华侨大学

硕士学位论文

基于聚类的XML文档集成管理方法研究

姓名：傅珊珊

申请学位级别：硕士

专业：计算机应用技术

指导教师：吴扬扬

20071201

摘　要

XML 作为 Internet 上信息表示和交换的一个标准，如何在关系数据库中有效地存储和查询这些数据已成为 XML 研究领域的一个重要问题。目前几乎所有的商业数据库产品（如三大商业 RDBMS 产品 SQL Server，Oracle 及 DB2）都进行了扩充，支持对 XML 的存储、查询和发布。通常，RDBMS 提供的 XML 数据存储功能有：将 XML 文档的内容或数据存储于数据库中；或以文本文件的形式对 XML 文档进行存储，并在数据库中保留对相应文件的索引或链接。然而随着 XML 应用的不断扩展，XML 数据的大量出现，由于 XML 数据可能来自各个不同的数据源，它们所遵循的文档模式（DTD 或 XML Schema）可能不同，文档之间存在一定的异构性，如果采用目前的 RDBMS 存储这些 XML 数据，则存在一定的局限性，即无法实现来自不同 DTD 的多个 XML 文档的有效集成存储。

本文研究来自不同 DTD 的多个 XML 文档的集成存储问题。主要工作包括：

1、提出了一种有效的 XML 文档频繁子树挖掘方法，具体做法是：首先对 XML 文档进行预处理，提取 XML 文档的有效结构 SST（Simplest Structural Tree 最简结构树）；然后提出 SSTMiner 算法，用于挖掘 SST 中的所有嵌入频繁子树。SSTMiner 算法不但继承了 TreeMiner 算法的优点，而且针对 TreeMiner 算法存在的瓶颈问题，以及结合当前所处理的 SST 的结构特点，对 TreeMiner 算法进行改进，提高了算法执行的效率；实验结果表明了该方法的有效性。

2、研究基于频繁结构的 XML 文档聚类方法，其频繁结构包括频繁路径和频繁子树，具体做法是：首先对 XML 文档进行频繁结构挖掘，获取最大频繁结构集，由于 SSTMiner 算法是一种非常有效的频繁子树挖掘算法，对 SSTMiner 算法稍加修改，即可得到 FrePathMiner 算法和 FreTreeMiner 算法，分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树；然后提出一种凝聚的层次聚类算法 XMLCluster，分别以最大频繁路径和最大频繁子树作为 XML 文档的特征，对文档进行聚类；实验表明本文提出的 FrePathMiner 和 FreTreeMiner 聚类方法，与传统的 ASPMiner 聚类方法相比，其聚类效果具有更大的优越性。

3、提出了一种 XML 文档集成存储方法，即在聚类的基础上，对 XML 文档进行集成存储，集成存储过程分为两个阶段进行：第一阶段为模式映射阶段（Schema Mapping），生成集成模式；第二阶段为数据存储阶段（XML Storage），从 XML 文档中抽取数据存入数据库，从而实现对来自不同 DTD 的多个 XML 文档的有效集成存储。最后，本文给出了基于聚类的 XML 文档集成管理系统的整个框架，并通过具体实例来说明基于聚类的 XML 文档集成存储方法。

关键词：XML，频繁子树挖掘，聚类，集成

Abstract

XML is a new standard of information issue and exchanging on the Internet, and how to store and query these data effectively in relational database has become an important problem in XML field. Today,nearly all commercial database products, such as SQL Server, Oracle and DB2, have been extended to provide support in storing and querying XML, and publishing data in XML format. Generally, there are two storage ways that RDBMS supports to XML data. One is to store the content of XML documents to RDBMS. The other is to store XML documents by whole files and store the indexes of these files in database. But with the rapid expansion of XML application and the huge emergence of XML data, these XML data may come from different data sources, so their document schemas (DTD or XML Schema) may be different and they are isomerous. So the current RDBMS products will be limit if we use them to store these XML data. That’s, they can't integrate XML documents, which conform to different DTDs, into relational storage efficiently.

This paper researches into the problem of integrating many XML documents, which conform to different DTD, into relational storage. The main research work is listed as follows.

1. This paper presents an efficient method to mine embedded frequent trees in a forest of XML documents. The method is that we first preprocess XML documents to get SSTs (Simplest Structural Trees) and

then mine frequent trees in SSTs. In this paper, we improve TreeMiner by breaking the bottle-neck of TreeMiner and considering the structure character of SST, and present an algorithm called SSTMiner. The experiments show that this method is efficient to mine frequent trees in XML documents.

2. This paper researches XML document clustering using frequent structure, which includes frequent path and frequent tree. The paper firstly mines frequent structures in XML documents. Because SSTMiner is an efficient method to mine all embedded frequent trees in XML documents, it can be modified a little to generate FrePathMiner algorithm and FreTreeMiner algorithm, which can be respectively used to mine common frequent path and common frequent tree. Then by using common frequent path and common frequent tree to characterize the XML documents, an agglomerative hierarchical clustering algorithm called XMLCluster is propesed to cluster XML documents. Finally, the experiment is conducted, in which three methods, including FrePathMiner, FreTreeMiner and traditional ASPMiner(Adaped Sequential Pattern Miner), are used in the XML document clustering. And the clustering results show that both FrePathMiner and FreTreeMiner can get higher clustering precision than ASPMiner.

3. This paper presents a solution to integrate XML documents based on clustering, which integrates XML documents into relational storage.

The process of integral storage can be divided into two steps. The first step is called schema mapping, in which the integral relational schema is generated. The second step is called XML storage, in which all XML information is extracted into relational database, and XML documents are eventually integrated into relational database efficiently.

Finally, this paper presents the complete frame of the system of integrating XML document using clustering and shows the method to be efficient by some detail examples.

Keywords: XML, frequent tree miner, clustering, integration

原创性声明

本人声明兹呈交的学位论文是本人在导师指导下完成的研究成果。论文写作中不包含其他人已经发表或撰写过的研究内容，如参考他人或集体的科研成果，均在论文中以明确的方式说明。本人依法享有和承担由此论文所产生的权利和责任。

学位论文作者签名：_____日期：_____

学位论文版权使用授权声明

本人同意授权华侨大学有权保留并向国家机关或机构送交学位论文和磁盘，允许学位论文被查阅和借阅。

论文作者签名：_____ 指导教师签名：_____

签 名 日 期：_____ 签 名 日 期：_____

第一章 绪论

1.1 论文背景与意义

XML(Extensible Markup Language，可扩展标记语言)[1]是 W3C(Word Wide Web Consortium)为适应 Internet 的发展，实现快速的电子商务和电子数据交换而推出的新型 Web 语言，是 ISO 所制定的 SGML（Standard Generalized Markup Language，通用语言标识标准）的一个精简集。作为 Internet 上信息表示和交换的一个标准，它的存储管理成为人们日益关心的问题。现在有四种管理 XML 数据的方式，它们分别是：

1) 将其保存为文件系统的 ASCII 文件；
2) 建立一个专门的数据库系统进行管理；
3) 转换到对象数据库中管理；
4) 转换到关系数据库中管理。

这四种方式中第一种最简单，但是缺点也最多，如每次浏览和查询时都需要解析，修改数据很困难等等。第二种虽然是针对 XML 专门设计的，有许多优点，但是技术还不成熟，应用得不多。第三种实现 XML 模式到对象的转换比较简单，且技术较为成熟，但是由于对象数据库应用还不普及，所以使用也很有限。第四种方式得到了许多学者的关注，因为关系数据库是已经发展了几十年的成熟技术，而且现有的大多数数据都是以关系数据库形式存放的，如果 XML 数据也能以这种关系形式存储，那么就可以实现对这两种数据的统一利用。因此，如何利用关系数据库有效地存储 XML 文档的问题成为研究领域的一个挑战。目前几乎所有的商业数据库产品（如三大商业 RDBMS 产品 SQL Server，Oracle 及 DB2）都进行了扩充，支持对 XML 的存储、查询和发布[2]。通常，RDBMS 提供的 XML 数据存储功能有：将 XML 文档的内容或数据存储于数据库中；或以文本文件的形式对 XML 文档进行存储，并在数据库中保留对相应文件的索引或链接。然而随着 XML 应用的不断扩展，XML 数据的大量出现，由于 XML 数据可能来自各个不同的数据源，它们所遵循的文档模式（DTD 或 XML Schema）可能不同，文档之间存在一定的异构性，如果采用目前的 RDBMS 存储这些 XML 数据，则存在一定的局限性，即无法实现来自不同 DTD 的多个 XML 文档的有效集成存储。

针对以上问题，本文考虑在 XML 文档的关系存储系统中引进了数据集成的思想。所谓数据集成[3]就是将多个数据源中的数据结合起来存放在一个一致的数据存储中。XML 数据集成的目的是运用一定的技术手段将来自不同数据源中的 XML 文档信息按照一定的规则组织成为一个整体，从而减少数据存储的冗余

和不一致，并使得用户能有效地对数据进行查询操作。采用数据集成的思想存储XML数据的好处表现为：1）聚集相似的XML文档可以获得更好的存储映射，这将大大减少存储需要的额外空间；2）一组相似的 XML 文档需要创建的表格数量减少，并且文档中表示同一对象的内容被存储到数据库中的同一位置，这就减少了查询处理时所需的表格连接的次数，从而提高了查询的效率；3）对一组相似的XML文档建立索引，可以大大减少索引占用的存储空间，并提高检索效率；4）聚集XML文档还有一个好处是，可以有效地抽取一组相似的XML文档的公共模式，一个更加具体更加精确的模式在查询中是非常有用，而且可以使得查询不仅仅局限于部分相关的数据，这同时也提高了查询处理的效率。可见，XML 数据集成存储的方法不但弥补了传统存储策略的不足，而且还给存储和查询带来更多的优越性。

本文研究来自不同DTD的多个XML文档的集成存储问题。我们首先对XML文档进行预处理，并利用XML文档频繁模式挖掘算法挖掘出文档集的所有频繁模式，然后以频繁模式作为文档的特征进行XML文档相似度计算，并利用XML文档聚类算法实现对XML文档的有效聚类。在聚类的基础上，利用集成算法分析归纳各个类，生成集成模式，最后，从XML文档中抽取数据存入数据库，从而实现对来自不同DTD的多个XML文档的有效集成存储。

1.2 论文研究内容

1.2.1 论文的主要工作

本文以Internet中信息表示与交换的新标准XML作为研究对象，对来自不同DTD的多个XML文档的有效集成存储问题进行了研究。本文基于数据集成的思想，结合XML相关的技术以及几种数据挖掘的经典技术，提出了一种基于聚类的XML文档集成存储方法，并建立实验系统。由于很多XML文档没有相应自带的DTD模式，本文在不考虑XML文档DTD的前提下，直接对XML文档进行处理，主要工作包括：

1．研究XML文档频繁子树挖掘方法

从由带标签有序树构成的森林中挖掘嵌入式频繁子树的具体做法是：首先对XML文档进行预处理，提取XML文档的有效结构SST（Simplest Structural Tree最简结构树），然后应用本文提出的SSTMiner算法挖掘SST中的所有嵌入频繁子树。

2．研究基于频繁结构的XML文档聚类方法

频繁结构包括频繁路径和频繁子树，基于频繁结构的XML文档聚类方法的具体做法是：首先对 XML 文档进行频繁结构挖掘，获取最大频繁结构集， 由

于 SSTMiner 算法是一种非常有效的频繁子树挖掘算法，对 SSTMiner 算法稍加修改，即可得到 FrePathMiner 算法和 FreTreeMiner 算法，分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树，然后采用一种凝聚的层次聚类算法 XMLCluster，分别以最大频繁路径和最大频繁子树作为 XML 文档的特征，对文档进行聚类。

3．研究基于聚类的 XML 文档集成存储方法

集成存储方法分两个阶段进行：第一阶段是模式映射阶段（Schema Mapping），首先提出关系模式生成算法 SchemaGenerator，在聚类的基础上，根据模式映射规则，分析各个类中的 XML 文档，生成相应的关系数据库模式，接着提出关系模式集成算法 SchemaIntegrator，分析各个类中各文档相应的关系数据库模式，归纳出各个类的集成关系模式，最后提出全局关系模式集成算法 GlobalSchemaIntegrator，合并各个类的集成关系模式，构成全局关系模式，从而在关系数据库中生成表格；第二阶段是数据存储阶段（XML Storage），主要提出了数据抽取算法 DataExtractor，在模式映射的基础上，抽取 XML 数据到关系数据库，从而实现对来自不同 DTD 的多个 XML 文档的有效集成管理。

1.2.2 国内外相关研究

目前在信息集成方面，已经提出了很多数据集成方案，如参考文献[4,5,6,7,8,9]主要研究了目前存在的异构数据源的集成管理，分别提出了各自的数据集成方法，主要考虑使用一种理想的中间模式，然后研究如何实现各源数据模式到中间模式的转换，从而实现异构数据源的统一管理，为用户提供统一的查询机制，让用户像使用一个大数据库系统一样，用统一的方法使用来自不同数据源的各类数据。由于来自不同数据源的XML文档之间也存在一定的异构性，人们已经提出了将信息集成的思想用于多源XML数据的管理的方法，并做了一些研究：

文献[10]提出了一种有效的XML文档集成方法XClust，该方法充分利用文档的DTD模式，具体做法是通过考虑DTD元素节点的语义信息，结构信息，以及上下文叶子节点信息，来计算DTD模式之间的相似度，再对DTD模式进行聚类，最后分别对各个类中的DTD模式进行分析，归纳出一个DTD集成模式，再对所有的DTD集成模式进行分析，归纳出一个最终的完整的DTD集成模式，从而完成对XML文档模式的有效集成。

文献[11]主要研究基于数据的XML文档集成方法，考虑到多个XML数据源可能描述了相同的数据对象，但是它们的结构相差很大，于是需要采用一种有效的方法对这些XML数据源进行集成，以便于用户方便地访问和获取更加完整和有用的信息，文献中提出了基于叶子节点聚集的有效的XML连接算法Lax，主要是将两棵XML文档树分别分解为两组子树系列，通过分析计算两组子树对应叶子

节点之间的相似度，从而获得两个XML文档之间的相似度，然后根据相似度计算的结果，提出了一个有效的聚类算法对所有XML文档进行聚集。

文献[12]中为了解决对大集合中的XML文档执行有效的查询，提出了一种XML数据处理的数据结构XSD，从而使得用户可以利用该结构更快更精确地检索到XML文档库中的相关文档集，这种结构的思想是在计算多个XML文档相似度的基础上，对这些文档进行聚类，将各类中相似的XML文档模式集成为一个合并模式，即这种合并模式代表了包含这些相似XML文档模式的一个类，只要一个文档的模式属于这个类，那么它对应的XML文档即可视为该类的一个实例。

文献[13]在利用数据库对XML文档存储时，引进了节点聚集的思想，并提出了详细的节点聚集算法，将文档的DTD模式映射到数据库表格中，从而减少索引占用的空间以及所需的表格数量，从而大大提高查询效率。

文献[14]提出了SAMAG系统，主要目的是通过建立数据集成系统，以便于在用户查询时，自动的实现中间查询模式(抽象的DTD模式)到XML数据源模式(XML对应的DTD模式)的映射，从而充分利用XML文档库，在将查询模式映射到XML数据源模式的过程中，充分考虑了文档的语义信息和结构信息。

对于 XML 文档聚类，国内外学者已经做了许多研究工作[16~27]。其一般策略是将 XML 文档视为一棵标记树或图，然后计算两两文档之间的相似度，在此基础上提出一种算法对文档进行聚类。在进行相似度计算时，现有的方法可以归纳如下：

1）引入编辑距离[15]的概念，探测将一个文档转化为另一个文档所作操作的最小代价，以此衡量 XML 文档之间的相似度，如文献[16~21]。文献[16,17]将 XML 文档表示为图，利用图形匹配算法计算将一个图转换为另一个图的最小代价。文献[18,19]提出了一种树的编辑距离方法。文献[20,21]中把 XML 文档结构表示为一个时间序列,每个标签的出现相应于一个脉冲，进而通过相应傅立叶变换的频域来计算文档间的相似性。这种方法的缺陷是基于编辑距离时，文档操作的代价很难确定；

2）直接利用文档的结构信息，语义信息或者叶子节点信息来计算 XML 文档之间的相似度，如文献[21~24]。文献[21]在计算 XML 结构相似度的基础上，实现了 XML 文档的聚类。文献[22,23]同时利用了文档的语义信息和结构信息，但在内容信息上还需要做进一步的研究工作。文献[10,24] 采用了 XML 文档模式中元素相似度测量方法，将元素节点在文档中的位置表示为三个层次结构，较为全面地考虑了元素节点的语义信息，结构信息，以及上下文叶子节点信息，二者的区别是，文献[10]充分利用了 XML 文档自带的 DTD，而文献[24]则不考虑 DTD 模式，而是直接对 XML 文档进行处理。这种方法对相似度计算公式的要求比较高，而且公式中一些权重的取值也很难选择；

