基于分层思想的XML文档变化检测算法—DL-Diff

陈振洲 李 磊
(中山大学软件研究所 广州510275)

摘 要 XML文档的变化检测可以广泛应用于构建XML数据仓库中的数据存储、文档模式提取以及增量查询等方面。随着被检测的数据规模的增加,对检测算法的效率也提出了更高的要求。本文利用分层的思想,提出了一种新的XML文档的变化检测方法DL-Diff,并验证了算法的有效性

关键词 变化检测,分层,编辑脚本,动态规划

A Change Detection Algorithm Based on Delaminating for XML Document: DL-Diff

CHEN Zhen-Zhou LI Lei
(Software Institute, Zhongshan University, Guangzhou 510275)

Abstract The change detection for XML document can be very useful to data storage of XML DW, pattern abstracting of document and incremental query evaluation. With increasing of the data scale to be detected, the efficient of the algorithm is more important. Using delaminating, this paper proposes a new change detection algorithm (DL-Diff) for XML document. We realize the algorithm and prove the efficiency of the algorithm.

Keywords Change detection, Delaminate, Edit script, Dynamic programming

1 引言

XML是Web上半结构化文档和数据的一种通用格式,它将取代HTML成为Web上发布和交换数据的标准格式[1]。由于在线信息的变化非常频繁,需要有一种工具来检测这种变化。随着被检测的数据规模的增加,对检测算法的效率也提出了更高的要求。

目前,对文档的变化检测算法主要有两类:一类是针对普通文件,比较典型的算法有CVS[2]和GUNdiff,文[3]指出这类算法不适合结构化数据形式;另一类是针对XML为代表的半结构化的数据,本文主要讨论这一类检测算法。

AT&T Internet Difference Engine[4]利用HtmlDiff[5]来比较两个HTML页面的区别。Htmldiff将两个HTML页面看成两个符号序列,并用加权LCS算法来寻找这两个序列的最佳匹配,但是这种方法并不适合XML文档。我们知道XML文档可以用树结构来描述,自然可以利用tree-to-tree修正方法[6,7]来检测XML文档的变化。XMLTreeDiff[8]用来计算两个XML文档的变化。首先,它利用DOMHash[9]来计算两个文档结点的hash值,并通过比较hash值来删除相同的子树;然后利用文[10]的算法来计算最终结果,但是XMLTreeDiff不一定能够得到优化的结果。Cobena等提出了另外一种检测XML文档变化算法XyDiff[11]。该算法首先采用自底向上底方式计算每个结点的签名(hash值)和权重(子树的大小),然后从两个文档的根结点开始比较两个结点的签名。由于在算法中使用了贪婪规则,XyDiff不能保证任何形式的最优和近似最优的结果。

XMLTree算法和XyDiff算法主要针对有序结点树。无序结点树的匹配被证明是一个NP问题,因此结点的无序性将会增加问题的难点,但是无序性在实际应用中有重要意义。Yuan Wang等[12]针对无序的结点模式提出了X-Diff算法,该算法是目前针对无序模式比较好的算法。我们在前面的工作中,提出了基于遗传算法的XML文档变化检测算法GA-Diff,该算法经过验证,对比较大的文档效果比较好;但是对于小文档,算法的效果一般,比X-Diff算法差。

在上述的这些算法中,大部分都是应用动态规划思想来解决问题,其中GA-Diff利用遗传算法思想。本文根据XML文档树的结构特点(假定XML文档树是分层的,所以操作都是与层次相关),提出了基于分层思想(delamination)的XML文档变化检测算法DL-Diff。

2 问题模型和基本定义

2.1 XML文档的树描述

一个XML文档(图1)可以用一棵带标号的树来表示(DOM树——图2),树中的每个结点对应文档中的元素,树中的每条边表示结点之间的关系。

```
<autos>
  <manufacturer name='Chevrolet' history="1978'>
    <model>2000 Convertible</model>
    <price>60,000</price>
    <horsePower>420</horsePower>
  </make>
  </manufacturer>
  <manufacturer name="Mazda" history='1970'>
    <model>test model</model>
    <price>30,000</price>
    <horsePower>350</horsePower>
    <fuelCapacity>15.5</fuelCapacity>
  </manufacturer>
</autos>
```

