图结构模糊XML文档上的模式匹配算法

缪丰羽[1] 王宏志[2]
(宁德师范学院计算机系 宁德 352100)[1] (哈尔滨工业大学计算机学院 哈尔滨 150001)[2]

摘 要 模糊XML文档是指包含不确定信息的XML文档。在模糊XML文档查询方面,现有的研究成果较少,并且都是基于树型结构的XML文档进行的。针对图结构下模糊XML文档的特征,设计了一组高效的图结构模糊XML文档上的模式匹配算法。该算法基于一种适合于图结构文档的索引方式,采用自底向上的结点匹配顺序,大大减少了结点的重复判断操作,也不需要进行局部匹配结果的归并以及针对PC关系设计额外的过滤函数。理论分析以及实验结果证明,提出的模式匹配算法不仅在小枝查询性能上优于现有的相关算法,而且能够较好地实现DAG模式匹配查询。

关键词 图结构,模糊数据,XML,模式匹配,DAG
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.11.055

Pattern Matching Algorithms for Graph Structured Fuzzy XML Documents

MIAO Feng-yu[1] WANG Hong-zhi[2]
(Department of Computer, Ningde Normal University, Ningde 352100, China)[1]
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)[2]

Abstract Fuzzy XML documents are XML documents which contain uncertain information. There are few research achievements of fuzzy XML documents, and all of them are based on tree-structure. According to the characteristics of the graph structured fuzzy XML documents, a group of efficient algorithms were proposed in this paper. These algorithms are based on an indexing scheme which is fit for graph-structured documents, and use the bottom-up search for nodes matchings to reduce the repeat judgements greatly. Such approaches neither need to merge the portions of matching results nor need to design the filter function for PC relations. The theoretical analysis and experimental results show that, the pattern matching algorithms presented in this paper outperform the relevant algorithms in twig query performance, and accomplish the query of DAG pattern matching effectively.

Keywords Graph structure, Fuzzy data, XML, Pattern matching, DAG

1 概述

在现实世界中,存在着各种不确定的信息。自从1965年扎德(Zadeh)提出模糊集合以及相关理论以来,对不确定信息的描述和处理就有了理论基础。

在XML数据研究领域,早先的各种研究都是基于确定性的XML文档。2001年以来,陆续有研究人员在模糊集合理论的指导下,将不确定信息结合到XML文档中来。2006年,Gaurav A等人第一次提出了"fuzzy XML"[1](模糊XML)的概念,用来表示这种包含不确定信息的XML数据。随后在模糊XML上人们又进行了各种深入的研究,取得了一定的研究成果[2-9]。

然而这些研究主要是基于树型结构的模糊XML文档,由于XML文档内部可能存在ID/IDREF类型的属性,事实上图结构才是XML文档的完整表示形式,因此对于模糊XML的研究也应该推广到图结构上才更具有普遍意义。对于树型结构的模糊XML文档,主要讨论的是路径查询和小枝查询,也可称为路径模式匹配和小枝模式匹配。其中路径模式是小枝模式的一种特殊情况,即不包含分支结点的小枝模式。而对于图结构的模糊XML文档,还需要讨论给定一个有向无环图,如何在模糊XML文档中寻找合适的匹配,也即dag模式匹配。

由于树型模糊XML文档上的查询算法不能处理图结构的文档,因此无法直接将其推广到图结构的模糊XML文档上。对于图结构的模糊XML文档,我们需要深入分析其结构特征,提出一种新的解决方案来处理模式匹配查询。

本文针对图结构模糊XML文档的特征,选择了一种索引方式将图结构数据分成两个部分,分别是生成树及FPI索引表,其中FPI索引表用来存储由引用边实现的结点可达关系。在此基础上设计了一组图结构模糊XML文档上的模式

到稿日期:2015-09-02 返修日期:2015-11-30 本文受国家973计划(2012CB316200),国家自然科学基金项目(61472099,61133002),国家科技支撑计划项目(2015BAH10F00),宁德师范学院2014年校级青年专项基金(2014Q51)资助。
缪丰羽(1983—),女,硕士,讲师,主要研究方向为XML数据库查询优化,E-mail:feathen1983@163.com;王宏志(1978—),男,博士,副教授,博士生导师,主要研究方向为大数据、数据质量、图数据管理。

匹配算法。实验证明，本文提出的算法不仅在小枝查询性能上优于同类算法，而且还可以处理 dag 模式匹配问题。本文的贡献可以归纳为如下几点：

(1)将现有的模糊 XML 数据的研究范围从树型结构扩展到图型结构，对于 XML 数据而言更具有一般性。

(2)选择了一种适合于图结构模糊 XML 文档的索引方式来实现文档数据的存储及可达查询。

(3)设计了图结构 XML 文档上的小枝模式匹配算法。该算法使用了自底向上的结点判断顺序，相比于一般算法中自顶向下的顺序，可以大大减少对结点的重复判断，提高了查询的效率，并能够有效处理查询模式中的 PC 关系。

(4)设计了图型模糊 XML 文档上的 dag 模式匹配算法，使之能够有效处理图结构数据上特有的 dag 查询。

本文第 2 节给出了图结构模糊 XML 文档的数据模型，并对其特点进行了深入分析；第 3 节提出了图结构模糊 XML 文档上的结点编码方式及索引结构；第 4 节给出了小枝模式匹配算法 GFTwig 及 dag 模式匹配算法 GFDag；第 5 节将本文提出的算法与同类算法进行实验分析及比较；最后总结全文。

