini dans la DTD associée ou dans le schéma qui lui est associé.

La DTD « ouvrage.dtd » est la suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ouvrage (index, titre, auteur+)>
<!ELEMENT index EMPTY>
<!ATTLIST index valeur CDATA #REQUIRED>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (nom, prenom+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
```

Voici un document XML « ouvrage.xml » relatif à un ouvrage valide par rapport à la DTD « ouvrage.dtd » :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ouvrage SYSTEM "ouvrage.dtd">
<ouvrage>
  <index valeur="125"/>
  <titre>Modélisation XML</titre>
  <auteur>
    <nom>Antoine</nom>
    <prenom>Lonjon</prenom>
  </auteur>
  <auteur>
    <nom>Jean-Jacques</nom>
    <prenom>Thomasson</prenom>
  </auteur>
</ouvrage>
```

9. Les schémas XML

Comme c'est le cas pour les DTD, les schémas XML permettent de spécifier la syntaxe du contenu des documents XML, en définissant les éléments et les attributs ainsi que les contraintes sur les valeurs de ceux-ci **[133]**.

À la différence des DTD dont la syntaxe est différente de XML et le seul type atomique est le type chaîne de caractères, les schémas XML introduisent le typage des données, ce qui permet la gestion de booléens, d'entiers, d'intervalles de temps,

etc. Il est même possible de créer de nouveaux types à partir de types existants tout en utilisant une syntaxe XML **[22]**.

Alors que les DTD ne tiennent pas compte des espaces de noms, les schémas XML tirent parti des espaces de noms et les utilisent fréquemment. Il existe deux espaces de noms dédiés aux schémas **[114]** :

- http://www.w3.org/2001/XMLSchema : espace de noms utilisé pour les éléments du schéma XML du W3C, il peut être utilisé comme espace de nom par défaut ou avec préfixe *xsd* (par exemple <xsd:element>…</xsd:element>).
- http://www.w3.org/2001/XMLSchema-instance : espace de noms utilisé pour les extensions du schéma XML du W3C employées dans des documents d'instance, il doit être défini avec un préfixe *xsi*.

D'autres espaces de noms cibles peuvent être déclarés pour correspondre aux éléments et attributs déclarés dans le document XML.

Un schéma XML contient une déclaration XML, une déclaration de l'élément racine, des déclarations des éléments, des déclarations de types simples, des déclarations de types complexes et des déclarations des attributs **[133]**.

9.1. Déclaration de l'élément racine <schema>

Elle contient des informations sur les espaces de nom, les valeurs par défaut des éléments et des attributs et la version du vocabulaire. Par exemple :

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
attributeFormDefault=qualified  elementFormDefault=qualified version="1.0">
<!-- déclarations d'éléments, d'attributs et de types ici -->
</xsd:schema>
```

Pour indiquer si dans les documents instances, les noms d'éléments et d'attributs devront nécessairement être qualifiés ou pas par des espaces de nom, les attributs *elementFormDefault* et *attributeFormDefault* peuvent prendre les valeurs *qualified* ou *unqualified* **[84]**.

Une déclaration de type, d'éléments, ou d'attribut est dite globale lorsqu'elle est déclarée comme enfant de l'élément racine <schema>. Elle est locale lorsqu'elle est fille d'une autre déclaration.

9.2. Déclaration des éléments

On déclare un composant <element> dans un schéma XML comme suit :
<xsd:element name="*nom-de-l'élément*" type="type-de-c*ontenu-de-l'élément*">

Nous dressons ci-dessous un tableau contenant tous les attributs du composant <element> **[84]**, ils sont facultatifs :

Tableau 1.1 : Les attributs d'un composant <element>.

Attribut	Description
name	Le nom de l'élément.
type	Le type de contenu de l'élément.
ref	Référence d'un élément défini ailleurs dans le schéma. Si *name* apparait, *ref* ne l'est pas et vice versa.
form	Indique si dans le document XML le nom d'un élément instance est qualifié par un préfixe ou non. L'attribut prend comme valeur *qualified* ou *unqualified* (il remplace la valeur de l'attribut *elementFormDefault* utilisé dans <schema>).
minOccurs	Nombre minimal des occurrences de l'élément dans le document XML. (attribut non utilisé dans les déclarations globales). L'attribut prend comme valeur un entier positifs ou nul.
maxOccurs	Nombre maximal des occurrences de l'élément dans le document XML. (attribut non utilisé dans les déclarations globales). L'attribut prend comme valeur un entier positifs ou nul ou *unbounded* qui signifie que le nombre d'occurrences de l'élément n'est pas limité.
default	La valeur par défaut du contenu de l'élément.
fixed	Le contenu de l'élément doit être égal à une valeur fixe.
nullable	Indique si l'élément peut être vide. L'attribut prend comme valeur *true* ou *false*.
final	Indique qu'il est interdit de définir des types par restriction ou extension ou les deux. L'attribut prend comme valeur *restriction* ou *extension* ou *#all*.
block	Interdit d'utiliser dans des instances des types dérivés de celui défini par l'élément. L'attribut prend comme valeur *restriction* ou *extension* ou *#all*.

9.3. Déclaration des types simples <simpleType>

Comme le montre le tableau ci-dessous, il existe deux groupes différents de types de données prédéfinis : les types primitifs qui sont la base de tous les autres types de données et les types dérivés qui sont construits à partir des types de données primitifs **[114]**.

Tableau 1.2 : Les types de données prédéfinis.

Les types de données primitifs	Les types de données dérivés
String, boolean, float, double, decimal, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, hexBinary, base64Binary, anyURI, QName, NOTATION.	normalizedString, token , language , ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, Name, NCName, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, positiveInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte.

<simpleType> permet de faire des dérivations c.à.d. définir un nouveau type simple dérivé d'un autre type simple existant. Les principales catégories de dérivation sont **[22]** :

- La liste permet de définir un nouveau type à partir d'un type de base. Il est également possible de créer une liste personnalisée par dérivation de types existants. Par exemple une liste des pays :

```
<xsd:simpleType name="numéroDeTéléphone">
   <xsd:list itemType="xsd:unsignedByte" />
</xsd:simpleType>
```

- L'union permet de définir un nouveau type incluant toutes les valeurs possibles d'un certain nombre d'autres types. Par exemple, sous réserve que le type simple *numéroDeTéléphone* ait été préalablement défini, on peut déclarer un numéro de téléphone qui autorise un nombre ou une chaine de caractères :

```
<xsd:simpleType name="numéroDeTéléphoneMnémoTechnique">
  <xsd:union memberTypes="xsd:string numéroDeTéléphone" />
</xsd:simpleType>
```

- La restriction limite les valeurs possibles d'un type donné (type de base) et est définie sur certaines propriétés du type de base, appelées facettes. Le tableau 1.3 répertorie les facettes utilisables dans les déclarations <simpleType> **[84]**.

Un exemple d'un ensemble des altitudes des points de la terre avec une restriction (le plus haut sommet a une altitude de 8850m) :

```
<xsd:simpleType name="altitude">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="8850"/>
   </xsd:restriction>
</xsd:simpleType>
```

- Une extension consiste à définir un nouveau type à partir d'un type existant en lui ajoutant éventuellement des sous-éléments et/ou des attributs. Une extension produit donc toujours un type complexe. Par exemple :

```
<xsd:complexType name="AdressePays">
   <xsd:complexContent>
   <xsd:extension base="Adresse">
       <xsd:sequence>
           <xsd:element name="pays" type="string"/>
      </xsd:sequence>
     </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Tableau 1.3 : Quelques facettes utilisables dans les déclarations <simpleType> [84].

Facette	Description
maxInclusive	Valeur maximale inclusive.
minInclusive	Valeur minimale inclusive.
maxExclusive	Valeur maximale exclusive.
minExclusive	Valeur maximale exclusive.
precision	Elle permet de fixer le nombre de chiffres figurant dans un nombre et prend comme valeur un entier positif. Elle s'applique aux types simples ordonnés : decimal, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte et positiveInteger.
Scale	Elle permet de fixer le nombre maximal de chiffres fractionnaires figurant dans un nombre décimal et prend comme valeur un entier positif ou nul. Elle s'applique aux types simples ordonnés : decimal, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte et positiveInteger.
length	Elle permet d'indiquer les longueurs strictes des types : string, binary, uriReference, ID, IDREF, ENTITY, NOTATION, language, IDREFS, ENTITIES, NMTOKEN, NMTOKENS, Name, QName et NCName.
maxLength	Elle permet d'indiquer les longueurs maximales des types : string, binary, uriReference, ID, IDREF, ENTITY, NOTATION, language, IDREFS, ENTITIES, NMTOKEN, NMTOKENS, Name, QName et NCName.
minLength	Elle permet d'indiquer les longueurs minimales des types : string, binary, uriReference, ID, IDREF, ENTITY, NOTATION, language, IDREFS, ENTITIES, NMTOKEN, NMTOKENS, Name, QName et NCName.
Enumeration	Elle permet d'énumérer les valeurs possibles du type dérivé. Elle s'applique à tous les types simples prédéfinis sauf booléen.
Encoding	Elle prend l'une des valeurs de base16 ou base64. Elle s'applique au type simple binary.
Pattern	Elle s'applique à tous les types simples. Elle restreint le type simple sur la base d'une expression régulière.
fractionDigits	Nombre de chiffres à droite du symbole décimal dans un type numérique.

9.4. Déclaration des types complexes <complexeType>

Un type complexe se définit pour déclarer des compositions d'éléments et/ou d'attributs. Une composition d'éléments est réalisée en utilisant les trois constructeurs suivants **[84]** :

- *choice* : Un seul élément du modèle de contenu doit être présent. Le nombre d'occurrences de chaque alternative est déterminé par les attributs *minOccurs* et *maxOccurs* de la déclaration de chaque élément du modèle de contenu. Dans cet exemple, un choix est présent entre l'élément *fils* et l'élément *fille* :

```
<xsd: complexType name="EnfantsType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded" >
    <xsd:element name="fils" type="PersonneType" />
    <xsd:element name="fille" type="PersonneType" />
  </xsd:choice>
</xsd:complexType>
```

- *sequence* : Les éléments du modèle de contenu doivent être tous présents dans l'ordre indiqué. Le nombre d'occurrences de chacun peut être déterminé par les attributs *minOccurs* et *maxOccurs*. Par exemple, le fragment de code suivant requiert que les éléments apparaissent dans l'ordre *Prenom* puis *Nom*.

```
<xsd:complexType name="NomType">
  <xsd:sequence>
    <xsd:element name="Prenom" type="xsd:string" />
    <xsd:element name="Nom" type="xsd:string" minOccurs="0"  />
  </xsd:sequence>
</xsd:complexType>
```

- *call* : Les éléments du modèle de contenu doivent être tous présents au plus une fois, dans un ordre quelconque. Les attributs *minOccurs* et *maxOccurs* doivent valoir 0 ou 1.

```
<xsd:complexType name="NomType">
  <xsd:all>
    <xsd:element name="Prenom" type="xsd:string" />
    <xsd:element name="Nom" type="xsd:string" />
  </xsd:all>
</xsd:complexType>
```

Le contenu de <complexeType> peut être :

- Un élément à contenu vide : l'exemple ci-dessous présente un contenu composé uniquement d'attributs :

```
<xsd:complexType name="prix">
  <xsd:attribute name="valeur" type="xsd:decimal" />
</xsd:complexType>
```

La déclaration d'un élément de ce type est :

```
<xsd:element name="monnaie" type="prix"/>
```

- Un ou plusieurs éléments : un exemple de déclaration du type dont le contenu de l'élément est composé uniquement de sous éléments :

```
<xsd:complexType name="EnteteType">
  <xsd:sequence>
    <xsd:element name="titre" type="xsd:string" />
    <xsd:element name="auteur" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

Une déclaration d'un élément de ce type s'écrit comme suit :

```
<xsd:element name="entete" type="EnteteType" />
```

- Un élément à contenu simple : suite de caractères, sans sous-éléments et avec attributs :

```
<xsd:simpleType name="NombreDeuxDecimales">
  <xsd:restriction base="xsd:decimal">
    <xsd:fractionDigits value="2" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="PrixType">
  <xsd:simpleContent>
    <xsd:extension base="NombreDeuxDecimales">
      <xsd:attribute name="monnaie" type="xsd:string" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Voici une déclaration d'un élément de ce type :

```
<xsd:element name="prix" type="PrixType" />
```

- Un élément à contenu mixte : composé de caractères et de sous-éléments.

```
<xsd:complexType name="TexteType" mixed="true">
  <xsd:sequence>
    <xsd:element name="important" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

La déclaration d'un élément de ce type :

```
<xsd:element name="texte" type="TexteType" />
```

- Un groupe d'éléments :

```
<xsd:group name="TitreAuteur">
<xsd:sequence>
<xsd:element name="Titre" type="xs:string"/>
<xsd:element name="Auteur" type="xs:string"/>
</xsd:sequence>
</xsd:group>

<xsd:element name="livre">
<xsd:complexType>
<xsd:sequence>
```

```
<xsd:group ref="TitreAuteur"/>
<xsd:element ref="traduction"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

9.5. Déclaration des attributs <attribute>

La déclaration d'un attribut est très semblable à celle d'un élément **[114]**. Sa syntaxe est la suivante :

```
<xsd:attribute name="nom-attribut" type="type-attribut " use="utilisation-attribut" />
```

Pour illustrer la définition d'attributs, prenons l'exemple suivant où on a deux attributs *titre* et *auteur* :

```
<xsd:element name="cours" >
  <xsd:complexType>
    <xsd:attribute name="titre" type="xsd:string" />
    <xsd:attribute name="auteur" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>
```

Avec *attribute* les attributs les plus couramment utilisés sont : *name*, *form*, *type*, *ref*, *use*, *default* et *fixed*. L'attribut *use* peut prendre les valeurs suivantes **[114]** :

- *prohibited* : l'attribut ainsi défini ne doit pas apparaître.
- *optional* : l'attribut peut apparaître au plus une fois et c'est la valeur par défaut quand l'attribut *use* n'est pas utilisé.
- *required* : l'attribut doit obligatoirement apparaître.

9.6. Exemple d'un schéma XML

Nous montrons ci-dessous un exemple complet d'un schéma XML et une instance XML valide par rapport à ce schéma. Le schéma XML « recherche.xsd » est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
elementFormDefault="qualified">
        <xsd:complexType name="infolabType">
                <xsd:sequence>
                        <xsd:element name="labo"/>
                        <xsd:element name="name"/>
                        <xsd:element name="adrprof"/>
                        <xsd:element name="tel"/>
                        <xsd:element name="fax"/>
                        <xsd:element name="email"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="recherches">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="infolab" type="infolabType"/>
                                <xsd:element name="themerech" />
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

Voici un document XML « recherche.xml » valide par rapport au schéma « recherche.xsd » :

```
<?xml version="1.0" encoding="UTF-8"?>
<recherches  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="recherche.xsd">
        <infolab>
                <labo>LIBD</labo>
                <name>Laboratoire d'Inginerie en Bases de Données</name>
                <adrprof>Université de Bab Ezzouar USTHB, Alger</adrprof>
                <tel>+213 66 55 44 33</tel>
                <fax>+213 66 77 88 77</fax>
                <email>xy@usthb.dz</email>
        </infolab>
        <themerech>
                 <projet1>Bases de données avancées</projet1>
                 <projet2>Entrepôts de données</projet2>
                 <projet3>Data Mining</projet3>
                 <projet4>XML</projet4>
                 <projet5>Data Mining XML</projet5>
    </themerech>
</recherches>
```

Tout document XML conforme à une DTD ou un schéma XML doit suivre des règles de notation et de structure permettant sa validation. Ces règles sont décrites ci-dessous.

10. Un document bien formé, un document valide

Chaque document XML doit être *« bien formé »*. C'est simplement un document qui suit toutes les règles de notation et de structure du langage XML **[60]** :

- Il doit commencer par une déclaration XML ;
- Les éléments non vides ont une balise ouvrante et une balise fermante ;
- Les éléments vides ne comportant qu'une balise doivent la clore par /> ;
- Il contient un seul élément racine encapsulant tous les autres éléments et leurs attributs
- Il contient un ou plusieurs éléments ;
- Les éléments non vides sont correctement imbriqués ;
- Un nom d'attribut apparaît uniquement dans la balise ouvrante et une seule fois dans cette balise ;
- Les valeurs des attributs sont entre guillemets ou apostrophes ;
- Les caractères réservés sont remplacés par des références d'entités (par ex. < pour <) ;
- s'il n'y a pas de DTD, les seules entités utilisées sont celles réservées de XML (&, <, >, ' et ").

Un document est « valide » si **[84]** :

- Il est bien formé ;
- Il est conforme à ce qui est défini dans la DTD associée ou conforme au schéma qui lui est associé.

11. Conclusion

Un document XML n'est ni un langage de programmation (tels Basic, C, Prolog, Perl, Java, etc.), ni un format propriétaire (tel le format de Word, ou PDF), ni une base de données. C'est un fichier texte qui permet le stockage des informations structurées où les noms des éléments sont définis selon les besoins de chaque domaine d'application.

Les balises XML décrivent la structure et la sémantique du contenu et non son aspect visuel. Les détails expliquant comment afficher le contenu des balises sur un navigateur web sont fournis dans un autre document externe qui est la feuille de style associée au même document XML.

Un document XML peut être associé à une DTD ou un schéma XML qui permettent de définir son propre langage. Il doit être bien formé avec une description syntaxiquement correcte et il peut être valide si sa description est conforme à une DTD.

Avant d'étudier le domaine de *XML mining*, nous allons présenter dans le chapitre suivant les concepts de base du *data mining* puisque les différents travaux existants de *XML mining* sont basés sur les techniques de *data mining*.

CHAPITRE 2
DATA MINING : CONCEPTS ET TECHNIQUES

1. Introduction

Les données peuvent être stockées de manières diverses dans des bases de données relationnelles, dans un ou plusieurs entrepôts de données (*datawarehouse*)… Elles peuvent être récupérées à partir de sources plus ou moins structurées comme Internet (les documents XML…), ou encore en temps réel (appel à un *call center*, retrait d'argent dans un distributeur de billets...).

Avec la croissance exponentielle des données dans les entreprises, le besoin de rentabiliser la collection de ces données, en les transformant sous une forme directement utilisable par l'entreprise, apparait de plus en plus urgent. D'où la nécessité d'un ensemble de méthodes et de techniques d'analyse de données et d'extraction d'information et de connaissances pour combler ce besoin. L'ensemble de ces techniques et ces méthodes est appelé le *data mining*.

Dans ce chapitre, nous allons étudier les concepts de base du *data mining* : le processus du *data mining*, les tâches accomplies et les différents algorithmes et les différentes techniques. La dernière partie du chapitre est consacrée au *text mining* et au multimédia, l'objet de cette partie est de compléter le panorama du *data mining*, en montrant comment celui-ci peut s'unir à la linguistique pour permettre l'analyse et l'utilisation automatique de grandes masses de données textuelles et voir l'application du *data mining* sur les données multimédia.

2. *Data mining*

Fouille de données, exploration de données, extraction de connaissances à partir de données, *knowledge data discovery*…etc, plusieurs termes utilisés pour désigner le *data mining*. Pour le décrire, nous employons les définitions suivantes :

Le *data mining*, ou fouille de données, est l'ensemble des méthodes et des techniques destinées à l'exploration et l'analyse de grandes bases de données informatiques, de façon automatique ou semi-automatique, en vue de détecter dans

ces données des règles, des associations, des tendances inconnues ou cachées, des structures particulières restituant l'essentiel de l'information utile tout en réduisant la quantité de données permettant d'étayer des prises de décisions **[116]**.

C'est un domaine pluridisciplinaire qui regroupe des techniques d'apprentissage automatique, de la reconnaissance de forme, des statistiques, des bases de données et de la visualisation pour apporter une réponse à l'extraction d'information provenant de base de données de grande taille **[15]**.

3. Les données en *data mining*

Les données en *data mining* sont de différents types **[116]** :

- Les données quantitatives peuvent être continues ou discrètes. Les données continues sont celles dont les valeurs forment un sous ensemble infini de l'ensemble *R* des réels. Les données discrètes sont celles dont les valeurs forment un sous ensemble fini ou infini de l'ensemble *N* des entiers naturels.
- Les données qualitatives (ou de catégorie, ou catégorielles) sont des données dont l'ensemble des valeurs est fini. Ces valeurs sont numériques ou alphanumériques. Quand elles sont numériques, ceux sont des codes et non des quantités (exemple : la catégorie socioprofessionnelle).
- Les données textuelles ont pour valeurs des textes non codés, écrits en langage naturel. L'analyse de données textuelles nécessite un mélange de linguistique et de statistique : le *text mining*.

Tous les algorithmes n'admettent pas tous les types de données en entrée. Néanmoins, certaines opérations permettent de passer d'un type à un autre.

4. Le processus du *data mining*

Une démarche de *data mining* suit globalement un processus de huit phases **[75]** :

- Poser le problème : La première phase consiste à comprendre et a formaliser le problème que l'organisation cherche à résoudre en terme de données et à définir les résultats attendus.
- La recherche des données : Il s'agit de déterminer la structure générale des données ainsi que les règles générales pour les constituer.
- La sélection des données pertinentes : C'est une phase de collection et de sélection de la population cible.

- Le nettoyage des données : Nettoyer les données en faisant le prétraitement et la mise en forme des données. Il permet de corriger ou de contourner les inexactitudes ou les erreurs qui sont glissées dans les données.
- Les actions sur les variables : Il s'agit d'intervenir sur les variables (normalisation, conversion, transformation, réduction, …) pour faciliter leur exploitation par les outils de modélisation.
- La recherche du modèle : Appelée aussi phase de modélisation qui consiste à choisir un algorithme de *data mining* et extraire la connaissance des données.
- L'évaluation du résultat : Elle permet d'estimer la qualité du modèle et valider les connaissances extraites. La validation prend généralement une forme qualitative (graphique, textuelle…) ou quantitative (intervalle de mesure de confiance, indicateur de pertinence…).
- Intégration de la connaissance : Il s'agit de la transition du domaine des études au domaine opérationnel avec l'implantation du modèle ou des résultats dans les systèmes informatiques ou dans les processus de l'entreprise pour une utilisation effective.

Le processus présenté est itératif et plusieurs retours en arrière dans les différentes étapes sont nécessaires pour affiner les résultats.

5. Les tâches du *data mining*

Les principales tâches du *data mining* peuvent se résumer comme suit **[72]** :

5.1. La description

C'est un moyen pour décrire des phénomènes ou des tendances gisant dans les données. Les résultats du modèle doivent décrire des caractéristiques claires qui puissent amener à une interprétation et à une explication intuitive. Certaines méthodes de *data mining* sont plus adaptées que d'autres pour une interprétation transparente. Par exemple, les arbres de décision fournissent une interprétation et une explication intuitive de leurs résultats **[72]**.

5.2. La classification

La classification est l'opération qui permet de placer chaque individu de la population étudiée dans une classe, parmi plusieurs classes prédéfinies, en fonction des caractéristiques de l'individu indiquées comme variables explicatives. Le plus

souvent, il y a deux classes prédéfinies, et le résultat de la classification est un ensemble de règles permettant d'affecter chaque individu à l'une ou l'autre des classes, sachant que la probabilité de « bon classement » dépend généralement de la règle qui s'applique à l'individu : certaines sont plus fiables que d'autres **[116]**.

Dans la classification, la variable cible est catégorielle, comme le revenu, qui peut être divisé en trois classes : revenu élevé, revenu moyen et revenu faible **[72]**. Les techniques utilisées pour la classification sont les *k-plus proches voisins*, les arbres de décision et les réseaux de neurones **[72]**.

5.3. L'estimation

L'estimation est similaire à la classification excepté la variable cible, continue au lieu d'être catégorielle **[72]**. Par exemple estimer les résultats du baccalauréat en fonction des résultats du brevet des collèges pour une population des lycéens.

Le domaine de l'analyse statistique classique fournit plusieurs méthodes d'estimation efficaces et très largement utilisées. Les réseaux de neurones peuvent aussi être utilisés pour l'estimation **[72]**.

5.4. La prévision (prédiction)

Elle est similaire à la classification et à l'estimation mise à part que pour la prévision, les résultats portent sur le futur **[72]**. Prévoir les gagnants du championnat de football, par rapport à une comparaison des résultats des équipes est un exemple de prévision.

Les techniques les plus appropriées à la prévision sont les *k-plus proches voisins*, les arbres de décision et les réseaux de neurones **[72]**.

5.5. Le *clustering*

Le *clustering* porte sur le regroupement d'enregistrements, d'observations ou de cas en groupe d'objets similaires. Il est différent de la classification parce qu'ils n'y a pas de variable cible pour segmenter. Le *clustering* a pour objectif de segmenter l'ensemble entier des données en des sous groupes relativement homogènes, des clusters, dans lesquels la similarité des enregistrements dans le groupe est maximisée et la similarité en dehors du groupe est minimisée **[72]**.

5.6. L'association

L'association est la tâche qui permet de trouver quelles variables vont ensemble. Elle extrait les corrélations entre les données. Très répandue dans le secteur de la distribution car leur principale application est « l'analyse du panier de la ménagère » qui consiste en la recherche d'associations entre produits sur les tickets de caisse. Trouver tous les articles achetés ensemble et ceux qui ne sont jamais achetés ensemble dans un supermarché est un exemple de la fonction d'association **[72]**. L'algorithme des règles d'association est utilisé pour générer des règles d'association.

6. Les principaux techniques et algorithmes de *data mining*

Les méthodes de *data mining* peuvent être classées en méthodes d'apprentissage supervisé[1] et méthodes d'apprentissage non supervisé[2] **[19]**, **[72]**.

6.1. Apprentissage supervisé

L'objectif général de la classification est d'être capable d'étiqueter des données en leur associant une classe. Si les classes possibles sont connues et si les exemples sont fournis avec l'étiquette de leur classe, on parle d'apprentissage supervisé ou d'analyse discriminante. Dans ce cas, il s'agit alors d'utiliser les exemples fournis et déjà classés pour apprendre un modèle qui permette ensuite d'associer à tout nouvel exemple rencontré sa classe la plus adaptée **[19]**. Il est souvent recommandé de constituer **[98]** :

- Un ensemble d'apprentissage.
- Un ensemble de test.
- Un ensemble de validation.

Au moins deux ensembles sont nécessaires : l'ensemble d'apprentissage permet de générer le modèle et l'ensemble de test permet d'évaluer l'erreur réelle du modèle sur un ensemble indépendant. Lorsqu'il s'agit de tester plusieurs modèles et de les comparer, le meilleur modèle peut être sélectionné selon ses performances sur l'ensemble de test et ensuite son erreur réelle est évaluée sur l'ensemble de validation **[98]**.

[1] Plusieurs synonymes existent : classification supervisée, (anglicisme : classification). Dans notre mémoire, nous utilisant l'appellation anglicisme « classification » pour désigner la classification supervisée.

[2] Appelé aussi classification non supervisée, segmentation chez les auteurs francophones et *clustering* chez les anglo-saxons. Le mot *clustering* est utilisé dans ce mémoire.

Les méthodes d'apprentissage supervisé que nous allons aborder sont : les arbres de décision, les réseaux de neurones, les *k-plus proches voisins* et les réseaux bayésiens.

6.1.1. Les *k-plus proches voisins* (kPPV)

La méthode des *k-plus proches voisins* (*k-NN : k-Nearest Neighbors*), appelée aussi RBM (*Raisonnement Basé sur la Mémoire*) **[116]** est utilisée le plus souvent en classification, bien qu'elle puisse aussi l'être pour l'estimation et la prévision **[72]**. L'ensemble d'apprentissage est mémorisé, de façon qu'une classification pour un nouvel enregistrement non classé puisse être trouvée simplement en le comparant aux enregistrements les plus similaires de l'ensemble d'apprentissage **[72]**. Dans l'exemple qui suit, l'individu « ? » est classé en « 0 », car entouré en majorité de « 0 ».

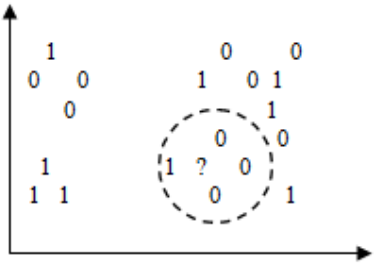


Figure 2.1 : Exemple de classification selon la méthode des *k-plus proches voisins*.

Le principe de l'algorithme des *k-plus proches voisins* **[28]** est de trouver, pour un nombre d'individus, les *k* plus proches voisins. Puis, combiner les classes de ces *k* exemples en une classe **[115]**.

Le choix de la distance est primordial au bon fonctionnement de la méthode afin de mesurer la similarité **[115]**. La fonction de distance la plus courante est la distance euclidienne, qui représente la manière habituelle par laquelle les humains pensent à la distance dans le monde réel **[72]** :

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (2.1)$$

Où $x = x_1, x_2, \ldots, x_m$ et $y = y_1, y_2, \ldots, y_m$ représentent les *m* valeurs des variables de deux enregistrements ou de deux individus.

La valeur de *k* est choisie de façon à obtenir la meilleure classification possible.

Le choix de la fonction de distance et de la valeur optimale de *k* est la principale difficulté de cet algorithme **[116]**.

Cet algorithme supporte bien les données hétérogènes, voire textuelles **[116]**, mais il est relativement lourd en terme de temps de calcul et d'utilisation d'espace mémoire si le nombre d'exemples est important **[98]**.

6.1.2. Les arbres de décision

Un arbre de décision est une représentation graphique d'une procédure de classification **[115]**. C'est un enchainement hiérarchique de règles logiques construites automatiquement à partir d'une base d'exemples. Un exemple est constitué d'une liste d'attributs, dont la valeur détermine l'appartenance à une classe donnée. La construction de l'arbre de décision consiste à utiliser les attributs pour subdiviser progressivement l'ensemble d'exemples en sous ensembles de plus en plus fins **[75]**.

Les nœuds internes de l'arbre sont des tests sur les champs ou attributs. Les feuilles sont les classes. Lorsque les tests sont binaires, le fils gauche correspond à une réponse positive au test et le fils droit à une réponse négative **[115]**.

Une règle logique comprend une prémisse (la première partie de la règle) et une conclusion (la seconde partie de la règle). La prémisse exprime une condition logique bâtie sur des tests portant sur des variables combinées par des opérateurs logiques (et, ou, non) **[75]**.

Un exemple d'arbre de décision est la population constituée d'un ensemble de patients où il y a deux classes : malade et bien portant. Les descriptions sont faites avec les deux attributs température (à valeurs décimales) et gorge irritée (attribut logique) :

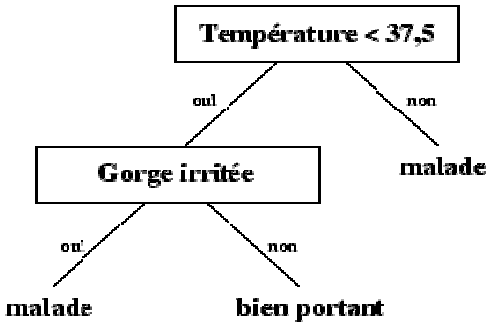


Figure 2.2 : Un exemple d'arbre de décision.

La traduction de cet arbre en règles de décision est :

- SI Température<37,5 ET gorge irritée ALORS malade.
- SI Température<37,5 ET NON(gorge irritée) ALORS bien portant.
- SI NON(Température<37,5) ALORS malade.

Pour choisir le test (le nœud), des fonctions qui mesurent le « degré de mélange » des différentes classes sont utilisées **[115]** :

- La fonction de *Gini* : $Gini(x) = 4x(1-x)$ (2.2)
- La fonction Entropie : $Entropie(x) = -x \log x - (1-x) \log (1-x)$ (2.3)

Où x désigne la proportion d'éléments dans l'une des deux classes. Ces deux fonctions sont à valeurs dans l'intervalle réel [0,1], prennent leur minimum pour x=0 ou x=1 (tous les exemples ou individus sont dans une même classe) et leur maximum lorsque x=1/2 (les exemples sont également répartis entre les deux classes) **[115]**.

Les algorithmes à base d'arbres de décision les plus importants sont **[116]** :

- CART (*Classification And Regression Trees*) **[13]** : les arbres de décision produits par CART sont strictement binaires, contenant deux branches pour chaque nœud de décision **[72]**. La fonction qui mesure le gain est la fonction de *Gini*. Pour l'élagage, un parcours ascendant de l'arbre construit est effectué. Pour décider si un sous arbre peut être élagué, il faut calculer l'erreur réelle estimée de l'arbre courant avec l'arbre élagué. L'estimation de l'erreur réelle est mesurée sur un ensemble de test **[115]**.

- C5 développée par Quinlan est une version plus récente qui améliore les versions ID3 **[99]** et C4.5 **[100]**. L'algorithme peut prendre en compte des attributs d'arité quelconque. La fonction qui mesure le gain est la fonction *entropie*. Cette fonction a tendance à privilégier les attributs possédant un grand nombre de valeurs. Pour éviter ce biais, une fonction gain d'information est également disponible. L'élagage est effectué avec l'ensemble d'apprentissage par une évaluation pessimiste de l'erreur. Bien que cette technique puisse sembler inadaptée, elle donne de bons résultats en pratique **[115]**.

Les algorithmes précédents semblent spécialisés pour les attributs discrets. Mais, ils ont été adaptés pour considérer également les attributs continus **[115]**.

Les arbres de décision possèdent l'avantage d'être compréhensibles mais sur un gros jeu de données, l'arbre devient gros et illisible **[98]**. En plus, l'algorithme n'est pas incrémental, c'est-à-dire, que si les données évoluent avec le temps, il est nécessaire de relancer une phase d'apprentissage sur l'échantillon complet (anciens exemples et nouveaux exemples) **[115]**.

6.1.3. Les réseaux de neurones

Les réseaux de neurones sont très utilisés pour la classification, l'estimation, la prédiction et le *clustering* **[115]**. Un réseau de neurones (ou réseau neuronal) a une architecture calquée sur celle du cerveau, organisée en neurones et synapses et se présente comme un ensemble de nœuds (ou neurones formels, ou unités) connectés entre eux [116].

Un neurone combine des entrées réelles x_1,..., x_n en une sortie réelle o. Les entrées n'ont pas toutes la même importance et à chaque entrée x_i est associé un poids (ou coefficient synaptique) w_i. D'abord, l'activité d'entrée est calculée. En règle générale, pour le neurone formel, l'activité en entrée est mesurée par la somme pondérée des entrées $\sum_i w_i x_i$ **[115]**.

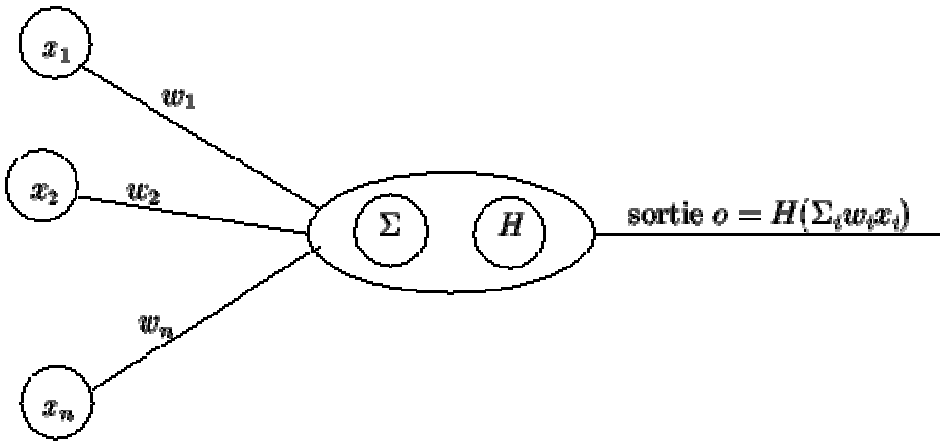


Figure 2.3 : Un neurone ou perceptron [115].