图1 旧的汽车供应数据片断

XML中的元素可以含有属性,并且属性在XML文档变化检测中具有很重要的作用。在传统的DOM树中,对每个文档中的元素都用一个结点来表示,而元素的属性仅仅表示成

陈振洲 博士生,主要研究领域为机器学习及数据挖掘;李 磊 教授,博士生导师,主要研究领域为数据库、知识库和数据挖掘。

结点的装饰。为了更好地考虑属性的作用，我们在树中为每个属性增加一个额外的结点[12,13]。

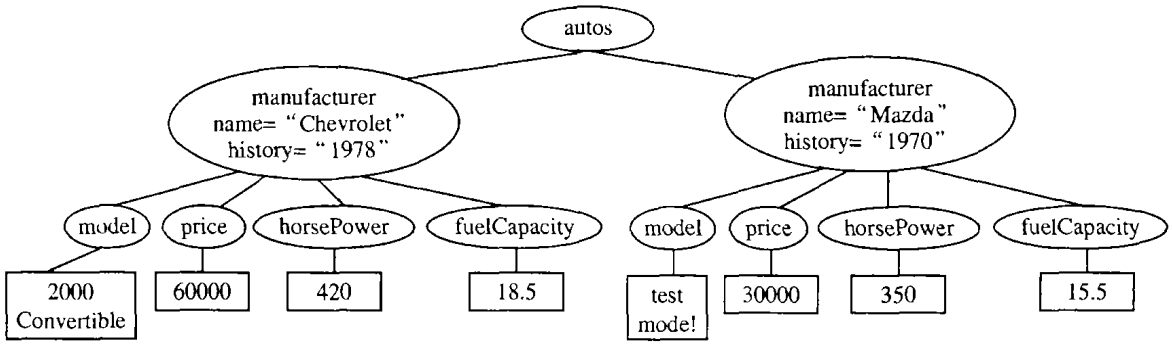


图2 对应图1文档的 DOM 树

总之，我们将每个 XML 文档表示成一棵带标号的树，文档中的每个元素和每个属性对应树中的一个结点(图2)，树中有3种类型的结点：元素结点，是带有一个标号(名字)的非叶子结点，用椭圆表示。文本结点，是带有一个标号(值)的叶结点，用实线框表示。属性结点，是叶结点，它包含一个标号(名字+值)，用虚线框表示。

根据 DOM 规范，元素结点和文本结点是有序的，但是实际应用中无序树比有序树更有意义。在本文中我们将扩展树看成一棵无序树，我们称这种树为 E-DOM 树。

2.2 树的编辑操作

在这节中，我们为 E-DOM 树定义3种基本的操作：

定义2.1(Modify(简称 M)操作) Modify(x, new_name)表示将结点 x 的标号改为 new_name，该操作只能在同一层次进行。

定义2.2(Delete(简称 D)操作) Delete(x)表示删除结点 x，x 必须是叶结点。

定义2.3(Insert(简称 I)操作) Insert(x(name), y)表示将结点 x 作为结点 y 的儿子插入，x 必须是叶结点，该操作只能在 x 的上一层次进行。

这里定义的这些基本操作和文[12]中的定义类似，但是存在以下区别：①M 操作可以针对任何结点，而不是仅仅针对叶结点。②I 操作不必指定位置，因为本文要处理的是无序树。③我们没有提供删除子树和插入子树的操作，因为对应子树的删除和插入操作可以转化为一系列的针对结点的删除和插入操作。

2.3 编辑脚本和编辑距离

一个编辑脚本就是能够将一棵树转换成另外一棵树的一系列编辑操作[3]。设 e_1 是一个操作，T_i 是一棵树，则 $e_1(T_1)=T_2$ 表示 e_1 可以将 T_1 转换为 T_2。设 E 是一个编辑脚本(编辑操作序列)，$E=e_1,\cdots,e_m$，则 $E(T_1)=T_{m+1}$ 表示可以将 T_1 转换为 T_{m+1}，且 $e_1(T_1)=T_2,\cdots,e_m(T_m)=T_{m+1}$。

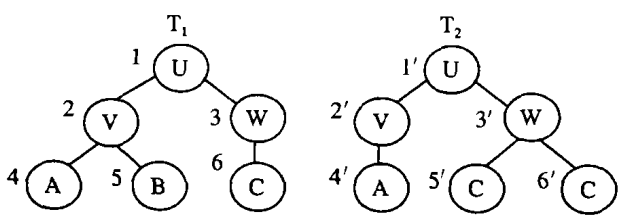


图3 编辑脚本的例子

设 $E_{T_i\to T_j}$ 表示 T_i 到 T_j 一个可能的编辑脚本，我们考虑图3中的两棵树 T_1 和 T_2，则一个可能的编辑脚本为：$E_{T_1\to T_2}$=Delete(5), Insert(5'(D), 3)，对于结点5'的插入操作只能是将它作为 V 或者 W 儿子。