2 图结构模糊 XML 数据模型

XML 数据中的模糊信息分为元素的模糊性和属性值的模糊性两种，可以用成员度和可能性分布分别进行表示[8]。同时引入一个取值为[0,1] 的可能性属性“Poss”和一个模糊构造子“Val” 共同说明一个给定元素存在于 XML 文档的可能性，Poss 值即该元素的成员度。以图 1 中的文档片段为例，第 3 行〈Val Poss=“0.9”〉表示〈open_auction ID=“o1”〉这个元素属于〈open_auctions ID=“o”〉的可能性为0.7。文档中的“Dist”元素用来说明一个可能性分布。一个 Dist 元素通常有多个 Val 元素作为孩子，每一个孩子的 Poss 值表示不同属性的可能性。例如，图 1 中第 11 行和第 15 行的 Val 表示 bidders 元素的孩子有可能是 bidder“b1”，也有可能是 bidder“b2”，可能性分别是 0.8 和 0.6。

```
1. 〈site ID="r" 〉
2. 〈open_auctions ID="o" 〉
3.   〈Val Poss="0.9"〉
4.     〈open_auction ID="o1"〉
5.       〈author ID="a" IDREF="p2"〉
6.       〈/author〉
7.       〈seller ID="s"〉
8.       〈/seller〉
9.       〈bidders ID="b"〉
10.        〈Dist〉
11.          〈Val Poss="0.8"〉
12.            〈bidder ID="b1"〉
13.            〈/bidder〉
14.          〈/Val〉
15.          〈Val Poss="0.6"〉
16.              〈bidder ID="b2" IDREF="w"〉
17.            〈/bidder〉
18.          /Val〉
19.        〈/Dist〉
20.      〈/bidders〉
21.    〈/open_aution〉
22.  〈/Val〉
23.  〈Val Poss="0.7"〉
24.    〈open_auction ID="o2"〉
25.    〈/open_aution〉
26.  〈/Val〉
27. 〈/open_auctions〉
28. 〈people ID="p"〉
29. 〈person ID="p1"〉
30.    〈watchs ID="w"〉
31.      〈watch ID="w1" IDREF="o2"〉
32.      〈/watch〉
33.      〈watch ID="w2" IDREF="o1"〉
34.      〈/watch〉
35.    〈/watchs〉
36.  〈/person〉
37.  〈person ID="p2"〉
38.   /person〉
39. 〈/people〉
40. 〈/site〉
```

图 1 图结构模糊 XML 文档片段

一般的模糊 XML 文档可以用包含精确结点(不含模糊构造子、Dist 和可能性属性 Poss 的结点)、模糊构造子(Val 和 Dist)以及可能性属性 Poss 的标签树来表示。其中模糊构造子和可能性属性结点称为模糊结点。文献[5,7,8]讨论了在一般模糊 XML 文档上的小枝匹配算法。但有些模糊 XML 文档中含有 ID/IDREF 属性(例如图 1 所示的文档片段)，可能在结点之间产生引用关系，使模糊 XML 文档从一般的树型结构变成图型结构。

由 XML 文档自身的特点可知，图结构的 XML 文档可以表示为一张结点有序的有向图，其中可能包括回路(即环)[10]。然而一般有向图中的强连通分量对查询是没有意义的。因此可以依照文献[10]提出的方法，消除图结构的模糊 XML 文档中的强连通分量，将其简化成有向无环图 DAG。处理完之后，图结构模糊 XML 文档上的查询处理也就转化成了对 DAG 的查询处理。对于这种图型结构的模糊 XML 文档，除了一般的路径模式匹配和小枝模式匹配，还存在一种查询方式即 dag 模式匹配。所谓的 dag 模式匹配是指查询模式为有向无环图的查询匹配，如图 2 所示。因此，需要针对图结构模糊 XML 文档的特征，选择一种适合的索引方式，进行高效的模式匹配查询，尤其是 dag 模式匹配的查询。

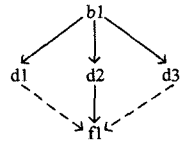

图 2 dag 查询模式

3 结点编码与索引结构

文献[5]提出了一种模糊 XML 文档的编码方式：用一个五元组(*DocID*, *LeftPos*, *RightPos*, *LevelNum*, *Fuzzy*, *FuzzySequence*)来表示模糊 XML 文档中的结点信息。其中：1)*DocID* 是文档的编号；2)*LeftPos* 是先序遍历文档时，第一次访问该结点的编号，*RightPos* 是第二次访问该结点的

编号，若 $LeftPos=RightPos$，则该结点为叶子结点；3）$LevelNum$ 是结点的层数，一般规定根结点的层数为 1；4）$Fuzzy$ 用来标识结点的性质，如果取值为 1 表示该结点为模糊结点，如果取值为 0 表示该结点为精确结点；5）$FuzzySequence$ 用来存储从根结点到当前结点的路径中经过的所有模糊结点。

这种编码方式的本质是在一种名为 zhang 编码[11]的区间编码基础上进行的扩展。这种编码方式可以有效地表达树型模糊 XML 文档中结点的可达信息，但是对于图型结构的数据，这种方式暴露其局限性。由于图数据中的结点可能有多条入边，在遍历的过程中，该结点有可能被多次访问，因此无法赋予结点唯一的 $LeftPos$ 和 $RightPos$ 值。