Pour le neurone biologique, lorsque l'activité en entrée dépasse un certain seuil, sa sortie est activée. Pour le neurone formel, une fonction de transfert H est appliquée à l'activité d'entrée. Cette fonction doit être telle que sa valeur de sortie soit 1 lorsque l'activité est suffisante et 0 sinon (parfois -1). Un premier choix possible est la fonction de *Heaviside* définie par $H(x) = 1$ si $x > 0$ et $H(x) = 0$ sinon. Plus souvent, des approximations dérivables de cette fonction sont utilisées, par exemple la fonction *sigmoïde* **[115]** :

$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}} \qquad (2.4)$$

Cette fonction prend ses valeurs dans l'intervalle [0,1], passe de 0 à 1 lorsque l'entrée est suffisante, tout en étant continue et dérivable.

Les réseaux de neurones traitent facilement les données réelles et les algorithmes sont robustes au bruit. Ce sont, par conséquent, des outils bien adaptés pour le traitement de données complexes éventuellement bruitées [1] comme la reconnaissance de formes (son, images sur une rétine, ...etc.) **[115]**.

Il existe différents modèles de réseaux de neurones [116] : les principaux, le perceptron multicouche (PMC), le réseau à fonction radiale RBF (*Radial Basis Function*) qui sont utilisés pour la classification et le réseau de Kohonen. Nous nous limitons dans ce chapitre aux réseaux de Kohonen qui sont utilisés pour le *clustering*.

6.1.4. Les réseaux bayésiens

Un réseau bayésien **[93]** est un graphe causal, orienté, acyclique dont les nœuds représentent des variables et les arcs orientés symbolisent des dépendances entre les variables **[75]**. Chaque nœud ne peut être alimenté que par les variables qui le précédent dans le graphe. La fréquence d'apparition d'une variable est mesurée par sa probabilité **[75]**. Le réseau bayésien permet de simplifier le calcul de la probabilité jointe en exprimant les indépendances conditionnelles entre les variables **[37]**. La figure suivante illustre un réseau bayésien (structure et tables de probabilités conditionnelles) :

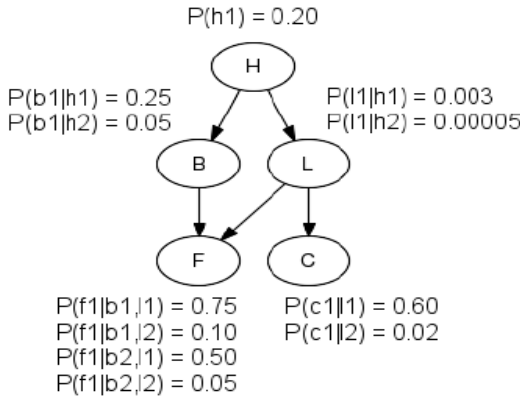


Figure 2.4 : Exemple de réseau bayésien.

[1] Ce sont des données dont certains attributs ont une valeur inconnue ou invalide **[98]**.

L'utilisation d'un réseau bayésien passe par deux étapes **[7]** :

• Construction du réseau bayésien : la détermination d'un réseau bayésien se déroule en quatre phases **[75]** :

- Création et préparation des variables représentant l'univers.

- Sélection des variables d'entrée et des variables de sortie et définition de l'ensemble des valeurs possibles pour chaque variable.

- Définition des relations de dépendances entre les variables.

-Attribution des probabilités conditionnelles en construisant une matrice des probabilités par un comptage des occurrences entre les différents nœuds.

Lors de la construction des réseaux bayésiens, il n'est pas toujours évident qu'un expert puisse fournir de façon numérique l'ensemble des paramètres nécessaires. Il est intéressant dans certains cas de déterminer ces paramètres à partir d'une base d'exemples. C'est grâce à la méthode d'apprentissage qui permet de remédier à ce problème **[29]**. Les deux types d'apprentissage utilisés sont **[7]** :

- Apprentissage de structure : qui permet de créer le graphe causal grâce à la détermination des variables d'intérêt du domaine, c.-à-d. les variables représentatives et les liens de causalité entre elles.

- Apprentissage des paramètres : qui permet de calculer les distributions locales de probabilités pour chaque nœud du graphe causal.

• Utilisation du réseau bayésien (inférence) : une fois qu'un réseau bayésien a été construit pour rendre compte d'un domaine, on utilise pour déterminer des probabilités qui correspondent à certains événements, certaines questions et certaines dépendances **[27]**. Dans ces cas, les inférences permettent effectivement de calculer la probabilité de chaque nœud lorsque les autres variables sont inconnues **[110]**. Il existe cependant quelques méthodes permettant d'effectuer ce calcul telles que : le conditionnement global, l'arbre de jonction et la méthode approchée **[29]**.

Les réseaux bayésiens sont appliqués pour le diagnostic. Ils peuvent déterminer les causes les plus probables ayant entrainé un problème dans un système en connaissant la panne. Toutefois, les réseaux bayésiens sont aussi utilisés pour faire

de la classification. Ils se basent sur un certain nombre de caractéristiques des textes pour pouvoir les classer dans une catégorie **[7]**.

Les réseaux bayésiens recherchent le meilleur graphe de connexion entre les variables. Ils apportent donc à l'utilisateur une connaissance des variables pertinentes et des liens qui unissent ces variables. Ils sont particulièrement recommandés pour faire face à des données manquantes ou bruitées. Néanmoins, la recherche du meilleur réseau est très consommatrice en puissance de calcul **[75]**.

6.2. Apprentissage non supervisé

Inversement, dans les méthodes non supervisées, aucune des données ou des variables à disposition n'a d'importance particulière par rapport aux autres. C'est à dire aucune variable cible n'est définie explicitement. C'est plutôt l'algorithme de *data mining* qui cherche à extraire des informations nouvelles et originales au sein de toutes les variables **[72]**. Il identifie des groupes tels que les exemples les plus similaires appartiennent au même groupe et les exemples les plus différents soient séparés dans différents groupes. La notion de similarité étant le plus souvent ramenée à une fonction de distance entre pairs d'exemples **[19]**.

Les techniques de *clustering* et les techniques de recherche d'association sont utilisées pour l'apprentissage non supervisé **[72]**.

Deux grandes familles de *clustering* sont distinguées **[16]** :

- Le *clustering* hiérarchique : dont le but est de former une hiérarchie de clusters telle que, plus on descend dans la hiérarchie, plus les clusters sont spécifiques à un certain nombre d'objets considérés comme similaires **[16]**. Le *clustering* hiérarchique se compose en deux méthodes **[16]**, **[19]**, **[66]** :

- Les méthodes ascendantes (ou agglomératives **[66]**) qui démarrent avec autant de clusters que d'objets puis fusionnent successivement l'ensemble des clusters jusqu'à ce qu'un critère d'arrêt soit satisfait.

- Les méthodes descendantes, qui démarrent avec un cluster unique regroupant l'ensemble des objets, puis divisent successivement les clusters selon un certain critère jusqu'à ce que tous les objets se retrouvent dans les clusters différents stockés aux feuilles de la hiérarchie.

Pour le *clustering* hiérarchique, plusieurs méthodes existent telles que : la méthode de **[123]**, BIRCH **[130]**, CURE **[54]**, ROCK **[55]**,...etc.

- Le *clustering* par partition : consiste à former une partition de l'espace des objets. Chaque partition représente un cluster. Dans cette famille, plusieurs méthodes se distinguent fortement **[16]**, **[19]**, **[66]** :

- Le *clustering* basé sur *k-means* **[82]** : le but est de définir *k* le nombre de clusters, auxquels sont ensuite associés l'ensemble des objets, selon leur proximités avec les points représentatifs considérés.

- Le *clustering* basé sur la densité : où il faut identifier dans l'espace, les zones de forte densité entourées par des zones de faible densité, qui formeront des clusters. Parmi ces techniques, on trouve : DBSCAN **[43]**, **[44]**, DENCLUE **[44]**...etc.

- Le *clustering* basé sur l'utilisation de grilles : l'idée est d'utiliser une grille pour partitionner l'espace en un ensemble de cellules, puis d'identifier les ensembles de cellules denses connectées pour former les clusters. Il est utilisé essentiellement pour des données spatiales **[66]**. Quelques exemples d'algorithmes : Wave-Cluster **[109]**, STING **[59]**, **[121]**, **[122]**.

- Le *clustering* statistique : qui fait l'hypothèse que les données ont été générées en suivant une certaine loi de distribution (avec une certaine probabilité), le but est alors de trouver les paramètres (cachés) de cette distribution.

- Le *clustering* via la théorie des graphes : consiste à former un graphe connectant les objets entre eux. La somme des valeurs des arcs correspondant aux distances entre les objets, est minimale. Les arcs de valeurs maximales sont ensuite supprimés pour former les clusters.

- Le *clustering* basé sur la recherche stochastique : qui parcoure l'espace des partitions possibles selon différentes heuristiques. Il sélectionne la meilleure qu'il trouve dans le temps qui leur est imparti, par exemple les algorithmes génétiques.

- Le *clustering* basé sur les réseaux de neurones, appelés « auto-adaptatifs » et qui utilise un réseau à une seule couche, duplique en sortie les données fournies en entrée et cherche à utiliser le moins de neurones possibles pour relier l'entrée et la sortie.

Les réseaux de Kohonen, *k-means* et les règles d'association sont les méthodes non supervisées que nous allons étudier.

6.2.1. Les réseaux de Kohonen

Les réseaux de Kohonen (ou cartes de Kohonen) représentent un type de carte auto-organisatrice (*SOM : Self Organisation Map*). Les cartes de Kohonen sont issues des travaux du chercheur T. Kohonen en 1982 **[70]**, qui a mis au point l'algorithme qui porte aujourd'hui son nom. Cet algorithme cherche à reproduire le processus de traitement des informations du cerveau **[75]**. Les réseaux de Kohonen sont destinés au *clustering*, ils permettent de déterminer automatiquement le nombre optimum de clusters au lieu de le fixer à priori **[116]**.

Le but des cartes auto-organisatrices est de convertir un signal d'entrée complexe avec un nombre élevé de dimensions en une carte discrétisée avec un faible nombre de dimensions **[61]**. Elles structurent les nœuds en sortie en classes de nœuds, dans lesquelles les nœuds de plus proche proximité sont plus similaires entre eux que les nœuds plus éloignés **[72]**. Une architecture typique de la carte auto-organisatrice est présentée dans la figure ci-dessous :

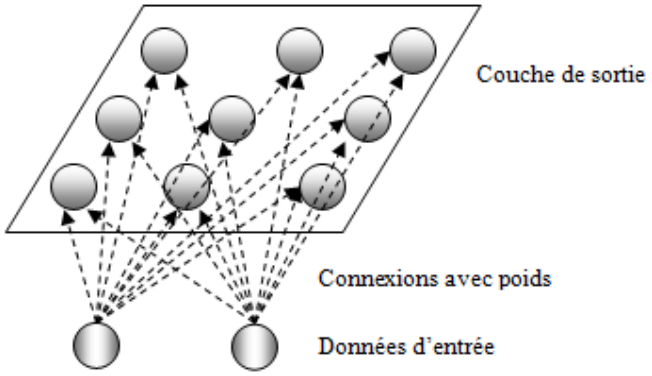


Figure 2.5 : Topologie d'une carte auto-organisatrice [72].

Les cartes auto-organisatrices sont basées sur un apprentissage compétitifs, où les nœuds en sortie sont en compétition entre eux pour être le nœud gagnant (ou le neurone), le seul nœud à être activé par une observation en entrée donnée **[72]**.

Les cartes auto-organisatrices n'ont pas de couches cachées. Les données à partir de la couche d'entrée passent directement vers la couche de sortie. Cette dernière est représentée sous la forme d'un treillis, en général d'une ou deux dimensions sous la forme d'un rectangle, hexagones…etc. (voir figure 2.5). Les cartes auto-organisatrices présentent trois processus caractéristiques **[72]** :

- La compétition : les nœuds en sortie se disputent entre eux pour produire la meilleure valeur pour une fonction de score donnée, généralement une fonction de distance euclidienne. Le nœud en sortie qui a la plus petite distance euclidienne entre les données d'entrée et les poids de connexion sera déclaré gagnant.

- La coopération : les nœuds voisins du nœud gagnant partagent des caractéristiques en commun dues au voisinage du nœud gagnant.

- L'adaptation : les nœuds voisins du nœud gagnant participent à l'adaptation (l'apprentissage). Les poids de ces nœuds sont ajustés de façon à améliorer la fonction de score.

Le principe de l'algorithme de Kohonen est que pour chaque vecteur d'entrée *x*, faire **[72]** :

- La compétition : pour chacun des nœuds en entrée *j*, calculer la valeur de $D(w_j, x_n)$ de la fonction de score. Par exemple, pour la distance euclidienne, $D(w_j, x_n) = \sqrt{\sum_i (w_{ij} - x_{ni})^2}$. Trouver le nœud gagnant *J* qui minimise $D(w_j, x_n)$ parmi tous les nœuds de sortie.

- La coopération : identifier tous les nœuds en sortie *j* dans le voisinage de *j* défini par le voisinage de taille *R*. Pour ces nœuds, faire le calcul suivant (adaptation, ajuster les poids) sur toutes les variables en entrée :

$$w_{ij,\ nouveau} = w_{ij,\ en\ cours} + \eta(x_{ni} - w_{ij,\ en\ cours}).$$

- Ajuster le taux d'apprentissage et la taille du voisinage, comme souhaité.

- Arrêter quand les critères de terminaison sont validés.

Les cartes de Kohonen disposent de capacités d'apprentissage automatique, d'un bon pouvoir illustratif des résultats et d'une forte sensibilité aux données fréquentes. Cependant, elles présentent certaines limites : la gestion des paramètres tels que la taille de la grille, les valeurs initiales des poids et le voisinage relèvent généralement d'une certaine expertise. D'autre part, l'interprétation des groupes constitués nécessite souvent le recours aux techniques statistiques pour comprendre les facteurs caractérisant chacun des groupes. De plus, ce système nécessite un grand ensemble d'apprentissage **[23]**.

6.2.2. Centres mobiles, *k-means* et nuées dynamiques

Pour résoudre certains problèmes complexes, il peut s'avérer utile de commencer par segmenter la population (la diviser en clusters) en espérant que le problème soit alors plus simple à résoudre sur les groupes ainsi constitués. Après application de l'algorithme et lorsque les clusters ont été construits, d'autres techniques ou une expertise, doivent dégager leur signification et leur éventuel intérêt. Une de ces techniques est la méthode des centres mobiles qui est très simple à mettre en œuvre et très utilisée **[115]**.

Les étapes de la méthode sont les suivantes **[116]** :

- Étape 1 : choisir k individus $c_1,..., c_k$;
- Étape 2 : regrouper les autres individus autour des centres définis à l'étape 1, de sorte que le groupe de c_i soit constitué des individus plus proches[1] de c_i que tout autre centre ;
- Étape 3 : remplacer les individus de l'étape 1 par les barycentres des groupes définis à l'étape 2 ;
- Étape 4 : répéter l'étape 2 ;
- …
- Étape *w* : s'arrêter quand l'inertie intraclasse, qui décroit d'une partition à la suivante, ne diminue plus sensiblement (en général $w \leq 10$).

La méthode des centres mobiles a des variantes **[116]** :

- *k-means* : dans la variante *k-means*, le barycentre de chaque groupe est recalculé à chaque nouvel individu introduit dans le groupe, au lieu d'attendre l'affectation de tous les individus et l'étape 3 avant de calculer les barycentres.
- *Nuées dynamiques* : elle se distingue principalement de celle des centres mobiles par le fait que chaque classe n'est plus présentée par son barycentre (éventuellement extérieur à la population). Elle est représentée par un sous ensemble de la classe, appelé noyau, qui s'il est bien composé (des individus les plus centraux, par exemple), sera plus représentatif de la classe que son barycentre.

[1] Selon la distance euclidienne ou une autre distance.

Dans ces algorithmes, la partition finale dépend beaucoup du choix initial (arbitraire) des centres c_i. On n'a donc pas un optimum global, mais seulement la meilleure partition possible à partir de celle de départ **[116].**

La méthode est sensible au choix des bons paramètres en particulier, le choix du nombre *k* de groupes à constituer. Un mauvais choix de *k* produit de mauvais résultats **[115]**. Il est possible de pallier à cet inconvénient en testant diverses valeurs de *k*, mais cela augmente la durée du traitement **[116]**.

6.2.3. Les associations

La recherche d'association vise à construire un modèle fondé sur des règles conditionnelles à partir d'un fichier de données. Une règle conditionnelle se définit sous la forme d'une suite « *si conditions alors résultat* » **[75]**.

Voici des exemples de règles d'association portant sur l'analyse du panier de la ménagère qui consiste en la recherche d'associations entre produits sur les tickets de caisse :

- *Si* un client achète des plantes *alors* il achète du terreau,
- *Si* un client achète une télévision *alors* il achètera un magnétoscope dans un an.

Une association s'apprécie au travers de deux indicateurs **[116]** :

- L'indice de support est la probabilité : $\text{proba (condition et résultat)}$
- L'indice de confiance est la probabilité : $\dfrac{\text{proba (condition et résultat)}}{\text{proba (condition)}}$

Par exemple dans le tableau ci-dessous, l'indice de confiance de l'association $C \Rightarrow B$ (*si C alors B*) est égal à 2/3 (le rapport entre le nombre de tickets contenant tous les articles figurant dans la règle et le nombre des tickets contenants les articles de la partie condition). Son indice de support est 2/5 (la fréquence d'apparition simultanée des produits dans l'ensemble des tickets). Donc, dans 66% des cas les clients qui achètent le produit *C* achètent aussi le produit *B* et cette règle apparait dans 40% de transactions.

Tableau 2.1 : Ensemble de tickets de caisse.

Tickets	Produits				
T1	*A*	*B*	*C*	*D*	*E*
T2	*B*	*C*	*E*	*F*	
T3	*B*	*E*			
T4	*A*	*B*	*D*		
T5	*C*	*D*			

L'objectif est de détecter les associations qui présentent un niveau de confiance et un niveau de support au dessus d'un seuil précisé par l'utilisateur **[75]**. La recherche de règles d'association dans un grand ensemble de données est un processus en deux étapes **[72]** :

1. Trouver tous les ensembles d'articles fréquents (*itemsets* fréquents) : tous les ensembles avec une fréquence d'articles supérieure à la moyenne.

2. À partir des ensembles d'articles fréquents, générer des règles d'association qui satisfassent les conditions de support et de seuil de confiance minimum.

A-Priori **[3]** est le premier algorithme proposé qui détermine les règles d'association présentes dans un jeu de données, pour un seuil de support et un seuil de confiance fixés. Il fonctionne en deux phases **[98]** :

- Construction des ensembles d'items fréquents (EIF) : la construction des EIF est itérative : construire les EIF contenant un seul item (1-item, item désigne un élément ou un article), puis ceux contenant 2 items (2-items), puis ceux contenant 3 items (3-items), ...etc. L'idée est donc de ne retenir que les items dont le support est supérieur au seuil fixé. L'algorithme *A-priori* qui construit tous les EIF est le suivant **[98]** :

```
Nécessite: un support seuil s
  L1 ← liste des items dont le support est > s
  i ← 1
  répéter
    i ++
    à partir de Li-1, déterminer l'ensemble Ci des EIF candidats comprenant i items.
    Li ← Ø
    pour tout élément e ∈ Ci faire
      si support (e) > seuil alors
        ajouter e à Li
      fin si
    fin pour
  jusqu'à Li ≠Ø
```

D'abord il faut déterminer l'ensemble L_1 des EIF de taille *1*. Ensuite, construire l'ensemble C_2 des EIF candidats de taille *2*, ce sont tous les couples construits à partir des EIF de taille *1*. Construire L_2 la liste des EIF de taille *2* en ne conservant que les éléments de C_2 dont le support est supérieur au seuil. Construire alors C_3, l'ensemble des triplets d'items dont les *3* sous paires sont dans L_2 et en ne retenant que ceux dont le support est supérieur au seuil, ce qui produit L_3. Et ainsi de suite, tant que L_i n'est pas vide.

- Génération des règles d'association à partir des EIF : c'est la phase de transformation des EIF en règles. Avec les notations précédentes, on dispose d'un ensemble de L_i pour des *i* croissant de 1 à une certaine valeur, chaque L_i étant une liste de *i-items* dont la fréquence d'apparition est supérieure à un certain seuil **[98]**. Supposons que L_3 contienne le triplet *(a, b, c)*. Plusieurs règles d'association peuvent être engendrées par ce triplet :

  - Si *a* et *b* alors *c*.
  - Si *a* alors *b* et *c*.
  - Si *b* et *c* alors *a*.
  - Si *b* alors *a* et *c*.
  - Si *a* et *c* alors *b*.
  - Si *c* alors *a* et *b*.

Parmi ces 6 règles candidates celles dont la confiance est supérieure au seuil fixée seront retenues. Si l'énumération des règles candidates est possible sur ce petit exemple, il faut bien être conscient que dans une application réelle cela est impossible : à nouveau, le nombre de règles candidates est beaucoup trop grand. Il faut donc une méthode plus judicieuse **[98]**.

Les règles d'association produites par la méthode permettent de donner des résultats clairs. Cependant, il faut noter que la méthode peut aussi produire des règles triviales (déjà bien connues des intervenants du domaine). Elle produit énormément de règles parmi lesquelles peu sont véritablement intéressantes **[98]**.

7. Text mining

Le *text mining* est l'ensemble des techniques et des méthodes destinées au traitement automatique de données textuelles en langage naturel, disponibles sous forme informatique, en assez grande quantité, en vue d'en dégager et de structurer le contenu et les thèmes dans une perspective d'analyse rapide, de découverte d'informations cachées ou de prise automatique de décision **[116]**.

Le processus du *text mining* se décompose en trois phases successives **[7]** :

• Traitement linguistique : Une analyse en *text mining* nécessite, tout comme une analyse de *data mining*, une phase de préparation des données avant l'analyse du texte proprement dit **[75]**. L'objectif de cette phase est de prétraiter les données pour pouvoir les manipuler par la suite. Le traitement linguistique se fait comme suit **[75]**, **[116]** :

- L'élimination des mots vides : il s'agit des articles (un, le, …), des adjectifs (ses, son, …), des démonstratifs (ces, ce, …) et des conjonctions (à, …).

- Lemmatisation : ramener les mots à leur forme canonique (substantifs ramenés au singulier, adjectifs ramenés au masculin et flexions des verbes ramenées à l'infinitif).

- Regroupement des variantes : recenser les variantes graphiques (clef=clé), les variantes syntaxiques, les variantes sémantiques, les synonymes, les parasynonymes (mots de sens voisins) et les développements de sigles.

- La recherche des mots signifiants : un opérateur humain doit donc dresser une liste de mots ou d'expressions pour intégrer les spécificités de la langue et tenir compte du contexte de l'étude.

- Désambiguïsation : de la polysémie (plusieurs significations) des mots, des ellipses, des homographes, des antiphrases, de l'ironie et des anaphores (la reprise d'un mot par un autre mot qui y renvoie le plus souvent un pronom : il, lui…).

• Lexicométrie : Elle mesure la fréquence d'apparition des mots, des expressions ou encore des co-occurrences[1] de certains mots au sein d'un même texte. Cette étape repose sur des techniques reconnues issues du monde de la recherche documentaire et qui possèdent certaines similarités avec le *text mining*. La recherche documentaire doit mesurer la similarité entre une requête et l'ensemble des documents disponibles. Quant au *text mining*, il doit être en mesure de déterminer la similarité entre un document et un ensemble de catégories de documents de manière à pouvoir le classer. Il existe deux grands types de représentation issus de la recherche documentaire **[7]** :

[1] Une occurrence désigne un élément linguistique ou un mot toutes les fois qu'il apparait dans un texte. Les mots ou éléments linguistiques qui figurent en même temps aux cotés de l'occurrence dans le texte sont les co-occurrences (ou sont dits ses co-occurrents) **[7]**.

- Le modèle vectoriel : procède à la transformation d'un texte en un vecteur. La première étape consiste à indexer le document puis à compter le nombre d'occurrences des termes d'index présents dans le document. On obtient ainsi un vecteur de profil d'occurrences. Généralement le vecteur n'est pas constitué du nombre d'occurrence d'un terme d'index mais plutôt de son poids relatif par rapport au nombre total d'occurrences de l'ensemble des termes d'index. Pour déterminer si deux textes sont proches, il suffit de réaliser un calcul de distance entre les deux vecteurs représentant les documents à comparer **[7]**.

- La sémantique distributionnelle : elle est une extension « sémantique » du modèle vectoriel. Elle va plus loin dans la recherche puisqu'elle tient compte du contexte qui entoure chaque terme. Elle va tenir compte des co-occurrences des termes d'index qui reflètent des liens sémantiques pouvant exister entre les termes. Concrètement, elle va réduire le nombre de termes d'index en regroupant les termes proches sémantiquement **[7]**. Ces systèmes nécessitent des dictionnaires sémantiques spécifiques **[116]**.

- Traitement des données : de nombreuses techniques de *data mining* permettent de classer les corpus dans différentes classes que le système définit automatiquement. Parmi ces méthodes, on peut citer les plus connues : les centres mobiles, les nuées dynamiques et les *k-means*. Pour créer automatiquement des classes de documents, ces méthodes tentent de minimiser la variance entre les classes. En d'autres termes, les systèmes tentent de mettre dans une catégorie les documents semblables et dans des catégories différentes les documents les moins semblables possibles **[7]**.

Les techniques de *text mining* étendent les techniques du *data mining* à la masse considérable des données textuelles en automatisant le processus de caractérisation des documents. Toutefois, à la différence du *data mining*, le *text mining* opère dans un univers de données moins structurées que celui des bases de données **[75]**.

8. Le *data mining* et le multimédia

Un certain nombre de recherches ou de produits émergeants traitent du *data mining* appliqué à des informations multimédias **[75]** :

- Reconnaissance du langage : Les recherches actuelles s'orientent vers des ordinateurs mobiles sans clavier à commandes vocales. Le projet *Galaxy* a mis en œuvre le système Jupiter qui permet d'obtenir la météo sur 600 villes dans le monde suite à une phrase énoncée lentement et distinctement en américain.
- *Image mining* : Il s'agit de la recherche des relations entre des images ou des séquences d'images. Par exemple, la recherche des similitudes entre des images médicales pour diagnostiquer une pathologie.
- *Video mining* : C'est l'extension de l'*image mining* au domaine de la vidéo. Il consiste aussi à rechercher des éléments communs ou à classer des vidéos en fonction de leur contenu.

9. Conclusion

Ce chapitre a permis de présenter les concepts de base de *data mining*, le processus de résolution d'un problème à l'aide du *data mining*, les tâches du *data mining*, les techniques et algorithmes de *data mining*, le *text mining* et le *data mining* et le multimédia.

Les méthodes présentées dans ce chapitre ne sont pas exhaustive. D'autres méthodes importantes ne sont pas étudiées, telles que le raisonnement à base de cas, les algorithmes génétiques… De plus, pour les méthodes proposées, il existe de nombreuses variantes, non présentées qui peuvent s'appliquer à des tâches différentes. En tout état de cause, un fait important communément admis : il n'existe pas une méthode supérieure à toutes les autres. Par conséquent, à tout jeu de données et à tout problème correspond une ou plusieurs méthodes. Le choix se fera en fonction de la tâche à résoudre, de la nature et de la disponibilité des données, des connaissances et des compétences disponibles et de la finalité du modèle construit.

Dans ce qui suit, nous allons voir les approches existantes du *XML mining*. Celles-ci sont basées sur les algorithmes de *data mining* étudiés dans ce présent chapitre.

CHAPITRE 3
LES APPROCHES DE FOUILLE DANS LES DOCUMENTS XML : ETAT DE L'ART

1. Introduction

XML s'est imposé comme le méta-langage permettant de représenter et d'échanger des données non seulement sur le web mais aussi de façon générale en entreprise **[50]**. Face aux quantités d'informations qui ne cessent d'augmenter dans les documents XML, l'extraction automatique de connaissances à partir de ces documents et les techniques de visualisation des résultats sont devenues indispensables. C'est la raison d'être de la fouille dans les documents XML ou *XML mining*.

Dans ce chapitre nous présentons les approches proposées dans la littérature pour la fouille dans les documents XML. Un survol de quelques projets et prototypes réalisés dans ce domaine permettra de voir différentes démarches de mise en œuvre. Aussi, un certain nombre de critères doivent être choisis pour effectuer une comparaison des approches existantes.

2. Les catégories de fouille dans les documents XML

La figure 3.1 montre les résultats d'un sondage effectué en 2008 pour les différents types de données sur lesquels le *data mining* est appliqué **[133]**. Le XML est classé parmi les derniers avec seulement 5.6 %, ce qui montre que les documents XML sont très peu pris en compte dans le domaine de *data mining*.

La fouille dans les documents XML est de deux catégories **[88]** : la fouille dans la structure est la fouille dans le contenu.

Les balises et leurs imbrications représentent la structure d'un document XML. La fouille dans la structure XML (*XML structure mining*) est essentiellement la fouille des schémas **[88]**. Elle s'intéresse à l'extraction d'informations à partir de la structure des documents XML **[51]**.

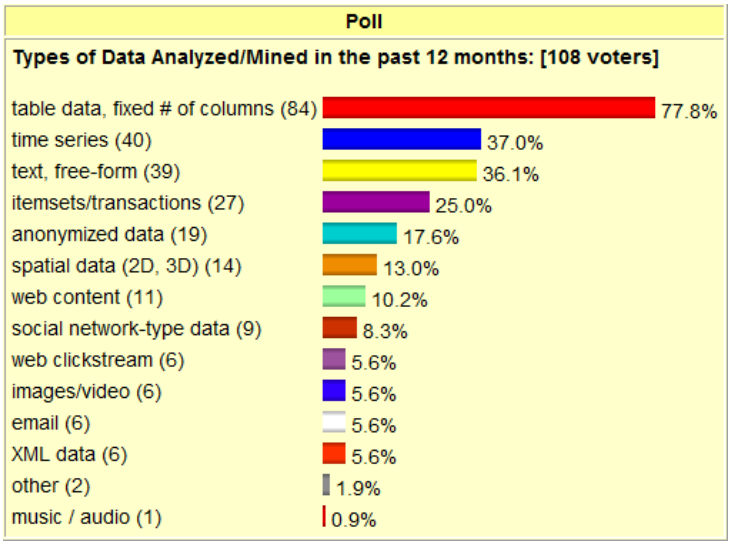


Figure 3.1 : Sondage sur les types de données sur lesquels le *data mining* est appliqué [133].

Dans un document XML, le contenu est le texte entre une balise ouvrante et une balise fermante. La fouille dans le contenu XML (*XML Content Mining*) est essentiellement la fouille des valeurs (une instance d'une relation) **[88]**. La fouille dans le contenu utilise habituellement les techniques de *text mining* pour extraire l'information contenue dans le document.

L'association de la fouille de structure et de la fouille de contenu devrait permettre d'élargir et d'améliorer les techniques de fouille de contenu XML pour améliorer la qualité sémantique des résultats obtenus **[9]**, **[20]**.

Nous commençons par étudier les approches de la première catégorie qui est la fouille dans la structure des documents XML.

3. Les approches de fouille de structure XML

L'objectif de cette partie est de faire un état de l'art sur les solutions proposées pour la fouille dans la structure des documents XML.

3.1. Les travaux de Termier et al. **[111]**

Cette approche propose de représenter un document XML par un arbre étiqueté. Puis d'appliquer l'algorithme *TreeFinder* qui traite le problème de recherche des arbres fréquents à partir des arbres de documents XML. Un arbre est étiqueté s'il existe une injection de l'ensemble des nœuds dans un ensemble d'étiquettes. Une étiquette d'un nœud est un élément d'un alphabet fini. L'algorithme *TreeFinder* est composé de deux étapes : le *clustering* et le calcul des arbres communs maximaux.

3.1.1. Clustering guidé par les occurrences de paires d'étiquettes fréquentes

Cette première étape prend en entrée un ensemble d'arbres $T = \{t_1, t_2, \ldots, t_n\}$, où chaque t_i est représenté par les items *l*m*. L'item est une paire d'étiquettes *(l, m)* de deux nœuds, tel que *l* est l'étiquette du nœud ancêtre d'un nœud descendant étiqueté par *m*. L'ensemble de tous les items existants dans *T* est appelé *itemsets*.

L'algorithme utilisé pour calculer les *itemsets* fréquents est *A-priori* **[4]**. Les paires d'étiquettes sont identifiées avec un seuil de fréquence ε prédéfini (support minimum).

À la fin de cette étape des ensembles-support sont construits. Un ensemble-support est un ensemble contenant des arbres XML dont lesquels les *itemsets* fréquents apparaissent. Une paire d'étiquettes est fréquente si elle apparait ε fois. Les ensembles-support forment les clusters.

3.1.2. Calcul des arbres communs maximaux

Cette étape calcule pour chaque cluster l'arbre maximal qui est le plus gros arbre inclus dans tous les arbres du cluster. Pour cela, les arbres de chaque cluster sont représentés sous forme de formules relationnelles. Pour un arbre *t*, l'encodage relationnel *Rel(t)* est la conjonction de tous les atomes *ab(u,v)*, tels que *u* et *v* sont des nœuds de *t*, *u* a l'étiquette *a*, *v* a l'étiquette b et *u* est le père de *v*. $Rel^{+}(t)$ décrit la fermeture transitive de la relation d'ancestralité, c'est la conjonction de tous les atomes *a*b(u,v)*, tel que *u* est un ancêtre de *v*.

La figure suivante illustre les deux fonctions d'encodage *Rel* et *Rel+* pour deux arbres étiquetés.

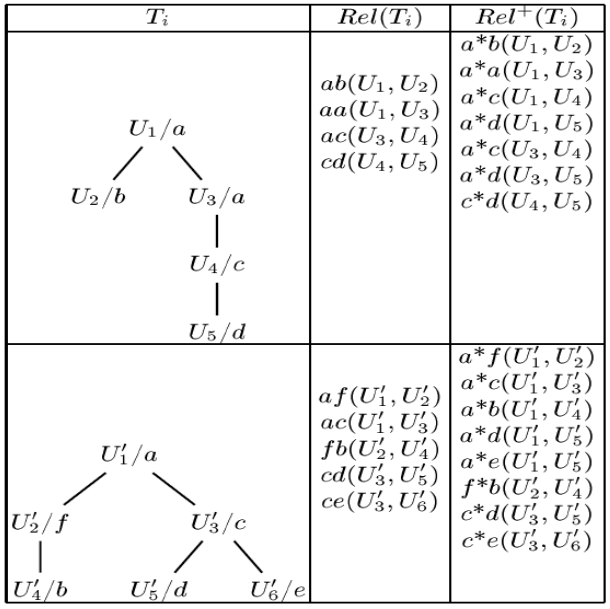


Figure 3.2 : Deux arbre T_1 et T_2 et leur encodage relationnel [111].

Pour chaque cluster la LGG (*Least General Generalization*) **[94]** des formules relationnelles encodant les arbres, est calculée :

$$LGG(Rel^+(t_1),\ldots, Rel^+(t_n)) \qquad (3.1)$$

La LGG de deux formules relationnelles est la formule la plus spécifique qui θ-subsume $Rel(f_1)$ et $Rel(f_2)$. Pour deux formules de la logique du premier ordre C et C', C θ-subsume C' s'il existe un mapping θ des variables de C dans les variables et constantes de C' tel que tout atome de $C\theta$ apparaisse dans C'. Si on prend l'exemple de la figure 3.2 : $LGG(Rel^+(t_1), Rel^+(t_2)) = a*b(U_1, U_2) \wedge a*c(U_1, U_3) \wedge a*d(U_1, U_4) \wedge c*d(U_3, U_4)$. Et donc, l'arbre résultant est :

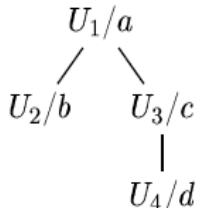


Figure 3.3 : Arbre résultant de LGG de deux formules relationnelles [111].

3.1.3. Discussion

Le but de l'algorithme *TreeFinder* est la découverte des arbres fréquents à partir des clusters d'une collection de documents XML. Les arbres fréquents sont les arbres communs inclus dans tous les arbres d'un cluster. Les clusters sont les ensembles d'arbres dans lesquels on retrouve les mêmes occurrences de paires

d'étiquettes. Dans l'étape du *clustering*, l'algorithme est guidé par la recherche d'occurrences fréquentes selon un certain seuil de paires d'étiquettes pour des nœuds en relation d'ancestralité. D'où la nécessité de l'intervention de l'utilisateur pour définir le seuil.