3）目前最新提出的基于频繁结构的方法，利用数据挖掘算法挖掘出文档集的所有频繁结构，以频繁结构作为文档的特征，进行 XML 文档相似度计算，如文献[25~27]。与前两种方法相比，对于本文的研究，基于频繁结构的方法具有更明显的优势，具体表现在：①通过高效的频繁结构挖掘算法找出文档频繁结构的集合，然后以频繁结构作为文档的特征，将文档转化为频繁结构的的特征向量表示，使得文档维数大大降低，文档相似度计算更简单；②将前两种中方法用于聚类算法中，聚类过程是透明的，用户只有被动地接受结果，而基于频繁结构的聚类算法的每一步都是有意义的，即含有类似频繁结构的文档才可能聚在一起，用户可以跟踪每一步获得的公共结构信息；③聚类结果的每个类可以用频繁结构来表示，其表达自然，容易理解，可以用于索引、查询和集成等系统中，帮助用户更有效地处理 XML 文档。

1.2.3 论文的特色和创新点

第一，提出一种有效的频繁子树挖掘方法，创新点包括：(1）提出最简结构树 SST 的定义，降低了 XML 文档的空间维数，从而大大降低了挖掘频繁子树时算法的复杂性；(2）提出 SSTMiner 算法，改进了 TreeMiner，减少了候选子树的产生，以及范围列表连接的次数，从而进一步提高了算法的效率。

第二，提出基于频繁结构的 XML 文档聚类方法，频繁结构包括频繁子树和频繁路径，创新点是：(1）通过对有效的 SSTMiner 算法进行修改，获得频繁结构的挖掘算法，该方法与传统的序列模式挖掘方法相比，其聚类结果具有更高的精确度；(2）设计一个自底向上的层次凝聚聚类算法，对 XML 文档模式进行聚类，这样，我们可以充分利用聚类过程产生的信息，来指导下一步聚类操作；聚类过程中，本文以频繁结构集作为文档的特征，构造了文档相似度计算公式和类相似度计算公式。

第三，在 XML 文档到关系数据库的存储系统中，引进了数据集成的思想，结合 XML 相关的技术以及几种数据挖掘的经典技术，提出了一种有效的基于聚类的 XML 文档集成存储方法，从而使 XML 文档在关系数据库中得到更有效地存储，并提高查询效率。

1.3 论文结构

全文总共分为五章，各章内容安排如下：

第一章为绪论，简要介绍了本文的研究背景与意义、本文的主要工作、国内外相关研究以及本文的特色和创新点。

第二章介绍了一种有效的 XML 文档频繁子树挖掘方法，提出了最简结构树

SST 的定义，改进了 TreeMiner 算法，获得 SSTMiner 算法，从而进一步提高了频繁子树挖掘的效率。

第三章介绍了一种基于频繁结构的 XML 文档聚类方法，该章首先对第二章提出的有效的频繁子树挖掘算法 SSTMiner 进行修改，分别获得最大频繁路径挖掘算法和最大频繁结构挖掘算法，用于挖掘 XML 文档的频繁结构，然后以挖掘到的频繁结构作为文档的特征，利用一种层次凝聚聚类算法对文档进行聚类，最后通过实验将本文提出的频繁结构挖掘算法以及传统的序列模式挖掘算法用于 XML 文档聚类的效果进行比较，证明了前者聚类结果具有更高的精确度。

第四章介绍了一种基于聚类的 XML 文档集成管理方法，该章首先将第三章提出的聚类方法用于 XML 文档集成管理系统中，在聚类的基础，提出了 XML 文档集成的相关算法，用于分析归纳聚类结果，最终实现 XML 文档到关系数据库的有效集成存储,然后给出了基于聚类的 XML 文档集成管理系统的总体框架，最后通过实验证明了该方法使得 XML 文档在关系数据库中得到更有效地存储，并提高了查询效率。

第五章是对研究工作的总结和展望。

第二章 挖掘 XML 文档频繁子树

频繁子树挖掘广泛应用于半结构化数据挖掘，web 挖掘，生物信息学，化合物结构分析等领域。本章主要研究从由带标签有序树构成的森林中挖掘嵌入式频繁子树，具体做法是：首先对 XML 文档进行预处理，生成最简结构树 SST(Simplest Structural Tree)，然后从 SST 中挖掘出频繁子树。本章提出了 SSTMiner 算法，该算法针对 TreeMiner 算法存在的瓶颈问题，结合当前所处理的 SST 的结构特点，进行改进，进一步提高了算法执行的效率。实验证明，本章提出的方法能够准确地高效地挖掘出 XML 文档中的频繁子树。

2.1 引言

频繁模式挖掘是数据挖掘中的一个重要问题，自从 Agrawal 等人开创性地提出著名的 Apriori 算法以来，其研究范围已逐渐从结构化数据，如频繁项集[28,29]和频繁序列模式挖掘[30]，扩展到半结构化数据，如频繁子树和频繁子图[31,32]。半结构化数据频繁模式发现的目的就是从大量的半结构化数据中发现有趣的模式，这些模式往往能够给出半结构化数据的概要信息，这类信息可以用来构成查询，也可以建立索引，还可以作为对数据库进行分类或聚类的基础等等。由于 XML 数据具有自描述性、灵活性、可扩展性的特点，因此被应用于越来越多的领域，并产生了丰富的 XML 数据，所以从 XML 数据中有效地挖掘频繁模式就成为了一项重要的工作。

关于频繁子树挖掘，国内外学者已经做了一些相关研究。Wang 和 Liu[33,34]提出了在单个 XML 文档中挖掘频繁子树的算法，他们的算法采用传统的类 Apriori 算法来解决这个问题，但挖掘的仅仅是直接子树。Asai[35]提出了一个类 apriori 的算法 FreqT 来挖掘有序标签树，他采用一种非常有用的最右路径扩展机制来产生候选树模式。还有一些挖掘不同类型的树模式的其他算法，例如 FreeTreeMiner[36]，该算法用于挖掘无序的直接的自由树（即不存在根节点），在[37]中 FreeTreeMiner 算法被用于图结构，从而在图库中抽取自由树；另外 PathJoin[38]，uFreqt[39]，uNot[40]和 HybridTreeMiner[41]，主要挖掘无序的直接的子树，TreeFinder[42]主要挖掘无序的嵌入子树等等。

目前，关于挖掘有序的嵌入的频繁子树的算法有 TreeMiner[43,44,45,46]，Xspanner[47]等等，其中 TreeMiner 算法是 Zaki 提出的，他和 Asai[32]几乎同时提出了有效挖掘频繁子树的方法，且都采用了最右候选生成模式，在[43,44,46]中，Zaki 给出了两种算法 TreeMiner 和 PatternMatcher，挖掘嵌入式频繁子树。TreeMiner 首先对文档进行字符串编码，然后经过类扩展，范围列表连接，来进行有效的候选等价类生成，支持度计算，以及频繁子树生成，其效率比 PatternMatcher 高出大约 4-20 倍，是迄今所发表的算法中最高效的挖掘嵌入式频繁子树的方法之一。在[46]中，TreeMiner 算法已被用于 XML 的分类器中。在国

内文献中，[48]吸收了 TreeMiner 中基于右分枝扩展方法构造完整的模式增长空间的思想，提出了 TG 算法挖掘频繁子树；[49]引用的 TreeMiner 的算法思想，提出了 XMLMINER 算法，从而完成对单个 XML 文档频繁模式的挖掘等等。然而，尽管 TreeMiner 算法非常有效，TreeMiner 算法仍存在不足之处：第一，TreeMiner 仍采用的是代价较高的候选模式产生一测试策略；第二，虽然通过对范围列表的求交运算进行支持度计算的效率大大高于一般的匹配操作，但在递归过程中其时间复杂度仍非常大，成为算法的又一个瓶颈。

针对以上问题，本章提出了一种从由带标签有序树构成的森林中挖掘嵌入式频繁子树的方法。具体做法是：首先对 XML 文档进行预处理，生成最简结构树 SST(Simplest Structural Tree)，然后从 SST 中挖掘出频繁子树。本章提出了 SSTMiner 算法，该算法针对 TreeMiner 算法存在的瓶颈问题，结合当前所处理的 SST 的结构特点，对 TreeMiner 算法进行改进，进一步提高了算法执行的效率。实验证明，本章提出的方法能够准确地高效地挖掘出 XML 文档中的频繁子树。本章的贡献是：①提出了最简结构树 SST，降低了 XML 文档的空间维数，从而大大降低了挖掘频繁子树时算法的复杂性。②提出了 SSTMiner 算法，改进了 TreeMiner，减少了候选子树的产生，以及范围列表连接的次数，从而进一步提高了算法的效率。

以下部分是这样组织的：第 2 节描述相关定义；第 3 节介绍 XML 文档预处理，生成最简结构树 SST；第 4 节介绍频繁子树挖掘算法。首先介绍传统的 TreeMiner 挖掘算法，然后给出改进策略以及改进后的算法 SSTMiner；第 5 节给出实验结果；最后是本章的结论。

2.2 相关定义

定义 2.1（XML 文档树） 一个 XML 文档可以表示成为一棵带标签的有序树 $T=(V,E,\mathring{a})$，其中，V 是节点集，$r\in V$ 为根节点；$E\subseteq V\times V$ 是有向边集合，每条边 $e\in E$ 代表节点间的嵌套关系，例如 $e=(x,y)$表示节点 x 到节点 y 的边，称 x 是 y 的父亲，y 是 x 的孩子，孩子之间是有序的；$\mathring{a}$是节点标签的有限集合；定义标签函数 $label:V?\ \mathring{a}$，将树中的每个节点分别指派一个标签。以下我们用 $label(x)$ 表示返回节点 x 的标签，用树的相关函数符号如：$parent(x)$表示返回节点 x 的父亲节点，$anc(x)$表示返回节点 x 的祖先节点集等等；

我们称树的集合为森林，记为 TDB，并且用$|TDB|$表示森林中包含的树的数目；用 $P=\{p_i=x_{i1}x_{i2}\ldots x_{ij}\ldots x_{im}|i=1,\ldots,n\}$表示路径集合，其中 n 表示某一棵树中包含的路径数量，m 表示某一条路径中包含的元素数量，p_i 表示第 i 条路径，x_{ij} 表示第 i 条路径的第 j 个节点，$\forall x,y\in V$，若存在一条从节点 x 到达节点 y 的路径，则称 x 为 y 的祖先，y 为 x 的后代；定义 n_i $(i=0\ldots k)$为先序遍历树 T 时节点对应的

遍历序号，其中 k 表示树 T 中节点的数目；若最后遍历的节点为 x，则称从 r 到 x 的路径为树 T 的最右路径；节点 x 的范围定义为 $scope(x)=[n_x\ ,\ n_r]$,其中 n_x 表示节点 x 的遍历次序，n_r 表示节点 x 的最后一个后代的位置。

定义 2.2（嵌入子树） 设 $T=(V,E,\mathring{a},)$和 $T'=(V',E',\mathring{a}')$是两棵有序标签树，如果存在一个 $V?\ V'$ 的映射函数 f，使得对于$\forall x,y\in V$ 它满足以下条件：

①$x\neq y\Leftrightarrow f(x)\neq f(y)$，

②$\forall x\in V\quad label(x)=label(f(x))$，

③$\forall x,y\in V\quad y=parent(x)\Rightarrow f(y)=anc(f(x))$，

则存在一个 T 到 T' 的树包含，T 为被包含树，T' 为包含树，T 称为 T' 的嵌入子树，记作 $T\subseteq T'$。若第③点中，x,y 在 T' 中仍保持父子关系，则称 T 为 T' 的直接子树。

定义 2.3（频繁子树） 设 TDB 是一个有序标签树集合（用$|TDB|$表示集合的元素个数），T 是一棵标签树，如果在 TDB 中存在 l 棵树$\{T_1\ ,\ldots,T_l\}$，使得 T 是$\{T_1\ ,\ldots,T_l\}$的嵌入子树，则T在TDB中的支持度为$\sup(T)=l/|TDB|$,设 e（$0<e\leq 1$）是用户定义的最小支持度阈值，如果 $\sup(T)\geq e$ ，则称 T 是 TDB 中的一棵 $e-$频繁子树。为了简便，我们定义最小支持计数为 $\min Sup=e\times|TDB|$，若 $l\geq\min Sup$，则称 T 是 TDB 中的一棵 $e-$频繁子树。

2.3 XML 文档预处理

在进行频繁模式挖掘前，为了降低算法复杂性，降低 XML 文档空间维数，我们首先使用 XML 解析器将 XML 文档解析成一棵 DOM 树，然后将 XML 文档最小化，提取 XML 文档的有效结构信息。通常，在一棵 XML 文档的 DOM 树中，会出现重复节点、重复路径以及与文档结构无关的字符信息。去除冗余路径和与文档结构无关的字符信息，可得到最简结构树 SST（Simplest Structural Tree）。

定义 2.4（最简结构树 SST） SST 是满足以下三个条件的一棵 DOM 树：在 SST 中，

①$\forall x\in V$，则 $x\notin V_T$，其中 V 表示所有节点集，V_T 表示文本类型节点集，即树中不含有任何与结构无关的文本节点，

②$\forall p_i=x_{i1}x_{i2}..x_{im}\in P$，则 $x_{ij}\neq x_{ij+1}(\ j=0,\ldots,m\text{-}1\ )$，其中 P 为 SST 的路径集合，即树中不含有接连同名元素节点，

③$\forall p_i,\ p_j\in P$，且 $i\neq j$，则 $p_i\neq p_j$，其中 P 为 SST 的路径集合，即树中不含有重复冗余路径（即任意两条路径不相同）。

算法 2.1 给出了获取 SST 的算法 getSST,这样,利用 getSST 算法对所有 XML 文档进行最小化，提取最简结构树 SST，降低了 XML 文档的空间维数，从而大

大降低了挖掘频繁子树时算法的复杂性。例如图 2.1(a)是一个 XML 文档对应的一棵 DOM 树，因为该树中包含了与结构无关的文本节点，另外树中还存在两种类型的冗余，一种情况是路径上接连重复出现同一个元素，如 ABB；另一种情况是存在相同重复的路径，如 AB，AE。经过算法 getSST 处理，过程与结果如图 2.1（b,c,d）所示。

算法 2.1 getSST

input: *T*

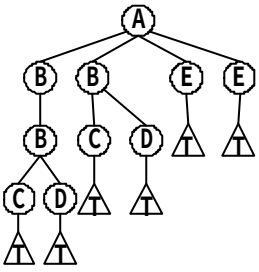
output: SST

getSST(*T*):

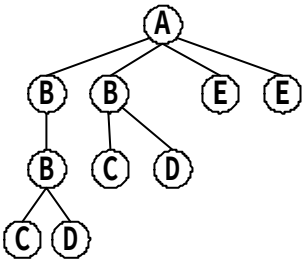
```
SST=Null;
//遍历DOM树T，去除文本节点
SST=removeTextNode(T);
//遍历SST，消除接连重复节点冗余
SST=removeRepeatedNode(SST);
//遍历SST，消除重复路径冗余
SST=removeRepeatedPath(SST);
```

removeRepeatedPath(SST):

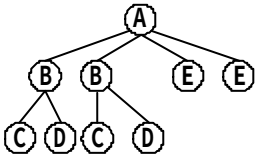
```
d=getDocNode(SST); //获取文档根节点
q.add(d); //加入队列
while(!q.isEmpty()) //层次遍历处理SST
 v=q.front(); //取队列元素
 pSet=∅; //初始化路径集合
 For every v's child node m Do //处理该节点的所有孩子节点
  p=getPath(m);
  If p ∉ pSet Then //路径不同，则路径加入路径集合，节点加入队列
   pSet=pSet∪{p};
   q.add(m);
  Else//路径相同，即冗余路径，合并冗余路径
   n=getNode(pSet(p));
   For every m's child node s Do
    copy s to n's child node set;
    delete m;
   EndIf
 EndFor
EndWhile
SST=getDoc(d); //获取消除重复路径冗余后的文档
```



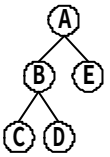
(a) 一个 XML 文档的 DOM 树示例



(b) 去除与结构无关的文本节点



(c) 消除接连重复节点冗余



(d) 消除重复路径冗余

图 2.1 获取最简结构树 SST 过程示例

2.4 XML 文档频繁子树挖掘

2.4.1 用 TreeMiner 算法挖掘频繁子树

Zaki 提出的 TreeMiner 算法[43,44,45,46]是迄今所发表的算法中最高效的挖掘嵌入式频繁子树的方法之一，本节首先简要介绍该算法。

TreeMiner 算法主要包含两个步骤:一是生成候选子树，二是对每个候选子树进行支持度计数，获得频繁子树。

1. 生成候选子树

定义 2.5（树的字串编码） 一个有序标签树 T 的字串表示记为 S ，通过如下过程得到：首先，初始化 $S=\varnothing$；然后从树的根节点开始先序遍历 T，并将当前节点的标签加入到 S 中；当遇到一个从子节点到父节点的回溯时，将一个特殊标签“ -1 ”加入到 S 中。