解决的方法有两种：一种是采用新的编码方式，使得结点编码可以表示图中的结点可达关系[10]。通常这种编码方式需要对结点赋予多个区间值。另一种是对图结构的文档构建索引[12]。这种索引可以分为 4 类：基于链的索引[13]、基于树的索引[14-16]、基于 path-tree 的索引[17]以及基于 hop 的索引[18]。

本文选择对图结构的模糊 XML 文档构建索引。由于许多 XML 文档的 ID/IDREF 边并不多，因此选择基于树的索引。

首先，对文档进行解析，将所有的元素、非 ID/IDREF 属性及其嵌套关系构建成一棵文档的生成树。可以将原来的五元组（$DocID, LeftPos:RightPos, LevelNum, Fuzzy, FuzzySequence$）修改为（$NID, LeftPos:RightPos, LevelNum, Fuzzy, FuzzySequence$）来表示树中的结点，其中 NID 表示文档中的结点 ID。因为对于查询而言，跨文档的查询结果是没有意义的。在解析的过程中同时收集各引用边信息，存储在一张引用边表中。这个步骤的时间复杂度为 $O(n)$，n 为文档的结点数。

然后，创建一个索引表 FPI，用来保存由引用边可达的结点信息。创建方法是：按先序遍历生成树上的所有结点，对于每个结点 w，检查其所有入边，若其中有引用边，则在 FPI 索引中创建一行元组，NID 部分为 w，$preds$ 值为所有的引用边起始结点。在创建 FPI 的过程中，若 w 的父结点在 FPI 索引中有对应的元组，则为 w 创建一行元组，$preds$ 值为 w 的父结点，或将 w 的父结点添加进 w 对应的元组中。例如对于图 1 所示的模糊 XML 文档片段，建成的 FPI 索引如表 1 所列。这个步骤的时间复杂度为 $O(m)$，m 为文档的边数。

表 1　FPI 索引

NID	preds
p2	a
w	b2
o2	w1
o1	w2

该索引方式的建立时间复杂度在最坏情况下为 $O(n+m)$。

对于给定的两个结点 i 和 j，若满足 $i.LeftPos<j.LeftPos$ 且 $i.RightPos>j.RighPos$，则 i 可达 j；否则还需要检查结点 i 是否可以通过引用边到达结点 j，所以接下来需要查找 FPI 索引来判断结点的可达关系。根据 FPI 索引找到 NID 为 j 的元组，取 $preds$ 列表中的结点依次判断。设结点 j 的 $preds$ 列表为 $p[j]$，当前判断的结点为 a，有：1）若结点 i 的 $LeftPos:RightPos$ 值包含结点 a 的 $LeftPos:RightPos$ 值，则 i 可达 j；2）若结点 i 的 $RightPos$ 值小于结点 a 的 $LeftPos$ 值，则 i 必不可达 j；3）若非前面两种情况，则还需判断在 FPI 索引中是否存在以 a 为 NID 的元组，如果没有，将 a 从 $p[j]$ 中删除，否则递归判断 a 的上一级前任中是否有结点编码在 i 的编码范围之内，如果都没有，则也可将 a 从 $p[j]$ 中删除，并且将 $p[a]$ 中剩余的结点添加到 $p[j]$ 中。根据以上描述可以设计出函数 $checkContainment(i,j)$ 来判断结点 i 和结点 j 之间是否可达。由于篇幅限制，该函数的具体实现代码未给出。若结点 i 可达结点 j，则意味着 i 和 j 之间满足 AD 关系。通过对结点层数的简单计算，也可以进一步推出 i,j 之间是否满足 PC 关系。

在最坏情况下，$checkContainment$ 函数需要找遍 FPI 索引中所有 $preds$ 值中的结点，而这些结点不会超过 $m-n$，其中 m 为图的总边数，n 为图的总结点数。所以在最坏情况下，$checkContainment$ 函数的时间复杂度为 $O(m-n)$。

4　模式匹配算法

在模糊 XML 数据中进行模式匹配查询，与普通的 XML 数据模式匹配查询的区别主要在于，必须增加一个整体成员度 δ_{whole} 与给定阈值的判断。整体成员度指的是匹配的候选结果中所有模糊结点的整体可能性，给定阈值则是用户给出的一个指标。只有当整体成员度大于给定阈值时，结果才能返回。因此，模糊 XML 数据的模式匹配查询首先需要定义整体成员度的计算方法。在文献[8]中采用下面的 Einstein 算子来计算两个成员度的交。

$$\delta_{whole}=\frac{\delta_i\times\delta_j}{1+(1-\delta_i)\times(1-\delta_j)}$$

在 Einstein 算子中，δ_i 和 δ_j 分别表示结点 i 和 j 的成员度，δ_{whole} 表示经过交运算后得到的整体成员度。

如果在查询模式的两个结点 i 和 j 之间存在多重模糊信息，则需要对这些模糊信息进行计算来求得这两个结点之间的局部修正成员度 $\delta_{revise}(i,j)$。

$$\delta_{revise}(i,j)=\bigcap_{t=1}^{x} f_t$$

其中，f_t 表示结点 i 和结点 j 之间的所有模糊结点的成员度的集合，$\delta_{revise}(i,j)$ 即这些成员度的 Einstein 交。最终的整体成员度也是根据修正后的成员度来进行计算的。