Dans sa deuxième étape, l'algorithme utilise une méthode empruntée à la programmation logique inductive, pour calculer *LGG* sur une représentation adaptée des arbres de chaque cluster. *LGG* est limité par son calcul qui coûte très cher.

L'algorithme ne prend pas l'ordre des fils en considération, il préserve la relation d'ancestralité plutôt que la relation de parenté. Par contre, il ignore les polysémies (une même étiquette dénotant différents concepts).

3.2. Les travaux de Francesca et al. **[47]**, **[48]**

Ces travaux proposent le *clustering* hiérarchique agglomératif des documents XML en identifiant les similarités structurelles avec l'utilisation de la notion d'appariement structurel entre les documents XML. Nous allons voir ci-dessous des préliminaires sur l'appariement des arbres.

Pour deux arbres XML, Un squelette commun est défini comme une collection de balises possédant les mêmes noms, le même niveau de profondeur et le même nœud parent. Un arbre t est un tuple $t = <r_t, V_t, E_t, \delta_t>$ avec r_t est le nœud racine de t, $V_t \subseteq N$ est l'ensemble des nœuds, $E_t \subseteq V_t \times V_t$ est l'ensemble des arcs, et $\delta_t : V_t \rightarrow \sum$ est la fonction étiquetée du nœud où $\sum$ est un alphabet.

Pour deux arbres t_1 et t_2, il existe un appariement significatif appelé $m(v,w)$ avec $v \in V_{t_1}$ et $w \in V_{t_2}$ si les conditions suivantes sont réalisées :

- $v = r_{t_1}$, $w = r_{t_2}$ et $\delta_{t_1}(v) = \delta_{t_2}(w)$;
- sinon $\delta_{t_1}(v) = \delta_{t_2}(w)$, $profondeur_{t_1}(v) = profondeur_{t_2}(w)$ et $m(a,b)$ maintient quand $(a,v) \in E_{t_1}$ et $(b,w) \in E_{t_2}$.

En général, un appariement multiple peut se produire quand un nœud de l'arbre a un appariement significatif avec plus d'un nœud d'un autre arbre.

La pertinence des appariements significatifs est évaluée à travers la fonction de poids $w_m : V \times V \rightarrow N$, pour un ensemble de nœuds V. Pour deux nœuds $v \in V_{t_1}$,

$w \in V_{t_2}$ le poids associé à *m(v,w)* est calculé par : $w_m(v,w) = 0$ si $\delta_{t_1}(v) \neq \delta_{t_2}(w)$, sinon $w_m(v,w) = 1 + sum_{v,w}$. $sum_{v,w}$ est la somme des degrés pertinents associés aux meilleurs appariements entre les descendants de *v* et *w*.

L'ensemble d'appariement significatif représente les chemins communs entre deux arbres. Le chemin d'un nœud est la suite des nœuds visités à partir de la racine pour atteindre ce nœud, en parcourant l'arbre de fils en fils **[119]**. Un arbre contenant seulement ces chemins communs est défini comme un arbre d'appariement (c'est le résultat de l'intersection des arbres). Un arbre d'appariement t_m est optimal si son poids est maximum (pour chaque arbre d'appariement $u_m \neq t_m$, on a $|t_m| \geq |u_m|$).

L'algorithme *XRep* proposé par ces travaux, est présenté dans la section suivante.

3.2.1. XRep

Initialement, chaque arbre XML est placé dans son propre cluster et une matrice contenant les distances des paires des arbres est calculée. Ensuite, l'algorithme *XRep* fusionne les clusters les moins dissimilaires d'une façon itérative (évalués sur la base des représentants des clusters). En conséquence, la matrice des distances est mise à jour pour refléter l'opération de fusion. Le processus est arrêté quand une partition optimum est atteinte (intra-distance minimisée entre les clusters et inter-distance maximisée).

Le noyau de *XRep* est le calcul du représentant du cluster. Un représentant du cluster des documents XML est un document XML qui reflète tous les contenus structurels des autres documents. La notion du représentant du cluster peut être formalisée comme suit : dans un domaine *U*, doté de la fonction de distance $U \times U \rightarrow R$, et un ensemble $S = \{t_1, \ldots, t_n\} \subseteq U$ d'arbres XML, le représentant de *S* est l'arbre qui minimise la somme des distances :

$$rep(S) = \min_{t \in u} \sum_{i=1}^{n} d(t_i, t) \qquad (3.2)$$

La distance d'édition entre les arbres calcule le coût minimum (*minimum-cost sequence*) des opérations requises pour convertir un arbre donné en un autre arbre **[129]**. Dans cette approche la définition de la distance d'édition proposée dans **[64]** est adoptée : pour deux arbres t_1 et t_2, les opérations d'édition sont l'insertion d'un nœud $w \in V_{t_2}$ dans t_1 et la suppression du nœud feuille de t_1.

Les étapes du calcul du représentant du cluster sont :

a. Appariement des arbres XML

Un algorithme est proposé pour construire l'arbre d'appariement optimal de deux arbres XML. L'idée est qu'à chaque niveau, tous les appariements significatifs sont détectés et le poids est calculé en calculant la matrice d'appariement et en enlevant les appariements multiples pour construire l'arbre d'appariement optimal.

b. Fusion des arbres XML

Un arbre fusion $t_{1,2}$ est formé pour inclure les nœuds de deux arbres t_1 et t_2. Pour un nœud v_i dans t_1 avec aucun appariement significatif ($V_m[i] = -1$), si le nœud parent de v_i apparie avec le nœud $w_j \in V_{t_2}$, alors v_i apparait dans $t_{1,2}$ en tant que fils de w_j. L'opération de la fusion des arbres est associative. Un premier arbre fusion t est construit à l'aide de deux arbres XML d'un cluster donné puis un autre arbre fusion est construit de t et d'un autre arbre. Ce processus est répété jusqu'à ce que tous les documents du cluster soient traités.

c. L'arbre représentant

Les nœuds feuilles sont enlevés s'ils permettent de minimiser la distance entre l'arbre de fusion et les arbres originaux dans le cluster. Ce processus est réitéré jusqu'à ce que la distance ne puisse plus être diminuée. L'arbre résultant est l'arbre représentant.

3.2.2. Discussion

Dans cette approche, une méthodologie de *clustering* des documents XML est proposée concentrant sur la notion du représentant du cluster XML. C'est un document XML prototype qui regroupe les composants les plus pertinents d'un ensemble de documents XML dans le cluster. C'est une technique conçue pour calculer un arbre représentant capable de capturer toutes les spécificités structurelles en utilisant la notion d'appariement structurel entre deux arbres XML. Initialement, les clusters sont définis et chaque arbre XML est placé dans son propre cluster.

Un seul arbre représentant peut être très limité pour inclure tous les composants d'un groupe de documents XML. Les stratégies basées sur la découverte d'un pattern fréquent sont mieux appropriées. La notion d'arbre

représentant peut être améliorée avec l'utilisation d'une forêt de sous arbres fréquents.

3.3. Les travaux de Aggarwal et al. **[1]**

Dans cette approche le classifieur *XRules* est proposé. Il est basé sur les règles structurelles en utilisant les sous structures fréquentes dans les documents XML. Une règle structurelle *R* est de la forme $T \Rightarrow c_i$, avec *T* un arbre enraciné étiqueté ordonné qui représente un document XML et c_i est l'une des *k* classes dans laquelle l'arbre *T* est associé. Un arbre enraciné est un ensemble de nœuds et un ensemble d'arcs satisfaisant les propriétés **[6]** :

- Il existe un nœud particulier appelé racine.
- Tout nœud *n*, autre que la racine, est relié par un arc à un unique antécédent *p* appelé père. *n* est appelé fils de *p*.
- On peut atteindre la racine à partir de n'importe quel nœud de l'arbre en se déplaçant de père en père.

Un arbre est ordonné s'il existe un ordre parmi les fils de chaque nœud **[6]**. Par exemple, l'ordre des indices des nœuds doit être compatible avec l'ordre des fils.

Dans ces travaux, la classification passe par deux phases : apprentissage et test.

3.3.1. Phase d'apprentissage

Dans cette phase, un ensemble de règles structurelles avec des classes connues est utilisé pour construire le modèle de classification. Elle est composée de trois étapes :

a. Fouille de règles structurelles fréquentes

Cette étape utilise l'algorithme *XMiner* qui est basé sur *TreeMiner* **[128]**. Il prend en entrée un ensemble d'arbres *D* avec leurs listes des scopes (pour chaque nœud, définir la position du nœud et la position de son nœud descendant droit le plus bas) et une liste des seuils de supports minimaux $\Pi_j^{\min}$, $\forall$ *j=1,…k* pour chaque classe. Il génère en sortie un ensemble de règles fréquentes $R_j = \{R^1,\ldots,R^{mj}\}$ pour chaque classe en faisant la jointure des listes des scopes de toutes les paires d'éléments. Les règles sont de la forme $R^i : T^i \overset{\Pi}{\Rightarrow} c^i$ et $\Pi \geq \Pi_j^{\min}$.

Pour un arbre T, son support absolu Π^A est défini comme le nombre d'arbres contenant T dans tout l'ensemble de données. Le support relatif Π est le rapport entre le support absolu et le nombre total d'arbres dans l'ensemble de données. T est fréquent si $\Pi \geq \Pi^{min}$, avec Π^{min} est un seuil de support minimum défini par l'utilisateur.

b. Ordonner et élaguer les règles

La force δ d'une règle peut être mesurée par :

- La confiance ρ : le rapport entre le nombre d'arbres contentant T de la classe c_i et le nombre d'arbres contenant T dans tout l'ensemble de données.
- Le rapport de probabilité γ : c'est le rapport entre le support relatif de T avec une classe c_i et le support relatif dans l'ensemble des arbres restants (qui n'appartiennent pas à c_i).
- La confiance pondérée ρ^w : c'est le rapport entre γ et $\gamma+1$.

Les règles ne respectant pas le seuil de confiance minimum ρ^{min} et le rapport de probabilité minimum γ^{min} définis par l'utilisateur sont élaguées. L'ordre des règles est basé sur une relation de précédence <<, pour dériver les règles finales R de l'ensemble des règles pour chaque classe en utilisant une méthode analogue à celle proposée dans **[80]**. R^i << R^j : une règle $R^i : T^i \overset{\Pi_i, \delta_i}{\Rightarrow} c^i$ précède une autre règle $R^j : T^j \overset{\Pi_j, \delta_j}{\Rightarrow} c^j$ si l'une des conditions suivantes est vérifiée :

- $\delta_i > \delta_j$,
- $\delta_i = \delta_j$, $\Pi_i > \Pi_j$.
- $\delta_i = \delta_j$, $\Pi_i = \Pi_j$, $|T^i| < |T^j|$ (nombre de nœuds).

c. Déterminer la classe par défaut

Un ensemble de règles R est prévu pour couvrir un exemple d'arbre S si au moins une règle apparie avec S. En général, un ensemble de règles ne peut pas nécessairement couvrir tous les exemples de l'ensemble d'apprentissage. Une classe appelée « classe par défaut » est choisie pour être l'étiquette d'un exemple de test si aucune des règles ne l'apparie.

Soit Δ l'ensemble des exemples de l'ensemble d'apprentissage *D* qui ne sont pas couverts par l'ensemble de règles ordonnées *R*. Si $\Delta \neq \emptyset$, alors la classe par défaut est donnée par : $default - class = \arg\max_{c_i} \left\{ \frac{w_i |\Delta_i|}{|D_i|} \right\}$ (3.3)

Avec w_i le poids de la classe c_i tel que $\sum_{i=1}^{k} w_i = 1$ où *k* est le nombre total des classes et Δ_i l'ensemble des exemples d'apprentissage non couverts avec la classe c_i.

Si $\Delta = \emptyset$, alors la classe par défaut est celle qui maximise le poids w_i (obtenu en fixant $\Delta_i = D_i$).

3.3.2. Phase de test

La phase de test prend en entrée le modèle de classification et un ensemble de données des exemples avec des classes inconnues dont le but est de prédire la classe pour chaque exemple de test. Elle consiste à définir l'ensemble des règles d'appariement $R(S) = \left\{ R^i : T^i \overset{\delta^i}{\Rightarrow} c^i \mid T^i \leq S \right\}$ pour chaque exemple de test *S*. Si $R(S) = \emptyset$, c.-à-d. il n'y a aucune règle d'appariement, la classe est prédite pour être la classe par défaut. Si $R(S) \neq \emptyset$, alors soit $|R(S)| = r$ et $R_i(S)$ les règles d'appariement dans $R(S)$ avec la classe c_i comme conséquence et soit $|R_i(S)| = r_i$. Chaque règle dans $R_i(S)$ est de la forme $T^j \overset{\delta^j}{\Rightarrow} c_i^j$ avec $\delta_j \geq \delta_{min}$.

Chaque règle d'appariement $T^k \in R(S) - R_i(S)$ est plus prédictive qu'une classe autre que c_i. *XMiner* trouve le support de T^k pour toutes les classes. Pour chaque classe c_i, la moyenne de la force des règles est calculée par : $\delta_i^u = \frac{\sum_{j=1}^{r} \delta^j}{r}$ (3.4)

Si $\delta_i^u \geq \delta^{min}$ alors classifier *S* dans c_i. Si δ_i^u a une valeur par défaut δ^{min} pour toutes les classes, ça veut dire que l'instance de test ne peut pas être prédite en utilisant les règles et la classe est assignée comme classe par défaut.

3.3.3 Discussion

Cette approche consiste à construire un classifieur en faisant une classification des documents XML basée sur des règles qui associent les structures fréquentes à des classes. Elle est composée de deux phases : d'apprentissage et de test. La

phase la plus consistante est la première phase qui consiste à créer le modèle de classification en appliquant la fouille de règles structurelles fréquentes. Pour appliquer cette dernière, l'utilisateur doit être vigilant lors de la spécification de la liste des seuils de supports minimaux pour chaque classe afin d'assurer le choix des arbres fréquents pour obtenir une bonne classification. Ainsi, d'autres paramètres doivent être spécifiés pour élaguer les règles construites tels que le seuil de confiance minimum ρ^{min} et le rapport de probabilité minimum γ^{min}.

La phase d'apprentissage doit aussi traiter une riche collection d'exemples afin de faciliter la prédiction des classes dans la phase de test.

3.4. Les travaux de Boussaid et al. **[9]**

Une nouvelle méthode d'extraction des règles d'association à partir de la structure des documents XML est proposée par Boussaid et al. La méthode permet de gérer les aspects hiérarchiques des documents tout en améliorant les mécanismes d'extraction grâce à la création d'une structure spéciale représentant la hiérarchie des balises rencontrées. La démarche passe par trois étapes.

3.4.1. Pré-formatage des données

Le pré-formatage consiste à extraire les balises à l'aide du parseur événementiel SAX[1]. Le parseur collecte des informations sur le nom du document, son chemin (en local ou sur internet) et le nombre de balises qu'il contient. Pour une balise, il collecte son nom et son nombre d'occurrences dans l'ensemble des documents (une balise est comptabilisée une seule fois par document) ainsi que ses parents et ses fils dans l'arbre du document. La figure suivante montre les résultats obtenus pour la balise *jour* d'un document XML.

[1] API standard qui permet de parcourir un document XML en utilisant un système basé sur un ensemble d'évènements **[133]**.

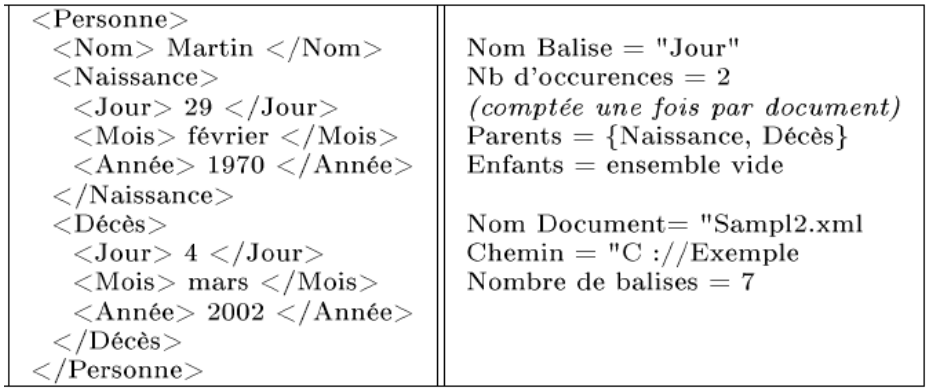


Figure 3.4 : Un document XML et les résultats obtenus pour la balise *jour* [9].

À la fin, une matrice booléenne est constituée. Elle permet d'accéder rapidement aux balises composant un document ou aux documents comprenant une balise donnée.

3.4.2. Création d'une DTD minimale

Afin de gérer les liens hiérarchiques existants entre les balises, une DTD minimale est créée. Elle contient les balises rencontrées en ne prenant en compte que leurs caractéristiques effectives. La construction de cette DTD se fait comme suit :

- Un arbre temporaire associé à chaque document est créé. Il sert à décrire la hiérarchie de l'ensemble des balises dans un document.
- Les arbres temporaires sont fusionnés dans un arbre général, où chaque nœud représente une balise. La racine correspond à l'élément principal et les feuilles sont les éléments atomiques. Ainsi, une branche de l'arbre représente une imbrication de balises. La liste des balises ne contient qu'une seule occurrence de chacune d'elles.
- Ensuite, la DTD est reconstruite en utilisant l'arbre général et les informations concernant la présence et la fréquence des balises dans chaque document (matrice générée lors du pré-formatage). La DTD minimale indique en plus si une balise est conditionnelle ou s'il existe une liste de balises.

3.4.3. Recherche des itemsets fréquents et extraction des règles d'association

En plus de l'adaptation de l'algorithme *A-priori* à ces structures de données, des traitements au niveau de la constitution des *itemsets* candidats sont ajoutés afin de prendre en compte la structure hiérarchique des documents XML.

Lors de la constitution des *1-itemsets* candidats, l'ensemble des balises extraites lors de la phase de pré-formatage est utilisé et celles qui sont renseignées dans l'ensemble des documents sont supprimées. Elles n'apportent aucune information supplémentaire. Lors de la constitution de l'ensemble des *k-itemsets* candidats, la DTD minimale est utilisée pour supprimer les redondances. Seulement les balises qui ne présentent pas de redondance par rapport à la hiérarchie sont sélectionnées.

L'extraction de deux types de règles d'association est faite. Les règles d'association "classiques" sont extraites grâce à l'algorithme d'extraction de règles d'association de **[2]**. Les règles d'association "hiérarchiques" permettent de représenter la fréquence des liens entre des balises directement imbriquées. Chaque balise, admettant des balises imbriquées, est considérée comme antécédent d'une règle. Toutes les balises imbriquées dans celle-ci sont considérées comme conséquence de cette règle. Dans ces deux cas, les règles d'association extraites sont présentées sous forme de documents XML.

3.4.4. Discussion

Cette méthode permet de représenter les liens existants entre les balises d'un ensemble de documents XML de même structure. Donc, elle n'est pas adaptée à une collection de documents XML possédants des structures hétérogènes.

La démarche consiste à pré-formater les données XML afin de les rendre exploitables par les algorithmes classiques de fouille et mettre en place des structures adéquates pour la gestion de la hiérarchie entre les balises, la DTD minimale en l'occurrence. Elle applique l'algorithme *A-priori* pour la recherche d'*itemsets* fréquents. Enfin, les règles d'association sont extraites et les résultats sont présentés sous forme de documents XML.

3.5. Les travaux de Dalamagas et al. **[30]**

Dalamagas et al. **[30]** représentent un document XML par un arbre enraciné étiqueté ordonné. Puis, le *clustering* est appliqué en utilisant des distances qui estiment la similarité des documents XML en termes d'hiérarchies des nœuds. Ils proposent trois étapes : le calcul du résumé structurel pour les arbres, le calcul de la distance structurelle et enfin le *clustering*.

3.5.1. Résumé structurel

Pour construire le résumé structurel de l'arbre, on déploie les deux pas suivants en parcourant l'arbre en pré-ordre[1] :

- *Reduce nesting* : consiste à réduire les nœuds imbriqués répétés. Un nœud imbriqué répété est un nœud qui n'est pas une feuille et qui a la même étiquette que l'un de ces ancêtres.

- *Reduce repetition* : consiste à réduire tous les nœuds répétés. Un nœud répété est un nœud qui a un chemin dupliqué dans l'arbre.

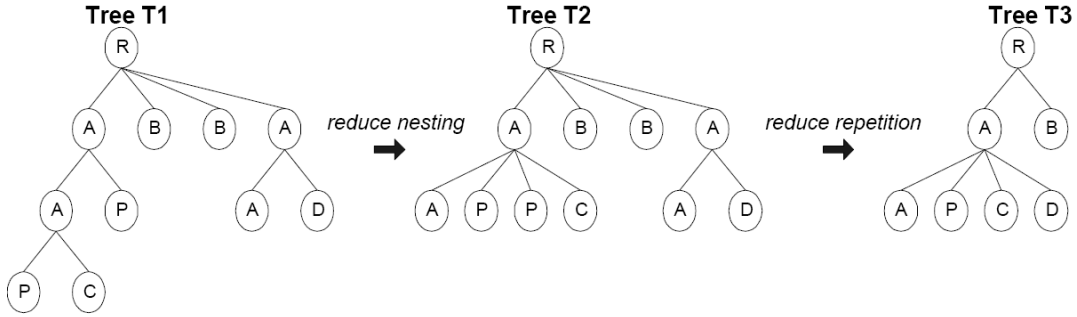


Figure 3.5 : Extraction du résumé structurel [30].

3.5.2. Distance structurelle

D'abord, il faut calculer la distance d'édition **[106]** entre les résumés structurels des arbres avec un algorithme qui est proche de celui de **[24]**. Les opérations d'édition pour transformer T_1 en T_2 sont : insérer, supprimer et modifier un nœud.

Un algorithme récursif est proposé pour calculer la distance d'édition (le coût minimal des coûts de toutes les séquences d'édition possibles pour transformer T_1 en T_2) de chaque paire de nœuds qui sont sur les mêmes profondeurs de deux arbres différents T_1 et T_2. L'algorithme donne en sortie $D(T_1,T_2)$ la valeur minimale des distances d'édition calculée pour tour les paires des nœuds. $D'(T_1,T_2)$ est le coût pour supprimer tous les nœuds de T_1 et insérer tous les nœuds de T_2. La distance structurelle S entre T_1 et T_2 est définie comme suit :

$$S(T_1,T_2) = \frac{D(T_1,T_2)}{D'(T_1,T_2)} \qquad (3.5)$$

[1] Le parcours pré-ordre consiste à parcourir les sous arbres de gauche à droite **[6]**.

3.5.3. *Clustering*

La méthode hiérarchique de lien unique SLHM (*Single Link Hierarchical Method*) **[58]**, **[102]** est utilisée pour le *clustering* sous un certain nombre de conditions **[104]**. Un algorithme de lien unique est implémenté en utilisant l'algorithme de *Prim* **[26]** pour calculer le MST d'un graphe (*Minimum Spanning Tree*). C'est un graphe connexe dont la somme des poids des arcs est minimale **[53]**.

Avec *n* résumés structurels des arbres enracinés étiquetés ordonnés, un graphe connexe est construit avec *n* nœuds et *n(n - 1)/2* arcs avec leurs poids. Le poids d'un arc correspond à la distance structurelle entre les arbres (les nœuds du graphe) que cet arc connecte. Pour un *clustering* de niveau l_1, les clusters de lien unique peuvent être identifiés en supprimant tous les arcs avec un poids $w \geq l_1$ de l'arbre MST du graphe. Les composants connectés du graphe restants sont les clusters de lien unique.

3.5.4. Discussion

Ces travaux présentent une méthodologie de *clustering* des documents XML par la structure en regroupant ensemble tous les données de structures similaires. L'application du *clustering* nécessite l'utilisation des distances qui estiment la similarité entre les structures des arbres. Les résumés structurels des arbres sont utilisés comme un index de structure pour améliorer et accélérer le calcul de la distance.

Un graphe est construit dont les nœuds sont les résumés structurels et les arcs sont les poids représentés par les distances. Les clusters sont définis à partir de ce graphe. Par exemple, pour un *clustering* de niveau l_1, les clusters peuvent être identifiés en supprimant tous les arcs avec un poids $w \geq l_1$ de l'arbre MST du graphe. Dans cette partie les auteurs n'ont pas mentionné comment le niveau du cluster est défini (par exemple l_1).

Cette approche est plus appropriée pour une collection de documents XML de structures homogènes. Pour ceux qui ont des structures totalement différentes, il est possible que l'approche ne puisse pas donner des résultats performants.

3.6. Les travaux de Lian et al. **[79]**

Dans ces travaux, S-GRACE un algorithme de *clustering* hiérarchique des documents XML est proposé. Il est basé sur l'information structurelle dans les données ainsi que sur la notion du graphe de structure (*s-graph*). L'algorithme permet de construire des clusters de documents de structures similaires.

3.6.1. Similarité entre les documents XML

Pour un ensemble *C* de documents XML, le graphe de structure *s-graph* de *C* : *sg(C)=(N,E)* est un graphe direct, tel que *N* est l'ensemble des éléments et des attributs dans les documents de *C* et *(a,b)* $\in$ *E* si et seulement si *a* est un élément parent de *b*, où *b* est un attribut de l'élément *a* dans le même document de *C*. Par exemple, le *s-graph* de deux documents *sg(doc$_1$, doc$_2$)* est l'ensemble des nœuds et des arcs apparaissant dans chaque document (figure 3.6).

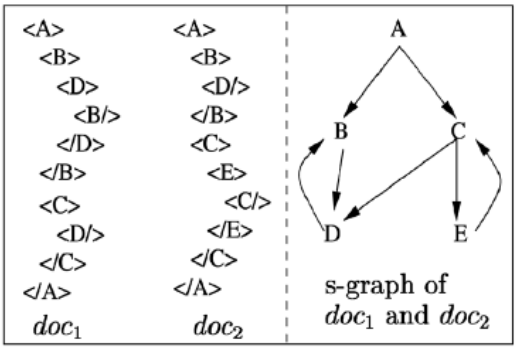


Figure 3.6 : Un exemple de *s-graph* de deux documents [79].

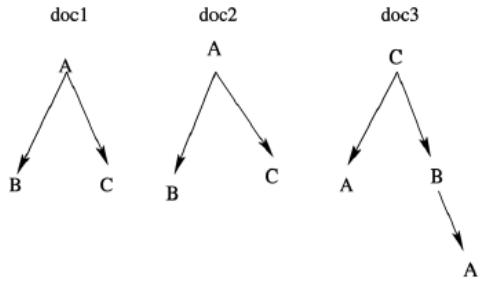
La distance entre deux documents XML C_1 et C_2 est définie par :

$$dist(C_1, C_2) = 1 - \frac{|sg(C_1) \cap sg(C_2)|}{\max\{|sg(C_1)|, |sg(C_2)|\}} \qquad (3.6)$$

Où $|sg(C_i)|$ est le nombre d'arcs dans $sg(C_i)$, $i=1,2$ et $sg(C_1) \cap sg(C_2)$ est l'ensemble des arcs de $sg(C_1)$ et $sg(C_2)$.

3.6.2. *Clustering*

Initialement, les *s-graphs* des documents XML sont calculés et entreposés dans une structure appelé SG. Un *s-graph* est représenté par des bits qui encodent les arcs du graphe. Chaque entrée dans le SG a deux champs : un bit représentant les arcs du *s-graph* et un ensemble contenant les identifiants de tous les documents dont leurs *s-graphs* sont représentés par ces bits.



SG

	entry 1	entry 2
A->A	0	0
A->B	1	0
A->C	1	0
B->A	0	1
B->B	0	0
B->C	0	0
C->A	0	1
C->B	0	1
C->C	0	0
	{doc1, doc2}	{doc3}

Figure 3.7 : Un exemple d'encodage de *s-graph* [79].

Une fois le SG est calculé, le *clustering* est appliqué sur les ensembles de bits. S-GRACE est un algorithme de clustering hiérarchique sur les documents XML qui applique ROCK **[55]** sur les *s-graphs* extraits des documents.

3.6.3. Discussion

Dans ces travaux, un framework de *clustering* de données XML est proposé. Il est basé sur la notion de *s-graph* pour représenter les documents XML et une métrique de distance pour exécuter le *clustering*. Le *s-graph* est encodé par des bits et donc le *clustering* est appliqué sur l'ensemble de bits de toute la collection des documents XML.

Les expériences ont montré que le *clustering* basé sur la notion de distance d'édition entre les arbres représentant les documents XML est très coûteux par rapport au *clustering* avec S-GRACE. Ce dernier peut découvrir les clusters difficilement identifiables manuellement.

3.7. Les travaux de Candillier et al. **[17]**

L'idée principale de ces travaux est de transformer un arbre d'un document XML en des ensembles de paires d'attribut-valeur puis d'y appliquer des méthodes de classification et de *clustering*. Les deux étapes de l'approche sont décrites ci-dessous.

3.7.1. Transformation de l'arbre

À partir d'un arbre XML, différents ensembles de paires d'attribut-valeur sont construits :

- Un ensemble de balises et leurs occurrences.
- Un ensemble de relations père-fils (paires de balises) et leurs occurrences.
- Un ensemble de relations frères suivants (paires de balises) et leurs occurrences.
- Un ensemble de chemins et sous chemins distincts commençant de la racine (séquences finies de balises) et leurs occurrences.
- Un ensemble de positions des nœuds et le nombre de leurs fils.

3.7.2. *Clustering* et classification

Après avoir transformé l'arbre XML, l'algorithme *boosted C5* **[101]** est utilisé pour la classification et l'algorithme SSC (*Statistical Subspace Clustering algorithm*) **[18]** pour la classification et le *clustering*.

SSC est basé sur l'utilisation du modèle probabiliste en supposant que les clusters suivent des distributions indépendantes sur chaque dimension (ou variable). Les clusters sont générés selon une distribution gaussienne modélisée par un centre et une matrice de covariance sur les dimensions. SSC utilise l'algorithme EM (*Expectation Maximization*) **[127]** qui permet de trouver les paramètres du modèle qui correspondent le mieux aux données. En sortie, il donne une description compréhensible des clusters identifiés. Supposant que *K* est le nombre de clusters attendus, les étapes de détection des clusters sont :

- Initialisation aléatoire du modèle (choisir les centroïdes initiaux et associer chaque objet au cluster dont le centroïde est le plus proche).
- Optimisation des valeurs du modèle en appliquant l'algorithme EM en répétant ses étapes jusqu'à trouver $LL^{t+1} - LL^{t} < \delta$: sélectionner le modèle qui maximise *LL* (*LL* : logarithme des probabilités, δ : constant positif très petit).

Une nouvelle phase est ajoutée à SSC qui consiste à garder seulement les meilleurs attributs pour chaque cluster en utilisant un paramètre défini par l'utilisateur *nb_ds* qui spécifie combien d'attributs il faut garder. Les attributs de poids élevé sont gardés et les autres sont ignorés. Les poids sont calculés par un rapport entre la variance globale et la variance locale.

Afin de fournir en sortie de l'algorithme SSC une description simple des clusters trouvés, les clusters sont représentés par des règles (hyper-cubes dans des sous-espaces de l'espace original), définies par un minimum de dimensions. À chaque

règle est associé un intervalle minimum contenant tous les points de données du cluster choisi et le support de la règle est calculé (l'ensemble des objets compris dans la règle). Puis pour toutes les dimensions, présentées dans l'ordre croissant de leurs poids, il faut supprimer la dimension qui n'altère pas le support de la règle.

SSC est adapté pour le *clustering* des documents XML. L'ensemble des attributs possibles associé à un document XML est composé de 5 classes d'attributs. Le résultat de la méthode du *clustering* est une hiérarchie dont chaque nœud représente un test sur les différents attributs. L'ensemble des données est coupé en deux parties construites de documents inclus dans le cluster et transformé avec l'ensemble d'attributs donné. Cette méthode consiste à calculer pour chaque cluster et pour chaque ensemble d'attributs possibles l'intérêt du partitionnement de clusters en deux parties selon l'ensemble des attributs, quand le nombre de clusters n'est pas encore atteint. L'intérêt de partitionnement est le rapport entre *LL* du modèle pour deux clusters et *LL* du modèle pour un cluster, mesuré par le nombre de points de données dans le cluster. Ensuite, il faut sélectionner la meilleure coupe et calculer la règle correspondante et l'utiliser comme un prochain test dans l'arbre généré.

SSC est aussi adapté à la classification des documents XML en suivant deux étapes :

- La première permet de mixer plusieurs classes dans un cluster. Par contre, elle ne permet pas à une classe d'être divisée entre différents clusters. Elle génère en sortie une hiérarchie et une partition courante.
- L'algorithme de la seconde étape qui est complètement guidée par les classes, prend comme entrée les sorties de l'étape précédente et consiste à séparer les classes qui forment une partie des mêmes clusters en utilisant des règles. Si une règle discrimine une classe des autres est trouvée alors elle sera utilisée comme un prochain test dans la hiérarchie. Si aucune règle n'est trouvée alors il est possible de discriminer une classe des autres dans un cluster donné. Le taux d'erreurs obtenu est testé avec le modèle probabiliste généré sur chaque ensemble d'attributs possible. Il faut sélectionner celui qui mène au plus petit taux d'erreur comme un test prochain dans la hiérarchie.

Dans la hiérarchie finale, les tests sur les nœuds peuvent correspondre à des tests membres de règles ou à des tests de probabilités sur des modèles probabilistes. Chacun de ces tests est appliqué seulement sur un seul ensemble d'attributs à la fois.

3.7.3. Discussion

Dans ces travaux, chaque arbre XML est transformé en un ensemble d'attribut-valeur. Les ensembles d'attribut-valeur sont construits en utilisant les différentes relations entre les nœuds de l'arbre. Un nombre très élevé d'attributs est construit à partir d'un arbre XML.

Une nouvelle méthode de *subspace clustering* est proposée. Elle est basée sur l'algorithme EM en ajoutant l'hypothèse que les données ont été générées selon des distributions indépendantes sur chaque dimension. L'algorithme SSC nécessite un paramètre donné par l'utilisateur *K* qui est le nombre de clusters recherchés mais qui requiert un choix judicieux. Lorsque *K* est supérieur au nombre réel des clusters recherchés, les règles associées aux clusters se chevauchent.

L'algorithme C5 est utilisé pour faire la classification sur l'ensemble de données à condition d'ignorer quelques attributs. Donc, il est plus convenable d'utiliser des algorithmes de classification qui peuvent être appliqués sur un nombre élevé d'attributs construits.

3.8. Les travaux de Hagenbuchner et al. **[56]**

L'approche de Hagenbuchner et al. **[56]** présente deux modèles basés sur SOM (*Self Organizing Maps*) les cartes auto organisatrices. Le premier modèle SOM-SD est le SOM basé sur les données structurées qui fait le *clustering* de données où un document XML est représenté par un graphe. Le deuxième modèle CSOM-SD ajoute la notion du contexte (l'état des parents, des fils, des voisins…).

3.8.1. SOM (*Self Organizing Maps*)

Un SOM **[71]** contient un nombre de neurones organisés sur une carte. Un vecteur *m*, appelé « *codebook* », est associé à chaque neurone avec la dimension de *m* égale à la dimension du $i^{ème}$ vecteur d'entrée x_i. Les neurones sur la carte sont liés ensemble par une topologie rectangulaire ou hexagonale. Ils sont mis à jour selon la fonction de voisinage *f( )*. SOM est un modèle formé sur un ensemble

d'exemples tel que pour chaque vecteur d'entrée x_i dans un ensemble de données, il faut obtenir le meilleur vecteur d'appariement *codebook* en calculant :

$$c = \arg\min_j \left\| x_i - m_j \right\| \tag{3.7}$$

Où || . || est la norme euclidienne. Les *codebooks* de la carte sont mis à jour comme suit : $\Delta m_j = \alpha(t)\, f(\Delta_{jc})\, (m_j - x_i)$ (3.8)

Où Δ_{jc} est la distance topologique entre un neurone *j* et un neurone gagnant *c* et $\alpha(t)$ est le coefficient d'apprentissage qui diminue à 0 avec le temps *t*. La fonction de voisinage $f(\Delta_{jc})$ prend la forme d'une fonction de Gauss :

$$f(\Delta_{jc}) = \exp\left(-\frac{\left\| l_j - l_c \right\|^2}{2\sigma^2} \right) \tag{3.9}$$

Où σ est la propagation, l_c est la position du neurone gagnant et l_j est la position du $j^{ème}$ neurone.