定义 2.6（前缀等价类） 设 T_1，T_2 是两棵 k-子树，其字串编码分别为 S_1，S_2，令 $?(S_1,k\text{-}1)$和 $?(S_2,k\text{-}1)$分别表示 S_1 和 S_2 的长度为 k-1 的前缀，若 $?(S_1,k\text{-}1)=?(S_2,k\text{-}1)$，则称 T_1，T_2 在同一个前缀等价类中，该等价类记为$[P]_k$ 。

等价类的表示包含两个域：一是长度为 k-1 的公共前缀 $?$，二是一个元素列表，其中每个元素用一个二元组(x, p)表示，x 表示第 k 个节点（即最后一个节点）的标签，p 表示 x 所依附的节点在深度优先遍历中的遍历序号，可见，每一个元素(x, p)代表一个 k-子树，该子树前缀为 $?$，最后一个节点的标签为 x，且连接到先序遍历序号为 p 的节点上。注意：每一个元素共享一个公共前缀，且当某个元素连接到公共前缀上时，则产生一个新的长度为 k 的前缀。

那么，给定$[P]_k$ ，如何获取$[P]_{k+1}$ 呢？候选树生成过程主要是考虑对元素列表中的每个元素进行求交扩展，包括自扩展。

定义 2.7（等价类扩展） 令(x, i)和(y, j)为等价类$[P]_k$ 的元素列表中的任何两个元素，$?$ 为等价类的前缀，令$[P^i_x]$表示由元素(x, i)扩展而来的等价类，对两个元素求交运算$(x, i)\otimes(y, j)$描述如下：

①当 $i=j$ 时，若 $?=\varnothing$，则将(y, n_x)加入$[P^i_x]$；若 $?\neq\varnothing$，则将(y, j)，(y, n_x)加入$[P^i_x]$，其中 n_x 表示节点 x 的先序遍历序号，

②当 $i>j$ 时，则将(y, j)加入$[P^i_x]$，

③当 $i<j$ 时，不产生候选子树。

所有由$[P]_k$ 衍生的 k+1-子树都可以通过对$[P]_k$ 的元素列表中的每对元素之间做求交运算得到。

2. 候选子树支持度计数

TreeMiner 算法采用了一种新颖的称为范围列表的树的表示方法，高效地支

持候选子树支持度计数。

定义 2.8（树的范围列表表示） 令(x, i)为等价类$[P]_k$的元素列表中的任何一个元素，$?$为等价类的前缀，则元素(x, i)代表了一棵 k-子树，那么用 $L(x, i)$表示该树的范围列表，该范围列表中的每个元素是一个三元组(t, s, m)，其中 t 表示树编号 id，s 表示节点 x 在编号为 id 的树中的范围，m 表示 k-子树的前$(k\text{-}1)$个节点在编号为 id 的树中的匹配位置。

由此该 k-子树的支持度计数可以通过范围列表中所有元素的 t 字段求得。

在 TreeMiner 算法中，当计算类扩展$(x, i) \otimes (y, j)$时，只可能有两种结果，即加入(y, j)或者(y, n_x)，分别代表两种情况，即 y 成为 x 的兄弟节点，或者 y 成为 x 的孩子节点，而且这两种情况分别代表了两棵新的候选子树，它们是否属于频繁子树取决于其支持度计数，而支持度计数要通过各自的范围列表求得，那么如何获取新的范围列表呢？新的范围列表的获取就要通过对范围列表连接运算求得，因为类扩展有两种可能的结果，所以范围列表连接对应两种不同的运算，分别称为范围内测试和范围外测试。

定义 2.9（范围列表连接运算） 令$(t_x, s_x, m_x) \in L(x)$，$(t_y, s_y, m_y) \in L(y)$，其中$s_x=[l_x, u_x]$，$s_y=[l_y, u_y]$，且$(x, i) \otimes (y, j)$运算后获得新子树 T，则范围列表连接运算 $L(x) \cap_{\otimes} L(y)$描述如下：

若类扩展加入(y, n_x)，即 T 中 y 成为 x 的孩子节点，则进行范围内测试，只需检查：

①$t_x = t_y = t$

②$m_x = m_y = m$

③$s_x \supset s_y$，即 $l_x = l_y$ 且 $u_x = u_y$

只要满足这三个条件，则增加三元组$(t_y, \{m_y \cup l_x\}, s_y)$到新的范围列表中，增加了元素$(y, n_x)$，意味着等价类中增加了子树$(?y)$；

若类扩展加入(y, j)，即 T 中 y 成为 x 的兄弟节点，则进行范围外测试，只需检查：

①$t_x = t_y = t$

②$m_x = m_y = m$

③$s_x \prec s_y$，即 $u_x < u_y$

只要满足这三个条件，则增加三元组$(t_y, \{m_y \cup l_x\}, s_y)$到新的范围列表中，增加了元素$(y, j)$，意味等价类中增加了子树$(?\text{-}1\text{-}1\ldots\text{-}1y)$，其中-1 的个数等于 j 到 x 的路径长度。

求得范围列表的所有元素后，就可根据范围列表中所有元素的 t 字段算出支持度计数，从而判断新的子树是否为频繁子树，若不是，则将它在等价类中删除。

3. TreeMiner 算法描述及实例分析

下面首先给出 TreeMiner **算法具体描述，见算法** 2.2**，然后给出将** TreeMiner **算法用于** XML **文档频繁模式挖掘的部分过程示例，见图** 2.2**。**

算法 2.2 TreeMiner

TREEMINER(D,minSup):

F_1 ={frequent 1-subtrees};

F_2 ={classes[P]$_1$ of frequent 2-subtrees};

For all [P]$_1 \in F_2$ **Do** Enumerate-Frequent-Subtrees([P]$_1$);

Enumerate-Frequent-Subtrees([P]):

For each element $(x, i) \in$ [P] **Do**

$[P_x^i] = \varnothing$;

For each element $(y, j) \in$ [P] **Do**

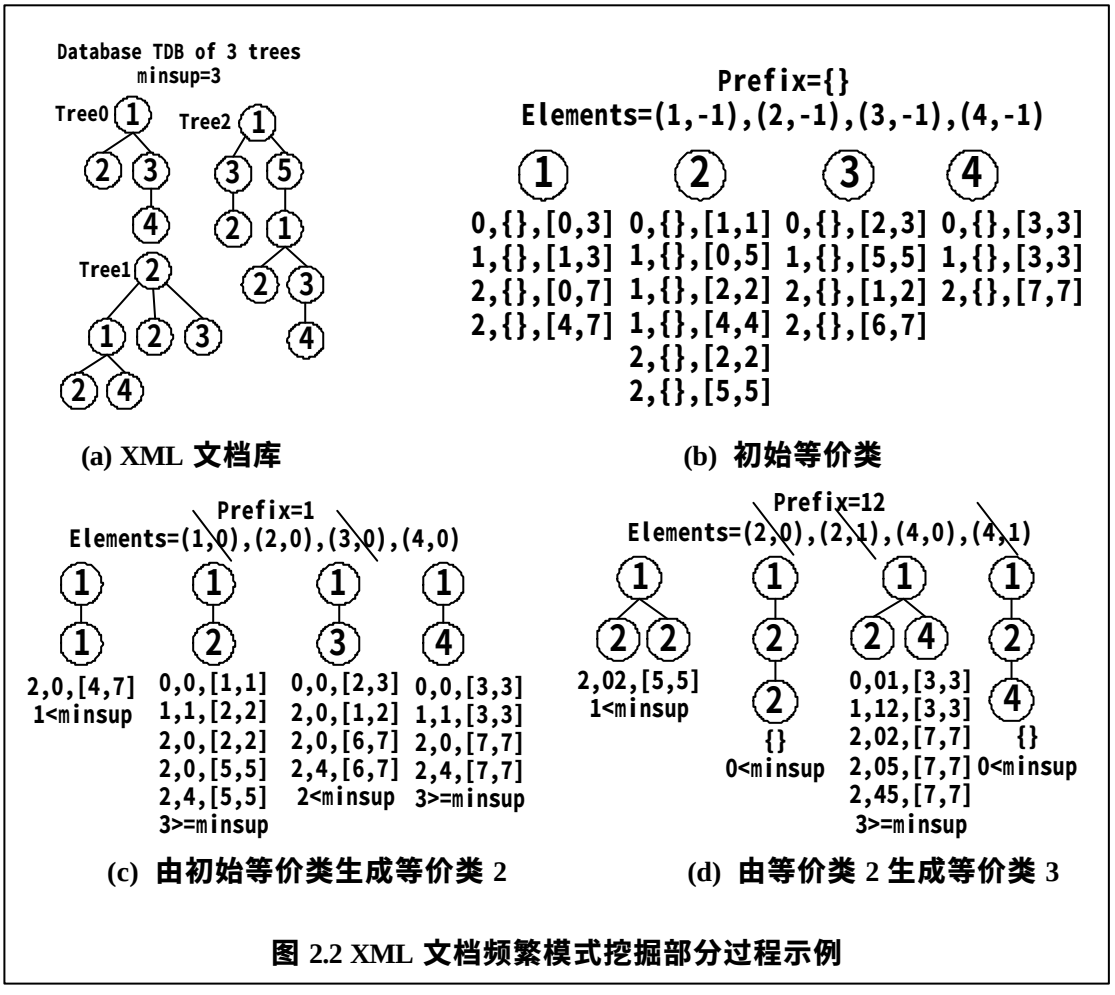
$R = \{(x, i) \otimes (y, j)\}$; //**元素类扩展**

$L(R) = \{L(x) \cap_{\otimes} L(y)\}$; //**范围列表连接**

If for any $r \in R, r$ is frequent **Then**

$[P_x^i] = [P_x^i] \cup \{r\}$; //**加入前缀等价类**

Enumerate-Frequent-Subtrees($[P_x^i]$); //**深度优先搜索等价类**



(a) XML 文档库 **(b) 初始等价类**

(c) 由初始等价类生成等价类 2 **(d) 由等价类 2 生成等价类 3**

图 2.2 XML 文档频繁模式挖掘部分过程示例

2.4.2 改进后的算法 SSTMiner

尽管 TreeMiner 算法非常有效，TreeMiner 算法仍存在不足之处：第一，TreeMiner 仍采用的是代价较高的候选模式产生——测试策略；第二，虽然通过对范围列表的求交运算进行支持度计算的效率大大高于一般的匹配操作，但在递归过程中其时间复杂度仍非常大，成为算法的又一个瓶颈。针对上述算法的不足，本文提出了改进算法 SSTMINER。

1. 改进策略一 ——减少候选子树的产生

针对 TreeMiner 算法的第一个不足之处，我们相应提出了改进策略一，考虑减少候选子树的产生。

首先对 XML 文档进行预处理，生成最简结构树 SST；其次，修改 TreeMiner 算法中的元素求交运算，除去自扩展。在 TreeMiner 算法中，候选树生成过程主要是考虑对元素列表中的每个元素进行求交扩展，包括自扩展。然而自扩展生成的子树结构必定包含了冗余路径，或者冗余的结构信息，例如当$(1,-1)$进行自扩展时，即$(1,-1)\otimes(1,-1)$，最终产生新元素$(1,0)$，由新元素$(1,0)$和前缀 1 构成的候选子树是一棵包含了冗余结构的子树。

回顾 SST 的定义可知，SST 是一棵不包含任何冗余结构的 XML 文档树，所以它的任何子树也不可能包含冗余结构，由此可知，从由 SST 构成的森林中挖掘出来的频繁子树，不可能包含冗余结构，这样，我们在候选树生成过程中，对元素列表中的每个元素进行求交扩展时，可以除去自扩展，从而大大减少了候选子树的数量。所以对等价类扩展定义修改如下：

定义 2.7$^{\#}$（等价类扩展） 令(x, i)和(y, j)为等价类$[P]_k$的元素列表中的任何两个元素，$?$ 为等价类的前缀，令$[P^i_x]$表示由元素(x, i)扩展而来的等价类，对两个元素求交运算$(x, i)\otimes(y, j)$描述如下：

①x ? y，

②当 $i=j$ 时，若 $?=\emptyset$，则将(y, n_x)加入$[P^i_x]$；若 $?\neq\emptyset$，则将(y, j)，(y, n_x)加入$[P^i_x]$，其中 n_x 表示节点 x 的先序遍历序号，

③当 $i>j$ 时，则将(y, j)加入$[P^i_x]$，

④当 $i<j$ 时，不产生候选子树。

2. 改进策略二 ——减少范围列表中元素之间连接的次数

TreeMiner 算法的第二个不足之处是，在进行候选子树支持计数计算时，首先要通过前一个等价类中的范围列表之间的求交运算，求得该候选子树的范围列表，范围列表之间的求交运算问题成为算法的瓶颈问题，由此提出了改进策略二，主要是减少范围列表中元素之间连接的次数。

在 TreeMiner 算法中，范围列表进行求交运算 $L(x)\cap_{\otimes}L(y)$，实际上就是范围

列表的所有元素之间进行互相连接，元素之间互相连接就是比较元素的各个字段的值，判断是否满足范围内测试或者范围外测试中的三个条件，假设 $L(x)$中有 m 个元素，$L(y)$中有 n 个元素，则范围列表求交运算时，逐个 $L(x)$中的元素，与 $L(y)$中的所有元素进行连接，所以需要进行$(m*n)$次元素之间的连接，那么对于简单的例子，连接的次数一般不会很大，但随着 XML 文档的海量增长，连接操作必将大大影响整个算法的效率。

由此，在 SSTMiner 算法中，我们考虑减少连接操作的次数。观察范围列表中的元素排列次序得知，元素的所在位置是严格按照第一字段 t 进行有序排列，根据元素排列恒为有序的性质，我们对原有的连接算法进行改进，改进连接算法的思路是：逐个取出 $L(x)$中的元素，与 $L(y)$中的元素进行连接，当满足条件 $t_x < t_y$ 时，则根据元素排序的有序性，$L(y)$中余下的所有元素都满足 $t_x < t_y$，所以必定不满足三个条件中的第一个，所以不必再取出 $L(y)$余下的所有元素与 $L(x)$中当前处理的元素进行连接，而转向处理 $L(x)$中的下一个元素，这样，使得在元素之间的连接操作只在 $t_x = t_y$ 的元素中进行，从而大大减少了连接次数。

3. SSTMiner 算法描述及实例分析

下面首先给出 SSTMiner 算法具体描述，见算法 2.3，然后给出将 SSTMiner 算法用于 XML 文档频繁模式挖掘的部分过程示例，见图 2.3。

算法 2.3 SSTMiner

input: SSTDB

output: $F=\{f_1,...f_m\}$

SSTMiner（SSTDB，minSup）:

$[P]_1$= getP1(SSTDB,minSup); //初始等价类

$F = F \cup F_{[P]_1}$; //从等价类1中获取1-频繁子树

Find_Frequent_Subtrees($[P]_1$); //从等价类1开始，深度优先搜索所有等价类，并获取频繁子树

Find_Frequent_Subtrees([P]):

For each element $(x,i)\in[P]$ **Do**

$[P_x^i]=\varnothing$;

For each element $(y,j)\in[P]$ **Do**

If$(x\neq y)$ **Then** //不做自扩展

$R=\{(x,i)\otimes(y,j)\}$; //元素类扩展

$L(R)=\{L(x)\cap_{\otimes} L(y)\}$; //范围列表连接，由函数ScopeListJoin()实现

For each $r\in R$ **Do** //对于元素类扩展结果的每个新元素

If Sup(r)>=minSup **Then** //若新元素的计数大于等于最小支持计数，则

$[P_x^i]=[P_x^i]\cup\{r\}$; //加入新元素到等价类，新元素和等价类中的前缀代表一棵频繁子树

$F = F \cup F_{[P_x^i]}$; //从新等价类$[P_x^i]$中获取频繁子树

Find_Frequent_Sbutrees($[P_x^i]$); //深度优先搜索等价类

ScopeListJoin($L(x),L(y)$):

$LNew = \varnothing$;