文献[8]在上述两个算子的基础上，设计了小枝查询算法 TwigFX 以及利用 X B-tree 索引的改进算法 TwigIndex。这两种算法存在以下几个较大的缺陷：1）算法的基本思路是自顶向下地处理查询结点对应的数据结点，只有某个查询结点所有的孩子结点存在满足条件的数据结点时，该查询结点才能入栈。该结点入栈后，又需要用同样的方式重新分析其孩子结点。这种方式存在大量的重复判断操作。2）算法采用的是寻找局部匹配结果，然后进行归并的方式，效率较低。3）这两种算法在只含有 AD 关系的查询中能获得较高的性能，但是如果查询中包含 PC 关系，则可能在最终的合并过程中产生较多的无效匹配，使查询效率大大降低，并需要增加设计相应的过滤算法。4）这两种算法只能针对传统的树型结构模糊 XML 文档进行查询，在图结构模糊 XML 文档查询中并不适用，也不能处理图结构文档中的 dag 查询。

本文结合第 2 节中介绍的索引方式设计了一种适合图结构模糊 XML 文档的小枝模式匹配算法，并针对图结构模糊

XML 文档环境下特有的 dag 查询，设计了相应的 dag 模式匹配算法。下面分别对这两个算法进行介绍。

4.1 GFTwig

用成员度来表示的模糊信息在模式匹配算法中体现为对结点的三重过滤，第一重过滤：当前结点 q 的成员度 δ_q 必须大于或等于模糊阈值 u；第二重过滤，从查询根结点 q_{root} 到当前结点 q 这条路径上的整体成员度 δ_{path} 必须大于或等于 u；第三重过滤，若当前结点 q 为查询模式中的分支结点，则 q 所在的小枝必须满足整体成员度 δ_{whole} 大于或等于 u。在文献[8]中，这三重过滤分散体现在算法的不同位置。本文算法将这三重过滤集中设计为一个返回值为 bool 类型的函数 *Filter*(q)，返回值为 1 表示结点 q 通过了三重过滤，可以进一步参与结果的匹配。最坏情况下，该函数的时间复杂度为 O(n)，其中 n 为查询模式中的结点数。由于篇幅限制，该函数的具体实现代码未给出。

GFTwig 算法的基本思想是：自底向上地处理所有查询结点对应的数据结点，在最大程度上避免重复的判断操作。

在算法 GFTwig 中，依据查询模式将各查询堆栈设置成树状结构，如图 3、图 4 所示。堆栈之间的关系同查询模式中查询结点之间的关系一一对应。每个堆栈 S_{qi} 设置一个指针集合 *Ptrs*，用来指向其所有的查询子结点栈。栈中的元素为二元组(*NID*, *ptrs*)，其中 *NID* 代表入栈结点的 *id*，*ptrs* 为指针集合，集合中的指针指向该元素的子孙结点在相应堆栈中的位置。此外，算法还设置了一个长度为查询结点数 n 的数组，用来存储每个查询堆栈的栈顶位置。

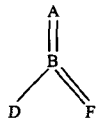

图 3 小枝查询

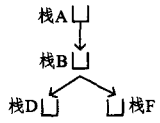

图 4 堆栈结构

算法首先将查询模式按深度优先顺序进行遍历，在遍历的过程中生成一个查询结点序列，并对每个查询叶结点做标记。如图 3 所示的小枝查询，按深度优先顺序生成的查询结点序列为：A、B、D、F，且 D、F 为查询叶结点。接着将所有查询叶结点对应的数据结点全部入栈。对于剩下的查询结点，则按照前面生成序列的逆序进行处理，即首先处理查询结点 B，然后处理查询结点 A。对于这些非叶查询结点，依次取其数据流中的每个数据结点进行判断，判断条件为：该查询结点的所有子查询结点栈中是否都有结点与当前处理的数据结点满足查询关系，并且该结点可以通过模糊信息的三重过滤。若有，则该数据结点入栈，并生成指向其子孙结点的指针。本文算法中的这个步骤同文献[8]中相关算法的不同之处在于：因本文算法采用自底向上的堆栈处理方法，该步骤中只需要判断第一代子查询结点栈，并不需要递归查找下面的所有子孙栈，因为下面的子孙栈一定满足条件，所以本文的算法可以最大限度地避免重复操作；并且本文的算法将分散在算法中的三重过滤集中用一个函数 *Filter* 实现，使主函数的实现更加清晰易懂。当所有查询结点都处理完时，调用相应的过程输出匹配结果，且查询匹配结果以查询结点序列的方式输出。