Les deux équations sont exécutées selon un nombre d'itérations. Le résultat est une carte *SOM* qui mappe les données en une carte de *n* dimensions (généralement deux dimensions).

3.8.2. SOM-SD (*SOM for Structured Data*)

SOM-SD pour les données structurées étend SOM pour encoder des graphes d'arbres structurés en traitant les nœuds du graphe l'un après l'autre. La réponse du réseau à un nœud *v* est un mapping de *v* en l'espace étalé. Ce mapping est appelé l'état de *v* qui contient les cordonnées du neurone gagnant.

Soit $y_v = f(x_v)$ la réponse du réseau lors du traitement des informations d'un nœud *v*, où *f( )* est une fonction générale non linéaire. Pour chaque nœud du graphe, le réseau en entrée x_v est défini comme un vecteur obtenu en concaténant l'étiquette de la donnée $l_v \in R^p$ qui est local à un nœud *v* avec les cordonnées du mapping des nœuds $y_{ch[v]}$ tel que $x_v = [l\ y_{ch[v]}]$, avec *ch[v]* les fils du nœud *v*.

Le vecteur *x* peut être de taille constante si son degré de sortie maximal est connu. Pour un nœud avec peu de dimensions par rapport à ceux assumés remplis par des valeurs appropriées, l'initialisation de *x* dépend de la disponibilité des états des fils. Ce qui implique le traitement des nœuds dans un ordre topologique inverse. Ce qui cause le découlement de l'information d'une façon strictement causale (à partir des nœuds feuilles aux nœuds racines).

SOM-SD est formé d'une manière similaire au SOM à la différence que les éléments du vecteur *I* et y_{ch} ont besoin d'être pondérés pour contrôler l'influence de ces composants à une mesure de similarité : $c = \arg\min_j(\|x_v - m_j\|\Lambda)$ (3.10)

Λ est une matrice diagonale $m \times m$ avec ses éléments diagonales $\lambda_{1,1} \cdot \cdot \cdot \lambda_{p,p}$ définis pour μ_1 et $\lambda_{p+1,p+1} \cdot \cdot \cdot \lambda_{m,m}$ pour μ_2. Les constants μ_1 et μ_2 contrôlent l'influence de I_v et $y_{ch[v]}$ pour la distance euclidienne dans l'équation. Le reste de l'algorithme est le même que SOM.

3.8.3. CSOM-SD (*Contextual SOM-SD*)

Pour CSOM-SD le réseau d'entrée est formé avec l'ajout d'une concaténation de l'état des nœuds parents et des nœuds fils au vecteur d'entrée x_v. Chaque nœud *v* possède un ensemble de voisins tel que le réseau d'entrée est : $x_v = [ly^K_{ne[v]}]$ avec *ne[v]* est l'état des nœuds voisins. L'algorithme de CSOM-SD est comme suit :

- 1er pas : initialiser tout *y* avec *k*, *k* = [−1,−1] les coordonnées de sortie impossibles.
- 2ème pas : pour chaque $v \in G_i$ (G_i est le $i^{ème}$ graphe dans l'ensemble) calculer $y^t = h(x_v^{t-1})$ avec $x^{t-1} = [l_v y^{t-1}_{ch[v]} y^{t-1}_{pa[v]}]$. *h( )* calcule l'état du nœud en mappant les sorties x_v^{t-1} ; $y_{pa[v]}$ sont les états des nœuds parents et $y_{ch[v]}$ sont les états des nœuds fils. Répéter ce pas *k* fois, avec *k* le chemin de longueur[1] maximale entre deux nœuds les plus distants de G_i. Appliquer ce pas à tous les graphes.
- 3ème pas : choisir un nœud *v* de l'ensemble de données, initialiser $x_v^K = [l\ y^K_{ch[v]} y^K_{pa[v]}]$ et calculer le meilleur appariement *codebook r* en trouvant le *codebook* le plus similaire m_r. Il peut être accompli en utilisant la distance euclidienne suivante : $r = \arg\min_i(\|x_v^K - m_i\|\Lambda)$.
- 4ème pas : mettre à jour les paramètres du réseau comme suit : $\Delta m_i = \alpha(t) g(\Delta_{ir})(m_i - x_v^K)$. *t* est l'itération courante, *α* est le taux d'apprentissage qui diminue progressivement à 0. *g( )* est la fonction de voisinage dépendante sur Δ_{ir} (la distance topologique entre le neurone *i* et le neurone *r*).

[1] La longueur d'un chemin est égale au nombre de nœuds qui le constituent **[119]**.

Répéter les pas 3 et 4 pour chaque nœud de l'ensemble de données. Les pas de l'algorithme sont répétés jusqu'à ce que le nombre d'itérations donné soit effectué ou jusqu'à ce que la précision du mapping atteigne un seuil donné.

3.8.4. Discussion

Les documents XML sont représentés par des graphes de données structurées. Le *clustering* est appliqué en utilisant l'approche des réseaux de neurones basée sur les cartes SOM, SOM-SD pour les données structurées et CSOM-SD en ajoutant la notion du contexte (les nœuds fils).

La limite majeure de ce modèle est qu'un bon apprentissage du réseau de neurones nécessite toujours un échantillon important. De plus, l'interprétation des groupes constitués est souvent difficile à faire.

3.9. Les travaux de Nayak et al. 1 **[89]**

Un nouvel algorithme de *clustering* agglomératif des documents XML hétérogènes par leurs structures est développé. L'algorithme est appelé XCLS (*the XML documents Clustering with Level Similarity*). Il est basé sur le calcul de *LevelSim* (*Level Similarity*) pour quantifier les similarités structurelles entre les documents XML et les clusters et grouper les documents XML dans des clusters de similarité de niveau maximal. Les documents sont représentés par des arbres étiquetés.

3.9.1. Le clustering avec *LevelSim*

Le *clustering* des arbres XML passe par les trois étapes suivantes :

a. La structure niveau

La première étape consiste à définir la « structure niveau » pour montrer les éléments dans chaque niveau de l'arbre en préservant la hiérarchie et le contexte des éléments des documents, et en concentrant sur les chemins des éléments avec les valeurs du contenu sans considérer les attributs dans un document XML. Pour un élément, les instances multiples des valeurs sont ignorées. La figure suivante montre un arbre *T_Actor* et sa « structure niveau »:

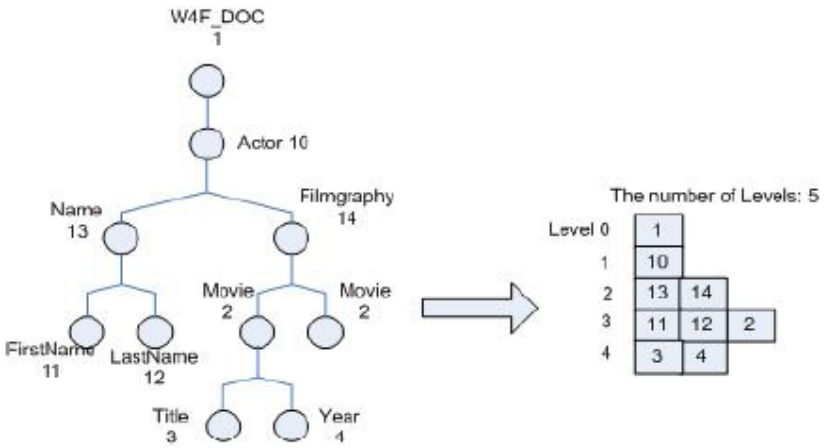


Figure 3.8 : *T_Actor* et sa structure niveau [89].

Les clusters sont aussi représentés par la « structure niveau ». Chaque niveau d'un cluster contient une collection d'éléments de même niveau pour tous les documents appartenant au cluster.

b. Les similarités structurelles

Les similarités structurelles entre deux objets XML (deux clusters, deux arbres, un arbre et un cluster) sont mesurées avec *LevelSim*. Elle assigne des poids différents aux éléments selon le niveau tel qu'un niveau élevé (racine) a plus de poids qu'un niveau bas (feuille). Les éléments sont appariés selon le niveau d'information de chaque objet. L'ordre d'appariement entre deux objets est important dû à l'information structurelle présente dans un document XML. En appariant l'objet 1 (arbre) avec l'objet 2 (cluster) *LevelSim* est défini comme suit :

$$LevelSim_{1\to 2} = \frac{0.5\times ComWeight_1 + 0.5\times ComWeight_2}{TreeWeight}$$
$$= \frac{0.5\times \sum_{i=0}^{L-1} N_1^i \times (r)^{L-i-1} + 0.5\times \sum_{j=0}^{L-1} N_2^j \times (r)^{L-j-1}}{\sum_{k=0}^{L-1} N^k \times (r)^{L-k-1}} \qquad (3.11)$$

$ComWeight_1$ et $ComWeight_2$ sont le total des poids des éléments communs dans tous les niveaux en considérant le niveau d'information de l'objet 1 et l'objet 2 respectivement.

TreeWeight est le total des poids de tous les items dans chaque niveau de l'arbre (l'objet 1).

N_1^i et N_2^j : la somme des occurrences de chaque élément commun dans le niveau *i* de l'objet 1 et le niveau *j* de l'objet 2 respectivement.

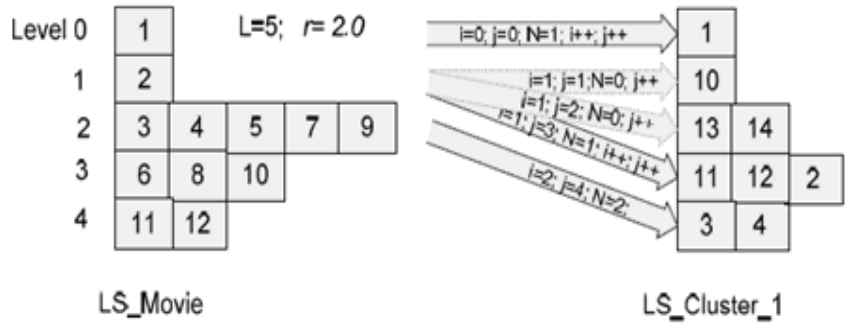
N^k : le nombre d'éléments dans le niveau *k* de l'arbre.

r : « base du poids » c'est le facteur croissant du poids. Il est toujours supérieur à *1* pour indiquer que les éléments de niveau élevé ont plus d'importance que les éléments du niveau bas.

L : le nombre de niveaux dans l'arbre.

c. L'appariement

La troisième étape consiste à faire l'appariement des éléments de l'arbre (objet 1) aux éléments du cluster (objet 2) pour calculer *LevelSim*. La figure suivante montre un cas d'appariement d'objet1 (arbre T_Movie) à l'objet 2 (cluster contenant l'arbre T_Actor) :



$$LevelSim_{1\to 2} = \frac{0.5\times(1\times 2^4+1\times 2^3+2\times 2^2+0\times 2^1+0\times 2^0)+0.5\times(1\times 2^4+0\times 2^3+0\times 2^2+1\times 2^1+2\times 2^0)}{1\times 2^4+1\times 2^3+5\times 2^2+3\times 2^1+4\times 2^0}$$

$$= 0.4259$$

Figure 3.9 : Le processus d'appariement d'un arbre à un cluster [89].

d. *Clustering*

La dernière étape consiste à grouper chaque document XML dans un nouveau cluster ou un cluster existant qui a le maximum de similarités de niveau (*LevelSim*) avec le document. L'algorithme XCLS inclut deux phases : « Allocation » où les clusters sont progressivement formés avec *LevelSim* et « Ajustement » où quelques itérations sont nécessaires pour raffiner le *clustering* et optimiser *LevelSim* entre le document et les clusters existants.

3.9.2. Discussion

Pour faire le *clustering* avec XCLS, chaque arbre XML est représenté par sa structure niveau. Puis, les similarités structurelles entre l'arbre XML et chaque cluster existant sont mesurées avec *LevelSim*. À la fin, l'arbre est assigné au cluster qui a le maximum de similarités ou à un nouveau cluster.

XCLS est plus approprié pour le *clustering* des données hétérogènes, car en plus de la mesure de similarité structurelle entre la relation père-fils, il inclut la relation ancêtre des données. En calculant la similarité, XCLS suppose que l'ordre des nœuds frères n'est pas important. XCLS peut grouper des documents de différentes classes dans un seul cluster, parce qu'il ne vérifie pas les similarités sémantiques entre les tags (synonyme…) et il ne compare pas les documents entre eux. Ces derniers sont comparés avec les clusters existants.

3.10. Les travaux de Garboni et al. **[49]**

Dans Garboni et al. **[49]**, les auteurs proposent une méthode de classification supervisée basée sur la structure seulement des documents XML où ce dernier est vu comme un arbre étiqueté représenté par ses balises seulement. Après l'étape de nettoyage, chaque cluster prédéfini est caractérisé par un pattern extrait des sous-séquences structurelles fréquentes. À la fin une classification basée sur le pattern de chaque cluster est appliquée.

3.10.1. Les étapes de la méthode

La méthode comporte trois étapes : nettoyage, *data mining* et enfin étape d'appariement.

- La première étape consiste à extraire les balises les plus fréquentes de la collection et élaguer les balises inutiles. Une balise qui est très fréquente dans toute la collection peut être considérée comme inutile parce qu'elle ne permet pas d'aider à séparer un document d'un autre (la balise n'est pas discriminante).
- Dans la deuxième étape, le but est de transformer, pour chaque cluster prédéfini, chaque document XML en une séquence. Pour appliquer cette transformation, le document XML est mappé en un arbre étiqueté. Les nœuds de l'arbre sont transformés en des identifiants. De plus, durant le mapping, les balises fréquentes extraites de la première étape sont élaguées. La réduction est alors appliquée, elle consiste à générer la séquence correspondante après une navigation

sur les profondeurs de l'arbre. La transformation d'un arbre est illustrée sur la figure ci-dessous. Une fois l'ensemble de séquences correspondant à chaque cluster est obtenu, un algorithme traditionnel d'extraction de pattern séquentiel capable d'extraire les séquences fréquentes est appliqué.

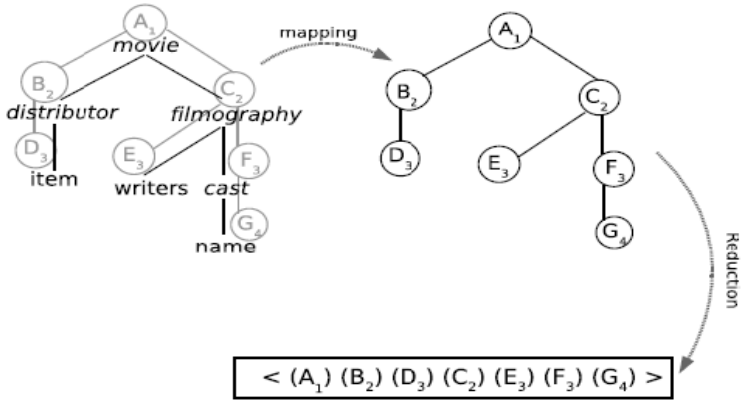


Figure 3.10 : Transformation d'un arbre XML en une séquence [49].

• La dernière étape de la méthode consiste à faire l'appariement entre chaque document de la collection et chaque pattern séquentiel en mesurant la distance entre les documents et les clusters. La mesure est basée sur la plus longue sous séquence commune $LCS(D_i, sp_{Cj}(k))$ entre un document D_i et le $k^{ème}$ pattern séquentiel d'un cluster C_j. Elle calcule la moyenne d'appariement entre le document et le pattern séquentiel qui décrit le cluster. Le score pour un document D_i dans le cluster C_j est défini comme suit :

$$score(D_i, C_j) = \frac{\sum_{k=1}^{|C_j|} \frac{\left|LCS(D_i, spc_j(k))\right|}{\left|spc_j k\right|}}{\left|C_j\right|} \qquad (3.12)$$

3.10.2. Discussion

Cette méthode de classification des documents XML, prend en considération qu'un ensemble de clusters d'une collection doit être prédéfini à l'avance pour pouvoir extraire à partir de chaque cluster un pattern structurel fréquent et évidement faire la classification en faisant l'appariement des documents XML avec les patterns.

Un problème d'appariement des documents XML peut se poser quand deux clusters ont des patterns séquentiels similaires. De plus, ce n'est pas assez judicieux de choisir la première classe arbitrairement quand deux scores sont égaux lors du calcul de la distance pour un même document XML.

3.11. Les travaux de Nayak et al. 2 **[92]**

Ces travaux ont pour but de déterminer les similarités entre des schémas XML hétérogènes (des DTD et des schémas XSD) en utilisant la méthodologie *XMine* qui est basée sur le *clustering* agglomératif. *XMine* est composée des trois phases suivantes : prétraitement, *clustering* et post-traitement.

3.11.1. Prétraitement

Cette phase consiste à déterminer les similarités entre différents schémas comme suit :

a. Analyse des structures

Dans la première étape, la structure d'un schéma est analysée et transformée en un arbre. Chaque nœud de l'arbre correspond à un élément, un attribut ou un opérateur de l'élément en dénotant la relation entre l'élément et son sous élément ou opérateur. Plusieurs arcs peuvent sortir d'un nœud si celui-ci est étiqueté par l'opérateur 'AND' ou 'OR'. Les éléments qui ont les propriétés type (#PCDATA ou ANY dans les DTD, ou *type* dans les XSD) sont considérés comme des feuilles de l'arbre.

b. Analyse des éléments

La deuxième étape consiste à mesurer le coefficient de similarité linguistique *lSim* (*linguistic similarity coefficient*) des éléments en supposant que les mêmes noms ont la même sémantique. *WordNet*[1] **[45]** est utilisée pour gérer les noms différents qui peuvent avoir la même sémantique (par exemple film et movie). Des synonymes sont définis par l'utilisateur et aussi des dictionnaires pour identifier les abréviations (par exemple emp : employé), les acronymes (par exemple ddn : date de naissance). Pour mesurer *lSim*, il faut suivre les trois pas suivants :

1. Construire un ensemble de jetons à partir des noms des éléments composés. Par exemple *PONumber : {PO, Number}*.

2. Développer les jetons en un ensemble linguistique *lingSet* (*linguistic set*) en utilisant le dictionnaire défini par l'utilisateur. Par exemple :

[1] *WordNet* est une ressource lexicale de langue anglaise, disponible sur internet, qui regroupe des termes dénotant un concept donné (noms, verbes, adjectifs et adverbes) en ensembles de synonymes appelés *synsets* **[105]**.

{PO, Number} : {Purchase, Order, Number}.

3. Mesurer *ISim* de deux ensembles de jetons T_1 et T_2 avec la formule suivante :

$$lSim(t_1,t_2)=\frac{\sum_{t_1\in T_1}\max_{t_2\in T_2} sim(t_1,t_2)+\sum_{t_2\in T_2}\max_{t_1\in T_1} sim(t_2,t_1)}{|T_1|+|T_2|} \quad (3.13)$$

$sim(t_1, t_2)$ est une fonction qui calcule la relation sémantique en utilisant *WordNet*, sinon la relation syntaxique est calculée en utilisant la fonction *edit_distance* **[103]**.

c. Identification des chemins similaires maximaux

La structure d'un arbre est représentée par un ensemble d'expressions de chemins $PE = \{p_1, p_2, \ldots, p_m\}$. Chaque expression P est représentée par une séquence d'éléments suivant les liens à partir du nœud racine x_1 jusqu'au nœud feuille x_n en traversant tous les nœuds du chemin : $P=\{x_1, x_2, \ldots, x_n\}$. Un chemin p_j est maximal s'il n'est pas contenu dans un autre chemin ou aucun super chemin de p_j n'est fréquent (il est fréquent s'il apparait dans l'ensemble plus que le seuil défini par l'utilisateur). Cette étape consiste à trouver les chemins similaires maximaux MPE (*Maximal Similar Paths*) parmi un ensemble d'expressions des chemins dans deux arbres, en modifiant les cinq phases de l'algorithme de **[5]** (*sequential pattern mining algorithm*) :

1. Les éléments contenus dans chaque chemin sont triés selon leurs positions hiérarchiques dans les niveaux de l'arbre. Le premier élément apparu dans le chemin représente toujours le nœud racine de l'arbre correspondant. Les éléments restants dans le chemin sont dénotés selon les descendants du nœud racine dans l'ordre.

2. Les éléments du chemin sont représentés par des entiers. Selon *ISim* les éléments de chemins similaires sont représentés par le même entier.

3. Chaque élément similaire dans les deux chemins est inclus dans l'ensemble de *1*-chemin large.

4. Déterminer progressivement les nouveaux chemins larges, le *2*-chemin large, le *3*-chemin large jusqu'à ce que le *n*-chemin large soit défini.

5. Les chemins similaires maximaux sont trouvés en appliquant la phase *backward* **[5]** à tous les chemins larges obtenus dans la phase précédente. Tous les sous chemins qui sont dans les chemins larges sont éliminés jusqu'à trouver les chemins maximaux.

d. Matrice de similarité

La dernière étape de la phase de prétraitement consiste à calculer les similarités entre deux schémas avec la formule suivante :

$$schemaSim(schema_B, schema_Q) = \frac{\sum_{k=1}^{|MPE|} MaxpathSim(MPE_k)}{Max(|PE^B|, |PE^Q|)} \quad (3.14)$$

PE^B et PE^Q sont des ensembles de chemins obtenus en traversant deux arbres *B* et *Q* respectivement. *MaxpathSim* (*similarity coefficient of all maximal similar paths*) est le coefficient de similarités des chemins similaires maximaux basé sur le calcul de *baseSim* (*base element similarity*) qui calcule la similarité sémantique entre deux noms et de PNC (*Path Name Coefficient*) qui différencie deux mêmes noms mais qui ont un niveau de position différent dans les chemins communs (par exemple *book.name* et *book.author.name*) ou dans leur contexte (par exemple *nom de patient* et *nom de medecin*).

Les similarités entre chaque paire de schémas sont mappées dans une matrice de similarité des schémas.

3.11.2. *Clustering*

La matrice de similarité des schémas construite précédemment est l'entrée du processus de *clustering* agglomératif **[131]** pour grouper les schémas similaires. Pour faire le *clustering* des documents XML, l'application *wCluto*[1] (*Web Interface for CLustering TOolKit*) est utilisée. La fonction « *Complete-Link* » est utilisée pour mesurer la distance entre les clusters.

3.11.3. Post-traitement

Dans la dernière phase, un arbre de cluster est construit. Chaque cluster qui contient un ensemble de schémas similaires forme un nœud dans la hiérarchie, où tous les nœuds sont dans le même niveau conceptuel. Chaque cluster peut être décomposé en sous clusters formant un niveau plus bas de la hiérarchie. Les clusters peuvent être groupés ensemble pour former un niveau plus élevé de la hiérarchie.

[1] wCluto, http://cluto.ccgb.umn.edu/cgi-bin/wCluto/wCluto.cgi.

3.11.4. Discussion

XMine est une méthodologie adaptée seulement pour les schémas XML (DTD ou schéma XSD). Un schéma XML est transformé en un arbre. Ce dernier est représenté par un ensemble d'expressions de chemins. Le coefficient de similarité linguistique est mesuré en comparant chaque paire d'éléments de deux schémas en utilisant *WordNet* et des dictionnaires définis par l'utilisateur. Les experts de domaine doivent être appelés pour définir les dictionnaires, ce qui peut présenter une lacune pour *XMine*.

Parmi un ensemble d'expressions de chemins dans deux arbres, il faut trouver les chemins similaires maximaux en utilisant une version modifiée de l'algorithme de **[5]**. Les similarités entre deux schémas sont calculées et mappées en une matrice de similarités des schémas. En utilisant cette matrice, le *clustering* est appliqué et un ensemble de clusters est construit.

La phase de prétraitement est une phase très consistante qui a une grande influence sur la qualité du *clustering*. Elle prend en considération la linguistique, le contexte des éléments et la similarité de la structure hiérarchique. Dans le calcul du coefficient de la similarité linguistique, il est supposé que les mêmes noms ont la même sémantique. Par contre, *XMine* permet de faire le *clustering* des schémas hétérogènes, alors qu'il ne permet pas de gérer les éléments de mêmes noms qui ont une sémantique différente (par exemple avocat : fruit et métier).

4. Comparaison entre les approches de fouille de structure XML

Nous présentons dans cette partie un tableau comparatif entre les différentes approches de fouille dans la structure des documents XML qui ont été présentées. Pour les comparer, nous devons tout d'abord choisir des critères objectifs qui reflètent le but de ce mémoire et qui mettent en évidence les différences entre les approches. Nous définissons les critères de comparaison retenus pour cette étude comme suit :

- **Type de documents XML** : La fouille peut être appliquée sur les documents XML (les instances) comme sur les grammaires XML (DTD ou schéma XSD). Cependant, elle peut être appliquée sur les deux : les instances et les grammaires.
- **Entrées** : Afin de pouvoir appliquer les techniques de *data mining* sur les documents XML, ces derniers ne sont pas utilisés tels qu'ils sont. Ils sont

généralement prétraités et transformés en des entrées. Parmi ces entrées, on trouve les structures arborescentes, les règles, les graphes…etc. Ces entrées sont ensuite utilisées par un algorithme existant de *data mining* ou par un nouveau algorithme proposé.

- **Tâche de fouille** : Pour chaque approche étudiée, nous allons spécifier la tâche de fouille accomplie. Le *clustering*, la classification ou l'association sont les tâches les plus souvent accomplies par les approches.

- **Modèle** : Les approches de fouille dans la structures des documents XML, proposées dans la littérature reposent sur quatre modèles pour représenter les documents XML. Ces modèles sont définis par Denoyer et al. **[38]** comme suit :

1. Modèle basé sur les vecteurs : un document XML est transformé en un vecteur.
2. Modèle basé sur la similarité : il faut définir une mesure de similarité entre les documents XML (ou des clusters) pour effectuer un *clustering* ou une classification.
3. Modèle basé sur les réseaux de neurones (tel que *Self Organizing Map*).
4. Modèle d'arbres fréquents : qui utilise un ensemble fréquent d'items d'un arbre XML.

- **À priori** : Afin d'achever la fouille dans les documents XML, quelques approches nécessitent des connaissances à priori ou des interventions de la part des utilisateurs ou des experts de domaine. Ces interventions sont nécessaires pour spécifier un ou plusieurs paramètres assurant le bon déroulement de l'approche. Si c'est le cas, le symbole « * » est ajouté dans la case à priori de notre tableau comparatif. Sinon la case reste vide.

- **Sémantique** : La plupart des approches étudiées ne permettent pas de gérer l'aspect sémantique des mots composant les éléments d'un document XML. Par exemple, la gestion des mêmes noms de balises ou des mots qui ont une sémantique différente, des noms de balises ou des mots différents qui peuvent avoir une même signification, etc. Pour ces approches, la case sémantique dans le tableau comparatif est vide. Si l'approche n'ignore pas l'aspect sémantique, le symbole « * » est mentionné dans la case.

- **Échantillonnage :** Quelques approches surtout celles qui appliquent la classification supervisée sur les documents XML passent par une phase d'apprentissage. Elles nécessitent toujours un échantillon important afin d'assurer un

bon apprentissage et un échantillon test. Pour ces approches nous mentionnons la case échantillonnage par le symbole « * ». La case est vide dans le cas échéant.

- **Résultats** : Nous allons spécifier pour chaque approche étudiée, les résultats obtenus à la fin.

Tableau 3.1 : Comparaison des approches de fouille dans la structure des documents XML.

	Approche	Type de documents	Entrées	Tâche de fouille	Modèle	à priori	Séman-tique	Échanti-llonnage	Résultats
Structure	**Termier et al. 2002**	Instances XML	Arbre étiqueté	*Clustering*	Modèle d'arbres fréquents	*			Clusters avec arbre maximal pour chacun
	Francesca et al. 2003	Instances XML	Arbre enraciné étiqueté	*Clustering*	Modèle basé sur la similarité				Clusters avec arbre représentant pour chacun
	Aggarwal et al. 2003	Instances XML	Règles structurelles	Classification	Modèle d'arbres fréquents	*		*	Classes
	Boussaid et al. 2004	Instances XML	Matrice booléenne et DTD minimale	Association	Modèle d'arbres fréquents	*			Document XML contenant les règles d'association
	Dalamagas et al. 2004	Instances XML	Graphe connexe	*Clustering*	Modèle basé sur la similarité				Clusters
	Lian et al. 2004	Instances XML	Structure SG de bits	*Clustering*	Modèle basé sur la similarité	*			Clusters
	Candillier et al. 2005	Instances XML	Ensembles de paires d'attribut-valeur	Classification	Modèle basé sur les vecteurs	*		*	Arbres de décision
				Clustering					Hiérarchie de clusters
	Hagenbuchner et al. 2005	Instances XML	Vecteur (étiquettes+fils) pour chaque nœud	*Clustering*	Modèle basé sur les réseaux de neurones			*	Clusters
	Nayak et al. 2005	Instances XML	Structures niveaux	*Clustering*	Modèle basé sur la similarité	*			Clusters
	Garboni et al. 2006	Instances XML	Vecteurs de séquences	Classification	Modèle basé sur la similarité			*	Clusters
	Nayak et al. 2007	Schémas XML	Matrice de similarité	*Clustering*	Modèle basé sur la similarité	*	*		Arbre de clusters

À la suite de cette section, nous présentons successivement les approches de la deuxième catégorie concernant la fouille dans le contenu XML.

5. Les approches de fouille de contenu XML

Il y a que peu de travaux existants qui abordent la problématique de fouille dans le contenu des documents XML. Dans cette partie, nous parcourons l'ensemble de ces travaux.

5.1. Les travaux de Yang et al. **[126]**

Dans cet article un nouveau modèle est proposé : SLVM (*Structured Link Vector Model*), où l'information dans la structure et les liens du document sont effectivement exploités. Un vecteur représente un document tel que les éléments du vecteur sont déterminés par les termes, la structure du document et les documents voisins. Le *clustering* est ensuite appliqué sur ces documents.

5.1.1. *Structured Link Vector Model*

SLVM est basé sur VSM (*Vector Space Model*) qui est une approche standard de représentation dans la classification des documents où un document est représenté par ses mots. VSM a trois phases **[124]**. La première consiste à construire l'index du document qui compte la fréquence des termes. Le calcul des poids pour les termes est la seconde étape et enfin l'évaluation de la similarité. SLVM représente un document par un vecteur SLV (*Structured Link Vector*) : *(d_struct, d_out, d_in)* avec *d_struct* le vecteur de structure, *d_out* le vecteur *Out-Link* et *d_in* le vecteur *In-Link* des documents.

Pour définir le vecteur de structure, les nœuds d'un arbre sont étiquetés en incluant les nœuds d'éléments, les nœuds d'attributs et les nœuds de texte. Pour un document *d* de la collection *D*, son vecteur de structure est de *n* dimensions :

$$d\_struct = (d\_struc_{(1)}, \ldots, d\_struc_{(n)})$$

avec $$d\_struc_{(i)} = \sum_{j=1}^{m} (TF(W_i, Doc.e_j).\varepsilon_j).IDF(W_i) \qquad (3.15)$$

Chaque $d\_struc_{(i)}$ du vecteur de structure du document correspond à un terme. $TF(W_i,Doc.e_j)$ est la fréquence du terme W_i dans le nœud e_j du document *Doc*.

$$IDF(W_i) = \log D/DF(W_i) \qquad (3.16)$$

D est Le nombre des documents, $DF(W_i)$ est le nombre de documents dans lesquels le terme W_i apparait au moins une fois.

ε_j est un vecteur d'unité qui correspond au nœud e_j et m est le nombre de nœuds. La relation des nœuds peut être reflétée par le vecteur d'unité ε_j. Une matrice de similarité est définie pour refléter les relations des nœuds dans les documents $(sjk = \varepsilon_j * \varepsilon_k)$ $(i \geq 0, j \leq m-1)$.

Les vecteurs *In-Link* et *Out-Link* sont proposés pour décrire la relation du lien entre les documents. Ils sont considérés indépendants du vecteur de structure.

Dans *Xlink*, *Xlink:from* définit la ressource source du lien (l'ancre de la source du document local dans un lien) et *Xlink:to* définit la ressource cible (l'ancre de la cible du document distant dans un lien). La ressource est aussi le document entier ou les éléments du document.

Le vecteur *Out-Link* d'un document *Doc* est défini par la somme des vecteurs de structure de sa ressource cible *Out-Link*. La ressource cible *Out-Link* d'un document *Doc* est définie comme la ressource cible ($Doc_n.e_j \in link.to$) des liens dont les ressources sources sont une partie ou la totalité du document *Doc* ($link.form \in Doc$).

$$d\_out = (d\_out_{(1)}, \ldots, d\_out_{(n)}) \tag{3.17}$$

$$d\_out_{(i)} = \lambda_{out} \cdot \sum_{link.from \in Doc} \sum_{Doc_n.e_j \in link.to} (TF(W_i, Doc_n.e_j).\varepsilon_j).IDF(W_i)$$

$$= \lambda_{out} \cdot \sum_{j=1}^{m} (TLout(W_i, Doc, e_j).\varepsilon_j).IDF(W_i)$$

λ_{out} est une constante : le poids du vecteur *Out-Link* comparé au vecteur de structure, Doc_n est un voisin du document *Doc*. $TLout(W_i,Doc,e_j)$ est la fréquence du terme W_i dans le nœud e_j de la ressource cible *Out-link* du *Doc*.

Le vecteur *In-Link* d'un document *Doc* est défini par la somme des vecteurs de structure de sa ressource source *In-Link*. La ressource source *In-Link* d'un document *Doc* est définie comme la ressource source ($Doc_n.e_j \in link.to$) des liens dont les ressources sources sont une partie ou la totalité du document *Doc* ($link.to \in Doc$).

$$d\_in = (d\_in_{(1)}, \ldots, d\_int_{(n)}) \tag{3.18}$$

$$d\_in_{(i)} = \lambda_{in} \cdot \sum_{link.to \in Doc} \sum_{Doc_n.e_j \in link.form} (TF(W_i, Doc_n.e_j).\varepsilon_j).IDF(W_i)$$

$$= \lambda_{in} \cdot \sum_{j=1}^{m} (TLin(W_i, Doc, e_j).\varepsilon_j).IDF(W_i)$$

λ_{in} est une constante : le poids du vecteur *In-Link* comparé au vecteur de structure. $TLin(W_i,Doc,e_j)$ est la fréquence du terme W_i dans le nœud e_j de la ressource source *In-link* du *Doc*.

5.1.2. *Clustering* basé sur SLVM

La clé de l'algorithme *k-means* est le calcul du centre de cluster et le calcul de la similarité. Le calcul du centre du cluster de *k-means* basé sur SLVM est :

$$c = \frac{1}{|S|}\sum_{d\in S} d \tag{3.19}$$

c est un vecteur de *n* dimensions $(c_{(1)},\ldots,c_{(n)})$, où :

$$c_{(i)} = \frac{1}{|S|}\sum_{d\in S} d_{(i)} = \left(\frac{1}{|S|}\sum_{d\in S} d\_struct_{(i)}, \frac{1}{|S|}\sum_{d\in S} d\_out_{(i)}, \frac{1}{|S|}\sum_{d\in S} d\_in_{(i)}\right)$$

La similarité entre deux documents Doc_x et Doc_y est :

$$cos(d_x,d_y)=(d_x*d_y)/(||d_x||\ ||d_y||) \tag{3.20}$$

Où :

$$(d_x * d_y) = (\sum_{i=1}^{n} d_{x(i)} * d_{y(i)})$$
$$= \sum_{i=1}^{n} (d\_struct_{x(i)} * d\_struct_{y(i)} + d\_out_{x(i)} * d\_out_{y(i)} + d\_in_{x(i)} * d\_in_{y(i)})$$

5.1.3. Discussion

Yang et al. présentent une nouvelle méthode de *text mining* qui consiste à faire le *clustering* sur un ensemble de vecteurs. Les vecteurs sont construits à partir des documents semi-structurés en utilisant SLVM le nouveau modèle proposé. Ce modèle prend en considération l'information structurelle et l'information des liens des documents. *k-means* est adopté pour illustrer SLVM.