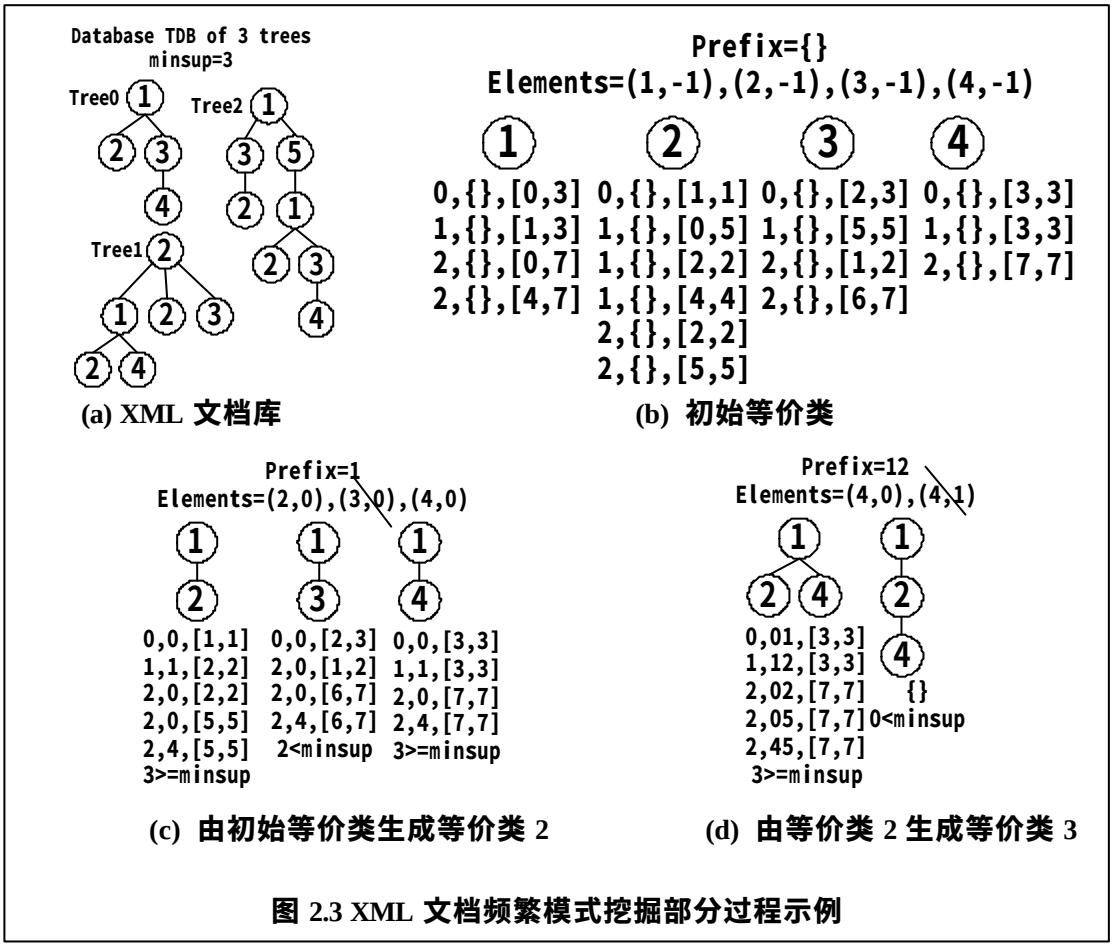
For each element $(t_x, s_x, m_x) \in L(x)$ **Do**

For each element $(t_y, s_y, m_y) \in L(y)$ **Do**

If $(t_x < t_y)$ **Thenbreak**; //若为真，则根据元素排列有序性得Ly余下元素的$t_y > t_x$，所以退出

If $(t_x == t_y)$ **And** $(m_x == m_y)$ **And** $(cmpScope(s_x, s_y) == true)$ **Then**

$LNew = LNew \cup (t_y, \{m_y \cup l_x\}, s_y)$;



(a) XML 文档库　(b) 初始等价类

(c) 由初始等价类生成等价类 2　(d) 由等价类 2 生成等价类 3

图 2.3 XML 文档频繁模式挖掘部分过程示例

2.5 实验结果及性能分析

我们在 Jbuilder 环境下，用 java 语言实现了 TreeMiner 算法及其改进后的 SSTMiner 算法，实验硬件平台是一台 512M 内存，主频 2.50GHz 的 Pentium(R) 4 微机，安装了 Window XP 操作系统。总共做了两组实验，对两个算法进行比较，第一组测试支持度大小对算法性能的影响，即不同支持度测试，第二组测试数据集规模大小对算法性能的影响，即不同规模测试。实验数据集来自文献[50]。

2.5.1 不同支持度测试

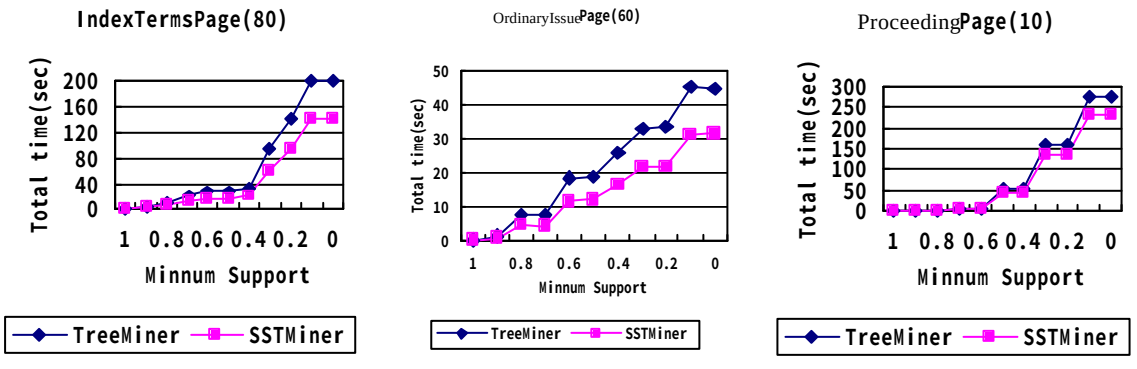
本组实验取出遵循 IndexTermPage.dtd，OrdinaryIssuePage.dtd，ProceedingPage.dtd 的 3 个 XML 文档集，每个文档集各包含 80，60，10 个 XML 文档，实验分 3 次进行，支持度取值范围为 1=min*Sup*=0。首先取出一个文档集进行文档预处理，生成 SST，由于文档集的所有文档统一于同一个 DTD，所以所有 SST 非常相似或基本相同，为了实验的效果更为明显，对生成的 SST 删除若干节点或增加若干新节点，使得它们具有一定的相似性但又不完全相同；在此基础上，测试在不同支持度下，分别用 TreeMiner 算法和 SSTMiner 算法对 SST 进行处理所需要的时间。

3 个文档集在不同的支持度下，可以挖掘出的频繁子树数量如表格 2.1 所示，分别用 TreeMiner 算法和 SSTMiner 算法挖掘这些数目的频繁子树所需要的时间如图 2.4 所示。

由实验结果可以看出，当文档数量不变时，在相同支持度下，算法 SSTMiner 的效率高于 TreeMiner 算法，而且随着支持度的减少，其性能优越性也更加明显。

Frequent Tree Num / MinSup / Dataset(Num)	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0
IndexTermsPage(80)	0	779	3342	9488	15120	15120	21013	114719	232991	622634	622634
OrdinaryIssuePage(60)	0	606	5189	5189	17229	17229	29901	50894	50894	132815	132815
ProceedingPage(10)	0	2232	2232	54402	54402	505765	505765	2139877	2139877	5260182	5260182

表 2.1 三个文档集在不同的支持度下的频繁子树数量



(a) (b) (c)

图 2.4 支持度测试

2.5.2 不同规模测试

本组实验取出遵循 IndexTermPage.dtd,OrdinaryIssuePage.dtd 的 1 个 XML 文档集，文档集包含了 10 个 XML 文档，实验分 6 次进行，文档集数量在这 10 个文档的基础上逐次递增，分别为 10，100，200，300，400，500，每次支持度取值均为 0.05。与第一组实验类似，首先对 10 个文档稍做修改，使得它们经过预处理后生成的 SST 具有一定的相似性但又不完全相同；在此基础上，测试在不同文档数量下，分别用 TreeMiner 算法和 SSTMiner 算法对 SST 进行处理所需要的时间。

10 个文档在最小支持度为 0.05 时，可以挖掘出的频繁子树数量为 288290，对于不同数量构成的文档集，分别用 TreeMiner 算法和 SSTMiner 算法挖掘这些数目的频繁子树所需要的时间如图 2.4 所示。

由实验结果可以看出,当支持度不变时,在相同文档数量下下,算法 SSTMiner 的效率高于 TreeMiner 算法，而且随着文档数量的增长，其性能优越性也更加明显。

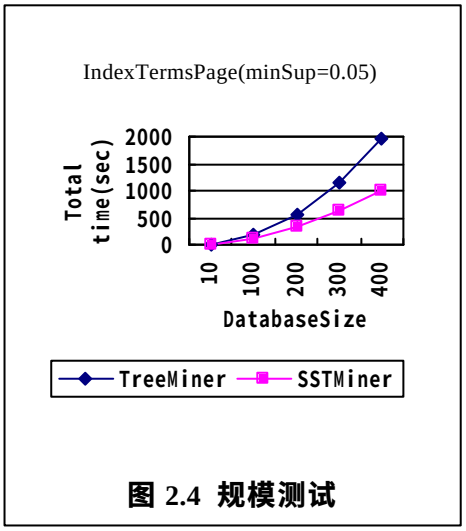


图 2.4 规模测试

2.6 本章小结

本章提出了一种从由带标签有序树构成的森林中挖掘嵌入式频繁子树的方法[51]:首先对 XML 文档进行预处理,生成最简结构树 SST;接着利用 SSTMiner 算法挖掘由 SST 构成的森林中的频繁子树。SSTMiner 算法不但继承了 TreeMiner 算法的优点，例如采用了 TreeMiner 中树的编码方式，前缀等价类的定义，为生成候选等价类而进行的类扩展运算,以及为获取候选子数支持度计数值而进行的范围列表连接运算等等，而且针对 TreeMiner 算法存在的瓶颈问题，以及结合当前所处理的 SST 的结构特点，对 TreeMiner 算法进行改进，改进策略主要考虑到

减少候选子树的产生以及减少范围列表元素连接的次数两个方面，从而进一步提高了算法执行的效率。最后本文通过实验证明该方法的有效性。

频繁子树挖掘广泛应用于半结构化数据挖掘，web 挖掘，生物信息学，化合物结构分析和挖掘等领域。下一章，我们将考虑把该方法用于对 XML 文档的聚类中，即先用该方法挖掘出 XML 文档的所有频繁子树，然后以这些频繁子树作为 XML 文档的特征，实现对 XML 文档的有效聚类。

第三章 基于频繁结构的 XML 文档聚类方法

本章研究基于频繁结构的 XML 文档聚类方法，其频繁结构包括频繁路径和频繁子树。第二章已经介绍一种有效的挖掘 XML 文档中所有嵌入频繁子树的算法 SSTMiner，为了降低聚类算法的复杂性，本章首先对 SSTMiner 算法进行修改，得到 FrePathMiner 算法和 FreTreeMiner 算法，分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树，在此基础上，提出一种凝聚的层次聚类算法 XMLCluster，分别以最大频繁路径和最大频繁子树作为 XML 文档的特征，对文档进行聚类。通过实验比较了分别将 FrePathMiner 算法，FreTreeMiner 算法，以及传统的改进的序列模式挖掘算法 ASPMiner(Adaped Sequential Pattern Miner)用于 XML 文档聚类的效果。结果表明 FrePathMiner 算法和 FreTreeMiner 算法找到频繁结构的数量都比传统的 ASPMiner 算法多，这就可以为文档聚类提供更多的结构特征，从而获得更高的聚类精度。

3.1 引言

目前针对基于频繁结构的聚类，文献[25]提出了 PBClustering（Path_Based Clustering）聚类算法，先用数据挖掘中的类增量算法挖掘出文档集中的公共路径，以公共路径作为文档的特征，对文档进行相似度计算并聚类。文献[26]也提出了基于频繁路径的聚类算法。在文献[22]中，Lee 等人也采用改进的序列模式挖掘算法，挖掘出 XML 文档的最大频繁路径，然后以最大频繁路径作为 XML 文档的特征，计算 XML 文档之间的相似度。文献[27]提出了 TreeFinder 算法，用于挖掘文档集中的频繁子树。文献[25]的实验部分将 PBClustering 算法和 TreeFinder 算法聚类结果进行对比，验证了 PBClustering 算法比 TreeFinder 算法具有更高的精确度。然而实际上，文献[27]已经表明 TreeFinder 算法挖掘频繁子树时存在丢失的现象，所以将 TreeFinder 频繁子树挖掘算法用于聚类，其聚类结果准确度难以保证问题也是不可避免的。

本章研究基于频繁结构的 XML 文档聚类方法，其频繁结构包括频繁路径和频繁子树。为了降低聚类算法的复杂性，首先对第二章提出的挖掘 XML 文档中所有嵌入频繁子树的算法 SSTMiner 进行修改，得到 FrePathMiner 算法和 FreTreeMiner 算法，分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树。然后，提出一种凝聚的层次聚类算法 XMLCluster，分别以最大频繁路径和最大频繁子树作为 XML 文档的特征，对文档进行聚类，最后通过实验比较分别将 FrePathMiner 算法，FreTreeMiner 算法，以及传统的改进的序列模式挖掘算法 ASPMiner(Adaped Sequential Pattern Miner)[22]用于 XML 文档聚类的效果。

本章以下部分组织如下：第 2 节介绍一些相关定义；第 3 节介绍最大频繁路径挖掘算法 FrePathMiner 和最大频繁子树挖掘算法 FreTreeMiner；第 4 节介绍基于频繁结构的文档聚类算法 XMLCluster；第 5 节是实验部分；最后是本章结论。

3.2 相关定义

定义 3.1（等价树） 设 $T=(V,E,\mathring{a},)$和 $T'=(V',E',\mathring{a}')$是两棵有序标签树，如果存在一个 $V?\ V'$ 的映射函数 f，使得对于$\forall x,y\in V$ 它满足以下条件：

①$x\neq y\Leftrightarrow f(x)\neq f(y)$，

②$\forall x\in V\quad label(x)=label(f(x))$，

③$\forall x,y\in V\quad y=parent(x)\Rightarrow f(y)=anc(f(x))$，则存在一个 T 到 T' 的树包含，T 为被包含树，T' 为包含树，T 称为 T' 的嵌入子树，记作 $T\subseteq T'$。若第③点中，x,y 在 T' 中仍保持父子关系，则称 T 为 T' 的直接子树。若 T 是 T' 的直接子树，且 T' 也是 T 的直接子树，则称 T 和 T'是等价树，即 T 和 T'完全相同，记为 $T'\equiv T$。

定义 3.2（最大公共子树） 设 $T,T_1,\ldots,T_l$ 是标签树，我们称 T 是 $T_1,\ldots,T_l$ 的最大公共子树当且仅当满足以下条件：

①$\forall i\in\{1,\ldots,l\}\ T\subseteq T_i$，

②如果存在一个标签树 T'，使得 $T\subseteq T'$，且$\forall i\in\{1,\ldots,l\}\ T\subseteq T'$，则 $T'\equiv T$。

定义 3.3（最大频繁子树） 设 TDB 是一个有序标签树集合（用$|TDB|$表示集合的元素个数），T 是一棵标签树，如果在 TDB 中存在 l 棵树$\{T_1,\ldots,T_l\}$，使得 T 是$\{T_1,\ldots,T_l\}$的最大公共子树，则 T 在 TDB 中的支持度为 $\sup(T)=l/|TDB|$；设 e（$0<e\leq 1$）是用户定义的最小支持度阈值，如果 $\sup(T)\geq e$，则称 T 是 TDB 中的一棵 $e-$ 最大频繁子树。为了简便，我们定义最小支持计数为 $\min Sup=e\times|TDB|$，若 $l\geq\min Sup$，则称 T 是 TDB 中的一棵$e-$最大频繁子树。

最大公共路径和最大频繁路径的定义类似子树的定义，这里略。

3.3 XML 文档最大频繁结构挖掘

第二章我们已经提出了一种从由带标签有序树构成的森林中挖掘嵌入式频繁子树的方法，该方法主要包括两个步骤：1）XML 文档预处理，提取文档的有效结构 SST（Simplest Structural Tree 最简结构树）；2）对 SST 进行处理，挖掘出文档中所有的频繁子树。在对 SST 的处理中，我们提出的 SSTMiner 算法是一种非常有效的挖掘 XML 文档所有嵌入频繁子树的算法。对它稍做修改，可以获得最大频繁路径挖掘算法 FrePathMiner 和最大频繁子树挖掘算法 FreTreeMiner，在此基础上，我们即可分别采用最大频繁路径和最大频繁子树作为 XML 文档的聚类特征，对文档进行有效地聚类。下面介绍如何获得 FrePathMiner 和 FreTreeMiner 算法。

3.3.1 最大频繁路径挖掘算法 FrePathMiner

FrePathMiner 算法用于挖掘 XML 文档中的所有最大频繁路径，对 SSTMiner 算法稍做修改即可获得，具体修改之处为：

1)：修改第二章的等价类扩展定义。等价类扩展用于生成所有可能存在的候选子树，其过程就是对两个元素求交运算，记为$(x, i) \otimes (y, j)$，其结果只有两种情况，即将(y, n_x)加入$[P^i_x]$和将(y, j)加入$[P^i_x]$，若加入(y, n_x)则表示，在原有子树的基础上，为节点 x 增加一个孩子节点 y，构成新的候选子树，若加入(y, j)则表示，在原有子树的基础上，为节点 x 增加一个兄弟节点 y，构成新的候选子树。由此可见，当加入(y, n_x)时，结果 y 在 x 的范围内，形成路径，当加入(y, j)时，结果 y 在 x 的范围外，形成子树，所以在 FrePathMiner 算法中，若使得等价类扩展结果只加入(y, n_x)而不加入(y, j)，即可生成所有可能存在的候选路径，且$(y, n_x)=(y, j+1)$。所以对等价类扩展定义修改如下：

令(x, i)和(y, j)为等价类$[P]_k$的元素列表中的任何两个元素，$?$ 为等价类的前缀，令$[P^i_x]$表示由元素(x, i)扩展而来的等价类，对两个元素求交运算$(x, i) \otimes' (y, j)$描述如下：