算法的具体主程序如下所示。

```
Algorithm GFTwig
1. 对查询模式进行拓扑排序，并得到查询叶结点集合 L 和剩余结点集合 Q
2.  for qi∈L
3.    for j=1 to |Tqi|
4.      push(Sqi,(next(Tqi),∅))
5.      advance(Tqi)
6.  for qi∈ Q in inverted sequence
7.    while(not end(Tqi))
8.      tq=next(Tqi)
9.      tem=∅
10.      allfound=true
11.      for qj∈ childs(qi)
12.        found=false
13.        for p=1 to |Sqj|
14.          h=Sqj. p. NID
15.            if ((tq. LeftPos < h. LeftPos and q. RightPos >
                 h. RightPos) or (checkContainment(tq,h))
16.                if (q//child(q) ∈ Q or tq. LevelNum =
                     h. LevelNum−1 or tq∈PL[h]) // PL[h]表示
                     FPI 索引中 NID 为 h 的元组对应的 preds 列
                     表
17.                  tem+=qj. p
18.                  found=true
19.        allfound=allfound and found
20.      if (allfound and Filter(tq))
21.        for ptr ∈ tem
22.          tq. ptrs+=ptr
23.        push(Sqi,(tq,ptrs))
24.      advance(Tqi)
25.  for i=1 to |Sqi|
26.    outputSolutionsT(Sqi,i)
```

```
Procedure outputSolutionsT (SN,SP)
1. if (S[SN] ∈ Q or S[SN]. SP. ptrs≠∅)
2.    index[SN]=SP
3.  else
4.    return
5.  if (SN==n)
6.    output (S[1]. index[1],…,S[n]. index[n])
7.  else
8.    for q∈childs(qSN)
9.      for i=1 to |Sq|
10.       if (q. i ∈ S[SN]. SP. ptrs)
11.         outputSolutionsT(Sq,i)
```

从算法主程序可以看出，该算法能够统一处理小枝查询中的 AD 关系以及 PC 关系，不需要针对 PC 关系设计额外的过滤函数，也不需要设计归并算法对局部匹配结果进行合并。

下面分析算法 GFTwig 的时间复杂度。记每个查询结点 q_i 对应的数据结点数为 b_i，所有查询结点对应的数据结点总数为 b；记所有叶查询结点对应的数据结点总数为 bL，所有的非叶查询结点对应的数据结点总数为 bQ，$bL+bQ=b$。

算法的第一部分是预处理，即对查询模式进行拓扑排序，这个步骤的时间复杂度为 O(n)，其中 n 为查询模式中的结点数目。算法的第二部分是将所有的查询叶结点对应的数据结点全部入栈，时间为 bL。算法的第三部分是依次判断非叶的查询结点 q_i 对应的所有数据结点 tq，检查其所有的子查询栈中是否都存在 tq 可达的元素。需要进行上述判断的结点数包括两个部分，一部分为叶查询栈中的结点，总数为 bL，另一

部分的上限为 bQ，在最坏情况下可用 b 来表示需要进行判断的结点总数。在每次可达判断中，首先根据两个结点的 $LeftPos$:$RightPos$ 值进行初步判断，如果两个结点的编码不满足可达条件，则需进一步通过 *checkContainment* 函数查找 FPI 索引进行判断。在 FPI 索引中被查找的结点总数不超过 $m+b*d$，其中 m 为图的总边数，d 为算法中递归查找的次数。*Filter* 函数中，需要判断的结点数为 bQ，所需要判断的次数为 $3bQ$。算法的第二部分 *outputSolutionsT* 函数代价为 $n*|outputlists|$，其中 $|outputlists|$ 为输出结果序列的数目。

综上所述，算法 GFTwig 的最坏时间复杂度为 $O(n+bl+b+m+b*d+3bQ+n*|outputlists|)$。最坏空间复杂度为 $O(\min(b,n*P_{max}))$，其中 P_{max} 为 XML 文档中最大的路径长度。

4.2 GFDag

对于图结构的模糊 XML 文档，还存在着一种模式匹配查询，即查询模式为一个有向无环图的查询，也称为 dag 查询。在这种查询模式中，有些结点具有多个父亲结点，即具有多条入边，这种结点称为汇结点。

GFDag 算法的基本思想是：首先将 dag 查询模式中的汇结点拆成多个具有相同标签的结点，选择拆分后的第一个结点继承原结点的查询子孙结点，其他的同标签结点成为查询叶结点。这样可以得到 dag 查询模式对应的一个小枝模式。例如：对于 DQ=//B(//D/F,//C//D)，即图 5 所示的 dag 查询模式，查询结点 D 为汇结点。我们将其拆分成两个具有相同标签的查询结点，并且其中一个结点继承了原汇结点所有的查询子孙结点，即 F，而另一个结点成为查询叶结点。结点拆分后对应的小枝模式如图 6 所示。之后执行修改过的小枝匹配算法得到所有的匹配结果，在这个修改后的小枝匹配算法中，对于相同标签的查询结点，必须在同一匹配结果中取相同值。

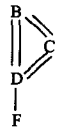

图 5 查询模式

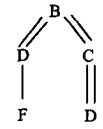

图 6 小枝模式

在拆分汇结点后执行的算法 GFDag 中，主程序部分与算法 GFTwig 基本相同，只有结果输出函数由函数 *outputSolutionsT* 修改而成。修改后的函数名为 *outputSolutionsD*。在函数 *outputSolutionsD* 中的输出结果必须满足：在所有相同标签的查询结点位置对应的数据结点都必须相同，因为这些查询结点对应的是同一个查询结点。具体的函数 *outputSolutionsD* 如下，其中的相同标签集合 *Rlable* 中记录的是 dag 中的所有汇结点，由于篇幅限制，本文暂不考虑另一类的相同标签结点。

```
Procedure outputSolutionsD (SN,SP)
1.  if (S[SN]∈Q or S[SN]. SP. ptrs≠∅)
2.    if (S[SN]. lable∈Rlable)
3.      for i=1 to SN−1
4.        if (S[i]. lable = = S[SN]. lable and S[i]. index[Si] ≠
          S[SN]. SP))
5.        return
6.      index[SN]=SP
7.    else
8.      index[SN]=SP
9.  else
10.   return
11. if (SN= =n)
12.   output (S[1]. index[1],…,S[n]. index[n])
13. else
14.   for q∈childs(qSN)
15.     for i=1 to |Sq|
16.       if (q. i∈S[SN]. SP. ptrs)
17.         outputSolutionsT(Sq,i)
```