La sémantique des nœuds telle que la gestion des mêmes mots qui ont une sémantique différente, les mots différents qui peuvent avoir la même signification…, n'est pas traitée dans ces travaux.

5.2. Les travaux de Braga et al. **[10]**, **[11]**, **[12]**

Dans ces travaux, un opérateur appelé XMINE qui permet d'extraire les règles d'association à partir de la structure et du contenu des documents XML est

développé. Il est basé sur *XPath*[1] et est inspiré de la syntaxe *XQuery*[2] et des travaux de **[83]**. La syntaxe de XMINE est comme suit :

```
XMINE RULE
IN Doc-Name (, Doc-Name)*
FOR ROOT IN XPathExpr (, FOR Variable IN XPathExpr)*
LET BODY := (XPathExpr,)+
      HEAD := XPathExpr (,XPathExpr)*
      (, Variable := XPathExpr)*
[WHERE XQuery-Where-Clause]
[GROUP Variable IN Target BY Criteria]
EXTRACTING RULES WITH
      SUPPORT = <real> AND CONFIDENCE = <real>
      [AND XQuery-Where-Clause]
[RETURN XQuery-Return-Clause]
```

La clause IN permet de spécifier la source du document XML. Dans la section FOR, la variable ROOT spécifie le fragment qui représente la transaction par des expressions *XPath*. Un fragment est une partie d'un document XML. Les variables BODY et HEAD identifie les fragments dans le document XML qui apparaissent respectivement dans le corps *X* (la condition) et l'entête *Y* (le résultat) de la règle d'association $X \Rightarrow Y$. La clause optionnelle WHERE spécifie une ou plusieurs conditions. La clause optionnelle GROUP BY permet la restructuration de la donnée source pour définir des groupes reliés à l'attribut de la transaction. La clause EXTRACTING RULES WITH spécifie les contraintes sur les règles résultantes telles que la valeur du support minimum et la valeur de confiance minimum. La clause RETURN définie la structure des règles d'associations produites.

5.2.1. Exemple de XMINE

Dans cet exemple le but est de déterminer avec XMINE, les personnes qui apparaissent comme coauteurs à partir d'un document XML « research.xml » qui représente des informations sur l'activité de recherche dans un département[3] (*PhDCourse*, *People*, *PhDStudent*, *FullProfessor*, *PersonalInfo*, *Publications*, *Book*, *Article*…). En pratique, il faut trouver les règles d'associations de la forme : "Wilson $\Rightarrow$ Holmes". Elle signifie que dans le département, les papiers qui ont Wilson comme auteur ont aussi Holmes comme auteur.

[1] C'est un langage d'adressage, qui a pour objectif la selection des nœuds dans un arbre XML en construisant l'expression de chemin **[133]**.
[2] C'est un langage d'intérrogation des documents XML **[133]**.
[3] http://www.atlantis.edu/research.xml

```
XMINE RULE
IN document("www.atlantis.edu/research.xml")
FOR ROOT IN //People/*/Publications/*
LET BODY := ROOT/Author,
HEAD := ROOT/Author
EXTRACTING RULES WITH SUPPORT = 0.1 AND CONFIDENCE = 0.2
RETURN
<RULE support={ SUPPORT } confidence={ CONFIDENCE }>
<BODY>
{ FOR $item IN BODY RETURN <Item> { $Item } <Item> }
</BODY>
<HEAD>
{ FOR $item IN HEAD RETURN <Item> { $Item } <Item> }
</HEAD>
</RULE>
```

BODY et HEAD sont les éléments auteurs qui apparaissent dans quelques publications. Une règle d'association est définie par la balise RULE, avec deux attributs support et confidence (confiance) et deux sous éléments <BODY> et <HEAD> qui spécifient les items dans le corps et l'entête de la règle. Une règle d'association produite à partir du document « research.xml » en utilisant la syntaxe XMINE de l'exemple est la suivante :

```
<RULE support="0.25" confidence="0.50">
<BODY>
<Item> <Author>Stolzmann</Author> </Item>
<Item> <Author>Wilson</Author> </Item>
</BODY>
<HEAD>
<Item> <Author>Holmes</Author> </Item>
</HEAD>
</RULE>
```

5.2.2 Support et confiance des règles d'association XML

L'ensemble F_D de la collection de tous les fragments XML définis par les expressions *XPath* dans X_R (X_R l'ensemble des expressions *XPath* spécifiées dans la section ROOT) est calculé par : $F_D = \bigcup_{p \in X_R} value(p, d)$ (3.21)

La fonction *value* qui, pour une expression *XPath p* et un document XML *d*, retourne l'ensemble des fragments XML dans *d* identifié par *p*.

L'ensemble F_I est la collection des fragments XML qui sont définis par les expressions *XPath* dans X_I ($X_I = X_B \cup X_H$, X_B l'ensemble des expressions *XPath*

spécifiées dans la section BODY et X_H l'ensemble des expressions *XPath* spécifiées dans la section HEAD) :

$$F_I = \bigcup_{p \in X_I} value(p, d) \tag{3.22}$$

$F_D^{'}$ et $F_I^{'}$ sont définis en filtrant les ensembles F_D et F_I avec la clause *XQuery where w* spécifiée dans la section WHERE. Donc, la fréquence d'un item *X*, $X \in F_I^{'}$ est :

$$freq(X, F_D^{'}) = \frac{\sum_{c \in F_D^{'}} \prod_{f \in X} contains(c, f)}{\left|F_D^{'}\right|} \tag{3.23}$$

Pour deux fragment XML *x* et *y*, la fonction *contains* retourne 1 si *x* contient *y*, 0 sinon.

5.2.3. Architecture de XMINE

Le prototype XMINE est implémenté avec un processus de trois étapes :

- Prétraitement : dans cette étape, les ensembles F_D et F_I sont générés en utilisant l'implémentation Java de *Xalan*[1] dans une table relationnelle *R*.
- *Data mining* : Avec la table *R* et les contraintes extraites de la clause EXTRACTING RULE, les règles d'association sont extraites. Cette étape est basée sur l'algorithme *A-priori*.
- Post-traitement : Les règles d'association sont mappées en une représentation XML.

5.2.4. Discussion

L'opérateur XMINE est un outil qui génère comme résultat un document XML contenant toutes les règles d'association extraites d'un ensemble de documents XML. Plusieurs exemples ont été présentés pour illustrer le principe de fonctionnement de *XMINE*.

Des connaissances à priori sont nécessaires par exemple pour spécifier les contraintes sur les règles résultantes telles que la valeur du support minimum et la valeur de confiance minimum.

[1] The Apache Software Foundation. The Apache XML Project. http://xml.apache.org/xalan-j/.

5.3. Les travaux de Denoyer et al. **[35]**

Un modèle basé sur les réseaux bayésiens est proposé dans ces travaux. Ce modèle est transformé en un classifieur en utilisant la méthode de *Fisher Kernel*. Un document est un arbre où chaque nœud représente une entité structurelle (paragraphe, titre, section...) du document et chaque arc représente une relation hiérarchique entre deux entités (par exemple, un paragraphe est inclus dans une section, deux paragraphes se suivent, etc.). Chaque nœud est composé d'une étiquette et d'un contenu.

5.3.1. Modélisation des documents avec les réseaux bayésiens

Les réseaux bayésiens sont convenables pour modéliser les dépendances et les relations entre les différents éléments dans un document structuré. Chaque document est représenté par un modèle génératif décrit par un réseau bayésien. La relation naturelle modélisée est « *is a descendant of* » dans l'arbre du document en le parcourant en pré-ordre. Soit :

- Λ l'ensemble de toutes les étiquettes possibles pour un nœud structurel.
- *V* est l'ensemble de tous les mots possibles. V^* dénote l'ensemble de toutes les séquences du mot possibles, y compris le vide.
- *d* un document structuré qui consiste en un ensemble de composants $(s_d^1,...s_d^{|d|},t_d^1,...,t_d^{|d|})$ avec s_d^i est l'étiquette de l'$i^{ème}$ nœud structurel de *d* ($s_d^i \in \Lambda$), t_d^i est le contenu textuel de ce $i^{ème}$ nœud ($t_d^i \in V^*$) et |*d*| est le nombre de nœuds structurels.

La méthode de classification utilisée correspond aux modèles génératifs. Ceux-ci ne permettent pas d'estimer directement la probabilité à posteriori *P*(*c*/*d*) mais passent par l'estimation des densités conditionnelles des classes *P*(*d* / *c*) et utilisent la règle de Bayes pour calculer le score de pertinence d'un document pour une classe.

Ce modèle correspond à une application récursive du processus suivant : à chaque nœud structurel *s*, un nombre de sous nœuds structurels, qui peut être zéro, et la longueur de la partie textuelle s'il y en a, sont choisis. Les étiquettes des sous nœuds et des mots sont prélevées de leur distribution respective qui dépend de *s* et la classe du document. En utilisant un tel réseau, la probabilité de jointure qui contient la probabilité structurelle et textuelle est :

$$P(c,d) = P(c)\prod_{i=1}^{|d|} P(s_d^i \mid pa(s_d^i),c)P(t_d^i \mid s_d^i,c) \quad (3.24)$$

Le modèle *Naïve Bayes* est utilisé pour les fragments de texte **[78]**. t_d^i est défini par la séquence de mots $t_d^i = (w_{d,1}^i,...,w_{d,|r_d^i|}^i)$ avec $w_{d,k}^i \in V$ ***et*** $|t_d^i|$ est le nombre des occurrences de mots. Avec *Naïve Bayes*, la probabilité de jointure pour ce modèle est réécrite comme suit :

$$P(c,d) = P(c)(\prod_{i=1}^{|d|} P(s_d^i \mid pa(s_d^i),c))(\prod_{i=1}^{|d|}\prod_{k=1}^{|t_d^i|} P(w_{d,k}^i \mid s_d^i,c)) \quad (3.25)$$

La figure 3.11 montre un réseau bayésien obtenu pour un document XML. Les cercles et les carrés sont respectivement des nœuds structurels et textuels.

5.3.2. Apprentissage

Afin d'utiliser ce modèle, une phase d'apprentissage permettant l'estimation des paramètres du réseau est nécessaire. Les paramètres θ sont comme suit :

$$\theta = \bigcup_c \left(\bigcup_{n\in V, m\in\Lambda} \theta_{n,m}^{c,s} \bigcup_{n\in V, m\in\Lambda} \theta_{n,m}^{c,w} \right) \quad (3.26)$$

Avec $\theta_{n,m}^{c,s}$ est l'estimation pour $P(s_d^i = n \mid pa(s_d^i) = m,c)$ et $\theta_{n,m}^{c,w}$ est l'estimation pour $P(w_{d,k}^i = n \mid s_d^i = m,c)$. *s* dans $\theta_{.,.}^{.,s}$ indique un paramètre structurel et *w* dans $\theta_{.,.}^{.,w}$ un paramètre textuel. Pour apprendre θ*s* en utilisant l'ensemble de documents d'apprentissage D_{TRAIN}, *L* (*log-likelihood*) est maximisée pour D_{TRAIN} :

$$L = \sum_{d\in D_{TRAIN}} \log P(c) + \left(\sum_{i=1}^{|d|} \log \theta_{s_d^i, pa(s_d^i)}^{c,s} \right) + \left(\sum_{i=1}^{|d|}\sum_{k=1}^{|t_d^i|} \log \theta_{w_{d,k}^i, s_d^i}^{c,w} \right) \quad (3.27)$$

L'algorithme d'apprentissage résout pour chaque paramètre $\theta_{n,m}^{c,.}$ de l'équation suivante (. correspond à *s* ou *w*) : $\frac{\partial L}{\partial \theta_{n,m}^{c,.}} = 0$ sous les contraintes :

$$\forall m \in \Lambda, \sum_{n\in\Lambda} \theta_{n,m}^{c,s} = 1, \sum_{n\in\Lambda} \theta_{n,m}^{c,w} = 1 \quad (3.28)$$

Soit D_{TRAIN}^c le sous ensemble de tous les documents dans D_{TRAIN} avec la classe *c*, $N_{n,m}^{d,c}$ est le nombre de fois qu'un nœud d'étiquette *n* possède un parent d'étiquette *m* dans le document *d*. Afin de rendre l'estimation de ces paramètres plus robuste, un lissage est utilisé :

$$\theta^{c,.}_{n,m} = \frac{\sum_{d \in D^c_{TRAIN}} N^{d,c}_{n,m} + 1}{\sum_i \sum_{d \in D^c_{TRAIN}} N^{d,c}_{i,m} + \Delta}$$ où Δ est égal à $|V|$ si $n \in V$ ou $|\Lambda|$ si $n \in \Lambda$. (3.29)

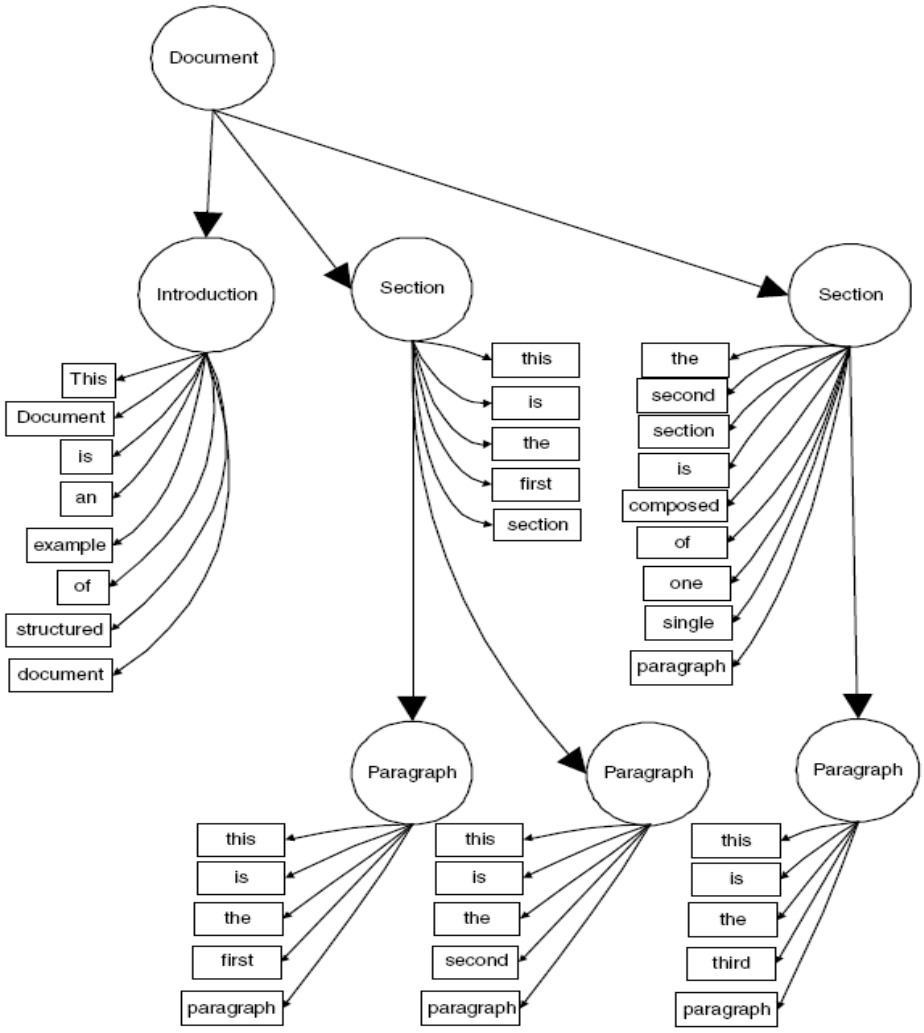


Figure 3.11 : Un réseau bayésien d'un document XML [35].

5.3.3. Les réseaux bayésiens et le noyau de *Fisher*

Le modèle génératif est transformé en un modèle discriminant en utilisant la méthode du noyau de *Fisher*. L'idée principale de **[65]** est d'associer à un modèle génératif un vecteur pour chaque exemple *x* et d'utiliser un classifieur discriminant classique sur ce vecteur. Pour un exemple *x* et un modèle génératif avec les paramètres θ, le score *Fisher* est défini par :

$$U_x = \nabla_\theta \log P(x \mid \theta)$$ (3.30)

Ce score est le gradient de la log-vraisemblance de *x* pour le modèle génératif. C'est un vecteur de dimension fixe (la dimension de *θ*) qui exprime combien un

paramètre du modèle génératif contribue à générer un exemple donné. Avec ce vecteur, une fonction *Kernel* pour définir la similarité entre deux exemples *x* et *y* est définie comme suit :

$$K(x,y) = U_x^T M^{-1} U_y \tag{3.31}$$

Avec *M* est la matrice d'information de *Fisher* : $M = E_X[U_X^T U_X]$ (3.32)

Le score *Fisher* considéré pour chaque document *d* et chaque classe *c* est comme suit :

$$U_d^c = \nabla_{\theta^{c,\cdot}_{\cdot,\cdot}} P(d,c \mid \theta) \tag{3.33}$$

Pour chaque paramètre $\theta^{c,\cdot}_{n,m}$, le score *Fisher* est calculé comme suit :

$$\frac{\partial \log P(d,c \mid \theta)}{\partial \sqrt{\theta^{c,\cdot}_{n,m}}} = \frac{N^d_{n,m}}{\sqrt{\theta^{c,\cdot}_{n,m}}} \tag{3.34}$$

Soit $\Lambda=\{\lambda_i\}_{i\in[1,\ldots,|\Lambda|]}$ et $V=\{v_i\}_{i\in[1,\ldots,|V|]}$; Pour chaque document *d* et une classe *c*, un vecteur $v^{c,d}$ correspondant à la représentation du document dans l'espace *Fisher* est obtenu.

$$\left\langle \frac{N^{d,c}_{\lambda_1,\lambda_1}}{\sqrt{\theta^{c,s}_{\lambda_1,\lambda_1}}} \cdots \frac{N^{d,c}_{\lambda_{|\Lambda|},\lambda_1}}{\sqrt{\theta^{c,s}_{\lambda_{|\Lambda|},\lambda_1}}} \frac{N^{d,c}_{\lambda_1,\lambda_2}}{\sqrt{\theta^{c,s}_{\lambda_1,\lambda_2}}} \cdots \frac{N^{d,c}_{\lambda_{|\Lambda|},\lambda_{|\Lambda|}}}{\sqrt{\theta^{c,s}_{\lambda_{|\Lambda|},\lambda_{|\Lambda|}}}} \middle| \frac{N^{d,c}_{v_1,\lambda_1}}{\sqrt{\theta^{c,t}_{v_1,\lambda_1}}} \cdots \frac{N^{d,c}_{v_{|\Lambda|},\lambda_{|\Lambda|}}}{\sqrt{\theta^{c,t}_{v_{|\Lambda|},\lambda_{|\Lambda|}}}} \right\rangle \tag{3.35}$$

Le score Fisher de la probabilité structurelle — Le score Fisher de la probabilité textuelle

5.3.4. Considération de différents types d'information console

En plus du texte, le modèle proposé peut être utilisé pour n'importe quel autre contenu. En utilisant le modèle génératif pour les images. La probabilité de jointure est :

$$P(c,d) = P(c)\left(\prod_{i=1}^{|d|} P(s_d^i \mid pa(s_d^i),c)\right)\left(\prod_{i=1/s_d^i\in\Lambda}^{|d|} P_{text}(t_d^i \mid s_d^i,c)\right)\left(\prod_{i=1/s_d^i=I}^{|d|} P_{image}(t_d^i \mid s_d^i,c)\right) \tag{3.36}$$

$P_{text}(t_d^i \mid s_d^i,c)$ est la probabilité textuelle et $P_{image}(t_d^i \mid s_d^i,c)$ est la probabilité d'image.

$$P_{image}(t_d^i \mid s_d^i,c) = \prod_{k=1}^{N_{color}} P(p_{d,k}^i \mid s_d^i,c), \forall \sum_{k=1}^{N_{color}} P_{d,k}^i = N_p \tag{3.37}$$

Toutes les images sont fixées à N_p pixels et une image est représentée par un histogramme qui consiste en N_{color} couleurs. $P(p_{d,k}^i \mid s_d^i,c)$ est la probabilité pour que l'image du nœud *i* contient $p_{d,k}^i$ pixels de couleur *k*.

5.3.5. Discussion

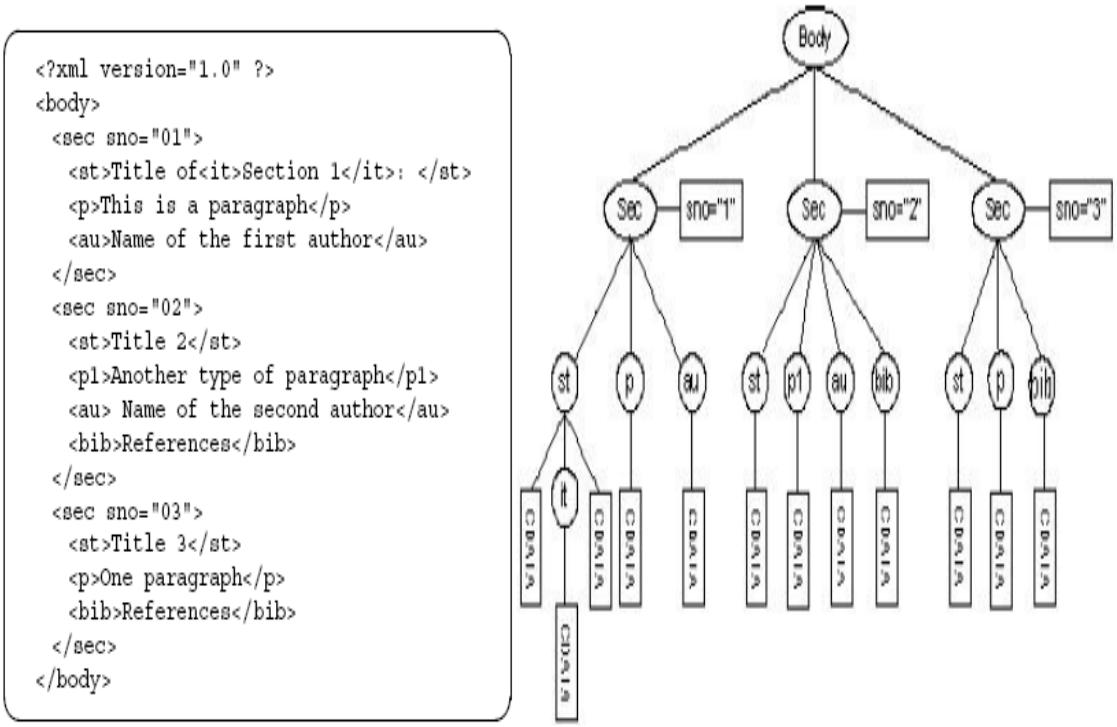
Les auteurs ont proposé un modèle d'apprentissage général pour la classification de documents structurés permettant de prendre en compte simultanément la structure et le contenu. Pour cela, ils définissent tout d'abord un modèle génératif de documents structurés à l'aide des réseaux bayésiens. Ensuite, ils transforment ce modèle génératif en un modèle discriminant en utilisant la méthode du noyau de *Fisher*. Le modèle peut être étendu facilement pour prendre en compte des types différents d'information comme la classification multimédia.

La profondeur du document peut être un autre paramètre du modèle. La longueur du document et la profondeur des distributions sont omises dans ce modèle.

5.4. Les travaux de Vercoustre et al. **[119]**

Les auteurs proposent un modèle de représentation de documents XML en vue du *clustering* qui permet de prendre en compte soit la structure seule, soit la structure et le contenu des documents. L'idée consiste à représenter un document XML par un ensemble des chemins générés à partir de son arbre. Le terme chemin est utilisé pour désigner indifféremment les chemins et les sous-chemins. Un chemin est terminal s'il se termine par une feuille. Il est initial s'il commence par le nœud racine. Un chemin textuel est un chemin prolongé par un mot du contenu textuel associé à un nœud.

Un exemple d'un document XML et de son arbre correspondant est donné dans la figure suivante :




```
<?xml version="1.0" ?>
<body>
 <sec sno="01">
  <st>Title of<it>Section 1</it>: </st>
  <p>This is a paragraph</p>
  <au>Name of the first author</au>
 </sec>
 <sec sno="02">
  <st>Title 2</st>
  <p1>Another type of paragraph</p1>
  <au> Name of the second author</au>
  <bib>References</bib>
 </sec>
 <sec sno="03">
  <st>Title 3</st>
  <p>One paragraph</p>
  <bib>References</bib>
 </sec>
</body>
```

Figure 3.12 : Le document Test.xml et sa représentation sous forme d'arbre [120].

Les tableaux 2 et 3 présentent quelques ensembles de chemins avec leur fréquence, pour l'exemple de la figure 3.12.

Tableau 3.2 : Chemins non textuels de longueur 3 (@ indique un attribut) [120].

Mots	Tf
body.sec.bib	2
body.sec.st	2
body.sec.p	3
body.sec.au	2
body.sec.sno@	3

Tableau 3.3 : Chemins textuels de longueur 4 [120].

Mots	Tf
body.sec.au."name"	2
body.sec.p."paragraph"	3
body.sec.bib."refer"	2
body.sec.sno@."01"	1
body.sec.st."tit"	3
body.sec.sno@."03"	1
body.sec.p."type"	1
body.sec.au."author"	2
body.sec.sno@."02"	1

Avant d'appliquer le *clustering* sur les chemins, une étape de prétraitement est nécessaire.

5.4.1. Prétraitement

Le prétraitement consiste à réduire le nombre des chemins générés en essayant de diminuer successivement le nombre de balises, de mots et de chemins différents. Le filtrage sur les balises utilise une connaissance sémantique de ces balises.

Enfin réduire le nombre final des chemins en utilisant les fréquences relatives TF-IDF de ces chemins. C'est le nombre d'occurrences du chemin dans un document sur leur nombre d'occurrences du chemin dans toute la collection. Les chemins trop fréquents ou trop rares sont éliminés.

5.4.2. *Clustering*

Le *clustering* dans ces travaux est basé sur l'algorithme proposé par **[21]**. Où la distance entre les clusters est basée sur les fréquences des mots du vocabulaire choisi. Le principe est proche de l'algorithme des *k-means*. Les clusters sont représentés par des prototypes qui résument, au mieux, l'information (les chemins) des documents appartenant à chacun des clusters.

Si le vocabulaire choisi est constitué de *p* mots alors chaque document *s* est représenté par le vecteur $x_s = (x_s^1,...,x_s^j,...,x_s^p)$ où x_s^j est le nombre d'occurrences du mot *j* dans le document *s*, alors le prototype *g* d'un cluster U_i est représenté par $g_i = (g_i^1,...,g_i^j,...g_i^p)$ où g_i^j est calculé par $g_i^j = \sum_{s \in U_i} x_s^j$.

Le prototype de chaque cluster étant fixé, chaque document est alors affecté au cluster dont la proximité du prototype avec ce document est la plus petite. La proximité est calculée par la distance euclidienne entre les distributions associées aux documents et aux prototypes.

5.4.3. Discussion

Vercoustre et al. proposent de représenter les documents XML par des ensembles de leurs chemins. Un algorithme de *clustering* tel que *k-means*, est appliquée sur ces ensembles. Une phase de prétraitement est nécessaire afin de réduire le nombre des chemins générés en essayant de diminuer successivement le nombre de balises, de mots et de chemins différents.

Dans cette phase, l'approche suppose la connaissance d'une sémantique implicite des éléments pour les sélectionner, mais ce n'est pas toujours le cas.

Du fait de la taille du vocabulaire généré, il n'est pas possible de faire un *clustering* sur la structure et le contenu d'une grande collection de documents XML.

5.5. Les travaux de Xing et al. **[124]**

L'approche proposée applique la classification des documents XML basée sur la similarité en utilisant le calcul de distance entre les documents XML et les schémas et en combinant la structure et le contenu pour la classification. Nous allons présenter dans ce qui suit les différentes étapes de cette approche.

5.5.1. Représentation des documents XML

Un document XML est représenté par un arbre étiqueté ordonné. Les nœuds sont dénombrés en basant sur un parcours post-ordre[1].

Une DTD est convertie en une grammaire NRHG (*Normalized Regular Hedge Grammar*) **[125]** qui a la forme d'un 5-uplet $(\sum, V_T, V_F, P, s)$ où :

- $\sum$ est un ensemble fini de terminaux (un terminal est utilisé pour étiqueter les nœuds de l'arbre).
- V_T est un ensemble fini de variables de l'arbre (les symboles de la grammaire pour générer les arbres).
- V_F est un ensemble fini de variables de la forêt (elles sont utilisées pour générer des forêts des séquences d'arbres qui correspondent à un string de variables de l'arbre).
- $s \in V_T$ est l'axiome.
- P est un ensemble de règles de production pour générer les arbres.

Si un arbre étiqueté ordonné n'est pas conforme à une NRHG, il peut être transformé avec une séquence des opérations d'édition tel que l'arbre résultant doit être conforme à la NRHG. Les mêmes opérations d'édition de **[108]** sont utilisées (insérer, supprimer et remplacer un nœud feuille de l'arbre). La distance d'édition entre un arbre et une NRGH est le coût minimal de la séquence des opérations d'édition.

5.5.2. Calcul de distance d'édition entre le document XML et le schéma

La distance d'édition de l'arbre est utilisée pour la mesure de similarité. Pour calculer la distance entre un arbre étiqueté ordonné et une NRHG, l'algorithme de calcul de distance d'édition de **[125]** est utilisé. L'algorithme prend en entrée une

[1] Le parcours post-ordre consiste à parcourir les sous arbres de droite à gauche **[6]**.

grammaire NRHG et un arbre étiqueté ordonné. Deux matrices sont retournées en sortie.

5.5.3. Classification

Pour obtenir un classifieur, l'approche suit les étapes suivantes :

- Dans la première étape, l'extraction des schémas à partir des documents XML est réalisée en inférant une expression régulière *R* d'une collection de séquences *I* avec des restrictions : *R* est simple, petite de taille, suffisamment générale pour accepter les séquences similaires et pour exclure les séquences non similaires de ceux qui sont dans *I*. Inférer une expression régulière à partir d'un ensemble de string est étudié dans **[25]** et **[52]**. Les auteurs de **[52]** introduisent la notion de MLD (*Minimum Length Description*). Avec MLD, le meilleur schéma (grammaire) est celui qui minimise la somme de la longueur de la grammaire L_g et la longueur de la donnée L_d lorsqu'elle est encodée avec la grammaire. Ce principe est utilisé dans l'approche courante. La méthode de **[113]** est aussi utilisée pour construire un automate fini non déterministe NFA (*non deterministic finite automata*).

- La deuxième étape consiste à calculer la distance entre les documents et les schémas pour chaque classe. Supposant que la collection contient *n* classes d'objets, un seul schéma est généré pour chaque classe. Pour chaque document, un vecteur de distances $(d_1, d_2, ..., d_n)$ est calculé. Il représente la distance entre chaque document et le centre des points de la classe. Pour le contenu, un vecteur est calculé en mesurant la similarité du contenu texte des documents XML avec des techniques standard telles que :

  - Le modèle d'espace vectoriel (VSM).
  - LSI (*Latent Semantic Indexing*) : regroupe les termes reliés sémantiquement (co-occurrence,…) dans une même dimension et se base sur la technique d'analyse de données SVD (*Singular Value Decomposition*).

Les vecteurs de la structure et du contenu peuvent être concaténés et être utilisés pour représenter un document XML.

- La troisième étape est la classification. Plusieurs packages logiciels ont été implémentés pour la classification des données. Dans ces travaux, le logiciel *Weka*[1] basé sur VSM est utilisé.

[1] WEKA Project, http://www.cs.waikato.ac.nz/ml/weka/

5.5.4. Discussion

Les auteurs proposent une méthode de classification basée sur le calcul de la distance d'édition entre un document XML représenté par un arbre étiqueté ordonné et un schéma représenté par une grammaire NRHG.

Le schéma qui représente un groupe de documents XML dépend non seulement de la méthode d'extraction du schéma mais aussi des documents sélectionnés pour l'extraction du schéma. Comment sélectionner les documents, qui peuvent capturer les propriétés structurelles de chaque classe, est une tâche importante.

Ce système fonctionne très bien pour deux classes. Cependant, la performance du système se dégrade considérablement quand le nombre de classes devient grand.

5.6. Les travaux de Doucet et al. **[42]**

Le modèle d'espace vectoriel (VSM) est utilisé dans cette approche. Un document XML est représenté par un vecteur de *N*-dimensions dans un espace vectoriel. *N* est le nombre de composants du document. Le vecteur contient le poids de chaque composant dans le document. Le poids est spécifié par TF-IDF qui est une combinaison du nombre d'occurrences du terme dans le document et de la valeur inverse du nombre de documents dans lesquels il est présent. Le *clustering* est enfin appliqué sur les vecteurs.

5.6.1. Représentation des documents

L'approche prend en considération le texte et son balisage structurel. Elle combine le texte et les balises en un seul vecteur en fusionnant les mots et les noms de balises « *text+tags* » à partir des deux vecteurs suivants :

- Le texte seulement « *text* » : dont il faut supprimer les mots vides et les suffixes des mots en utilisant l'algorithme de **[97]**. La dimension du vecteur représente les termes des mots réduits.
- Les balises seulement « *tags* » : la dimension du vecteur représente les balises d'un document XML.

Deux autres techniques sont aussi utilisées :

- La première était proposée en 2002 (*2-step approach*) « *tags then text* » **[41]**. Son algorithme a comme entrée les documents de la collection, le nombre des

clusters souhaités *n* et le seuil de similarité interne σ. Le premier pas de l'algorithme consiste à faire le *clustering* avec les balises. Pour ce faire, *k-means* est appliqué avec *k* = *n*, puis les *m* clusters qui ont une similarité interne supérieure à σ sont gardés. Le deuxième pas est le *clustering* du texte. *k-means* est appliqué avec *k* = *n*-*m*. À la fin les *m* clusters basés sur les balises et les *n*-*m* clusters basés sur le texte sont combinés pour former le *n*-cluster final.

- La deuxième technique « *text+T/E* » consiste à utiliser un indicateur structurel T/E (*Text nodes/Element nodes*) de la proportion du contenu mixé dans un fragment XML. Selon **[76]** et **[77]**, la mesure T/E est la probabilité du texte d'un élément XML, basée sur le fait qu'un élément peut être exclu d'un index de texte. La valeur de T/E est intégrée comme une dimension additionnelle au vecteur qui représente un document XML.

5.6.2. *Clustering*

L'algorithme de *k-means* est utilisé avec *k* un paramètre défini en entrée qui indique le nombre de clusters désirés. Pour faciliter la comparaison, *k* est défini comme un nombre de classes désirées. L'algorithme est appliqué sur une partition initiale de la collection qui est ajustée à plusieurs reprises jusqu'à trouver une solution stable.

5.6.3. Discussion

Les auteurs étudient le problème de la classification non supervisée des documents XML. Ils proposent plusieurs représentations des documents XML telles que l'utilisation d'un sac de balises (*bag of structure*). L'utilisation d'un sac de mots (*bag of words*) et la combinaison du texte et des balises. L'algorithme *k-means* est appliqué sur ces représentations.

La qualité du *clustering* augmente avec l'utilisation des composants texte et l'ajout du vecteur structurel ne permet pas d'améliorer de manière significative les résultats du *clustering*.

Une faiblesse de cette technique est que la structure arborescente des éléments est ignorée et les mots ne sont pas reliés à leur chemin dans un arbre XML.