①x ? y，

②当 $i=j$ 时，将$(y, j+1)$加入$[P^i_x]$，

③当 $i>j$ 或者 $i<j$ 时，不产生候选子树。

2)：修改第二章的范围列表连接定义。范围列表表示方法，高效地支持候选子树支持度计数，在进行支持度计数前，先要进行范围列表连接运算，记为$L(x) \cap_{\otimes} L(y)$。因为第二章中类扩展可能生成两种候选子树，所以范围列表连接也有两种相应的运算，分别称为范围内测试和范围外测试。由于本节经过改进后的等价类扩展只可能产生一种结果，即加入元素$(y, j+1)$，使得 y 成为 x 的孩子节点，所以在 FrePathMiner 算法中，范围列表连接运算只进行范围内测试，而不存在范围外测试，所以对范围列表连接定义修改如下：

令$(t_x, s_x, m_x) \in L(x)$，$(t_y, s_y, m_y) \in L(y)$，其中 $s_x=[l_x, u_x]$，$s_y=[l_y, u_y]$，且$(x, i) \otimes' (y, j)$运算后获得新子树 T，则范围列表连接运算 $L(x) \cap_{\otimes}' L(y)$描述如下：

若类扩展加入$(y, j+1)$，即 T 中 y 成为 x 的孩子节点，则进行范围内测试，只需检查：

①$t_x = t_y = t$

②$m_x = m_y = m$

③$s_x \supset s_y$，即 $l_x = l_y$ 且 $u_x = u_y$

只要满足这三个条件，则增加三元组$(t_y, \{m_y \cup l_x\}, s_y)$到新的范围列表中，

增加了元素(y, n_x)，意味着等价类中增加了子树$(?y)$；

SSTMiner 算法经过以上两处修改后能挖掘出了所有的频繁路径，为了降低聚类算法的复杂性，我们进一步对获得的所有的频繁路径进行处理，找出最大的频繁路径。获得最大频繁路径的具体算法描述见算法 3.1。

算法3.1 getMaxFre

input: $F=\{f_1,...f_m\}$

output: max$F=\{f_1,...f_n\}$

getMaxFre (F)**:**

For each element $f_i, f_j \in F$ Do

If($f_i \subseteq$ (any $T' \in \{T_1,...,T_l\}$)) And ($f_j \subseteq$ (any $T' \in \{T_1,...,T_l\}$)) And ($f_i \subseteq f_j$) Then

//若两个频繁模式都包含于文档库中的同一个文档子集，且一个模式包含于另一个模式

$F = F - \{f_i\}$; //则从频繁模式集合中去除前一个模式，保留后一个模式

max$F = F$;

3.3.2 最大频繁子树挖掘算法 FreTreeMiner

FreTreeMiner 算法用于挖掘 XML 文档中的所有最大频繁子树，也是对 SSTMiner 算法稍做修改即可获得。由于 SSTMiner 算法已经挖掘出了所有的频繁子树，所以 FreTreeMiner 算法只要在 SSTMiner 算法的基础上，进一步处理所有的频繁子树，找出最大的频繁子树即可。获得最大频繁子树的具体算法描述类似算法 3.1，这里略。

3.3.3 实例分析

表 3.1 列举了三个简单文档实例，并给出采用 FrePathMiner 算法，FreTreeMiner 算法，以及传统的改进的序列模式挖掘算法 ASPMiner(Adaped Sequential Pattern Miner)[22]可以挖掘到的最大频繁结构，其中 FreTreeMiner 挖掘到最大频繁子树采用第二章中树的字符串编码方式进行描述。比较 FrePathMiner 算法和 ASPMiner 算法的挖掘结果可以看出，前者挖掘到更多的最大频繁路径，其根本原因是两种算法的最大频繁路径的定义不同，FrePathMiner 采用最大公共子树的概念来定义最大频繁路径，既考虑频繁路径之间的包含关系，也考虑路径在文档集中出现的情况，例如虽然频繁路径 A,B,C 包含频繁路径 A,C，但是 A,B,C 是 T_0,T_1 的最大公共子树，而 A,C 则是 T_0,T_1,T_2 的最大公共子树，两条路径分别是两个不同文档集合的最大频繁子树，文档集合中文档的数目都满足最小支持计数，所以它们都被认为是最大频繁路径，而 ASPMiner 算法是一种类增量算法，若两条频繁路径序列互相包含，则去除被包含的那条路径而不考虑该路径在文档集中出现的情况，如频繁路径 A,C 被频繁路径 A,B,C，则去除 A,C。

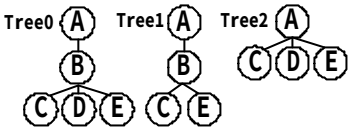
Database / MaxFreq / Structure / Algorithm	Database TDB of 3 trees minsup=2 
FrePathMiner	A,C; A,E; A,B,C; A,B,E; A,D;
FreTreeMiner	A,C,-1,E; A,B,C,-1,E; A,C,-1,D,-1,E;
ASPMiner	A,B,C; A,B,E; A,D;

表 3.1 三种算法挖掘到的最大频繁结构

3.4 XML 文档聚类

在最大频繁路径挖掘算法 FrePathMiner 和最大频繁子树挖掘算法 FreTreeMiner 的基础上，我们提出了 XML 文档聚类算法 XMLCluster。其基本思想是，分别利用 FrePathMiner 算法和 FreTreeMiner 算法作为文档的特征抽取器，将挖掘到的两种频繁结构（即频繁路径和频繁子树）用于 XML 文档的特征表示，从而在降低文档的空间维数且保留文档的必要结构信息和语义信息的基础上，对文档进行聚类，最终实现基于最大频繁路径的文档聚类和基于最大频繁子树的文档聚类两种方法。文档特征的数量可以由频繁结构挖掘算法中的最小支持度 e 来控制。下面先介绍两个相似度计算公式，然后详细介绍 XMLCluster 算法。

3.4.1 相似度计算

相似度计算包括文档相似度计算和聚类相似度计算，分别如公式 1 和公式 2 所示：

文档之间相似度计算公式：$$Sim(T_1,T_2)=\frac{|F(T_1)\cap F(T_2)|}{|F(T_1)\cup F(T_2)|} \quad (1)$$

其中 T_1，T_2 表示两棵文档树，$F(T_1)$，$F(T_2)$分别表示这两个文档的频繁结构集合，$|F(T_1)\cap F(T_2)|$表示这两个文档公共的频繁结构数目，$|F(T_1)\cup F(T_2)|$表示这两个文档所有的频繁结构数目。

聚类之间相似度计算公式：$$Sim(C_1,C_2)=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}Sim(T_i,T_j)}{m*n} \quad (2)$$

其中，C_1，C_2 表示两个类，$C_1=\{T_1,\ldots,T_i,\ldots T_m\}$，$C_2=\{T_1,\ldots,T_j,\ldots T_n\}$，$Sim(T_i, T_j)$表示文档 T_i，T_j 之间的相似度。

3.4.2 聚类算法 XMLCluster

凝聚的层次聚类算法是一个经典的聚类算法，它是一种自底向上不断合并类的过程，最终生成的类的数目取决于一个终止标准。本章基于凝聚的层次聚类算法的思想，提出了 XMLCluster 算法对 XML 文档进行聚类。算法实现过程归纳为三个步骤：1）建立文档相似度矩阵；2）建立初始类；3）聚类。

建立文档相似度矩阵的具体方法是，首先在频繁结构挖掘算法的基础上，将各文档表示成频繁结构特征集合形式，然后根据文档相似度计算公式（见 3.4.1 节）计算文档集中两两文档之间的相似度，进而建立文档相似度矩阵。设 $TDB=\{T_1,\ldots,T_n\}$，n 表示文档数目，则文档相似度矩阵是一个 $n\times n$ 维的矩阵 $Maxtri[n][n]$；令 $Sim(T_i, T_j)$ $(1=i, j=n)$表示文档 T_i，T_j 之间的相似度，则 $Maxtri[i][j]=Sim(T_i, T_j)$。由此可见，文档相似度矩阵的特点是：①$Maxtri[n][n]$是一个对称矩阵，且 $Maxtri[i][i]=1$；②当文档 T_i，T_j 越相似，则矩阵相应位置的值越接近于 1，反之越接近于 0。

建立初始类的具体方法是，为每个文档建立一个初始类，类中的元素即为该文档。设 $TDB=\{T_1,\ldots,T_n\}$，n 表示文档数目，则定义初始化聚类集合 $C=\{C_1,\ldots,C_n\}$，其中 $C_i=\{T_i\}$。

聚类的过程是，①判断聚类集合中的聚类数目$|C|$是否大于用户指定的最小聚类数目阈值 minNum，若是则执行②③，否则聚类终止；②根据聚类相似度计算公式（见 3.4.1 节），通过扫描文档相似度矩阵，计算聚类集合中两两聚类之间的相似度；③合并最相似的两个聚类，从而构成新的聚类集合，并对新的聚类集合进行递归聚类。

XMLCluster 的完整算法描述见算法 3.2。

算法3.2 XMLCluster

input: $TDB=\{T_1,\ldots,T_n\}$，$F=\{f_1,\ldots,f_m\}$，minNum

output: $C=\{C_1,\ldots,C_k\}$

XMLCluster(*TDB*，*F*，*minNum*):

For each element $T_i\in TDB$ **Do** //初始化各文档的频繁结构特征表示

$F(T_i)=\{f \mid f\in F\wedge f\subseteq T_i\}$;

CreateMaxtri(F(TDB)); //建立文档相似度矩阵

$C=\{C_1,\ldots,C_n\}$; //初始化聚类

For each element $C_i\in C$ **Do**

$C_i=\{T_i\}$;

Cluster(C); //聚类

Cluster(*C*)：

If $|C| > minNum$ **Then** //**聚类终止条件**

For each element $C_i, C_j \in C$ **Do** //**计算各类之间的相似度**

$Sim(C_i, C_j)$;

//**对最大相似度的两个类进行合并，获取新的聚类**

For $\max Sim(C_i, C_j)$ **Do**

$C_k = Merge(C_i, C_j)$;

$C = C - \{C_i, C_j\} \cup C_k$;

Cluster(*C*); //**递归聚类**

3.4.3 聚类结果在 XML 文档处理中的应用

基于频繁结构的 XML 文档聚类方法最大的特点是，可以从每个类中抽取出一个公共的结构模式。在 XMLCluster 算法的基础上，一个 XML 文档集 *TDB* 被处理成一个聚类集合 $C=\{C_1,\ldots,C_i,\ldots,C_k\}$，一个 XML 文档类 C_i 可以表示成频繁结构的一个子集 $F(C_i)= F(T_1)\cap\ldots\cap F(T_w)$，其中 $T_1,\ldots,T_w\in C_i$，子集 $F(C_i)$中的元素即为类 C_i 中所有文档公共的频繁结构，这些公共的频繁结构表示了该文档类的领域语义信息和知识描述，在各种 XML 文档处理中起着重要的作用，例如为 XML 文档库建立索引[18,12]，对 XML 文档进行有效地集成[10]等等，从而提高查询效率。文献[18]在聚类的基础上，为每个类建立一个有效的 DTD 模式，该 DTD 模式可以用于查询评估，提高查询效率等等。文献[12]提出了 XML Schema Directory（XSD）数据结构，该结构的思想是在聚类的基础上，将各个类中的 XML 文档模式合并成为一个公共模式，公共模式代表了包含这些 XML 文档模式的一个类，只要一个 XML 文档的模式属于这个类，那么它所对应的 XML 文档即为该类的一个实例，从而将该公共模式用于 XML 查询，提高查询效率。文献[10]首先对聚类结果的各个类进行分析，归纳出各个类各自的集成模式，然后对所有的集成模式进行分析，归纳出一个全局的集成模式，从而实现对 XML 文档的有效集成，同时提高查询效率。

3.5 实验结果及性能分析

在实验部分，我们主要将 FrePathMiner 算法，FreTreeMiner 算法，以及传统的改进的序列模式挖掘算法 ASPMiner(Adaped Sequential Pattern Miner)[22]用于挖掘频繁结构，并分别以这三种频繁结构作为 XML 文档的聚类特征，采用 XMLCluster 算法对文档进行聚类，比较这三种聚类方法的聚类结果。

FrePathMiner 算法和 FreTreeMiner 算法是对 SSTMiner 算法的改进，分别用于挖掘最大频繁路径和最大频繁子树。ASPMiner 算法是对数据挖掘中的序列模式挖掘算法的改进，用于挖掘最大频繁路径。本章在 JBuilder 环境下，用 java 语言分别实现了各算法，实验硬件平台是一台 512M 内存，主频 2.50GHz 的 Pentium(R) 4 微机，安装了 Window XP 操作系统。实验数据集来自文献[52]，从[52]中取出 movies.dtd 和 actors.dtd 文档，对 movies.dtd 文档稍做修改生成 editMovies.dtd 文档，从而获得三个 DTD 文档，由于 movieds.dtd 介绍某部电影的情况及其演员信息，actors.dtd 介绍某个演员的个人信息及其拍摄电影的情况，editMovies.dtd 又是对 movies.dtd 的一个修订版本,所以以这三个 DTD 文档作为标准,生成的 XML 文档之间存在一定的相似性，本章共生成 300 个 XML 文档，每 100 个 XML 文档各遵循一个 DTD，在此基础上，分别采用以上三种 XML 文档聚类方法对这 300 个文档进行聚类,聚类终止于最小聚类数目阈值 3(即对应于三个 DTD 文档)，并在不同的支持度下对聚类结果进行比较。

3.5.1 不同支持度下的频繁结构挖掘数量

本实验在不同的支持度下，比较从 XML 文档库中用 FrePathMiner，FreTreeMiner，以及 ASPMiner 三种算法挖掘出来的频繁结构数量，实验结果如图 3.1 所示，可以分为两个阶段分析：第一阶段最小支持度范围是 0.2~0.9，由于支持度较大，三种方法挖掘到的频繁结构数量较少，且相差不多，传统的 ASPMiner 算法挖掘到的频繁路径数量最少，FrePathMiner 算法找到的频繁路径数量较 ASPMiner 算法多， FreTreeMiner 用于挖掘频繁子树，其挖掘到的频繁子树数量略多于 FrePathMiner 算法；第二阶段最小支持度范围是 0.05~0.2，由于支持度较小,三种方法挖掘到的频繁结构数量总体上较第一阶段多,但是 ASPMiner 算法找到的频繁路径数量增加不多，FrePathMiner 算法找到的频繁路径数量则明显增加，FreTreeMiner 算法则急剧增加。由两个阶段的分析结果可得，本章提出的 FreTreeMiner 算法和 FrePathMiner 算法找到频繁结构的数量都比传统的 ASPMiner 算法多，这就可以为文档聚类提供更多的结构特征，从而提高聚类精度。

3.5.2 不同支持度下的聚类精度

本实验在不同的支持度下，以 FrePathMiner，FreTreeMiner，以及 ASPMiner 三种算法作为特征提取器，对 XML 文档进行聚类，进而比较三种方法聚类结果的精确度，为了描述聚类结果的精确度，本文建立了一个混淆矩阵 $entry[m][m]$，其中 m 表示聚类结果类的数目，$entry[i][j](1= i, j = m$ 且 $i ? j)$表示将属于类 j 的，

但被错误归入到类 i 中的文档的数量，$entry[i][i]$表示将属于类 i 的，且被正确归入到类 i 中的文档的数量，这样，文档聚类结果精确度的衡量准则如公式 3 所示：

$$Accuracy = \sum_{i=1}^{m} entry[i][i] \Big/ \sum_{i=1}^{m}\sum_{j=1}^{m} entry[i][j] \quad (3)$$

例如表 3.2 中，i 表示 movies.dtd 类，j 表示 actors.dtd 类，k 表示 editmovies.dtd 类，则聚类结果获得的精确度为(80+30+50)/300=0.533。

实验结果如图 3.2 所示，同样分为两个阶段分析：第一阶段最小支持度范围是 0.2~0.9，由于这一阶段三种算法找到的频繁结构特征数量较少，所以精确度相对较低，这一阶段三种聚类方法各自的特点是，采用传统的 ASPMiner 聚类方法，其精确度较低，且随着支持度的降低其大小不稳定，本章提出的 FrePathMiner 和 FreTreeMiner 两种聚类方法精确度都较高，且随着支持度的降低而逐渐增大，由于 FreTreeMiner 算法找到的频繁结构数量多于 FrePathMiner 算法，所以前者部分支持度下的聚类结果精确度略高于后者；第二阶段最小支持度范围是 0.05~0.2，由于支持度较小，三种算法挖掘到的频繁结构数量较多，从而使得聚类精度都达到了 1，即聚类结果完全正确。由两个阶段的分析结果可得，本章提出的 FreTreeMiner 和 FrePathMiner 两种聚类方法可以获得较高的精确度，且随着支持度的降低而增大，使得精确度大小人为可控，与传统的 ASPMiner 聚类方法相比，具有更优的聚类效果。