GFDag 算法与 GFTwig 算法的时间复杂度和空间复杂度基本相同，不再赘述。

5 实验分析

实验的硬件平台：AMD Athlon™II X4 640 Processor，主频为 3GHz，3.25GB 内存；软件平台：Microsoft Windows XP SP3，SAX2 API，SQL Server 2000 以及 JAVA 编程语言。

实验采用了 XML 数据集 XMark，XMark 是一种由生成工具产生的图结构的 XML 数据，它具有复杂的路径结构和很多具有多入边的结点。本文选取几个不同大小的测试数据，如表 2 所列。

表 2 测试文档参数

数据大小(M)	5	10	50	100
结点数目	83533	167865	832911	1666315
边数	93507	186905	929101	1858173

实验首先对 Xmark 文档进行预处理，即消除文档中的强连通分量。这个处理过程的时间代价为 $O(n+nc)$，其中 n 为文档中的结点总数，nc 为所有强连通分量中的结点数。

之后再选用文献[19]中使用的 Random Walk 方法来将处理过的 XMark 文档构造成测试用的模糊 XML 文档。该方法可以随机地在文档的两个结点之间生成一个模糊结点。该模糊结点的成员度取值范围为[0,1]。生成过程中可以自行设置模糊结点的数量。

目前为止，由于在图结构的模糊 XML 文档查询领域并没有相关的模式匹配算法，因此我们选择了树结构模糊 XML 数据查询领域较有代表性的几个算法 EVALDP[19]算法、C-Twig[5]算法以及 TwigIndex[8]算法与本文提出的 GFTwig 算法进行比较。在进行对比实验时，需要先对测试文档进行预处理，提取其生成树部分对这些算法进行测试，且只能进行小枝查询。具体使用到的查询如表 3 所列。

表 3 实验中用到的查询表达式

编号	查询表达式
Q1	//site//person//age
Q2	//site//people/person/name
Q3	//site(//item//description,//category//name)
Q4	//site(//item/description,//category/name)
Q5	//site(//item//category,//category//name)
Q6	//site(//incategory/category,//category/name)

首先选择 100M 的模糊 Xmark 文档，将其进行处理，得到对应的生成树部分进行各算法的对比测试。由于文献[8]提出的算法 TwigIndex 需要针对每次查询分别构建相应结点的 XB-tree 索引来帮助过滤部分结点，因此查询时间还应该包括额外的索引建立时间。而本文提出的查询匹配算法只需要针对一个文档一次性生成索引即可，因此不需要在查询中另外计算索引生成时间。

从图 7 可知，GFTwig 算法的查询性能总体上优于其他 3 种算法，并且对于含有 PC 关系的查询，响应时间上并没有太大的影响。而其他几种算法，尤其是 TwigIndex，对于 Q2 和 Q4 这两个带 PC 边的查询，响应时间明显增大。

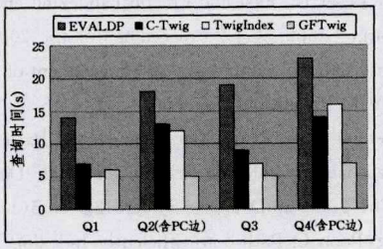

图 7　100M 测试文档上的查询时间对比

图 8 为 Q4 查询在不同大小的测试文档上各算法的查询响应时间对比。从图中可以看出，整体上各算法的查询响应时间根据文档大小的增大而增大，并且 GFTwig 算法在查询效率上占有明显的优势；而对于 Q4 这种含有 PC 边的查询，TwigIndex 需要过滤匹配结果，因此查询性能有所下降。

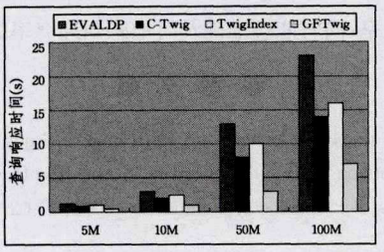

图 8　Q4 上不同大小文档的查询时间

由于 EVALDP 算法、C-Twig 算法以及 TwigIndex 算法不能处理图结构上的模糊 XML 文档，因此只能单独对 GFTwig 算法和 GFDag 算法做查询性能上的测试。图 9 给出了 GFTwig 算法在不同大小的文档上针对 Q1－Q4 4 种查询的查询响应时间，图 10 为 GFDag 算法在不同大小的文档上针对 Q5 和 Q6 查询的响应时间。