5.7. Les travaux de Knijf **[67]**, **[68]**

FAT-CAT (*Frequent Attribute Trees based Classification*) est une méthode de classification des arbres XML proposée dans les travaux de Knijf. Elle est basée sur

l'algorithme *FAT-miner* qui est utilisé pour la découverte de pattern fréquent dans les arbres de données structurées en prenant en considération les attributs.

Un arbre d'attributs enraciné étiqueté ordonné $T= \{V, E , \leq , L , v_0 , M\}$ est un graphe connexe acyclique directe qui contient un ensemble de nœuds V et un ensemble d'arcs E.

$L : V \rightarrow \Sigma$ est la fonction étiquetée qui assigne les étiquettes de l'alphabet Σ aux nœuds dans V. v_0 est le nœud racine de l'arbre.

$\leq \subset V^2$, $\leq$ est une relation binaire qui représente un ordre entre les frères.

L'ensemble des attributs est défini par : $A = \{a_1, \ldots, a_n\}$ avec un ordre entre les attributs (par exemple : $a_j < a_k$).

Pour chaque nœud v dans V, un sous ensemble non vide M de A est assigné, c'est l'ensemble des attributs de $v : M : V \rightarrow P(A) \setminus \{\varnothing\}$.

Pour deux arbres enracinés étiquetés T_1 et T_2, T_2 est un sous arbre de T_1 et T_1 est un super-arbre de T_2, noté par $T_2 \leq T_1$, s'il existe une fonction injective d'appariement Φ de V_{T2} dans V_{T1} satisfaisant les conditions pour tout v, v_1, $v_2 \in V_{T2}$:

1. Φ préserve les étiquettes : $L_{T2}(v) = L_{T1}(\Phi(v))$.
2. Φ préserve l'ordre entre les frères : si $v_1 \leq_{T2} v_2$ alors $\Phi(v_1) \leq_{T1} \Phi(v_2)$.
3. Φ préserve la relation parent-fils (v_1 est un parent de v_2) : $(v_1, v_2) \in E_{T2}$ ssi $(\Phi(v_1), \Phi(v_2)) \in E_{T1}$.
4. Φ préserve les attributs: $\forall v \in V_{T2} : M(v) \subseteq M(\Phi(v_2))$.

Dans la section suivante, nous allons voir les étapes de classification de la méthode FAT-CAT.

5.7.1. Classification avec FAT-CAT

La procédure globale de la classification des documents XML est comme suit :

- Calculer les patterns fréquents pour les différentes classes sur l'ensemble d'apprentissage en utilisant l'algorithme *FAT-Miner* : Soit $D = \{d_1, \ldots, d_m\}$ une base de données où chaque enregistrement d_i est un arbre enraciné étiqueté ordonné. Soit $C=\{c_1, \ldots, c_k\}$ les k classes de données. Avec D_{ci} est l'ensemble d'enregistrements dans la base de données qui a la classe d'étiquette c_i. Un arbre enraciné étiqueté ordonné T apparait dans d_i s'il est un sous arbre de d_i. Le support de T dans une

classe c_i est défini par $\psi(T/c_i) = \Sigma_{d \in Dci} \sigma_d(T)$, qui est le nombre d'enregistrements de la classe c_i dans lesquels T apparait une ou plusieurs fois avec $\sigma_{di}(T) = 1$ si $T \leq d_i$ et 0 sinon. T est fréquent dans la classe c_i si $\psi(T/c_i)/|D_{ci}| \geq minsup$ (support minimum défini par l'utilisateur). Le but est de déterminer tous les sous arbres qui apparaissent fréquemment dans les classes.

- Sélectionner le pattern émergeant **[40]** de ces patterns fréquents : le pattern émergeant est celui que son support augmente significativement d'une classe à une autre. Pour ce faire, il faut mesurer en observant le pattern, quel facteur modifie la probabilité de la classe comparée par la probabilité de la classe antérieure ($P(class|T)/P(class)$).
- Le pattern émergeant est utilisé pour apprendre à faire le modèle de classification sur l'ensemble de test en construisant des composants binaires pour chaque document XML. Chaque composant indique la présence ou l'absence d'un pattern émergeant dans un enregistrement. L'implémentation de **[8]** est utilisée pour construire des arbres de décisions. À chaque nœud de l'arbre de décision, un nombre d'attributs qui discrimine bien entre les classes est choisi.
- Évaluer le modèle de classification sur l'ensemble de test.

5.7.2. Discussion

À partir d'un ensemble d'arbres de documents XML, cette approche de classification consiste à trouver tous les sous arbres qui apparaissent au moins *n*% dans les enregistrements de données en prenant en considération les attributs. *n* est la contrainte du support minimum que l'utilisateur doit spécifier.

Pour spécifier une valeur appropriée du support minimum de chaque classe, il faut donner une valeur très élevée et la diminuer graduellement jusqu'à ce qu'un nombre suffisant de pattern de qualité soit produit.

6. Comparaison des approches de fouille de contenu XML

En tenant compte des points discutés précédemment dans l'étude des approches de fouille dans le contenu, nous constatons que nous pouvons comparer ces approches en utilisons les mêmes huit critères de comparaison utilisés pour comparer les approches de fouille dans la structure XML. Seulement pour le quatrième critère « Modèle », où nous avons utilisé les quatre modèles de

représentation proposés par Denoyer et al. **[38],** nous ajoutons un autre modèle vu qu'il y a des travaux récents qui utilisent ce modèle. Le modèle est :

- Modèle basé sur les réseaux bayésiens.

À l'aide de ces critères, le tableau ci-dessous présente une comparaison des approches de fouille dans le contenu des documents XML.

Tableau 3.4 : Comparaison des approches de fouille dans le contenu des documents XML.

	Approche	Type de documents	Entrées	Tâche de fouille	Modèle	à priori	Séman-tique	Échanti-llonnage	Résultats
Structure et contenu	**Yang et al. 2002**	Instances XML	Structured Link Vector	*Clustering*	Modèle basé sur les vecteurs	*			Clusters
	Braga et al. 2002	Instances XML	Table relationnelle	Association	Modèle d'arbres fréquents	*			Document XML contenant les règles d'association
	Denoyer et al. 2004	Instances XML	Arbres	Classification	Modèle basé sur les réseaux bayésiens			*	Classes
	Vercoustre et al. 2006	Instances XML	Ensembles de chemins	*Clustering*	Modèle basé sur les vecteurs	*	*		Clusters
	Xing et al. 2006	Instances et schémas XML	Arbre étiqueté ordonné et grammaire NRHG	Classification	Modèle basé sur la similarité			*	Classes
	Doucet et al. 2006	Instances XML	Vecteurs	*Clustering*	Modèle basé sur les vecteurs	*			Clusters
	Knijf 2007	Instances XML	Arbre d'attributs enraciné étiqueté ordonné	Classification	Modèle d'arbres fréquents	*		*	Arbres de décision

7. Conclusion

Ce chapitre a permis de voir l'évolution des travaux existants dans le domaine de *XML mining* en faisant une analyse approfondie de l'état de l'art dans ce domaine et en parcourant un nombre important des travaux. Des 18 travaux différents étudiés dont 11 concernent la fouille dans la structure et 7 concernent la fouille dans le contenu des documents XML, nous pouvons déduire quelques remarques importantes :

- La fouille dans le contenu reste encore un volet moins traité par rapport à celle dans la structure. Cependant, les techniques de *text mining* peuvent étendre les techniques de *data mining* pour le contenu textuel. La combinaison des deux, peut donner des résultats plus performants.
- La plupart des travaux sont adaptés à la fouille sur les instances des documents XML bien que les grammaires XML (DTD ou schéma XML) sont mieux adaptés pour la fouille dans la structure, ce qui est montré dans les travaux de Nayak et al. 2 **[92]**.
- Le modèle basé sur la similarité est utilisé par la majorité des travaux de fouille dans la structure. Ces derniers utilisent la distance d'édition malgré que les expériences ont montré que cette distance est très coûteuse par rapport à d'autres.
- Plusieurs travaux proposent une méthodologie de *clustering* regroupant ensemble tous les documents XML les plus similaires. La classification est aussi utilisée pourtant elle nécessite un échantillon important pour un bon apprentissage c.à.d. elle doit traiter une riche collection d'exemples afin de faciliter la prédiction des classes dans la phase de test. L'association est la moins utilisée dans la fouille des documents XML.
- Un aspect primordial pour la qualité de la fouille surtout dans le contenu textuel est l'aspect sémantique. Celui-ci est ignoré dans la majorité des travaux étudiés.
- Malgré son importance, la structure arborescente des éléments (la notion du chemin…) est parfois ignorée.
- Des connaissances à priori sont nécessaires dans la plupart des approches pour appliquer la fouille telles que la supposition d'une sémantique implicite des éléments, des paramètres à spécifier par les utilisateurs et des dictionnaires définis par les experts du domaine…etc.

Dans le chapitre suivant, nous allons présenter notre approche de fouille dans les documents XML. Cette approche permet la fouille aussi bien dans la structure que dans le contenu des documents XML. Elle s'inspire de certains travaux présentés dans ce mémoire.

CHAPITRE 4
F-CheX : UNE APPROCHE DE FOUILLE DANS LES DOCUMENTS XML

1. Introduction

Dans le chapitre précédant, nous avons réalisé une analyse approfondie de l'état de l'art du domaine de *XML mining*. En prenant comme base cette analyse, nous allons proposer dans ce chapitre une nouvelle approche de *XML mining*. Celle-ci prend en considération la structure et le contenu. Elle est basée principalement sur la représentation des documents XML par des chemins. L'approche fait appel à des notions importantes pour gérer l'aspect sémantique des mots tel qu'un thésaurus de mots.

2. *F-CheX*, une approche de fouille par clustering des Chemins XML

F-CheX (Fouille dans les Chemins XML) est une nouvelle approche qui consiste à effectuer le *clustering* sur les documents XML. Ces derniers sont représentés par des ensembles de chemins en prenant en compte simultanément la structure et le contenu et en faisant appel à un thésaurus créé au préalable pour unifier les mots de chemins.

L'approche *F-CheX* basée sur le modèle des vecteurs. Elle s'inspire des travaux présentés dans Vercoustre et al. **[119]** qui ont proposé le *clustering* des documents XML représentés par des ensembles de chemins générés à partir de leurs arbres en ajoutant le contenu textuel et les attributs comme des nœuds de l'arbre. L'utilisation des chemins permet de conserver la structure arborescente des éléments composant un document XML.

L'originalité de notre approche réside dans l'utilisation d'un thésaurus pour gérer l'aspect sémantique des mots présents dans la collection XML. Le thésaurus est créé à partir de deux sacs de mots générés de la collection XML. Un sac pour le contenu structurel et un autre pour le contenu textuel. Nous générons les deux sacs en se basant sur l'approche de Doucet et al. **[42]**. Ces auteurs proposent plusieurs représentations des documents XML telles que l'utilisation d'un sac de

balises (*bag of tags*), l'utilisation d'un sac de mots (*bag of words*) et la combinaison du texte et des balises (*text+tags*) pour représenter les documents XML.

En utilisant le thésaurus nous unifions tous les termes synonymes dans l'ensemble des chemins générés. Afin de faciliter l'étape du *clustering*, l'ensemble des chemins est mappé dans une matrice binaire sur laquelle une méthode de *clustering* est alors appliquée pour organiser automatiquement la collection XML en des petits sous ensembles homogènes. L'algorithme *k-means* **[82]** est utilisé vu sa simplicité et sa robustesse.

Donc globalement, l'approche *F-CheX* est constituée de trois grandes phases dont chacune est composée de différentes étapes. La figure 4.1 présente ces trois phases qui sont : la construction d'un thésaurus, la génération de la matrice des chemins et enfin le *clustering*. Les trois phases se résument comme suit :

Phase A : Construction d'un thésaurus

1. Construction des sacs de mots

2. Traitement des sacs de mots

Phase B : Génération de la matrice des chemins

3. Arbre XML

4. Arbre réduit

5. Matrice des chemins

Phase C : Clustering

6. Application d'un algorithme de *clustering* sur la matrice des chemins.

Dans ce qui suit, nous allons détailler les différentes phases et étapes constituant l'approche *F-CheX.*

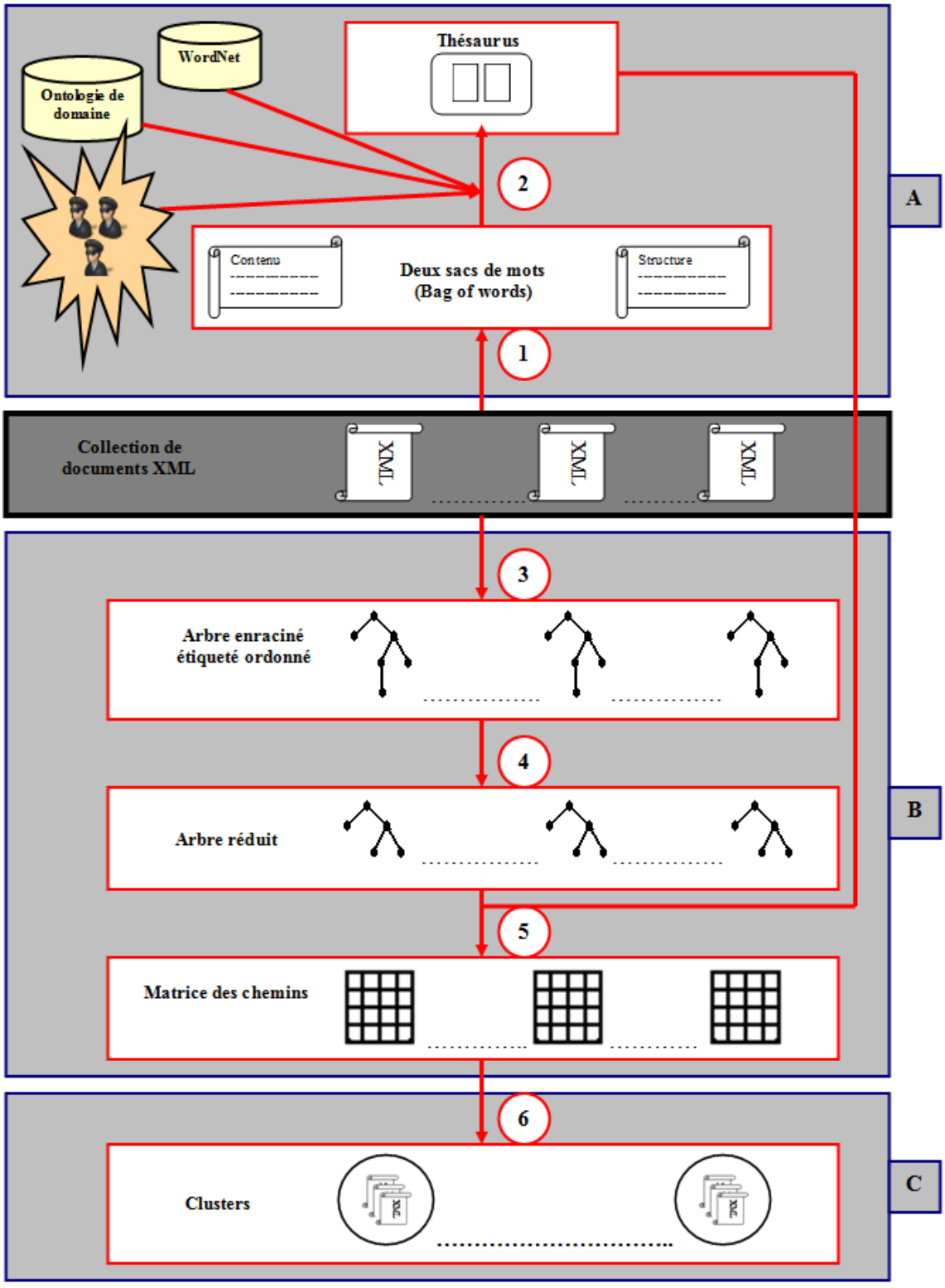


Figure 4.1 : Schéma global de *F-CheX*.

3. Construction d'un thésaurus

Afin de gérer l'aspect sémantique des mots dans une collection de documents XML, nous proposons de définir dans cette phase un thésaurus de mots. Le thésaurus contient deux sacs de mots (*bag of words*). Chaque sac de mots comporte un certain nombre d'éléments. Un élément est composé d'un ou de plusieurs termes. Les éléments sont de trois types : maître, esclave et neutre. La construction d'un tel thésaurus passe par les deux étapes suivantes :

3.1. Construction des sacs de mots

Doucet et al. **[42]** proposent plusieurs représentations des documents XML telles que l'utilisation d'un sac de balises (*bag of tags*), l'utilisation d'un sac de mots (*bag of words*) et la combinaison du texte et des balises (*text+tags*) pour représenter les documents XML. Ils appliquent ensuite directement l'algorithme *k-means* sur ces représentations.

Dans notre cas, nous proposons de construire deux sacs de mots à partir de toute la collection des documents XML : sac de structure et sac de contenu. Les deux sacs contiennent respectivement les mots des balises (structure) et ceux du contenu des balises (contenu) :

- **Le sac de structure** : contient des éléments qui représentent les termes de toutes les balises présentes dans la collection ;
- **Le sac de contenu** : contient des éléments qui sont le contenu textuel entre les balises XML.

Les éléments des deux sacs peuvent être composés d'un ou de plusieurs termes. Par exemple, l'élément « Bibliothèque » est composé d'un seul terme alors que l'élément « XML cours et exercices » est composé de quatre termes.

3.2. Traitement des sacs de mots

Les deux sacs de mots construits dans l'étape précédente sont alors traités. Ils permettent de constituer un thésaurus excluant toute sorte de redondances telle que chaque élément n'apparaisse qu'une seule fois dans un sac de mots. Le thésaurus contient trois types d'éléments que nous définissons comme suit :

- **Un élément maître** : les éléments différents et ayant la même sémantique sont regroupés sous un seul élément appelé maître. L'élément maître possède un ou plusieurs éléments synonymes.
- **Un élément esclave** : les éléments synonymes qui sont regroupés sous un élément maître sont appelés des éléments esclaves.
- **Un élément neutre** : un élément qui n'est ni maître ni esclave est appelé un élément neutre.

Pour définir le thésaurus en utilisant les deux sacs de mots, le traitement des éléments fait appel à :

- une ontologie du domaine si elle existe ou à d'autres ontologies telles que *WordNet*, puisque elle est disponible gratuitement sur internet.
- ou des avis d'experts du domaine.

La figure suivante illustre bien un exemple d'un thésaurus :

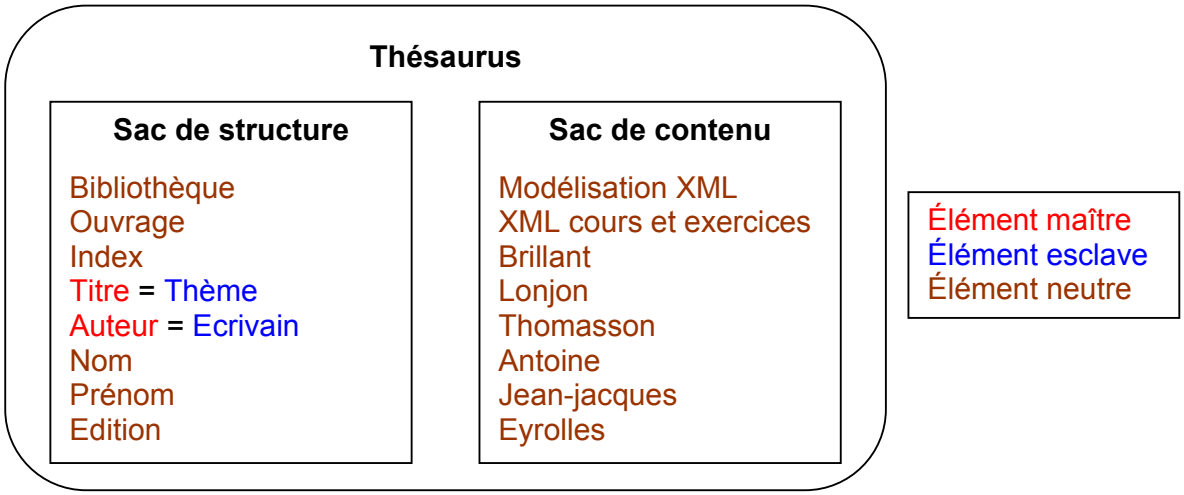


Figure 4.2 : Un exemple d'un thésaurus.

Dans le sac de structure, l'élément « Titre » est un élément maître, « Thème » est son élément esclave. De même, pour l'élément « Ecrivain » est un élément esclave de l'élément maître « Auteur ». Les éléments restants sont des éléments neutres. Tous les éléments du sac de contenu sont des éléments neutres.

Nous allons présenter maintenant la deuxième phase de notre approche qui consiste à générer la matrice des chemins.

4. Génération de la matrice des chemins

Cette phase consiste à générer un ensemble des chemins à partir de chaque document XML en le transformant en un arbre réduit et en utilisant le thésaurus construit à l'avance pour pouvoir unifier tous les termes utilisés lors de la construction de l'ensemble des chemins. Les ensembles des chemins sont ensuite mappés en une matrice des chemins pour toute la collection XML. Les trois étapes constituant cette phase sont décrites ci-dessous.

4.1. Arbre XML

Puisque les documents XML ont une structure arborescente, nous représentons un document XML par un arbre enraciné étiqueté ordonné **[47]**, **[1]**, **[30]**, **[124]**, **[67]** en prenant en considération la structure et le contenu.

```
<?xml version="1.0" encoding="ISO-8859-
1"?>
<Bibliothèque>
    <Ouvrage>
        <Index valeur="1"/>
        <Titre> Modélisation XML</titre>
        <Auteur>
            <Nom>Lonjon</Nom>
            <Prenom>Antoine</Prenom>
        </Auteur>
        <Auteur>
            <Nom>Thomasson</Nom>
            <Prenom>jean-jacques</Prenom>
        </Auteur>
        <Edition>Eyrolles</Edition>
    </Ouvrage>
    <Ouvrage>
        <Index valeur="2"/>
        <Titre> XML Cours et exercices
</Titre>
        <Auteur>
            <Nom>Brillant</Nom>
            <Prenom>Allexandre</Prenom>
        </Auteur>
        <Edition>Eyrolles</Edition>
    </Ouvrage>
</Bibliothèque>
```

Figure 4.3 : Le document XML « Bibliothèque».

Un arbre enraciné est un ensemble de nœuds et un ensemble d'arcs satisfaisant les propriétés **[6]** :

- Il existe un nœud particulier appelé racine.
- Tout nœud *n*, autre que la racine, est relié par un arc à un antécédent unique *p* appelé père. *n* est appelé fils de *p*.
- On peut atteindre la racine à partir de n'importe quel nœud de l'arbre en se déplaçant de père en père.

Un arbre est étiqueté s'il existe une injection de l'ensemble des nœuds dans un ensemble d'étiquettes. Une étiquette d'un nœud est un élément d'un alphabet fini **[111]**.

Un arbre est ordonné s'il existe un ordre parmi les fils de chaque nœud **[6]**.

Un arbre enraciné étiqueté ordonné *T* est un *n-uplet* $T=<n_0, N, A, \delta>$ **[47]**, où :

- n_0 est le nœud racine de l'arbre *T*.
- $N \subseteq E$ est l'ensemble des nœuds de l'arbre *T* dont *E* est l'ensemble d'étiquettes regroupant les noms des balises, les termes du contenu textuel et les attributs. Une étiquette d'un nœud est un élément d'un alphabet fini $\sum$.
- $A \subseteq N \times N$ est l'ensemble des arcs.
- δ est une fonction étiquetée qui assigne à chaque nœud $n \in N$ de l'arbre *T* une étiquette $e \in \sum$ tel que $\delta(n) = e$.

Nous représentons les nœuds structurels (les balises) par des cercles, les nœuds textuels (contenu) par des rectangles et les nœuds attributs par des triangles. La figure 4.4 illustre la représentation du document XML de la figure 4.3 par un arbre enraciné étiqueté ordonné.

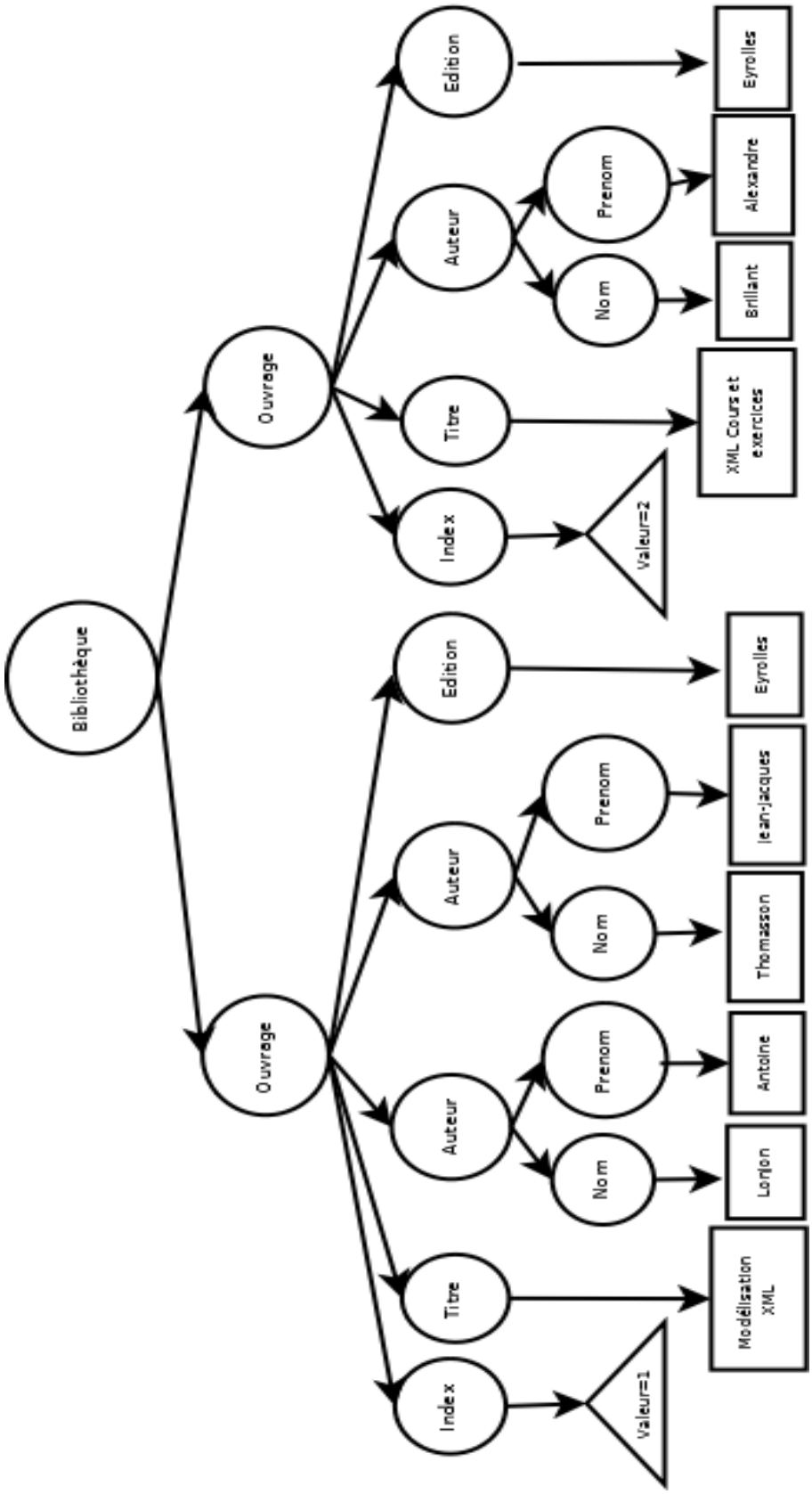


Figure 4.4 : Représentation du document XML « Bibliothèque» par un arbre enraciné étiqueté ordonné.

Bibliothèque, *Ouvrage*, *Index*, *Titre*, *Auteur*, *Nom*, *Prenom* et *Edition* sont les nœuds structurels. *valeur="1"* et *valeur="2"* sont les nœuds attributs. *Modélisation XML*, *Lonjon*, *Antoine*, *Thomasson*, *jean-jacques*, *Eyrolles*, *XML Cours et exercices Brillant*, *Allexandre* et *Eyrolles* sont les nœuds textuels.

4.2. Arbre réduit

Dans cette étape, nous proposons de réduire chaque arbre enraciné étiqueté ordonné qui représente le document XML en étendant la notion du résumé structurel utilisée dans les travaux de **[30]**, où un arbre représente seulement la structure d'un document XML. Le résumé de l'arbre dans notre cas consiste à construire un arbre réduit en prenant en considération les trois types de nœuds : les nœuds structurels, les nœuds textuels et les nœuds attributs. L'idée est d'éliminer tous les nœuds dupliqués de l'arbre enraciné étiqueté ordonné construit en utilisant l'algorithme *ReduiceTree*.

Dans un arbre, un nœud dupliqué est un nœud qui possède un chemin dupliqué. Ce dernier doit commencer du nœud racine de l'arbre. En supprimant l'un des deux nœuds dupliqués, si celui-ci possède des fils, il faut affecter ces derniers à l'autre nœud maintenu.

L'algorithme qui permet de réduire un arbre enraciné étiqueté ordonné est donné ci-dessous ; Il est dénommé *ReduiceTree*.

Algorithme *ReduiceTree*

```
Entrée : Arbre T
Sortie : Arbre réduit
n0 : nœud racine
ni , nj : nœud

Parcourir T
Si il existe deux nœuds égaux ni et nj
   Alors
    Vérifier les chemins des deux nœuds jusqu'à la racine n0
    Si les chemins sont égaux
      Alors
       Si  ni est un nœud feuille
          Alors
            Supprimer ni
          Sinon
           Affecter les fils de ni à  nj
           Supprimer ni
       Fin Si

   Fin Si
Fin Si
```

Dans l'arbre de la figure 4.4, nous constatons qu'il possède les nœuds dupliqués : *Ouvrage*, *Index*, *Titre*, *Auteur*, *Nom*, *Prenom*, *Edition* et *Eyrolles*. Son arbre réduit est le suivant :

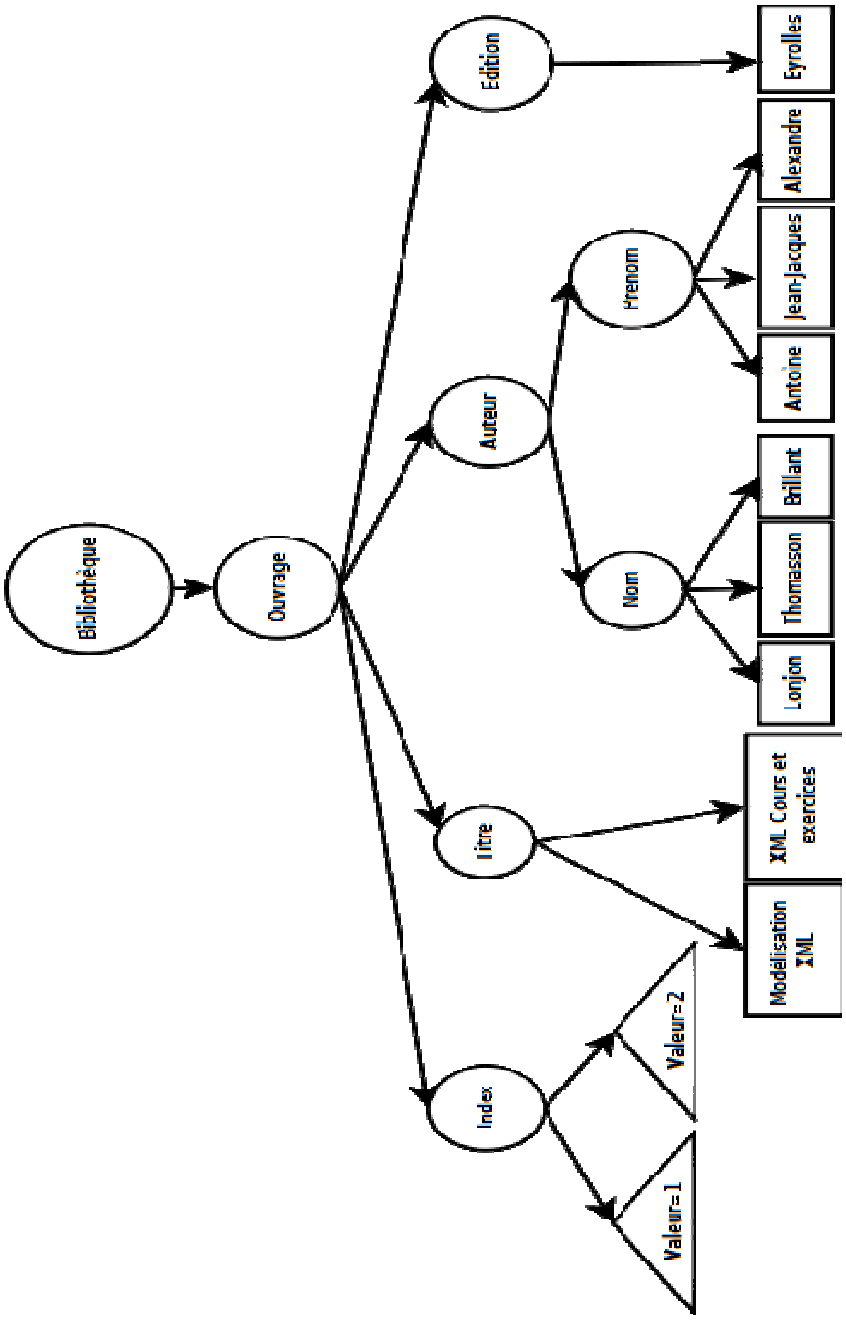


Figure 4.5 : Arbre réduit de l'arbre enraciné étiqueté ordonné de la figure 4.4.

Le fait de construire à l'avance un arbre réduit à partir d'un arbre enraciné étiqueté ordonné en supprimant tous les nœuds et chemins dupliqués, permet d'éviter en quelque sorte la génération des chemins identiques lors de la transformation de l'arbre en un ensemble des chemins dans l'étape suivante.

4.3. Matrice des chemins

Après avoir transformé un arbre XML en un arbre réduit, nous proposons de générer un ensemble des chemins en s'inspirant des travaux de Vercoustre et al. **[119]**. L'ensemble des chemins est généré en préservant la structure hiérarchique de l'arbre. Le parcours de l'arbre se fait en en pré-ordre[1]. Un ensemble de chemins est de la forme suivante :

$$EC=< c_1, c_2,\ldots, c_n>=<(a /ec_1), (a/ec_2),\ldots,(a/ec_n)> \quad (4.1)$$

Où *a* le nœud racine de l'arbre qui est un nœud structurel.

Et ec_i un sous chemin du nœud *a*. Un sous chemin comporte des nœuds structurels, des nœuds textuels notés entre doubles cottes (" ") et des nœuds attributs préfixés par @ (nous avons conservé les mêmes notations de **[119]**).

Les chemins générés peuvent être de trois types :

- **Chemin structurel** : c'est un chemin qui se termine par un nœud antécédent d'un nœud feuille de l'arbre réduit (nœud structurel).
- **Chemin textuel** : c'est un chemin qui se termine toujours par un nœud feuille de l'arbre réduit (nœud textuel).
- **Chemin attribut** : c'est un chemin qui se termine par un nœud attribut de l'arbre réduit.

[1] Parcourir l'arbre de gauche à droite.

L'ensemble des chemins généré de l'arbre réduit de la figure 4.5 est la suivante :

EC=< (Bibliothèque/Ouvrage/Index/@valeur=1),
(Bibliothèque/Ouvrage/Index/@valeur=2),
(Bibliothèque/Ouvrage/Titre/"Modélisation XML"),
(Bibliothèque/Ouvrage/Titre),
(Bibliothèques/Ouvrage/Titre/"XML Cours et exercices"),
(Bibliothèque/Ouvrage/Auteur/Nom),
(Bibliothèque/Ouvrage/Auteur/Nom/"Brillant"),
(Bibliothèque/Ouvrage/Auteur/Nom/"Lonjon"),
(Bibliothèque/Ouvrage/Auteur/Nom/ "Thomasson"),
(Bibliothèque/Ouvrage/Auteur/Prenom),
(Bibliothèque/Ouvrage/Auteur/Prenom/"Antoine"),
(Bibliothèque/Ouvrage/Auteur/Prenom/"jean-jacques"),
(Bibliothèque/Ouvrage/Auteur/Prenom/"Allexandre"),
(Bibliothèque/Ouvrage/Edition),
(Bibliothèque/Ouvrage/Edition/"Eyrolles")>

Figure 4.6 : L'ensemble des chemins de l'arbre réduit de la figure 4.5.