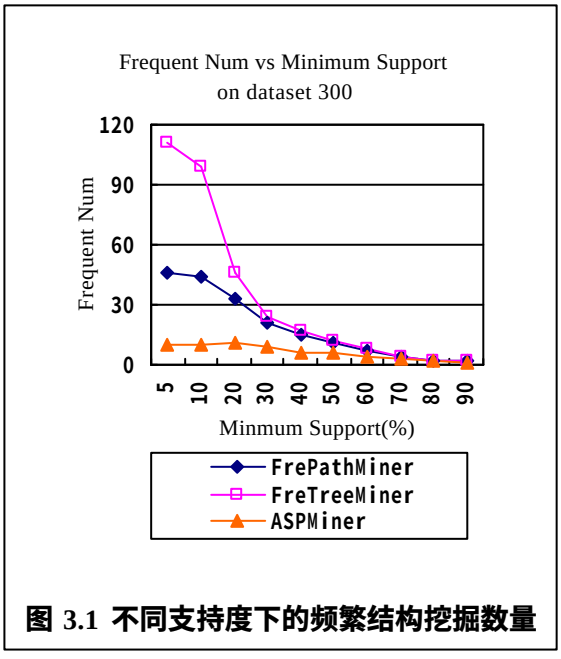


图 3.1 不同支持度下的频繁结构挖掘数量

		Actual clusters			
		i	j	k	Total
Result clusters	i	80	50	70	200
	j	0	0	30	30
	k	20	50	0	70
	Total	100	100	100	300

表 3.2 一个计算精确度的混淆矩阵的例子

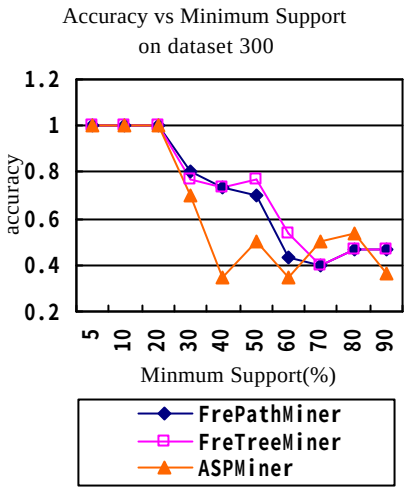


图 3.2 不同支持度下的聚类精确度

3.6 本章小结

与其他 XML 文档聚类方法相比，基于频繁结构的方法具有更明显的优势。目前采用的频繁结构有频繁元素，频繁路径，频繁子树等等。本章研究基于频繁路径和基于频繁子树的 XML 文档聚类方法[53]。本章首先对频繁子树挖掘算法 SSTMiner 进行改进，生成 FrePathMiner 算法和 FreTreeMiner 算法，分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树，在此基础上，本章提出一种凝聚的层次聚类算法 XMLCluster，分别以最大频繁路径和最大频繁子树作为 XML 文档的特征，对文档进行聚类，最后通过实验表明，本章提出的 FrePathMiner 和 FreTreeMiner 聚类方法，与传统的 ASPMiner 聚类方法相比，其聚类效果具有更大的优越性。

XML 文档聚类结果可以用于解决 XML 处理工作中各种各样的问题，如建立索引，快速查询，信息集成等等。下一章，我们将考虑把该方法用于 XML 文档集成存储管理系统中，实现对 XML 文档的有效存储和查询。

第四章 基于聚类的 XML 文档集成管理方法

本章研究基于聚类的 XML 文档集成管理方法。第三章已经介绍了基于频繁结构的 XML 文档聚类方法，本章将第三章提出的聚类方法用于 XML 文档集成管理系统中，从而提出了一种基于聚类的 XML 文档集成管理方法，即在聚类的基础上，对 XML 文档进行集成存储，集成存储过程分为两个阶段进行：第一阶段为模式映射阶段（Schema Mapping），首先提出关系模式生成算法 SchemaGenerator，根据模式映射规则，分析各个类中的 XML 文档，生成相应的关系数据库模式；然后提出关系模式集成算法 SchemaIntegrator，分析各个类中各个文档相应的关系数据库模式，归纳出各个类的集成关系模式；最后提出全局关系模式集成算法 GlobalSchemaIntegrator，合并各个类的集成关系模式，构成全局关系模式，从而在关系数据库中生成表格；第二阶段为数据存储阶段（XML Storage），主要提出了数据抽取算法 DataExtractor，在模式映射的基础上，抽取 XML 数据到关系数据库，从而实现对来自不同 DTD 的多个 XML 文档的有效集成管理。本章接着给出了基于聚类的 XML 文档集成管理系统的整个框架。最后通过具体实例来说明基于聚类的 XML 文档集成存储方法的有效性。

4.1 引言

XML 是 Internet 上信息表示和交换的标准，随着大量 XML 数据的出现，如何有效地存储和查询这些 XML 数据就成为目前值得研究的一个重要问题[54]。把 XML 数据映射为关系数据，利用经典的关系数据库系统来管理 XML 数据是目前常见的解决方法之一。目前基于关系数据库的 XML 存储的一般策略[13,55,56]是首先建立映射规则，直接将 XML 文档模式映射到关系数据库模式中，然后根据映射规则，抽取 XML 文档数据到关系数据库中。然而随着 XML 应用的不断扩展，XML 数据的大量出现，由于 XML 数据可能来自各个不同的数据源，它们所遵循的模文档式（DTD 或 XML Schema）可能不同，文档之间存在一定的异构性，如果采用传统的存储策略，还存在一定的缺陷：1）文档的异构性使得存储需要的额外空间非常大；2）文档中的内容被分散到数据库各个不同的表格中，查询效率大大地降低，重建一个文档也变得非常费时。目前，在 XML 数据管理方面，由于来自不同数据源的 XML 文档之间存在一定的异构性，人们已经提出了将信息集成的思想用于多源 XML 数据管理的方法，并做了一些研究[10~14]，然而，在 XML 文档到关系数据库的存储应用中，引进数据集成技术的研究还相对较少，我们在文献[57]中已经提出了一种集成多个相似的 XML 文档到关系数据库的方法，即将一个聚类中的相似文档集成存储到关系数据库中，然而，文献[57]中对于如何获得聚类的问题没有具体说明，对于多个聚类的集成方案也没有具体讨论。

针对以上问题，本章考虑在 XML 文档到关系数据库的存储系统中，引进数据集成的思想，进而有效地解决来自不同 DTD 的多个 XML 文档的关系存储和查询问题。本章将第三章提出的聚类方法用于 XML 文档集成管理系统中，提出了

一种基于聚类的 XML 文档集成管理方法，即在聚类的基础上，对 XML 文档进行集成存储，集成存储过程分为两个阶段进行：第一阶段为模式映射阶段（Schema Mapping），首先提出关系模式生成算法 SchemaGenerator，根据模式映射规则，分析各个类中的 XML 文档，生成相应的关系数据库模式；然后提出关系模式集成算法 SchemaIntegrator，分析各个类中各个文档相应的关系数据库模式，归纳出各个类的集成关系模式；最后提出全局关系模式集成算法 GlobalSchemaIntegrator，合并各个类的集成关系模式，构成全局关系模式，从而在关系数据库中生成表格；第二阶段为数据存储阶段（XML Storage），主要提出了数据抽取算法 DataExtractor，在模式映射的基础上，抽取 XML 数据到关系数据库，从而实现对来自不同 DTD 的多个 XML 文档的有效集成管理。本章还给出了基于聚类的 XML 文档集成管理系统的整个框架，并通过具体实例来说明基于聚类的 XML 文档集成存储方法的有效性。

本章以下部分组织如下：第 2 节介绍一些相关定义；第 3 节介绍 XML 文档到关系数据库的集成存储，其过程主要分两个阶段进行。首先介绍第一阶段模式映射（Schema Mapping），并给出了相关算法，包括关系模式生成算法 SchemaGenerator，关系模式集成算法 SchemaIntegrator，全局关系模式集成算法 GlobalSchemaIntegrator；接着介绍第二阶段数据存储（XML Storage），并给出了数据抽取算法 DataExtractor；第 4 节给出了基于聚类的 XML 文档集成管理系统的整个框架；第 5 节是实验部分；最后是本章的结论。

4.2 相关定义

一棵 XML 文档树是一个节点的集合，其中除根节点外每个内节点至少有一个父节点，并且可以有很多有序的孩子节点。XML 文档树的基本节点类型包括：文档节点，元素节点，属性节点，文本节点，注释节点，处理指令节点，文档片段节点等等，其中，元素节点，属性节点以及节点之间的父子关系包含了丰富的语义信息和结构信息，所以本文主要考虑这三种节点类型。一个 XML 文档模式可以描述如下：

定义 4.1（XML 模式）：一个 XML 文档模式可以表示成一棵树 $T=(V,E)$，其中，V 是节点集，$\forall v\in V$，则 $v=(type,\ name,\ \{a_1,\ldots,\ a_p\},\ \{e_1,\ldots,\ e_q\})$，其中 $type$ 是节点类型，可以是元素类型，属性类型或者文本类型，分别记作 $Etype$，$Atype$ 和 $Ttype$，$name$ 是节点名称，$\{a_1,\ldots,\ a_p\}$是节点的属性集合（p=0，表示属性个数），$\{e_1,\ldots,\ e_q\}$是节点的子元素集合（q=0，表示子元素个数），这里要注意的是，当 $type=Atype$ 或者 $Ttype$ 时，则 $p+q=0$，即属性集合和子元素集合为空集，这说明属性类型节点和文本类型节点不包含子元素和属性，换句话说就是，只有元素节点才有可能有属性节点和子元素节点；$E\subseteq V\times V$ 是有向边集合，每条边表示节点

间的嵌套关系，$\forall(v_i, v_j)\in E$，则 v_i 是 v_j 的父亲节点，v_j 是 v_i 的某一孩子节点。另外，本文用 v_0 表示文档根元素节点，用 *parent*(v)和 *children*(v)分别表示返回节点 v 的父节点和所有孩子节点。

关系数据库是一系列关系表（二维表格）的集合，关系数据库有如下特征：①每张关系表都有表名；②每张关系表仅有一个模式；③属性（或称字段）的个数是固定的，每个属性有属性名，在一个关系表中，各属性名是不同的；④每个属性有类型及长度；⑤某些属性集唯一表示元组，成为主键。本文用 r 来描述一个关系模式，其形式定义如下：

定义 4.2（关系数据库模式）：一个关系数据库模式可以形式化表示为 *r=Table*(*ID, F, PID*)，其中，*Table* 是关系表名，*ID* 是关系表的关键字，F 是关系表的属性集合，*PID* 是关系表的外部关键字。

上面我们已经定义了 XML 模式和关系数据库模式，由此可以将 XML 模式映射到关系数据库模式，映射过程主要处理 XML 模式中的元素类型节点，本文将 XML 文档中的元素节点分为两类：①简单元素节点：元素节点只包含文本节点，不包含属性节点和子元素节点；②混合元素节点：元素节点既包含文本节点，也包含属性节点或者子元素节点。这样，当进行模式映射时，若处理的节点是混合元素节点，则该节点映射为数据库中的一个表，它所包含的每个属性和子元素节点映射为表中的各个字段，同时为该表格增加一个主键和外键，外键取决于该节点的父节点映射到数据库中的表格的主键，另外要注意的是，因为文档根节点没有父节点，所以文档根节点映射到数据库中的表格的外键总是为空。整个映射过程用映射规则具体描述如下：

定义 4.3（映射规则）：令 $T=(V,E)$是一个 XML 模式，v_0 表示文档根元素节点，$R=\{r|\ r=Table(ID, F, PID)\}$是由 XML 模式映射生成的关系数据库模式集合，则由 T 生成 R 所采用的映射规则是，$\forall v\in V$，$v=(type, name, \{a_1,\ldots, a_p\}, \{e_1,\ldots, e_q\})$，生成相应关系数据库模式 $r\in R$ 如下：

①当 $v.type=Atype$ 或者 *Ttype* 时，则 v 为属性节点或者文本节点，映射结果为 *r=null*，即不生成新表格；

②当 $v.type=Etype$，且 $v.p+v.q=0$ 时，则 v 为简单元素节点，映射结果为 *r=null*，即不生成新表格；

③当 $v.type=Etype$，且 $v.p+v.q>0$ 时，则 v 为混合元素节点，映射结果为 *r=Table*(*ID, F, PID*)，即将节点 v 映射到关系数据库的一个新表，其中 *Table=v.name*，即表格名为节点元素名，*ID=v.name+"id"*，即关键字名为节点元素名加上"id"两个字符，$F=\{v.a_1.name,\ldots,v.a_p.name\}\cup\{v.e_1.name,\ldots,v.e_q.name\}$，即节点包含的属性节点名和子元素节点名构成表格字段，*PID=parent*(v)*.name+"id"*，即外部关键字取决于 v 父节点映射到关系数据库中的

表格的主码，当 $v=v_0$ 时，$PID=null$，即根元素节点没有外部关键字。

这样根据映射规则，将一个文档的 XML 模式映射到关系数据库模式集合时，不仅把文档树中各个结点本身的信息存储下来，而且还把各个结点之间的嵌套关系通过关系表外部码反映出来。

4.3 XML 文档到关系数据库的集成存储

在聚类的基础上，对 XML 文档进行集成存储，集成存储过程分为两个阶段进行：第一阶段为模式映射阶段（Schema Mapping），第二阶段为数据存储阶段（XML Storage）。下面分别进行详细介绍。

4.3.1 模式映射（Schema Mapping）

模式映射就是处理聚类结果的各个类，根据模式映射规则，结合一些集成算法，最终生成全局关系模式，从而在关系数据库中生成表格。模式映射算法见算法 4.1，输入的是聚类结果的各个类，输出的是全局关系模式。

算法4.1 SchemaMapping
input: $C=={C_1,...,C_n}$
output:C_R
SchemaMapping(*C*):
$C\_R=\varnothing$;
For each element $C_i \in C$ **Do** //处理每个类
$Ci\_R=\varnothing$;
For each element $T_j \in C_i$ **Do** //处理类中的每个文档
$Ci\_R=Ci\_R \cup$ **SchemaGenerator(**T_j**)**; //获取文档的关系模式，并加入类的所有关系模式
$C\_R=C\_R \cup$ **SchemaIntegrator(**Ci_R**)**; //获取类的集成模式，并加入全局集成模式

模式映射算法中所使用的相关集成算法包括关系模式生成算法 SchemaGenerator，关系模式集成算法 SchemaIntegrator，全局关系模式集成算法 GlobalSchemaIntegrator，分别介绍如下：

1. 关系模式生成算法 SchemaGenerator

关系模式生成算法 SchemaGenerator 就是在聚类的基础上，根据模式映射规则，分析各个类中的 XML 文档，生成相应的关系数据库模式，见算法 4.2。

算法4.2 SchemaGenerator

input: $T=(V,E)\in Ci, v=(type,name,\{a_1,\ldots,a_p\},\{e_1,\ldots,e_q\})$

output: $R=\{r \mid r=Table(ID,F,PID)\}$

SchemaGenerator($T\in Ci$):

$R=\varnothing$;

D =getDocNode(T);

preOrderTraverse(D,i);

Return R;

preOrderTraverse(Node$v$,*clusterId* *cId*):

//只对元素类型节点处理，即若为属性节点或者文本节点，则不做处理

If ($v.type==Etype$) **Then** //v是元素节点类型

If ($v.p+v.q==0$) **Exit**; //v没有属性节点和子元素节点，即为简单元素节点，则不做处理

$r=Table(ID,F,PID)$; //将节点v映射到关系数据库中的一个表格

$r.Table=v.name+cId$; //表格名为v节点元素名+文档所属类标号

$r.ID=v.name+"Id"$; //关键字名为v节点元素名+"Id"

$r.F=\{v.a_1.name,\ldots,v.a_p.name\}\cup\{v.e_1.name,\ldots,v.e_q.name\}$; //表格字段为$v$节点的属性节点和子元素节点

If ($v==v_0$) **Then** $r.PID=null$; //若为根节点，则外部关键字为空

Else $r.PID=parent(v).name+"Id"$; //否则，外部关键字名为父节点映射到关系数据库中的表格主码

$R=R\cup\{r\}$; //加入一个新表格

For any $w\in children(v)$ **Do**

preOrderTraverse(w); //递归处理v节点的孩子节点

2. 关系模式集成算法 SchemaIntegrator

关系模式集成算法 SchemaIntegrator 就是在关系模式生成算法的基础上，分析各个类中各个文档相应的关系数据库模式，归纳出各个类的集成关系模式，见算法 4.3。

算法4.3 SchemaIntegrator

input: $Ci\_R=R_1\cup\ldots\cup R_{ci}$

output: $Ci\_R=\{r \mid r=Table(ID,F,PID)\}$

SchemaIntegrator(*Ci_R*):

//由所有关系数据库模式分析归纳出一个集成关系数据库模式

For each element $r_i,r_j\in Ci\_R$ **Do**

r = integrateRiRj(r_i,r_j);

If (r!=null) $Ci\_R=Ci\_R-\{r_i,r_j\}\cup\{r\}$;

Return Ci_R;

integrateRiRj(r_i, r_j):

r = null;

If ($r_i.ID==r_j.ID$) **And** ($r_i.PID==r_j.PID$) **Then** //两个模式表格名，关键字，外部关键字都相同，则合并