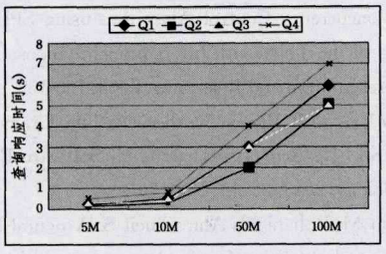

图 9　GFTwig 算法的查询时间

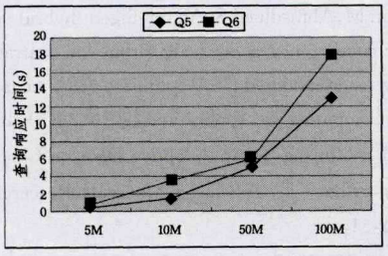

图 10　GFDag 算法的查询时间

从图 9、图 10 可以看出，所提算法可以实现图结构模糊 XML 文档上的模式匹配查询，包括带 PC 边或者不带 PC 边的查询，以及 dag 查询。查询响应时间在合理的预期范围之内，具有良好的查询性能。

综上所述，本文提出的算法在处理树型结构模糊 XML 文档查询时，可以得到优于 TwigIndex 等算法的查询效率，同时还可以处理图型结构的模糊 XML 文档，在其上进行小枝模式匹配以及 dag 模式匹配，并取得良好的查询性能。

结束语　本文分析了图结构模糊 XML 文档的特征，以及当前模糊 XML 文档上普遍采用的建模方式和典型的小枝匹配算法，针对其若干不足进行了扩展和改进。首先将模糊 XML 数据的研究范围从一般的树型结构扩展到图型结构；其次针对文献[8]中算法在处理 PC 关系上的不足进行了补充；再次，从算法结构上对文献[8]中小枝匹配算法进行了改进，将其自顶向下的结点判断方式改成了自底向上的判断，大大减少了算法中对结点的重复判断，提高了查询的效率；最后还设计了图型模糊 XML 文档上的 dag 模式匹配算法。实验证明，本文提出的算法在处理树型模糊 XML 文档上优于文献[8]提出的小枝匹配算法，并能较好地实现图结构模糊 XML 文档的模式匹配查询。

参 考 文 献

[1] Gaurav A, Alhajj R. Incorporating Fuzziness in XML and Mapping Fuzzy Relational Data into Fuzzy XML[C]// Proceedings of the 2006 ACM Symposium on Applied Computing. Dijon, France, 2006: 456-460

[2] Li Ya-wen, Wang Guo-ren, Xin Juan-chang. Holistically Twig Matching in Probabilistic XML[C]// Proceedings of the 25th International Conference on Data Engineering. Shanghai: IEEE, 2009: 1649-1656

[3] Liu Si-qi, Wang Guo-ren. Boosting Twig Joins in Probabilistic XML[C]// Proceedings of the 22nd International Conference on Database and Expert Systems Applications. France: Springer Verlag, 2011: 51-58

[4] Ma Zong-ming, Yan Li. Fuzzy XML Data Modeling with The UML and Relational Data Models[J]. Data & Knowledge Engineering, 2007, 63(3): 972-996

[5] Liu Jian, Ma Zong-ming, Yan Li. Efficient Processing of Twig Pattern Matching in Fuzzy XML[C]// Proceedings of the 18th ACM Conference on Information and Knowledge Management. Hong Kong, China, 2009: 193-204

[6] Yan Li, Liu Jian. Conceptual Design Methodology for Fuzzy XML Model[J]. Computer Science, 2011, 38(12): 156-161(in Chinese)
严丽，刘健. 一种模糊 XML 模型的概念设计方法[J]. 计算机科学，2011，38(12)：156-161

[7] Ma Zong-ming, Liu Jian, Yan Li. Matching Twigs in Fuzzy XML [J]. Information Sciences, 2011, 181(1): 184-200

[8] Liu Jian, Ma Zong-min, Qu Qiu-long. Querying Twigs in Fuzzy XML Documents[J]. Chinese Journal of Computers, 2014, 9(37): 1972-1985(in Chinese)
刘健，马宗民，璩秋龙. 基于模糊 XML 的小枝查询处理[J]. 计算机学报，2014，9(37)：1972-1985

[9] Liu Li-xin, Zhang Xiao-lin, Lv qin, et al. Matching Twig Patterns in Uncertain XML without Merging[J]. Computer Science, 2013, 40(5): 198-200(in Chinese)
刘立新，张晓琳，吕庆，等. 一种非归并不确定 XML 小枝模式查询算法[J]. 计算机科学，2013，40(5)：198-200

[10] Wang Hong-zhi, Wang Wei. Labeling Scheme and Structural Joins for Graph-Structured XML Data[J]. Lecture Notes in

Computer Science,2005,3399:277-289
[11] Zhang C,Naughton J,DeWitt D,et al. On Supporting Containment Queries in Relational Database Management Systems[C]// Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. Santa Barbara, California, United States:ACM Press,2001:425-436
[12] Fu Li-zhen,Meng Xiao-feng. Reachability Indexing for Large-Scale Graphs:Studies and Forecasts[J]. Journal of Computer Research and Development,2015,52(1):116-129(in Chinese)
富丽贞,孟小峰. 大规模图数据可达性索引技术[J]. 计算机研究与发展,2015,52(1):116-129
[13] Chen Yang-jun,Chen Yi-bin. An Efficient Algorithm for Answering Graph Reachability Queries[C]// Proceedings of IEEE ICDE08. Piscataway,NJ:IEEE,2008:893-902
[14] Wang Hong-zhi,Li Jian-zhong,Luo Ji-zhou,et al. Hash-base Subgraph Query Processing Method for Graph-structured XML Documents[J]. Proceedings of the VLDB Endownment,2008,1(1):478-489
[15] Chen Li,Gupta A. Stack-based Algorithms for Pattern Matching on DAGs[C]// Proceedings of the 31st International Conference on Very Large Data Bases (VLDB). Trondheim,Norway:VLDB Endownment,2005:493-504
[16] Trißl S, Leser U. Fast and Practical Indexing and Querying of Very Large Graphs[C]// Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. Beijing, China:ACM Press,2007:845-856
[17] Jin Ruo-ming,Ruan Ning,Xiang Yang,et al. Path-tree:An Efficient Reachability Indexing Scheme for Large Directed Graphs [J]. ACM Trans on Database System,2011,36(1):73-84
[18] Cai Jing, Poon C. Path-hop:Efficiently Indexing Large Graphs for Reachability Queries[C]// Proc of ACM CIKM'10. New York:ACM,2010:119-128
[19] Kimelfeld B, Kosharovshy Y, Sagiv Y. Query efficiency in probabilistic XML models[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. Vancouver,Canada,2008:701-714