Le chemin *(Bibliothèque/Ouvrage/Titre)* est un chemin structurel, le chemin (*Bibliothèque/Ouvrage/Titre/"Modélisation XML")* est un chemin textuel et le chemin (*Bibliothèque/Ouvrage/Index/@valeur=2)* est un chemin attribut.

Dans l'ensemble des chemins, nous remplaçons tous les éléments esclaves par leurs éléments maîtres en utilisant le thésaurus. Pour ce faire, nous appliquons l'algorithme *TraitEC* présenté ci-dessous :

Algorithme *TraitEC*

```
Entrée : ensembles de chemins EC de la collection XML, Thésaurus
Sortie : ensembles de chemins traités
EC_i : ensemble des chemins
c_i : chemin
e_i : élément

Pour chaque EC_i
Faire
  Pour chaque c_i
  Faire
    Pour chaque e_i de c_i
    Faire
      Si e_i commence par “
        Alors
          Si e_i est un élément esclave dans le sac de contenu
            Alors
              Remplacer e_i par son élément maître
          Fin Si
      Fin Si
      Si e_i ne commence ni par @ ni par “
        Alors
          Si e_i est un élément esclave dans le sac de structure
            Alors remplacer e_i par son élément maître
          Fin Si
      Fin Si
    Fait
  Fait
Fait
```

À la fin de cette étape, l'ensemble des chemins de chaque document est mappé dans une matrice binaire. La matrice des chemins est définie pour calculer la fréquence d'apparition des chemins dans chaque document XML tel que les *m* lignes sont les documents et les *n* colonnes sont les chemins de toute la collection XML.

$$MC = \begin{bmatrix} & c_1 & \cdots\cdots & c_n \\ d_1 & 1 & \cdots\cdots & 0 \\ \vdots & \vdots & \cdots\cdots & \vdots \\ d_m & 0 & \cdots\cdots & 1 \end{bmatrix} \qquad (4.2)$$

La dernière phase de l'approche consiste à appliquer le *clustering* sur la matrice afin d'identifier des groupes ou des clusters de documents similaires.

5. *Clustering*

Le *clustering* est un très bon moyen pour organiser automatiquement des très grandes collections de données XML en des petits sous ensembles homogènes. C'est un processus de regroupement d'un ensemble de documents dans différents clusters. Un cluster de documents est une collection de documents qui sont similaires l'un à l'autre dans le même cluster et dissimilaire aux documents des autres clusters. La notion de similarité étant le plus souvent ramenée à une fonction de distance entre paires de documents.

Il existe plusieurs algorithmes de *clustering*. *k-means* est un algorithme de *clustering* utilisé pour des grands ensembles de données. Il présente l'avantage d'être de complexité linéaire.

Dans la dernière phase de *F-CheX*, nous proposons d'appliquer l'algorithme *k-means* sur la matrice des chemins pour identifier des groupes (ou des clusters) de documents similaires. La matrice des chemins, qui regroupe les documents XML représentés par leurs ensembles de chemins, est l'entrée de l'algorithme de *clustering*. Dans ce cas, nous cherchons à cibler les groupes homogènes de documents. C'est-à-dire à identifier des groupes tels que les documents possédant les ensembles de chemins les plus similaires appartiennent au même groupe et que les documents possédants les ensembles de chemins les plus différents soient séparés dans des groupes différents.

L'algorithme *k-means* est utilisé avec *k* un paramètre défini arbitrairement en entrée qui indique le nombre de clusters à construire. La distance euclidienne est utilisée pour mesurer la distance entre paires de documents où *n* est le nombre des chemins. Elle est calculée par la formule suivante :

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (4.3)$$

Les étapes de *k-means* sont les suivantes :

1. Choisir aléatoirement *k* documents qui formeront l'ensemble des centroïdes initiaux représentant les *k* clusters à construire.
2. Assigner chaque document au cluster dont le centroïde le plus proche selon la distance *d* (si un minimum de *d* est trouvé entre deux objets).
3. Si aucun document ne change de cluster d'une itération à l'autre alors arrêt et sortir les clusters. Sinon, mettre à jour les centroïdes des clusters en fonction des objets qui leur sont associés.
4. Aller à 2.

À la fin de cette phase, des clusters sont construis dont chacun regroupe les documents XML les plus similaires.

6. Conclusion

Dans ce chapitre nous avons proposé les différentes phases constituant notre nouvelle approche de fouille dans les documents XML et plus précisément la fouille dans les instances des documents en prenant en considération le contenu textuel en plus du contenu structurel.

F-CheX conserve la structure arborescente des éléments en utilisant la notion du chemin. Elle fait appel à des experts ou des ontologies de domaine pour construire le thésaurus qui permet de traiter l'aspect sémantique en unifiant tous les termes utilisés surtout dans le contenu textuel. Le *clustering* sur la matrice des chemins est utilisé pour construire des clusters regroupant les documents XML les plus similaires.

L'approche *F-CheX* nécessite une phase de validation sur des corpus de documents XML. Pour ce faire, nous avons développé un prototype d'évaluation. Dans le chapitre suivant, nous allons présenter le prototype développé ainsi que la validation sur les corpus XML.

CHAPITRE 5
EVALUATION EXPERIMENTALE

1. Introduction

L'évaluation expérimentale est une étape importante pour valider notre approche proposée sur des collections de documents XML. Dans ce chapitre, nous allons présenter le processus d'expérimentation, le prototype logiciel implémenté et le logiciel de *data mining* utilisé. Nous présentons aussi les collections de tests, les mesures d'évaluation et les résultats obtenus.

2. Processus d'expérimentation

Le schéma de la figure 5.1 illustre les principales étapes du processus d'expérimentation. En prenant comme entrée une collection de documents XML, le prototype *F-CheX* génère une matrice des chemins. De son tour, le logiciel Tanagra[1] génère des clusters en appliquant l'algorithme *k-means* sur la matrice.

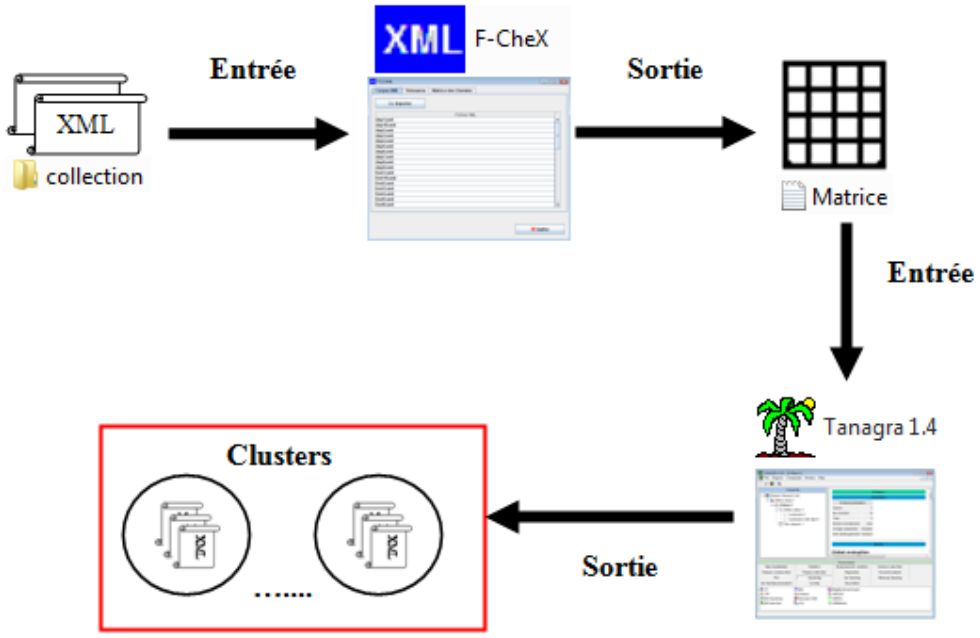


Figure 5.1: Le processus d'expérimentation.

[1] http://eric.univ-lyon2.fr/~ricco/tanagra/

Commençant par notre prototype développé, nous allons discuter ci-dessous sur les différentes fonctionnalités de *F-CheX*.

3. Présentation du prototype *F-CheX*

Pour la réalisation de notre prototype, nous avons opté pour l'environnement de développement suivant : langage de programmation Java, Eclipse 3.2.2 et JDK 1.5.0 sous le système d'exploitation Windows Vista. Nous avons utilisé l'API Java SAX pour lire, analyser et traiter les documents XML.

Le prototype *F-CheX* permet de générer à partir d'une collection XML, le thésaurus et la matrice des chemins.

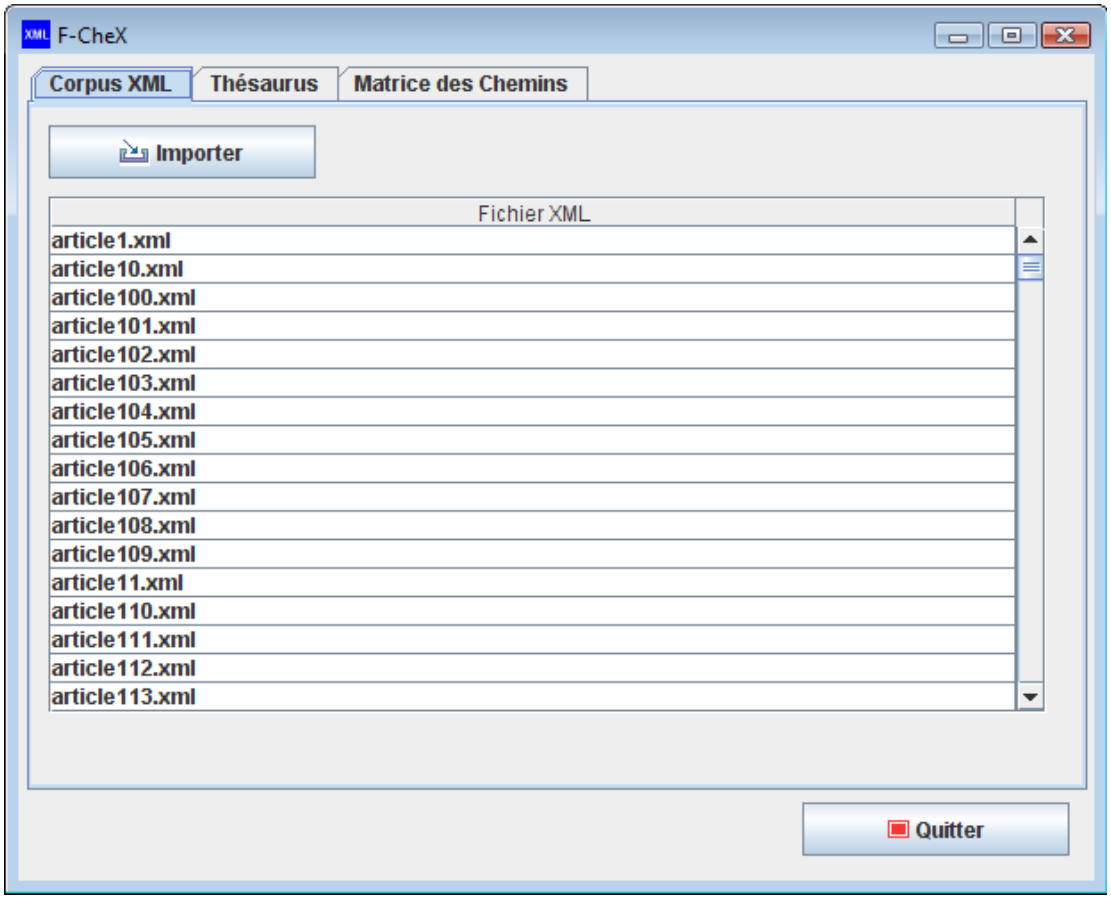


Figure 5.2 : La fenêtre principale du prototype *F-CheX*.

La fenêtre principale du prototype contient trois onglets : « Corpus XML », « Thésaurus » et « Matrice des chemins ». « Corpus XML » permet d'importer la collection des documents XML.

Comme l'indique la figure 5.2, la fenêtre du thésaurus permet de :

- Générer le sac de structure.
- Générer le sac de contenu.
- Epurer le sac de structure (suppression des doubles).
- Epurer le sac de contenu (suppression des doubles).
- Enregistrer le sac de structure.
- Enregistrer le sac de contenu.

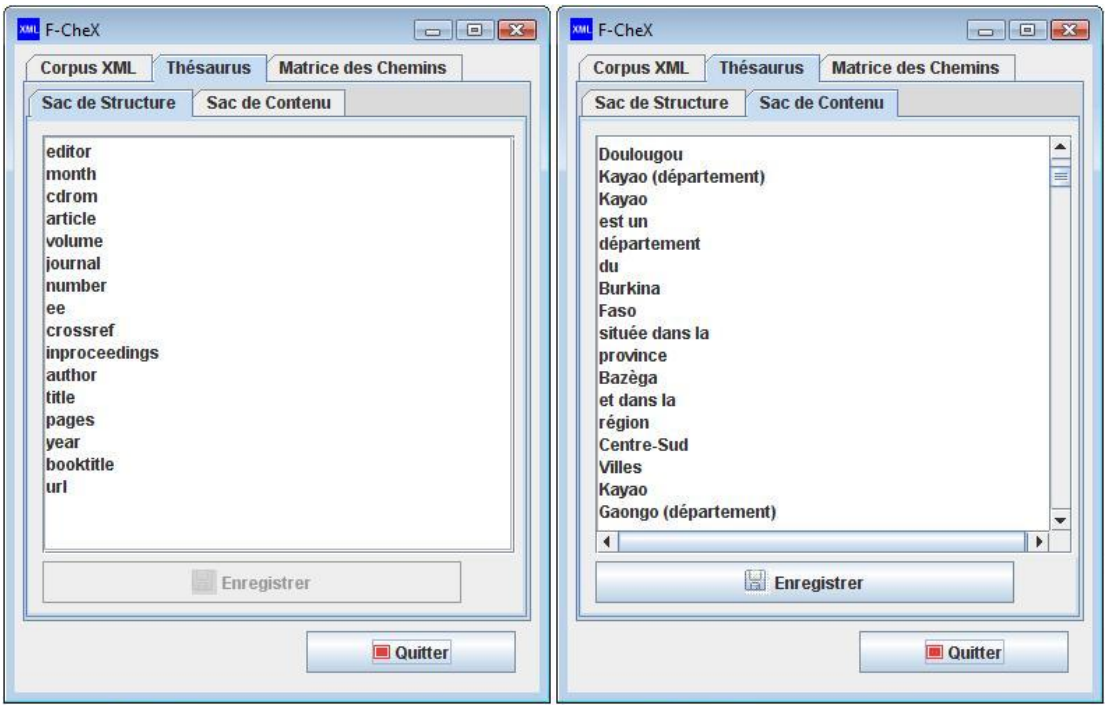


Figure 5.3 : La fenêtre Thésaurus.

Les onglets de la fenêtre matrice des chemins permettent de :

- Générer l'arbre enraciné étiqueté ordonné pour un document XML (Figure 5.3).
- Générer l'ensemble des chemins pour un document XML (Figure 5.4).
- Générer les ensembles de chemins pour toute la collection des documents XML (Figure 5.5).
- Traiter les ensembles de chemins (en unifiant les mots avec le thésaurus) (Figure 5.5).
- Enregistrer les ensembles de chemins (Figure 5.5).
- Générer la matrice des chemins (Figure 5.6).

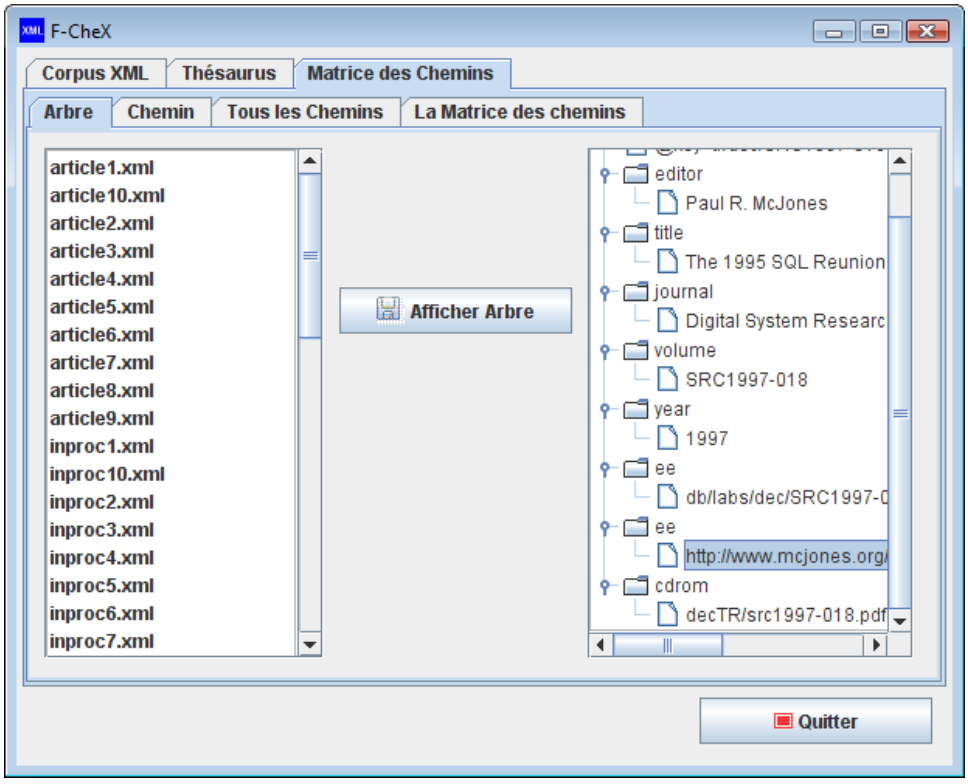


Figure 5.4 : L'onglet arbre.

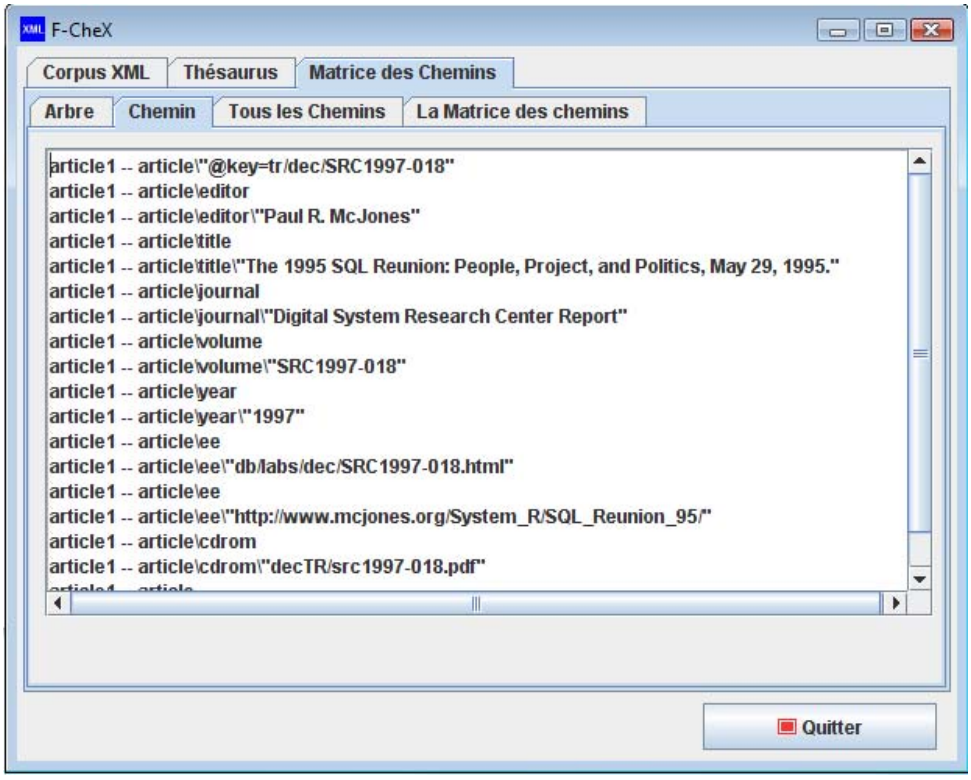


Figure 5.5 : L'onglet Chemin.

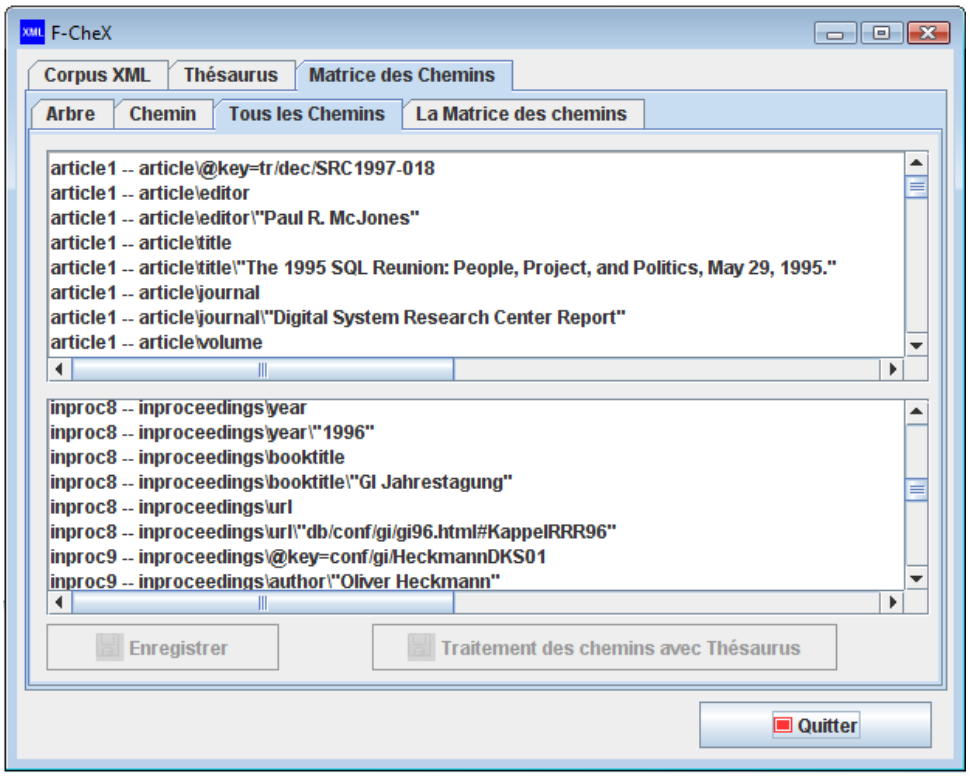


Figure 5.6 : L'onglet tous les chemins.

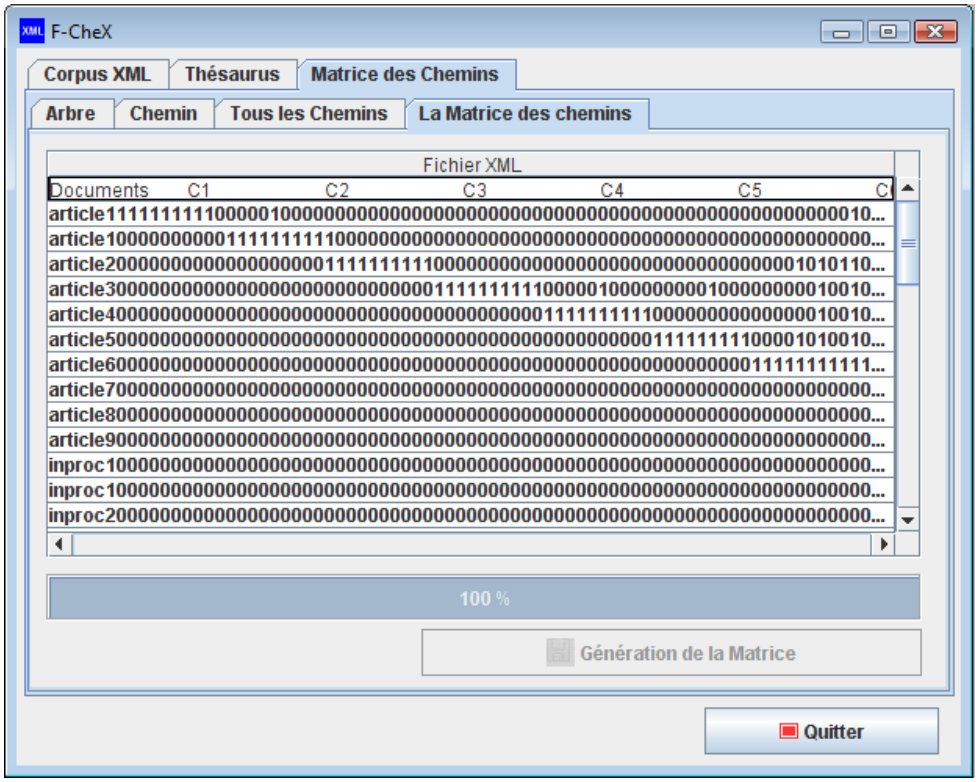


Figure 5.7 : L'onglet matrice des chemins.

Après avoir généré la matrice des chemins, nous appliquons le *clustering* à l'aide du logiciel Tanagra sur la matrice.

4. Tanagra

Tanagra est un logiciel gratuit de *data mining* destiné à l'enseignement et à la recherche. Il implémente une série de méthodes de fouilles de données issues du domaine de la statistique exploratoire, de l'analyse de données, de l'apprentissage automatique et des bases de données.

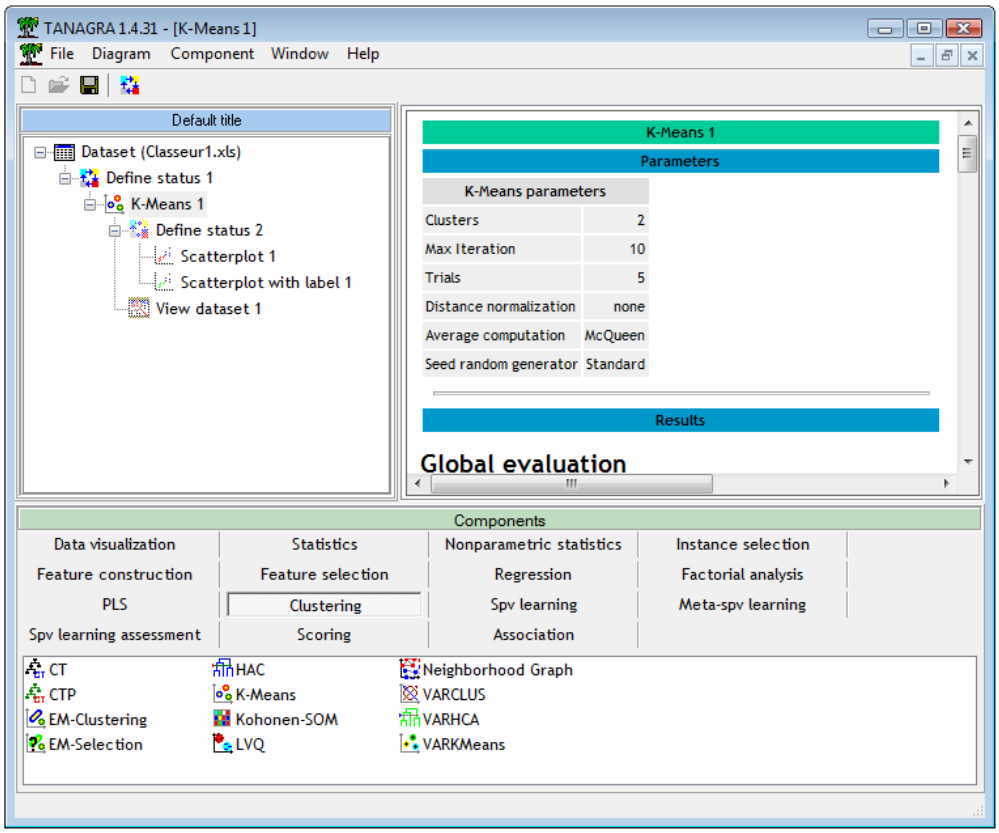


Figure 5.8 : Interface de Tanagra.

L'objectif principal du Tanagra est d'offrir aux chercheurs et aux étudiants une plate-forme de *data mining* d'accès facile, respectant les standards des logiciels du domaine, notamment en matière d'interface et de mode de fonctionnement et permettant de mener des études sur des données réelles ou synthétiques.

Le second objectif de Tanagra est de proposer aux chercheurs une architecture leur permettant d'implémenter aisément les techniques qu'ils veulent étudier et de comparer les performances des algorithmes. Tanagra se comporte

plus comme une plate-forme d'expérimentation qui leur permettrait d'aller à l'essentiel en leur épargnant toute la partie ingrate de la programmation de ce type d'outil : la gestion des données.

Le troisième objectif vise à diffuser une méthodologie possible d'élaboration de ce type de logiciel. L'accès au code leur permettra de voir comment se construit ce type de logiciel, quels sont les écueils à éviter, quelles sont les principales étapes d'un tel projet et quels sont les outils et les bibliothèques qu'il faut préparer pour le mener à bien.

Nous avons utilisé la version 1.4.31 de Tanagra pour appliquer l'algorithme de *clustering k-means* sur les matrices de chemins générées des collections XML utilisées pour faire nos tests.

5. Les collections de tests

Nous allons évaluer notre approche sur deux collections XML disponibles gratuitement sur internet. Les collections sont : «*DBLP Computer Science Bibliography*[1]» et «*Wikipédia XML Corpus*[2]».

À partir de chaque collection, nous constituons 4 jeux de tests. Nous augmentons graduellement le nombre de documents XML composant les jeux de tests. Les 4 jeux de tests *jeu1*, *jeu2*, *jeu3* et *jeu4* sont composés respectivement de 50, 100, 500 et 1000 documents.

Chaque jeu comporte cinq types de documents XML. Les jeux contiennent un même nombre de documents pour chacun des types. Par, exemple *jeu2* contient 100 documents XML avec 20 documents pour chacun des 5 types.

5.1. *DBLP Computer Science Bibliography*

La première collection des documents XML est générée à partir la base de données DBLP contenant les références bibliographiques des publications scientifiques. DBLP est constituée de plus de 1200 000[3] publications en Anglais de différents types : *article*, *inproceedings*, *proceedings*, *book*, *incollection*, *phdthesis*, *mastersthesis* et *www*.

[1] DBLP XML records, http://dblp.uni-trier.de/xml/
[2] http://www-connex.lip6.fr/~denoyer/wikipediaXML/
[3] http://dblp.uni-trier.de/xml/docu/dblpxmlreq.pdf

Pour cette première collection de documents XML, les cinq types utilisés sont : *articles*, *inproceedings*, *phdthesis*, *proceedings* et *www*.

5.2. *Wikipédia XML Corpus*

La deuxième collection est le corpus Wikipédia qui est composé essentiellement de 8 collections qui correspondent à 8 différents langues : Anglais, Français, Allemand, Hollandais, Espagnol, Chinois, Arabe et Japonais.

Nous allons utiliser le corpus textuel Français avec les cinq types : *départements*, *foot*, *peintres*, *références* et *rivières*.

Puisque les deux collections ne disposent pas d'une ontologie de domaine, nous allons définir manuellement les éléments maîtres et esclaves des sacs de mots qui constituent le thésaurus. C'est pour cette raison que nous ne traitons au maximum que 1000 documents XML. La construction manuelle d'un tel thésaurus est assez fastidieuse. Il est possible d'imaginer un moyen pour produire automatiquement un thésaurus d'un grand nombre de documents XML.

Pour évaluer l'approche sur les deux collections de tests, nous utilisons les mesures d'évaluation décrites ci-dessous.

6. Les mesures d'évaluation

Le rappel (*Recall*), la précision (*Precision*) et la *F-mesure* (*F-measure*) sont utilisées pour évaluer les résultats obtenus. Ces mesures sont fréquemment utilisées pour évaluer la qualité de la classification en fouille de documents. Le rappel *R* et la précision *P* sont définis par les formules suivantes **[104]** :

$$P = \frac{\sum_i a_i}{\sum_i a_i + \sum_i b_i}, \quad R = \frac{\sum_i a_i}{\sum_i a_i + \sum_i c_i} \tag{5.1}$$

Pour un cluster construit C_i **[30]**, **[124]** :

- a_i (TP : *True Positive*) est le nombre de documents XML qui sont correctement attribués au cluster C_i.
- b_i (FP : *False Positive*) est le nombre de documents XML qui sont dans le cluster C_i pourtant ils ne doivent pas être membres.
- c_i (FN : *False Negative*) est le nombre de documents XML qui doivent être membre du cluster C_i bien qu'ils ne se trouvent pas.

Une précision élevée signifie une justesse élevée du *clustering* alors qu'un rappel faible signifie qu'il y a plusieurs documents XML qui ne sont pas dans le cluster approprié **[30]**. Une précision et un rappel élevés indiquent que la qualité du *clustering* est excellente **[30]**.

La *F-mesure* (*F1*) combine les mesures de précision et de rappel pour essayer d'assigner chaque classe calculée à une classe prédéfinie de telle sorte que deux classes calculées ne peuvent pas être assignées à la même classe prédéfinie **[120]**. La *F-mesure* mesure un équilibre entre la précision et le rappel. Elle calcule la moyenne harmonique des deux précédentes valeurs **[69]** :

$$F1 = \frac{2 \times P \times R}{P + R} \tag{5.2}$$

Dans la section suivante, nous allons discuter les résultats de l'évaluation expérimentale.

7. Résultats

Le tableau ci-dessous représente les résultats de deux jeux de tests (*jeu1* et *jeu4*) des deux collections, qui contiennent respectivement 50 et 1000 documents XML. Ces résultats désignent le nombre de nœuds, d'éléments du thésaurus construit, de chemins générés, les profondeurs minimale et maximale des chemins générés et enfin le nombre de lignes et de colonnes des deux matrices de chemins. Ces chiffres sont obtenus lors du déploiement de notre approche *F-CheX*.

Tableau 5.1 : Résultats pour *jeu1* et *jeu4* des deux collections de tests.

		DBLP		Wikipédia	
		Jeu1	**Jeu4**	**Jeu1**	**Jeu4**
Les nœuds	Nombre total de nœuds (non dupliqués)	366	1549	287	2553
	Nombre de nœuds attributs	51	1001	105	2100
	Nombre de nœuds structurels	23	23	20	22
	Nombre de nœuds textuels	292	525	162	431
Les éléments	Nombre total d'éléments du thésaurus	315	548	182	453
	Nombre d'éléments maîtres	10	25	20	32
	Nombre d'éléments esclaves	28	82	31	41
	Nombre d'éléments neutres	277	441	131	380
Les chemins	Nombre total de chemins	719	14208	1048	20155
	Nombre de chemins structurels	309	6180	281	5620
	Nombre de chemins textuels	350	6828	485	8895
	Nombre de chemins attributs	60	1200	282	5640
	Profondeur minimale d'un chemin	02	02	02	02
	Profondeur maximale d'un chemin	03	03	07	07
La matrice des chemins	Nombre de lignes	50	1000	50	1000
	Nombre de colonnes	402	677	336	814

Nous remarquons que le nombre des chemins textuels de la deuxième collection est supérieur à celui de la première collection. Tandis que cette dernière comporte plus de chemins structurels par rapport à la deuxième collection. Les chemins de la deuxième collection sont de profondeur plus large.

En appliquant *k-means* à l'aide du Tanagra sur les deux matrices de chemins calculées pour chaque jeu de test des deux collections et en mettant le nombre de clusters *k* à 5, cinq clusters sont construits avec un nombre de documents équivalent à la partition initiale. La figure ci-dessous montre les résultats du *clustering* à l'aide de Tanagra pour le premier jeu de test des deux collections.