$r=Table(ID,F,PID)$;

$r.Table=r_i.Table$;

$r.ID=r_i.ID$;

$r.PID=r_i.PID$;

$r.F=r_i.F\cup r_j.F$; //对两个字段列表做并运算，从而获得关系模式r_i,r_j字段列表的最小覆盖

Retrun r;

3. 全局关系模式集成算法 GlobalSchemaIntegrator

全局关系模式集成算法 GlobalSchemaIntegrator 就是在关系模式集成算法 SchemaIntegrator 的基础上，合并各个类的集成关系模式，构成全局关系模式，从而在关系数据库中生成表格，见算法 4.4。

算法4.4 GlobalSchemaGenerator

$C\_R = C1\_R \cup ... \cup Cm\_R$

4.3.2 数据存储（XML Storage）

数据存储就是处理聚类结果的各个类，在模式映射的基础上，抽取 XML 文档信息到关系数据库中，从而实现对来自不同 DTD 的多个 XML 文档的有效集成管理，具体描述见算法 4.5。

算法4.5 XMLStorage
input: $C = = \{C_1,...,C_n\}$
XMLStorage(*C*):
For each element $C_i \in C$ **Do** //处理每个类
For each element $T_j \in C_i$ **Do** //处理类中的每个文档
DataExtractor(T_j); //抽取文档数据到关系数据库

算法 4.5 中使用了数据抽取算法 DataExtractor 用于抽取 XML 文档数据到数据库中，具体过程见算法 4.6。

算法4.6 DataExtractor
input: $T = (V,E) \in Ci$
DataExtractor(*T* ∈ *Ci*):
D =getDocNode(*T*);
preOrderTraverse(*D*, *i*);
preOrderTraverse(Node *v*, clusterId cId):
//只对元素类型节点处理，即若为属性节点或者文本节点，则不做处理
If (*v.type*==*Etype*) **Then** //*v*是元素节点类型
If (*v.p*+*v.q*==*0*) **Exit**; //*v*没有属性节点和子元素节点，即为简单元素节点，则不做处理
Table=*v.name*+*cId*; //表格名为*v*节点元素名+文档所属类标号
insert into *Table* with the information of *v*; //将*v*节点的相关信息插入数据库表格
//包括关键字，字段值，外部关键字
//关键字取*v*节点编号，字段值包括*v*属性值和*v*子元素包含的所有文本节点的值，外部关键字取*v*父节点编号
For any $w \in children(v)$ **Do**
preOrderTraverse(*w*); //递归处理*v*节点的孩子节点

4.4 系统功能结构设计

基于聚类的 XML 文档集成管理系统的整个结构框架如图 4.1 所示：

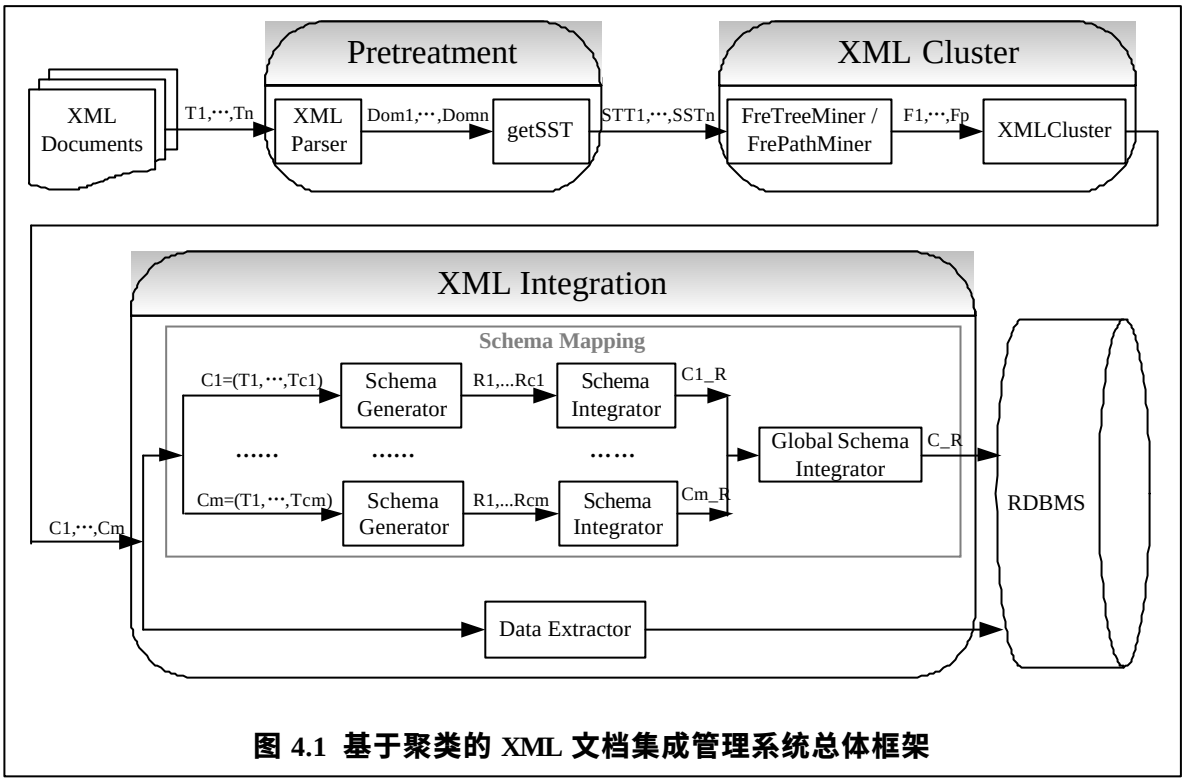


图 4.1 基于聚类的 XML 文档集成管理系统总体框架

整个结构框架主要包括三个部分：

1．预处理（Pretreatment）

1) XML 解析（XML Parser）：利用 XML 解析器对文档进行解析，生成 DOM 树；

2) 最简结构树 SST 生成（getSST）：遍历 DOM 树，提取 XML 有效结构信息。

2．聚类（XML Cluster）

1) 频繁路径/子树挖掘（FrePathMiner/FreTreeMiner）：在生成 SST 的基础上，挖掘由 XML 文档构成的森林中的所有最大频繁路径/子树；

2) XML 文档聚类（XMLCluster）：以最大频繁路径/子树作为 XML 文档的特征，对文档进行聚类。

3．数据集成（XML Integration）

1) 模式映射（Schema Mapping）

a) 关系模式生成（Schema Generator）：在聚类的基础上，根据模式映射规则，分析各个类中的 XML 文档，生成相应的关系数据库模式；

b) 关系模式集成（Schema Integrator）：分析各个类中各文档相应的关系数据库模式，归纳出各个类的集成关系模式；

c) 全局关系模式集成（Global Schema Integrator）:合并各个类的集成关系模式，构成全局关系模式，从而在关系数据库中生成表格。

2) 数据存储（XML Storage）

数据抽取（Data Extractor）：在模式映射的基础上，抽取 XML 数据到关系数据库，从而实现对来自不同 DTD 的多个 XML 文档的有效集成管理。

4.5 实验结果及性能分析

本章在 Jbuilder 环境下，用 java 语言实现了 XML 集成存储相关算法，从而建立了基于聚类的 XML 文档集成管理系统，实验硬件平台是一台 512M 内存，主频 2.50GHz 的 Pentium(R) 4 微机，安装了 Window XP 操作系统。实验数据集来自文献[52]，从[52]中取出 movies.dtd 和 actors.dtd 文档，对 movies.dtd 文档稍做修改生成 editMovies.dtd 文档，从而获得三个 DTD 文档，由于 movieds.dtd 介绍某部电影的情况及其演员信息，actors.dtd 介绍某个演员的个人信息及其拍摄电影的情况，editMovies.dtd 又是对 movies.dtd 的一个修订版本，所以以这三个 DTD 文档作为标准，生成的 XML 文档之间存在一定的相似性，本实验共生成 6 个 XML 文档实例构成文档库，见图 4.2，每 2 个 XML 文档各遵循一个 DTD。

T1:

```
<W4F_DOC>
 <Movie>
  <Title>Nosferatu, eine Symphonie des Grauens</Title>
  <Year>1922</Year>
  <Directed_By>
   <Director>F.W. Murnau</Director>
  </Directed_By>
  <Genres>
   <Genre>(more)</Genre>
  </Genres>
  <Cast>
   <Actor>
    <FirstName>Gustav</FirstName>
    <LastName>von Wangenheim</LastName>
    <E_mail>gvw@sina.com</E_mail>
    <Address>Canada</Address>
   </Actor>
  </Cast>
 </Movie>
</W4F_DOC>
```

T2:

```
<W4F_DOC>
 <Actor>
  <Name>
   <FirstName>Claudio</FirstName>
  </Name>
  <Filmography>
   <Movie>
    <Title>Casablanca</Title>
   </Movie>
  </Filmography>
 </Actor>
</W4F_DOC>
```

T3:

```
<Movie>
 <Directed_By>
  <Director_name>F.W. Murnau</Director_name>
  <position>post</position>
 </Directed_By>
 <Actor>
  <Name>
   <FirstName>Gustav</FirstName>
   <LastName>von Wangenheim</LastName>
  </Name>
 </Actor>
</Movie>
```

T4:

```
<W4F_DOC>
 <Movie>
  <Title>Bronenosets Potyomkin</Title>
  <Year>1925</Year>
  <Directed_By>
   <Director>Grigori Aleksandrov</Director>
  </Directed_By>
  <Genres>
   <Genre>Drama</Genre>
  </Genres>
 </Movie>
</W4F_DOC>
```

T5:

```
<W4F_DOC>
 <Actor>
  <Name>
   <FirstName>Frank</FirstName>
   <LastName>Albertson (I)</LastName>
  </Name>
  <Filmography>
   <Movie>
    <Title>Farmer's Daughter, The</Title>
    <Year>1928</Year>
   </Movie>
  </Filmography>
 </Actor>
</W4F_DOC>
```

T6:

```
<Movie>
 <Directed_By>
  <Director_name>F.W. Murnau</Director_name>
 </Directed_By>
 <Actor>
  <Name>
   <FirstName>Gustav</FirstName>
   <LastName>von Wangenheim</LastName>
  </Name>
 </Actor>
 <Title>Nosferatu, eine Symphonie des Grauens</Title>
 <Year>1922</Year>
</Movie>
```

图 4.2 文档库

在实验中，本章首先采用第二章提出的 XMLCluster 聚类方法对这 6 个文档进行聚类，聚类支持度为 0.8，且终止于最小聚类数目阈值 3（即对应于三个 DTD 文档），这样，最终聚类结果三个类中的文档分别是 T1 和 T4，T2 和 T5，T3 和 T6，说明文档库中的文档被正确的归类。

在此基础上，利用 XML 集成存储相关算法实现对聚类结果各类中 XML 文档的有效集成存储，部分集成过程见图 4.3，表 4.1 给出了集成存储结果生成的所有关系表以及文档数据在关系表中对应的存储内容。

从表 4.1 可以看出，XML 文档中所有的内容信息都无损地存储到关系表中，XML 数据的层次结构信息也都通过父表和子表的参照约束关系得以完整保存，极大的方便了对 XML 数据的使用和处理；表格关键字由两个标识符组成，并用“/”分开，第一个标识了该节点所属的文档标号，第二个标识了该节点在相应文档中的位置，例如在 Movie2 表格中，MovieId=”0/1”表示该记录对应的元素节点 movie 在标号是 0 的文档中，且在文档中的位置编号为 1，这样有利于查询中，将查询到的关系数据转换为原来的 XML 文档结构形式；另外将这 6 个 XML 文档经过频繁结构挖掘、聚类并集成存储，整个过程历时只需约 1 秒多。可见本章提出的基于聚类的 XML 文档集成存储方法是可行的，从而有效地解决了来自不同 DTD 的多个 XML 文档到关系数据库的转换存储问题。

文档T3的关系模式：
Movie0(MovieId,Directed_By,Actor,Title,Y ear)
Directed_By0(Directed_ById,Director_name,MovieId)
Actor0(ActorId,Name,MovieId)
Name0(NameId,FirstName,LastName,ActorId)
文档T6的关系模式：
Movie0(MovieId,Directed_By,Actor)
Directed_By0(Directed_ById,Director_name,position,MovieId)
Actor0(ActorId,Name,MovieId)
Name0(NameId,FirstName,LastName,ActorId)
类一（包含文档T3和T6）的集成模式：
Movie0(MovieId,Directed_By,Actor,Title,Y ear)
Directed_By0(Directed_ById,Director_name,position,MovieId)
Actor0(ActorId,Name,MovieId)
Name0(NameId,FirstName,LastName,ActorId)
类二（包含文档T2和T5）的集成模式：
W4F_DOC1(W4F_DOCId,Actor)
Actor1(ActorId,Name,Filmography,W4F_DOCId)
Name1(NameId,FirstName,LastName,ActorId)
Filmography1(FilmographyId,Movie,ActorId)
Movie1(MovieId,Title,Year,FilmographyId)
类三（包含文档T1和T4）的集成模式：
W4F_DOC2(W4F_DOCId,Movie)
Movie2(MovieId,Title,Year,Directed_By,Genres,Cast,W4F_DOCId)
Directed_By2(Directed_ById,Director,MovieId)
Genres2(GenresId,Genre,MovieId)
Cast2(CastId,Actor,MovieId)
Actor2(ActorId,FirstName,LastName,E_mail,Address,CastId)

三个类合并结果形成全局模式，从而在数据库中生成表格：
Movie0(MovieId,Directed_By,Actor,Title,Year)
Directed_By0(Directed_ById,Director_name,position,MovieId)
Actor0(ActorId,Name,MovieId)
Name0(NameId,FirstName,LastName,ActorId)
W4F_DOC1(W4F_DOCId,Actor)
Actor1(ActorId,Name,Filmography,W4F_DOCId)
Name1(NameId,FirstName,LastName,ActorId)
Filmography1(FilmographyId,Movie,ActorId)
Movie1(MovieId,Title,Year,FilmographyId)
W4F_DOC2(W4F_DOCId,Movie)
Movie2(MovieId,Title,Year,Directed_By,Genres,Cast,W4F_DOCId)
Directed_By2(Directed_ById,Director,MovieId)
Genres2(GenresId,Genre,MovieId)
Cast2(CastId,Actor,MovieId)
Actor2(ActorId,FirstName,LastName,E_mail,Address,CastId)

图 4.3 部分集成过程示例

W4F_DOC1

W4F_DOCId	Actor
1/0	
4/0	

W4F_DOC2

W4F_DOCId	Movie
0/0	
3/0	

Movie1

MovieId	Title	Year	FilmographyId
4/4	Casablanca		4/3

Movie0

MovieId	Directed_By	Actor	Title	Year
2/0			Nosferatu, eine Symphonie des Grauens	1922
5/0				

Movie2

MovieId	Title	Year	Directed_By	Genres	Cast	W4F_DOCId
0/1	Nosferatu, eine Symphonie des Grauens	1922				0/0
3/1	Bronenosets Potyomkin	1925				3/0

Actor0

ActorId	Name	MovieId
2/2		2/0
5/2		5/0

Actor1

ActorId	Name	Filmography	W4F_DOCId
1/1			1/0
4/1			4/0

Actor2

ActorId	FirstName	LastName	E_mail	Address	CastId
0/5	Gustav	von Wangenheim	gvw@sina.com	Canada	0/4

Name0

NameId	FirstName	LastName	ActorId
2/3	Gustav	von Wangenheim	2/2
5/3	Gustav	von Wangenheim	5/2

Name1

NameId	FirstName	LastName	ActorId
1/2	Frank	Albertson (I)	1/1
4/2	Claudio		4/1

Directed_By0

Directed_ById	Director_name	position	MovieId
2/1	F.W. Murnau		2/0
5/1	F.W. Murnau	post	5/0

Directed_By2

Directed_ById	Director	MovieId
0/2	F.W. Murnau	0/1
3/2	Grigori Aleksandrov	3/1

Cast2

CastId	Actor	MovieId
0/4		0/1

Genres2

GenresId	Genre	MovieId
0/3	(more)	0/1
3/3	Drama	3/1

Filmography1

FilmographyId	Movie	ActorId
1/3		1/1
4/3		4/1

表 4.1 生成的关系表

4.6 本章小结

本章考虑在 XML **文档到关系数据库的存储系统中，引进数据集成的思想，**

进而有效地解决来自不同 DTD 的多个 XML 文档的关系存储和查询问题。本章提出了一种基于聚类的 XML 文档集成管理方法，即在聚类的基础上，对 XML 文档进行集成存储，集成存储过程分为两个阶段进行：第一阶段为模式映射阶段（Schema Mapping），使用的三个相关集成算法包括关系模式生成算法 SchemaGenerator，关系模式集成算法 SchemaIntegrator 以及全局关系模式集成算法 GlobalSchemaIntegrator；第二阶段为数据存储阶段（XML Storage），主要使用了数据抽取算法 DataExtractor。本章接着给出了基于聚类的 XML 文档集成管理系统的整个框架。最后本章通过具体实例来说明基于聚类的 XML 文档集成存储方法的有效性。