(上接第 283 页)

算法会表现出比 HCC 算法更高的时间效率。ACC-PRC 算法通过双向阈值的设置减少了事务冲突率变化导致的策略切换时间开销,保证了算法的平稳执行。

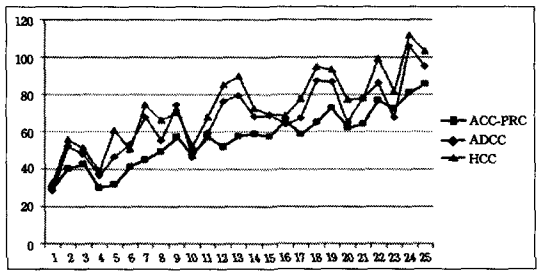

图 5 事务平均执行时间对比图

从图 4 和图 5 的对比实验中可以看出,ACC-PRC 算法具有较优的性能。HCC 算法采用 ART2 神经网络预测冲突率,虽然保证了较高的准确性,但是神经网络结构复杂,计算时间长,对于需要实时计算结果的并发情况来说,处理时间过长必然会导致事务执行效率的降低。ADCC 算法在冲突率变化剧烈时可能频繁地切换系统策略导致事务的执行效率低下。而 ACC-PRC 算法结构简单,并能够及时将预测结果反馈给系统,在保证计算精度的情况下,通过双向阈值避免了不必要的系统开销,提高了事务执行效率。

结束语 本文研究了现有的并发控制算法,并针对 HCC 算法和 ADCC 算法的不足,提出了一种新的并发控制算法:ACC-PRC 算法。该算法在 ADCC 算法的基础上将并发控制过程分为信息收集和策略选择两个阶段。信息收集阶段利用先验事务队列保证事务执行的可串行化,并且收集系统的事务执行状态。策略选择阶段在循环冲突队列上运用改进的加权移动平均法预测下一阶段冲突率,并根据双向阈值决策下一阶段的并发策略(乐观/悲观并发策略)。ACC-PRC 算法利用反馈值提高了预测冲突率的精度,并利用双向阈值避免了系统并发策略的频繁切换。对比实验表明,ACC-PRC 算法在预测精度、事务平均执行时间等方面优于 HCC 算法和 ADCC 算法。ACC-PRC 算法能够在事务到达率较高时保持良好的事务执行效率,同时能够准确及时地感知冲突率的变化。

参 考 文 献

[1] Xu Shu-ting,Sun Yong-qiang. Performance Study of Real-Time Multiversion Concurrency Control Protocols in Parallel Database Systems[J]. Chinese J. Computers, 2002, 25(2): 173-180(in Chinese)
徐淑顽,孙永强. 并行数据库实时多版本并发控制协议性能研究[J]. 计算机学报,2002,25(2):173-180
[2] Nystrom D,Nolin M,Teoanovio A,et al. Pessimistic concurrency control and versioning to support database pointers in real-time databases; proceedings of the Real-Time Systems[C]// 16th Euromicro Conference on Real-time Systems. 2004, 16: 261-270
[3] Makni A,Bouaziz R,Gargouri F. Formal Verification of an Optimistic Concurrency Control Algorithm using SPIN[C]// Proceedings of the Thirteenth International Symposium on Temporal Representation and Reasoning. 2006:160-167
[4] Demsky B,Lam P. Views:Synthesizing Fine-Grained Concurrency Control [J]. Acm Transactions on Software Engineering & Methodology,2013,22(1):139-176
[5] Sheikhan M,Rohani M,Ahmadlueil S. A neural-based concurrency control algorithm for database systems [J]. Neural Computing & Applications,2013,22(1):161-174
[6] Sheikhan M,Ahmadlueil S. An intelligent hybrid optimistic/pessimistic concurrency control algorithm for centralized database systems using modified GSA-optimized ART neural model [J]. Neural Computing & Applications,2013,23(6):1815-1829
[7] Chen Y R,Zhuang Y. An Adaptive Decision Concurrency Control Algorithm [J]. Advanced Materials Research, 2014, 1046(1):512-515
[8] Liu Ya-qi,Xie Bo. Iron ore CIF price forecasting based on improved weighted moving average method[J]. Journal of Shanghai Maritime University,2015(2):55-59(in Chinese)
刘雅琪,谢波. 基于改进加权移动平均法的铁矿石到岸价格预测[J]. 上海海事大学学报,2015(2):55-59