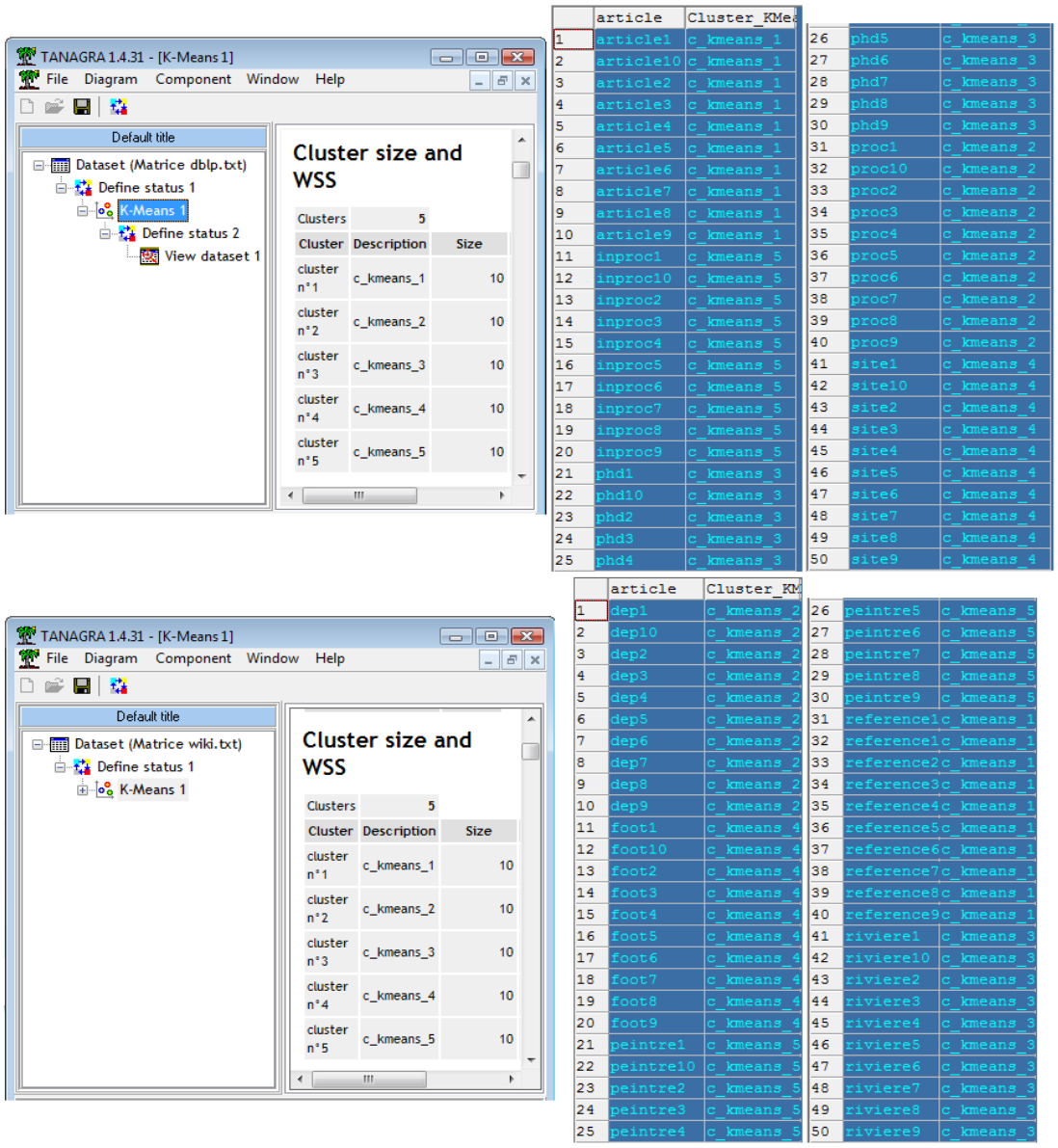


Figure 5.9 : *k-means* à l'aide de Tanagra pour le premier jeu des deux collections.

Comme l'indique le tableau 5.2, selon les mesures d'évaluation nous pouvons dire qu'avec notre approche les résultats de *clustering* obtenus sont de bonne qualité. Le rappel et la précision ont atteint de bonnes valeurs pour les deux collections.

Tableau 5.2 : Les résultats du *clustering* avec *k-means*.

		Nombre de clusters	Rappel	Précision	F-mesure
DBLP	**Jeu1**	5	1.00	1.00	1.00
	Jeu2	5	1.00	1.00	1.00
	Jeu3	5	1.00	1.00	1.00
	Jeu4	5	1.00	1.00	1.00
Wikipédia	**Jeu1**	5	1.00	1.00	1.00
	Jeu2	5	0.99	0.99	0.99
	Jeu3	5	0.99	0.99	0.99
	Jeu4	5	0.99	0.99	0.99

Nous constatons que les résultats sont toujours bons avec :

- De petits arbres XML (profondeur =3) ou avec des arbres de profondeur large (profondeur=7).
- Un nombre important de chemins attributs ou avec un petit nombre.
- L'information de structure (les nœuds structurels et les chemins structurels) est plus grande que l'information de contenu (les nœuds textuels et les chemins textuels) ou l'inverse.

En termes de temps d'exécution, nous remarquons qu'au-delà de 500 documents XML, le temps de calcul de la matrice devient de plus en plus lent.

8. Conclusion

Dans ce chapitre, nous avons évalué notre approche avec le prototype *F-CheX* en utilisant quatre jeux de tests pour chacune des deux collections : DBLP en Anglais et *Wikipédia* en Français. Nous avons augmenté graduellement le nombre de documents XML composant les jeux de tests.

Le *clustering* avec *k-means* est appliqué sur les matrices générées par *F-CheX* avec le logiciel *Tanagra*. De bonnes valeurs des mesures d'évaluation ont été obtenues, ce qui prouve que notre approche *F-CheX* garantit des résultats très intéressants.

CONCLUSION

Tout au long de ce mémoire, nous nous somme intéressés au domaine de la fouille dans les documents XML (*XML mining*) qui est devenu un domaine émergeant. XML est un format de plus en plus utilisé pour représenter, stocker et transmettre les données. La quantité d'information ne cesse d'augmenter dans ces documents. Le besoin de pouvoir extraire automatiquement des connaissances à partir de ces données augmente constamment. C'est la raison d'être de la fouille dans les documents XML. Cependant, bien que XML soit très utilisé, seuls quelques travaux de *XML mining* existent.

La fouille dans les documents XML comprend deux catégories : la fouille dans la structure XML (*XML Structure Mining*) et la fouille dans le contenu XML (*XML Content Mining*). Notre travail a été guidé par une étude préalable des travaux existants traitant les deux catégories de fouille dans les documents XML. Chacun de ces travaux a été étudié et analysé d'une façon approfondie et un tableau comparatif a été dressé pour les travaux de chaque catégorie en se basant sur des critères de comparaison. La fouille dans le contenu reste encore moins traitée et quelques aspects sont ignorés dans la majorité des travaux, tels que la sémantique et la structure arborescente des éléments.

Nous avons proposé dans ce mémoire une approche de fouille dans les documents XML qui permet de prendre en compte la structure et le contenu. L'approche consiste à appliquer le *clustering* sur les documents XML. Puisque ces derniers ont une structure arborescente, nous les représentons par des arbres enracinés étiquetés ordonnés en prenant en considération la structure et le contenu. Puis, nous proposons de réduire chaque arbre en éliminant tous les nœuds dupliqués avec l'algorithme *ReduiceTree*. Le fait de construire un arbre réduit permet d'éviter la génération des chemins similaires et dupliqués lors de la transformation de l'arbre en un ensemble des chemins. Les ensembles de chemins conservant la structure arborescente des éléments composant les documents XML sont mappés dans une matrice de chemins sur laquelle le

clustering est appliqué. L'algorithme *k-means* est utilisé avec *k* un paramètre défini arbitrairement en entrée qui indique le nombre de clusters désirés. La distance euclidienne est utilisée pour mesurer la distance entre paires de documents.

L'originalité de notre approche réside dans l'utilisation d'un thésaurus créé au préalable pour gérer l'aspect sémantique et unifier tous les mots présents dans les ensembles de chemins générés de la collection XML. Le thésaurus est créé à partir de deux sacs de mots construits de la collection XML : sac de structure et sac de contenu. C'est la combinaison des concepts de thésaurus, de sacs de mots et de chemins XML, qui assure en réalité l'originalité de notre approche.

Pour évaluer notre approche, nous avons développé le prototype *F-CheX* avec *Java*. Nous avons évalué notre approche en utilisant deux collections différentes : DBLP en Anglais et Wikipédia en Français. Le *clustering* avec *k-means* est appliqué sur les deux matrices générées par *F-CheX* avec le logiciel Tanagra. De bonnes valeurs des mesures d'évaluation ont été obtenues.

Le travail présenté débouche sur plusieurs perspectives de recherche. Il serait intéressant de :

- Améliorer le temps d'exécution et plus précisément le temps de calcul de la matrice des chemins générés à partir d'une collection XML.

- Évaluer l'approche sur des collections de tests comportant un nombre plus élevé de documents et de tester si la performance de l'approche est toujours maintenue quand le nombre de clusters devient de plus en plus grand (passage à l'échelle).

- Il est clair que dans notre approche, le thésaurus est utilisé pour gérer l'aspect sémantique en faisant appel soit à *WordNet* soit à des experts de domaine. Il serait donc plus judicieux de tester l'approche sur une collection de documents XML possédant déjà une ontologie de domaine ou bien en lui créant une ontologie propre.

- Une perspective incontournable, est d'appliquer d'autres méthodes de *data mining* sur les ensembles de chemins. Parmi ces méthodes, nous citons : la classification supervisée, en générant le pattern des chemins communs pour les documents de chaque cluster prédéfini et en appliquant la classification des documents selon ces patterns...

Il est à noter que ce travail a été concrétisé par deux articles scientifiques parus dans les conférences suivantes :

Amina Madani, Omar Boussaid, Hafida Abed, Sabine Loudcher, “Fouille dans les documents XML : Etat de l’art”, Quatrième Atelier des Systèmes Décisionnels (ASD 2009), Novembre 2009, Jijel, Algérie. Proceedings : Les Systèmes Décisionnels, Applications et perspectives, pp 53-65, 5226-2009 ISBN : 978-9961-9913-0-5.

Amina Madani, Omar Boussaid, Hafida Abed, “F-CheX : Une approche de fouille dans les documents XML”, EDA'10 : 6èmes Journées francophones sur les Entrepôts de Données et l’Analyse en ligne, 11 - 13 Juin 2010, Djerba, Tunisie.

RÉFÉRENCES

1. Aggarwal, C.C., Zaki M.J., “XRules: An Effective Structural Classifier for XML Data”, SIGKDD 03, pp. 316–325, ACM Press, New York, 2003.
2. Agrawal, R., Imielinand, T., Swami, A., “Mining association rules between sets of items in large databases”. In P. Buneman et S. Jajodia editors, editors, SIGMOD93, pages 207_216, Washington DC, USA, Mai 1993.
3. Agrawal, R., Srikant, R., “Fast algorithms for mining association rules in large databases”, Research Report RJ 9839, IBM Almaden Research Center, San Jose, California, June 1994.
4. Agrawal, R., Srikant, R., “Fast algorithms for mining association rules”, In Proceedings of 20th VLDB Conference, pages 487- 499, Santiago, Chile, 1994.
5. Agrawal, R., Srikant, R., “Mining Sequential Patterns: Generalizations and Performance Improvements”, Paper presented at the fifth International Conference on Extending Database Technology (EDBT’96), pages 3-17, France, 1996.
6. Ben Hassena, A., Miclet, L., “Les techniques d'appariement entre arbres”, Rapport Bibliographique, Publication interne IRISA numéro 1911, Projet CORDIAL, IRISA, France. Novembre 2008.
7. Balmisse, G., “Gestion des connaissances – Outils et applications du knowledge management”, Edition Vuibert, ISBN : 2-7117-8697-8, 2002.
8. Borgelt, C., “A decision tree plug-in for dataengine”, In: Proc. 6th European Congress on Intelligent Techniques and Soft Computing, volume 2, pages 1299–1303, 1998.
9. Boussaid, O., Duffoux, A., Lallich, S., Bentayeb,F., “Fouille dans la structure de documents xml”, In Actes de la 4èmes Journées Francophones d'Extraction et de Gestion des Connaissances, (EGC 04) dans la Revue des Nouvelles Technologies de l'Information, volume 2, pages 519-524, Clermont-Ferrand, France, Janvier 2004.

10. Braga, D., Campi, A., Ceri, S., Klemettinen, M., Lanzi, PL., “A Tool for Extracting XML Association Rules from XML Documents”, Research paper in Proceedings of IEEE-ICTAI 2002, pp. 57-64, Washington DC, USA, Nov 2002.
11. Braga, D., Campi, A., Ceri, S., Klemettinen, M., Lanzi, P. L., “Discovering Interesting Information in XML Data with Association Rules”, Technical Report 2002-15, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2002.
12. Braga, D., Campi, A., Klemettinen, M., Lanzi, P. L., “Mining association rules from xml data”. In Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2002), Springer LNCS 2454, pp. 21-30, September 4-6, Aixen-Provence, France, 2002.
13. Breiman, L., Friedman, J., Olshen, R., Stone, C., “Classification and regression Trees”, Chapman & Hall/CRC Press, Boca Raton, FL, 1984.
14. Brillant, A., “XML Cours et exercices”, Editions Eyrolles, ISBN : 978-2-212-12151-3, 2007.
15. Cabena, P., Hadjinian, P., Stadler, R., Verhees, J. et Zanasi, A., “Discovering Data Mining : From Concept to Implementation”, Prentice Hall Upper Saddle River, NJ, pp. 195, 1998.
16. Candillier, L., “La classification non supervisée”, Technical report, GRAPPA, Université de Lille 3, Ville neuve d'Ascq, France, 2004.
17. Candillier, L., Tellier, I., Torre, F., “Transforming XML trees for efficient classification and clustering”. In Proceedings of the 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX'05, pp. 469-480, 2005.
18. Candillier, L., Tellier, I., Torre, F., Bousquet, O., “SSC : Statistical Subspace Clustering”. In Perner, P., Imiya, A., eds.: 4th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM'2005). Volume LNAI 3587 of LNCS. Leipzig, Germany, Springer Verlag pp. 100-109, 2005.
19. Candillier, L., “Contextualisation, visualisation et évaluation en apprentissage non supervisé”, Thèse, Université Charles de Gaulle - Lille 3, 2006.

20. Candillier, L., Denoyer, L., Gallinari, P., Rousset, M.C., Termier, A., Vercoustre, A.M., “Mining XML Documents”, in Data Mining Patterns: New Methods and Applications Information Science Reference (Ed.) pp. 198-219, 2007.
21. Celeux, G., Diday, E., Govaert, G., Lechevallier, Y., Ralambondrainy, H., “Classification Automatique des Données, Environnement statistique et informatique”. Dunod informatique, Bordas, Paris, FRANCE, 1989.
22. Chagnon, G., “Cours de XML”, http://www.gchagnon.fr/cours/xml/index.html mars 2008.
23. Charrad, M., “Techniques d'extraction de connaissances appliquées aux données du Web”, Mémoire de master en informatique, Université de la Manouba, Tunis, Décembre 2005.
24. Chawathe, S.S., “Comparing hierarchical data in external memory”, in: Proceedings of the VLDB Conference, Edinburgh, Scotland, UK, pp. 90-101, 1999.
25. Chidlovskii, B., “Schema Extraction from XML Data : A Grammatical Inference Approach”, KRDB’01 Workshop, Rome, Italy, September 15, 2001.
26. Cormen, T., Leiserson, C., Rivest, R., “Introduction to algorithms”, MIT Press, 1990.
27. Cornuéjols, A., Miclet, L., Kodratoff, Y., “Apprentissage artificiel: Concepts et algorithmes”, ISBN : 2-212-11020-0, édition Eyrolles, 2002.
28. Cover, T.M., Hart, P.E., “Nearest neighbor pattern classification”, IEEE Transactions on Information Theory, 13 : 21_27.1967.
29. Cram, D., May, M., Guelton, R., Touch, S., “Résumé : Réseaux bayésiens”, Support de cours, liris.cnrs.fr/alain.mille/enseignements/master_ia/Alain/exposes_2005/R_seaux_
Bay_sien.pdf, 2005.
30. Dalamagas, T., Cheng, T., Winkel, K.J., Sellis, T.K., “Clustering XML Documents using Structural Summaries”, In Proc. of ClustWeb - International Workshop on Clustering Information over the Web in conjunction with EDBT 04, pp. 547–556, Crete, Greece, 2004.

31. Dalamagas, T., Cheng, T., Winkel, K.J., Sellis, T.K., “A methodology for clustering XML documents by structure”, Information Systems 31(3), pp. 187–228, 2006.
32. Denoyer, L., Gallinari, P., Piwowarski, B., “Un modèle pour la Recherche d’Information pour les Documents Structurés”, 6ème journées internationales d’analyse des données textuelles (JADT 2002), Saint Malo, France, 2002.
33. Denoyer, L., Gallinari, P., Vittaut, J.N., Brunessaux, S., Brunessaux, St., “Structured Multimedia Document Classification”, In Proceedings of ACM Document Engineering, 2003, Pages: 153 – 160, Grenoble, France.
34. Denoyer, L., Gallinari, P., Trang-Huyen, V., “Un modèle statistique pour la classification de documents structurés”, Journées francophones d'Extraction et de Gestion des Connaissances (EGC 2003) 2003, pp. 169-180, Lyon, France.
35. Denoyer, L., Gallinari, P., “Bayesian Network Model for Semi-Structured Document Classification”. Revue Information Processing & Management, Special Issue on Bayesian Networks and Information Retrieval, Pages 807-827, Elsevier, 2004.
36. Denoyer, L., Gallinari, P., “Un Modèle de Mixture de Modèles Génératifs pour les Documents Structurés Multimedia : Application à la Classification de Documents XML et HTML”, Revue Document Numérique, Numéro spécial « Fouille de Textes et Organisation de Documents », Hermès, 2004.
37. Denoyer, L., “Apprentissage et inférence statistique dans les bases de documents structurés : Application aux corpus de documents textuels”, Thèse de doctorat, Université de Paris 6, Décembre 2004.
38. Denoyer, L., Gallinari, P., Vercoustre, A.M., “Report on the XML Mining Track at INEX 2005 and INEX 2006 - Categorization and Clustering of XML Documents”, In: Proceedings of INEX, pp. 432-443, 2006.
39. Denoyer, L., Gallinari, P., “The Wikipedia XML Corpus”, SIGIR Forum, pp. 12-19, 2006.
40. Dong, G., Li, J., “Efficient mining of emerging patterns: Discovering trends and differences”, In: ACM SIGKDD International Conference on

Knowledge Discovery and Data Mining, pp. 43–52, 1999.

41. Doucet, A., Ahonen-Myka, H., “Naive clustering of a large xml document collection”, In: Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), Schloss Dagsuhl, Germany, pp. 81–87, 2002.
42. Doucet, A., Lehtonen, M., “Unsupervised classification of text-centric XML document collections”, In: Workshop of the INitiative for the Evaluation of XML Retrieval, 2006.
43. Ester, M., Kriegel, H.P., Sander, J., Xu, X., “A density-based algorithm for discovering clusters in large spatial databases with noise”. In Evangelos imoudis, Jiawei Han, and Usama Fayyad, editors, Second International Conference on Knowledge Discovery and Data Mining, pages 226–231, Portland, Oregon, 1996. AAAI Press.
44. Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X., “Incremental clustering for mining in a data warehousing environment”. In Proc. 24th Int. Conf. Very Large Data Bases, VLDB, pages 323–333, 24–27 1998.
45. Fellbaum, C., “WordNet : An Electronic Lexical Database”, MIT Press, 1988.
46. Flynn, P., “Foire aux questions XML”, Version Française, Cahiers GUTenberg n°33-34 Congrès GUT’99 Journée XML, 19 mai 1999.
47. De Francesca, F., Gordano, G., Ortale, R., Tagarelli, A., “A general framework for XML document clustering”, Technical report, n.8, ICAR-CNR, 2003.
48. De Francesca, F., Gordano, G., Ortale, R., and Tagarelli, A., “Distance-based Clustering of XML Documents”, 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03) Cavtat-Dubrovnik, pp. 75–78, Croatia, September 22 and 23, 2003.
49. Garboni, C., Masseglia, F., Trousse, B., “Sequential pattern mining for structure based XML document classification”, In: INEX 2005 Workshop of the INitiative for the Evaluation of XML Retrieval, pp. 458–468, 2006.
50. Gardarin, G., Dang-Ngoc, T.,T., “Architecture de médiation "tout-XML" Conception et évaluation”, Ingénierie des Systèmes d'Information 8(5-6):

11-25, 2003.

51. Garofalakis, M., Rastogi, Seshredi, S., Shim, K., “Data mining and the web : past, present and future”, Kansas city,USA, 2nd international workshop on web information and data management, p.43-47, 1999.
52. Garofalakis, M., Gionis, N., A., Rastogi, R., Seshadri, S., Shim, K., “Xtract: A System for Extracting Document Type Descriptors from XML Documents”, SIGMOD Conference 2000, Dallas, Texas, USA, pp. 165-176, 16-18 May 2000.
53. Gower, J.C., Ross, G.J.S., “Minimum spanning trees and single linkage cluster analysis”, Applied Statistics 18, pp. 54-64, 1969.
54. Guha, S., Rastogi, R., Shim, K., “CURE: an efficient clustering algorithm for large databases”. Proceedings of ACM SIGMOD International Conference on Management of Data, pages 73–84, 1998.
55. Guha S., Rastogi, R., Shim, K., “ROCK: A Robust Clustering Algorithm For Categorical Attributes”, Proc. 15th Int’l Conf. Data Eng., pp. 512-521, 1999.
56. Hagenbuchner, M., Trentini, F., Sperduti, A., Tsoi, A., Scarselli, F., Gori, M., “Clustering XML Documents using Self-Organizing Maps for Structures”. INEX 2005 Workshop on Mining XML documents, pp. 432–442, November 2005.
57. Hagenbuchner, M., Trentini, F., Sperduti, A., Scarselli, F., Tsoi, A.C., “A Self-Organising Map Approach for Clustering of XML Documents”, International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21, pp. 1805 – 1812, 2006.
58. Halkidi, M., Batistakis, Y., Vazirgiannis, M., “Clustering algorithms and validity measures”, Tutorial paper in the Proceedings of the SSDBM Conference, Virginia, USA, 2001.
59. Han, J., Kamber, M., “Data mining : Concepts and techniques”, Morgan kaufmann publishers, USA. 2001.
60. Harold, E.R., “XML Le guide de l’utilisateur”, Editions Osman Eyrolles Multimedia, ISBN : 2-7464-0070-7, 2000.
61. Haykin, S., “Neural Networks : A Comprehensive Foundation”, Prentice Hall, Upper Saddle River, NJ, 1990.

62. Hubert, L.J., Levin, J. R., “A general statistical framework for accessing categorical clustering in free recall”, Psychological Bulletin 83, pp. 1072-1082, 1976.

63. Hubert, L., Arabie, P., “Comparing Partitions”, Journal of Classification, 2:193–218, 1985.

64. Isert, C., “The editing distance between trees”, Technical report, Ferienakademie, for course 2: Bume: Algorithmik Und Kombinatorik, Italy, 1999.

65. Jaakkola, T.S., Diekhans, M., Haussler, D., “Using the Fisher kernel method to detect remote protein homologies”. In Intelligent systems for molecular biology conference (ISMB’99). Heidelberg, Germany: AAAI, 1999.

66. Jouve, P.E., “Apprentissage Non Supervisé et Extraction de Connaissances à partir de Données”, Thèse de doctorat, Université Lumière Lyon2, décembre 2003.

67. Knijf, J.De., “FAT-CAT: Frequent Attributes Tree Based Classification”, In Fuhr, N., Lalmas, M., and Trotman, A., editors, Comparative Evaluation of XML Information Retrieval Systems, 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006, pages 485–496. 2007.

68. Knijf, J.De., “FAT-miner: Mining Frequent Attribute Trees”, In Cho, Y., Wainwright, R., Haddad, H., Shin, S., and Koo, Y., editors, SAC ’07: Proceedings of the 2007 ACM symposium on Applied computing, pages 417–422. 2007.

69. Knijf, J.De., “Studies in Frequent Tree Mining”, UU Universiteit Utrecht (169 pag.) (Utrecht: Utrecht University). Prom./coprom.: prof. dr. A.P.J.M. Siebes & dr. A.J. Feelders. 2008.

70. Kohonen, T., “Self-organized formation of topologically correct feature maps”, Biological Cybernetics, Vol. 43, p.59-69, 1982.

71. Kohonen, T., “Self-Organisation and Associative Memory”, Springer, 3rd edition, 1990.

72. Larose, D.T., “Des données à la connaissance : Une introduction au data

mining", Traduction et adaptation de Thierry Vallaud, Vuiber, ISBN 2-7117-4855-3, Septembre 2005.

73. St-Laurent, S., "Introduction au XML", Editions Osman Eyrolles Multimedia, ISBN : 2-7464-0094-4, 2000.
74. Lee, L.M., Yang, L.H., Hsu, W., Yang, X., "XClust: clustering XML schemas for effective integration", Paper presented at the 11th ACM International Conference on Information and Knowledge Management (CIKM'02), Virginia, November 2002.
75. Lefébure, R., Venturi, G., "Data mining", Editions Eyrolles, ISBN : 2-212-09176-1, 2001.
76. Lehtonen, M., "Preparing Heterogeneous XML for Full-Text Search", ACM Transactions on Information Systems 24, pp. 1–21, 2006.
77. Lehtonen, M., "Indexing Heterogeneous XML for Full-Text Search", Doctoral dissertation, University of Helsinki, Faculty of Science, Department of Computer Science, November 2006.
78. Lewis, D.D., "Naive (Bayes) at forty: the independence assumption in information retrieval". In Proceedings of ECML-98 (pp. 4–15). Heidelberg, 1998.
79. Lian, W., Cheung, D.W., "An Efficient and Scalable Algorithm for Clustering XML Documents by Structure", in IEEE Transactions on Knowledge and Data Engineering, pp. 82-96, 2004.
80. Liu, B., Hsu, W., Ma, Y., "Integrating Classification and Association Rule Mining", SIGKDD, pp.80-86, 1998.
81. Lonjon, A., Thomasson, J.J., "Modélisation XML", Editions Eyrolles, ISBN : 2-212-11521-0, 2006.
82. MacQueen, J.B., "Some methods for classification and analysis of multivariate observations". In Proceedings of 5th Berkley Symposium on Mathematical Statistics and Probability, volume I : Statistics, pages 281–297, 1967.
83. Meo, R., Psaila, G., Ceri, S., "An extension to SQL for mining association rules", Data Mining and Knowledge Discovery, 2(2) : 195. 224, 1998.
84. Michard, A., "XML langage et applications", Editions Eyrolles, ISBN : 2-212-09206-7, 2001.

85. Milligan, G.W., Cooper, M.C., “An examination of procedures for determining the number of clusters in a data set”, Psychometrika 50, pp. 159-179, 1985

86. Muhlenbach, F., “Évaluation de la qualité de la représentation en fouille de données”, THÈSE, Université Lumière Lyon II, Décembre 2002.

87. Naïm, P., Wuillemin, P.H., Leray, P., Pourret, O., Becker, A., “Réseaux bayésiens”, Editions Eyrolles, ISBN : 2-212-1137-1, 2004.

88. Nayak, R., Witt, R., and Tonev, A., “Data mining and XML documents. In Proceedings International Conference on Internet Computing”, IC'2002 3, pp. 660-666, Las Vegas, Nevada. 2002.

89. Nayak, R., Xu, S., “XML documents clustering by structures with XCLS”, In: Workshop of the INitiative for the Evaluation of XML Retrieval INEX, pp. 432–442, November 2005.

90. Nayak, R., Iryadi, W., “XMine : A methodology for mining XML structure”, The Eighth Asia Pacific Web Conference, China. January 2006.

91. Nayak, R., Xu, S., “XCLS : A Fast and Effective Clustering Algorithm for Heterogeneous XML Documents”, The 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), April, Singapore, 2006.

92. Nayak, R., Iryadi, W., “XML schema clustering with semantic and hierarchical similarity measures”, Knowledge-Based Systems, Volume 20, Issue 4, Pages 336-349, May 2007.

93. Pearl, J., “Probabilistic reasoning in intelligent systems : Networks of plausible inference”. Morgan Kauffman, 1988.

94. Plotkin, G., “A note on inductive generalisation”, Machine Intelligence, 5:153-163, 1970.

95. Popescu-Belis, A., “cours XML L'impact d'XML pour les technologies multilingues”, Université de Genève, http://www.issco.unige.ch/staff/andrei/xml, 2005.

96. Porter, M.F., “An algorithm for suffix stripping”, Program, 14(3) pp. 130−137, 1980.

97. Porter, M.F., “An algorithm for suffix stripping”, In Readings in information retrieval, San Francisco, CA, USA, pp. 313–316. Morgan Kaufmann Publishers Inc, 1997.

98. Preux, P., “Fouille de données - Notes de cours”, Université de Lille 3, http://www.grappa.univ-lille3.fr/~ppreux/fouille, 9 octobre 2008.
99. Quinlan, J.R., “Induction of decision trees”, Machine Learning, 1(1) : 81–106, 1986.
100. Quinlan, J.R., “C4.5 : Programs for Machine Learning”, Morgan Kaufmann, San Francisco, CA, 1992.
101. Quinlan, J.R., “Data mining tools see5 and c5.0”, 2004.
102. Rasmussen, E., “Clustering algorithms”, In: W. Frakes, R. Baeza-Yates (Eds.), Information Retrieval: Data Structures and Algorithms, Prentice Hall, 1992.
103. Rice, S.V., Bunke, H., Nartker, T.A., “Classes of cost functions for string edit distance”, Algorithmica 18 (2) 271–280, 1997.
104. van Rijsbergen, C.J., “Information Retrieval, Butterworths”, London, 1979.
105. Safar, B., Reynaud, C., Calvier, F.E., “Techniques d’alignement d’ontologies basées sur la structure d’une ressource complémentaire”, 1ères Journées Francophones sur les Ontologies, JFO’2007, Sousse, Tunisie, 18-20 octobre 2007.
106. Sankoff, D., Kruskal, J., Warps, T., “String Edits and Macromolecules, The Theory and Practice of Sequence Comparison”, CSLI Publications, 1999.
107. Sévigny, M., “Introduction à XML Notes de cours”, www.ajlsm.com/formation/xml/notes/xml/notes.pdf, 13 septembre 2002.
108. Shasha, D., Zhang, K., “Approximate Tree Pattern Matching”, In: Apostolico, A.,Galil, Z. (eds.) Chapter 14 Pattern Matching Algorithms, Oxford University Press, Oxford, 1997.
109. Sheikholeslami, C., Chatterjee, S., Zhang, A., Wavecluster, “A multiresolution clustering approach for very large spatial database”. 24th VLDB Conference, New York, USA, 1998.
110. Stephenson, T.A., “An introduction to Bayesian network theory and usage”. IDIAP-RR 00-03, February 2003.
111. Termier, A., Rousset, M.C., Sebag M., “TreeFinder : a First Step towards XML Data Mining”, Dans International Conference on Data Mining ICDM’02, Maebashi, Japon, pp. 450-457, 2002.
112. Termier, A., “Extraction d’arbres fréquents dans un corpus hétérogène de

données semi-structurées : application à la fouille de documents XML", Thèse, Université Paris-Sud, France, 2004.

113. Thompson, K., "Regular Expression Search Algorithm". Communications of ACM 11(6), pp. 419–422, 1968.
114. Tittel, E., "SCHAUM'S XML", Traduit par Patrick Fabre, Editions Dunod, ISBN : 2-10-0006934-9, 2003.
115. Tommasi, M., Gilleron, R., "Découverte de connaissances à partir de données", http://www.grappa.univ-lille3.fr/polys, 2000.
116. Tufféry, S., "Data mining et Scoring", Editions Dunod, ISBN : 2 10 008184 5, 2002.
117. Tufféry, S., "Data mining & statistique décisionnelle", http://data.mining.free.fr/cours/Presentation.pdf, 2006.
118. Vercoustre, A.M., Despeyroux, T., Lechevallier, Y., Trousse, B., "Experiments in clustering homogeneous xml documents to validate an existing typology", In Proceedings of the 5th International Conference on Knowledge Management (I-Know), Vienne, Autriche, (postprint), Journal of Universal Computer Science, , n° 1, July 2005.
119. Vercoustre, A.M., Fegas, M., Gul, S., Lechevallier, Y., "A flexible structured-based representation for XML document mining". In Proceedings of the 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX'05, Schloss Dagstuhl, Germany, LNCS (3977), Springer, pp. 443-457, 2006.
120. Vercoustre, A.M., Fegas, M., Lechevallier, Y., and Despeyroux, T., "Classification de documents XML à partir d'une représentation linéaire des arbres de ces documents", In Actes des 6ème journées Extraction et Gestion des Connaissances (EGC 2006), Revue des Nouvelles Technologies de l'Information (RNTI-E-6), pages 433–444, Lille, France, January 2006.
121. Wang, W., Yang, J., Muntz, R.R., "STING : A statistical information grid approach to spatial data mining". In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, Twenty-Third International Conference on Very Large Data Bases, pages 186–195, Athens, Greece, 1997. Morgan

Kaufmann.

122. Wang, W., Yang, J., Muntz, R., “STING+: An approach to active spatial data mining”. In Fifteenth International Conference on Data Engineering, pages 116–125, Sydney, Australia, pp. 116-125, 1999. IEEE Computer Society.
123. Williams, W.T., Lambert, J.M., “Multivariate methods in plant ecology”. Journal of Ecology, 47:83–101, 1959.
124. Xing, G., Xia, Z., “Classifying XML documents based on structure/content similarity”, In: Workshop of the INitiative for the Evaluation of XML Retrieval, Springer, pp. 444-457, 2006.
125. Xing, G., “Fast Approximate Matching Between XML Documents and Schemata”, In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 425–436, Springer, Heidelberg, 2006.
126. Yang, J., Chen, X., “A semi-structured document model for text mining”, Journal of Computer Science and Technology archive, Volume 17(5), 603-610, May 2002.
127. Ye, L., Spetsakis, M., “Clustering on unobserved data using mixture of gaussians”, Technical report, York University, Toronto, Canada, Oct. 2003.
128. Zaki, M. J., “Efficiently Mining Frequent Trees in a Forest”, SIGKDD, pages 71-80, 2002.
129. Zhang, K., Shasha, D., “Simple fast algorithms for the editing distance between trees and related problems”, SIAM J. Comput., 18(6):1245–1262, 1989.
130. Zhang, T., Ramakrishnan, R., Livny, M., “BIRCH : an efficient data clustering method for very large databases”. In ACM SIGMOD International Conference on Management of Data, pp. 103–114, Montreal, Canada, June 1996.
131. Zhao, Y., Karypis, G., “Evaluation of hierarchical clustering algorithms for document datasets”, Paper presented at the 2002 ACM CIKM, Virginia, pp. 515-524, USA, 2002.
132. Data Types Analyzed/Mined, http://www.kdnuggets.com/polls/2008/data-mining-applications.htm, September 2008.

133. World Wide Web Consortium, “Extensible Markup Language (XML)”, http://www.w3c.org/xml/.

134. Amina Madani, Omar Boussaid, Hafida Abed, Sabine Loudcher, “Fouille dans les documents XML : Etat de l’art”, Quatrième Atelier des Systèmes Décisionnels (ASD 2009), Novembre 2009, Jijel, Algérie. Proceedings : Les Systèmes Décisionnels, Applications et perspectives, pp. 53-65, 5226-2009 ISBN : 978-9961-9913-0-5.

135. Amina Madani, Omar Boussaid, Hafida Abed, “F-CheX : Une approche de fouille dans les documents XML”, EDA'10 : 6èmes Journées francophones sur les Entrepôts de Données et l’Analyse en ligne, 11 - 13 Juin 2010, Djerba, Tunisie.