基于关系数据库的 XML 文档管理通常包括以下三个步骤：生成关系模式，创建关系表用于存储数据；抽取 XML 数据，存入对应表中的相应位置；处理用户 XML 查询，转换为等价的关系数据库查询。本章提出的基于聚类的 XML 文档集成管理系统中，已经解决了关系模式的生成，以及 XML 文档数据的有效存储问题，下一步，我们将考虑 XML 文档的有效查询，即在聚类的基础上，充分利用聚类过程产生的信息，根据 XML 文档存储映射规则，为各个类建立索引，从而提高 XML 文档查询效率。

第五章 结论与展望

5.1 总结

本文以 Internet 中信息表示与交换的新标准 XML 作为研究对象，基于数据集成的思想，结合 XML 相关的技术以及几种数据挖掘的经典技术，对来自不同 DTD 的多个 XML 文档的有效集成存储问题进行了研究。主要研究成果是：

首先提出了一种有效的 XML 文档频繁子树挖掘方法，具体做法是：首先对 XML 文档进行预处理，提取 XML 文档的有效结构 SST（Simplest Structural Tree 最简结构树）；然后提出 SSTMiner 算法，用于挖掘 SST 中的所有嵌入频繁子树。SSTMiner 算法不但继承了 TreeMiner 算法的优点，而且针对 TreeMiner 算法存在的瓶颈问题，以及结合当前所处理的 SST 的结构特点，对 TreeMiner 算法进行改进，改进策略主要考虑到减少候选子树的产生以及减少范围列表元素连接的次数两个方面，从而进一步提高了算法执行的效率；最后本文通过实验证明该方法的有效性。

接着研究基于频繁结构的 XML 文档聚类方法，其频繁结构包括频繁路径和频繁子树，具体做法是：首先对 XML 文档进行频繁结构挖掘，获取最大频繁结构集，由于 SSTMiner 算法是一种非常有效的频繁子树挖掘算法，对 SSTMiner 算法进行改进，即可得到 FrePathMiner 算法和 FreTreeMiner 算法，分别用于挖掘 XML 文档中最大频繁路径和最大频繁子树；然后提出一种凝聚的层次聚类算法 XMLCluster，分别以最大频繁路径和最大频繁子树作为 XML 文档的特征，对文档进行聚类；最后通过实验表明，本文提出的 FrePathMiner 和 FreTreeMiner 聚类方法，与传统的 ASPMiner 聚类方法相比，其聚类效果具有更大的优越性。

最后提出了一种基于聚类的 XML 文档集成管理方法，即在聚类的基础上，对 XML 文档进行集成存储，集成存储过程分为两个阶段进行：第一阶段为模式映射阶段（Schema Mapping），使用的三个相关集成算法包括关系模式生成算法 SchemaGenerator，关系模式集成算法 SchemaIntegrator 以及全局关系模式集成算法 GlobalSchemaIntegrator；第二阶段为数据存储阶段（XML Storage），主要使用了数据抽取算法 DataExtractor，从而实现对来自不同 DTD 的多个 XML 文档的有效集成管理。同时，本文给出了基于聚类的 XML 文档集成管理系统的整个框架，并通过具体实例来说明基于聚类的 XML 文档集成存储方法的有效性。

5.2 进一步工作展望

本文提出了一种基于聚类的 XML 文档集成管理方法，并建立了实验系统，

从而实现了对来自不同 DTD 的多个 XML 文档到关系数据库的有效集成存储。基于关系数据库的 XML 文档管理通常包括以下三个步骤：生成关系模式，创建关系表用于存储数据；抽取 XML 数据，存入对应表中的相应位置；处理用户 XML 查询，转换为等价的关系数据库查询。本文已经解决了前两个步骤，下一步，我们将考虑 XML 文档的有效查询问题。

参考文献

[1] Bray T, Paoli J, Sperberg-McQueen CM, Maler E, eds.Extensible markup language (XML) 1.0 (Fourth Edition). W3C Recommendation, 2006. http://www.w3.org/TR/2006/REC-xml-20060816/.

[2] 许卓明，廖述梅，陶皖，董逸生．商业 RDBMS 产品的 XML 支持[J]．计算机应用研究，2004．

[3] Jiawei Han, Mickeline Kamber. Data Mining: Concepts and Techniques[M]. Morgan Kaufmann Pulishers,2001.

[4] Marti A.Hearst. Information Integration[J]. IEEE Intelligent Systems,1998.

[5] Seung-Jin Lim, Yiu-Kai Ng. An Automated Integration Approach for Semi-Structured and Structured Data[C]. Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS.01),2001.

[6] Ian Gorton, Justin Almquist, Kevin Dorow, Peng Gong, Dave Thurman. An Architecture for Dynamic Data Source Integration[C]. Proceedings of the 38th Hawaii International Conference on System Sciences,2005.

[7] Jurgen Gores. Patten-based Information Integration in Dynamic Environments[C]. Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS'05),2005.

[8] Peter Mork, Arnon Rosenthal, Len Seligman, Joel Korb, Ken Samuel. Integration Workbench: Integrating Schema Integration Tools[C]. Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06),2006.

[9] Jun Yuan, Ali Bahrami, Changzhou Wang, Marie Murray, Anne Hunt. A Semantic Information Integration Tool Suite[C]. VLDB.06, September 12–15, 2006.

[10] Mong Li Lee, Liang Huai Yang, Wynne Hsu, Xia Yang. XClust: Clustering XML Schemas for Effective Integration[J]. CIKM'02, November 4-9, 2002.

[11]Wenxin Liang and Haruo Yokota. LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. 2-12-1 Oh-okayama, Meguro-ku, Tokyo 152-8552, Japan. wxliang@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp.

[12]Evangelos Kotsakis, Klemens Bohm. XML Schema Directory: A Data Structure for XML Data Processing[C]. 1st International Conference on Web Information Systems Engineering (WISE'00),2000.

[13]Guangming Xing, Jinhua Guo, Ronghua Wang. Managing XML Documents using RDBMS[C]. Proceedings of the Sixth International Conference on Software

Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05),2005.

[14]Chantal Reynaud, Jean-Pierre Sirot, Dan Vodislav. Semantic Integration of XML Heterogeneous Data Sources[C]. Proceedings of the International Database Engineering and Applications Symposium (IDEAS'01),2001.

[15]Zhang K, Shasha D. Simple fast algorithms for the editing distance between trees and related problems[J]. SIAM Journal of Computing, 1989, 18(6):1245 - 1262.

[16]Chawathe S S, Rajaraman A, Garcia-Molina H, et al. Change detection in hierarchically structured information[A]. In Procs of the Int'l Conf on Management of Data (SIGMOD'96) [C]. 1996. 493-504.

[17]Cobena G, Abiteboul S, Marian A. Detecting changes in XML document[A] . In 18th Int l Conf on Data Engineering(ICDE 2002)[C]. San Jose:2002.

[18]Nierman A , Jagadish H V. Evaluating structural similarity in XML documents [A]. In Int'l Workshop on the Web and Databases (WebDB)[C]. Madison:2002. 61-66.

[19]COSTA G, MANCO G, ORTALE R, et al. A Tree-based Approach to Clustering XML Documents by Structure[Z]. Rapporto Tecnico N. 04: RT-ICAR-CS-04-04 Aprile 2004.

[20]Flesca S, Manco G, Masciari E, et al. Detecting structural similarities between XML documents[C]. In Proc 5th Int. Workshop on the Web and Databases (WebDB'02).

[21]Sergio Flesca, Giuseppe Manco, Elio Masciari, Luigi Pontieri, and Andrea Pugliese, Student Member, IEEE. Fast Detection of XML Structural Similarity[J]. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,2005.

[22]Jung-Won Lee, Kiho Lee, Won Kim. Preparations for Semantics-Based XML Mining[C]. Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM'01).2001.

[23]Yinghua Ma & Richard Chbeir. Content and Structure Based Approach For XML Similarity[C]. Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05),2005.

[24]Uchang Park, Yeojin Seo. An Implementation of XML Documents Search System based on Similarity in Structure and Semantics[C]. Proceedings of the 2005

International Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05),2005.

[25]Ho-pong Leung, Fu-lai Chung, Stephen C.F. Chan and Robert Luk. XML Document Clustering Using Common Xpath[C]. In: Proceedings of the 2005 International Workshop on Challenges in Web Information Retrieval and Integration(WIRI'05), 2005.

[26]Davood Rafiei, Daniel L.Moise, Dabo Sun. Finding Syntactic Similarities Between XML Documents[C]. Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA'06), 2006.

[27]H.P Leung, F.L Chung and S.C.F. Chan, On the use of hierarchical information in sequential mining-based XML document similarity computation. Knowledge and Information Systems, vol.7, no.4, 2005.

[28]R. Agrawal, T.Imielinski,and A.Swami. Mining association rules between sets of items in large databases[C]. Proceedings of the ACM-SIGMOD International Conference Management of Date. Washington DC, May 1993. 207-216.

[29]R. Agrawal, R.Srikant. Fast algorithm for mining association rules[C]. In: Proceedings of the 20th International Conference on VLDB. Santiago, 1994. 487-499.

[30]R. Agrawal and R.Srikant. Mining Sequential Patterns[C]. Proceedings of the 11th International Conference on Data Engineering. Los Alamitos,CA:IEEE Computer Society Press, 1995.

[31]A. Inokuchi,T.Washio,H.Motoda. An apriori-based algorithm for mining frequent substructures from graph data. 4th European Conference on Principles of Knowledge Discovery and Data Mining. September 2000.

[32]A. Inokuchi, T.Washio,and H.Motoda. Complete mining of frequent patterns from graphs: Mining graph data. Machine Learning,vol.50,no.3,321-354,2003.

[33]K.Wang and H.Liu. Schema discovery for semistructured data[C]. In Proceedings of Conference on Knowledge. Discovery and Data Mining, l 997. 271-274.

[34]K.Wang and H. Liu. Discovering Typical Structures of Documents: A Road Map Approach[C]. Proc. ACM SIGIR Conf. Information Retrieval, 1998.

[35]T.Asia. Efficient substructure discovery from large semi-structured data[CJ. International Conference on Data Mining (SDM2002), Proceedings of the Second SLAM. 2002, 158-174.

[36]Y. Chi, Y. Yang, and R.R. Muntz. Indexing and Mining Free Trees. Proc. Third IEEE Int'l Conf. Data Mining, 2003.

[37]U. Ruckert and S. Kramer. Frequent Free Tree Discovery in Graph Data. Special Track on Data Mining, Proc. ACM Symp. Applied Computing, 2004.

[38]Y. Xiao, et al. Efficient Data Mining for Maximal Frequent Subtrees. The 3rd IEEE Int'l Conf on Data Mining(ICDM 2003). Melbourne,Florida,USA,2003.

[39]S. Nijssen and J.N. Kok. Efficient Discovery of Frequent Unordered Trees. Proc. First Int'l Workshop Mining Graphs, Trees and Sequences, 2003.

[40]T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering Frequent Substructures in Large Unordered Trees. Proc. Sixth Int'l Conf. Discovery Science, Oct. 2003.

[41]Y. Chi, Y. Yang, and R.R. Muntz. HybridTreeMiner: An Efficient Algorihtm for Mining Frequent Rooted Trees and Free Trees Using Canonical Forms. Proc. 16th Int'l Conf. Scientific and Statistical Database Management, 2004.

[42]A. Termier, M-C. Rousset, and M. Sebag. Treefinder: A First Step Towards XML Data Mining. Proc. IEEE Int'l Conf. Data Mining,2002.

[43]M. Zaki. SPADE: An efficient algorithm for mining frequent sequences [J]. Machine Learning, 2001 .40:31—60.

[44]M.J. Zaki. Efficiently mining frequent trees in a forest[C]. In Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, 71-80.

[45]M.J. Zaki and C.C. Aggarwal, "XRules: An Effective Structural Classifier for XML Data," Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, Aug. 2003.

[46] M.J. Zaki. Efficiently mining frequent trees in a forest:Algorithms and Applications[J]. IEEE Transactions On Knowledge and Data Engineering. Vol.17,No.8, August 2005.

[47]C. Wang, M. Hong, J. Pei, H. Zhou, W. Wang, and B. Shi. Efficient Pattern-Growth Methods for Frequent Tree Pattern Mining. Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining, 2004.

[48]马海兵．频繁模式挖掘相关技术研究.复旦大学博士论文[D]．上海:复旦大学．2005．

[49]张忆．基于XML的频繁模式发现研究.合肥工业大学硕士论文[D]．安徽:合肥工业大学．2006．

[50]Sigmod XML 数据集. Available at: http://www. acm..org/sigmod/record/xml.

[51]傅珊珊，吴扬扬．一种挖掘 XML 文档频繁子树的方法[J]．计算机工程与科学，2007.11．

[52]Available at: http://www.cs.wisc.edu/niagara/data.html .

[53]傅珊珊，吴扬扬．基于频繁结构的 XML 文档聚类[J]．计算机工程与应用．

[54]Surajit Chaudhuri, Kyuseok Shim.Storage and Retrieval of XML Data using Relational Databases[C].In: Proceedings International Conference on Data Engineering,2003:802

[55]Haifeng Jiang,Hongjun,WeiWang and Jeffrey Xu Yu. Xparent:An Efficient RDBMS-Based XML Database System. Proceedings of the 18th International Conference on Database Engineering (ICDE 02),2002.

[56]万常选.<<XML 数据库技术>>[M].北京:清华大学出版社，2005 年 1 月．

[57]傅珊珊，吴扬扬．抽取 XML 文档到关系数据库[J]．计算机工程与设计，2006 Vol.27 No.21．

攻读硕士期间参加的课题和发表的论文

参与的科研项目：

[1] 基于主动数据库的广播数据收集分类系统，福建省科技计划重点项目，课题编号 20041014

[2] 识别和抽取 Web 中的关系信息及出现模式，国务院侨办科研基金项目，课题编号 03QZR5

发表论文：

[1] 傅珊珊，吴扬扬． 抽取 XML 文档到关系数据库[J]．计算机工程与设计，2006 Vol.27 No.21．

[2] 傅珊珊，吴扬扬．从 XML 到关系数据模型的映射[J]．计算机科学，2006 Vol.33 No.11(增刊)．

[3] 苏其良，傅珊珊，吴扬扬．基于主动机制的网络数据广播过滤系统[J]．广西师范大学学报(自然科学版)，2006 Vol.24 No.2．

[4] 傅珊珊，吴扬扬．一种挖掘 XML 文档频繁子树的方法[J]．计算机工程与科学，2007.11．

[5] 傅珊珊，吴扬扬．基于频繁结构的 XML 文档聚类[J]．计算机工程与应用．（已录用）

致　谢

时光荏苒，岁月穿梭，转眼间我在华侨大学求学已经六年多了，在这段时间里，我得到了许多人的帮助，在此我要向他们表示感谢。

首先我要特别感谢我的导师吴扬扬教授，我在学习中的每一点进步都凝聚着她辛勤的汗水，吴老师忘我的工作热情、严谨的治学态度和踏实的工作作风，不仅给我留下了深刻的印象，而且将激励着我在今后的学习工作中刻苦钻研、勤学奋进。在学习和生活中，吴老师给予我大量无私的关怀和帮助，“一日为师，终身为母”，在此我要向吴老师致以最深的谢意！

感谢课题组的老师，他们是罗伟、雷庆、缑锦等，感谢课题组的同门兄弟姐妹，他们是颜毛智、谷峰、黄臻臻、林一旻、刘晨曦、苏其良、陈良图、黄毅芳、吴楚坤、石尚凯等，他们在我的研究生学习工作中给我提出了许多宝贵的意见和建议，与他们在学术上的交流使我受益匪浅，在此向他们表示衷心的感谢！

感谢华侨大学计算机系的各位老师，他们的辛勤施教使我学到了许多宝贵的知识，尤其是潘孝铭老师在本科阶段给予我学习和生活上的热心帮助和指导，在此向潘老师以及计算机系的所有老师表示真诚的谢意！

感谢我的舍友王亚惠、庄翠雅、李小丽、吴叶青等，两年来她们在生活中给了我很大的帮助，我们相伴着一起度过了许多美好的时光，我想，两年时间虽然短暂，但它将永远珍藏在我们每个人的心中！

感谢 05 级研究生同学付婷、卢晓菁、黄一青、薛清福、池静、关栋栋、匡春临、庄群洪等，他们在学习和生活中给了我很大的支持和鼓励，伴我度过了研究生阶段快乐而充实的时光，在此向他们表示感谢！

最深情的谢意献给多年来一直关心我、爱护我的家人：特别要感谢我敬爱的父母，是他们无私的爱与奉献让我在学习生活中，克服困难、勇往直前、不断地成长。“谁言寸草心，报得三春晖”，衷心祝愿你们永远幸福、安康快乐；感谢我的哥哥傅志焰，在我求学的这么多年里，是他一直陪伴在父母身边，是他的付出与鼓励让我安心的完成学业；同时，感谢我的嫂嫂伊雪凡，谢谢她的支持与鼓励！

最后，再一次感谢吴老师，再一次感谢所有关心、支持和帮助过我的人们，谢谢你们，祝福你们！