定义2.4(编辑距离) 给定一个编辑脚本 $E=e_1\cdots,e_n$，且 e_i 是上面定义的 M、D 和三种基本操作之一，则编辑距离 $Cost(E)=n$

定义2.5(最小编辑距离) 设 E 是 $T_1\to T_2$ 的一个编辑脚本，若对任意的 $T_1\to T_2$ 的编辑脚本 E'，都有 $Cost(E')\geqslant Cost(E)$，则称 E 为 $T_1\to T_2$ 的最小编辑脚本，$Cost(E)$ 为 $T_1\to T_2$ 的最小编辑距离。

定义2.6(树的距离) 设 E 为 $T_1\to T_2$ 的最小编辑脚本，称 $Dist(T_1,T_2)=Cost(E)$ 为 T_1 和 T_2 之间的距离。

3 DL-Diff 的设计和实现

算法的基本思想是将树看成是层次可分的结构(如图3，T_1 有三层，$((U),(V,W),(A,B,C))$，T_2 也是有三层$((U),(V,W),(A,D,C))$)，对于树的基本操作是有层次限制的，这样可以一定程度减少搜索空间，从而提高算法的效率。

3.1 算法的基本过程

给定两个 XML 文档 DOC_1 和 DOC_2，T_1 和 T_2 是它们的树描述。DL-Diff 检测 DOC_1 和 DOC_2 是否相同，如果不同，DL-Diff 找出 $T_1\to T_2$ 的具有最小编辑距离的编辑脚本。

```
Iuput:(DOC1, DOC2)
  /* Parsing and preprocessing */
  ①. parse DOC1 to T1 and dilaminate T1;
     parse DOC2 to T2 and delaminate T2;
  /* Using DL to solve the problem */
  ②. If(T1=T2)
        DOC1 is equivalent with DOC2, stop.
     Else
        Compare the nodes layer by layer and find out a mini-
        mum-cost edit script;
```

图4 DL-Diff 算法的基本过程

如图4所示，整个算法包括几个步骤：

①*解析和预处理* 算法首先解析 DOC_1 和 DOC_2，将它们表示成 T_1 和 T_2，并将结点进行分层。在解析过程中，要对结点的属性做一些特殊处理。

②*分层算法 DL 求解* 在这个阶段，算法首先判断 T_1 是否等于 T_2，如果不相等，则利用 DL 求解，求出树的距离 $Dist(T_1,T_2)$ 和最小代价编辑脚本。

3.2 解析和预处理

在 DL-Diff 的这一步中，两个 XML 文档 DOC_1 和 DOC_2 首先被解析成 T_1 和 T_2(包括对中的带有属性的结点进行处理)，然后进行分层处理，并将处理结果保存起来。在处理过程中，我们要对每个结点求 Hash 值，一个结点的 Hash 值由结点本身的值和其孩子结点的 Hash 值决定。

3.3 算法求解

在这一步中，我们首先判断树 T_1 和 T_2 是否相等(Hash(T_1)是否等于 Hash(T_2))，如果不相等，则说明树 T_1 和 T_2 不相等，我们利用算法 DL 对问题进行求解。

如图5，要找到 T_1 到 T_2 的最小编辑距离(距离)，需要以下步骤：

①首先我们要过滤掉 T_1 和 T_2 第二层中相同的子树(可以通过比较 Hash 值来判断是否相等)，这样可以减少搜索空间，避免一些不必要的操作。

②其次，我们对 T_1 和 T_2 剩下的子树两两计算其距离，最后我们可以求出最小编辑离(树的距离 $Dist(T_1,T_2)$)。在计算子树的距离的时候我们可以利用第一步的方法。

在分层(DL)算法求解过程中，我们利用动态规划来计算 $Dist(T_1,T_2)$。先从倒数第一层开始，对 T_1 和 T_2 中叶结点对求最小编辑距离；然后对 T_1 和 T_2 中倒数第二层中的结点对求最小编辑距离；以此类推计算，最终可以求出 T_1 和 T_2 的最小编辑距离(树的距离 $Dist(T_1,T_2)$)。

```
Input:(T1,T2) //T1 and T2 have been delaminated
Output:a minimum-cost edit script and a series of operations
Initialize:set initial working sets
N1={all the nodes of the last layer of T1};
N2={all the nodes of the last layer of T2};
Set the Distance Table DT={};
Set the Operation Table OT={};
/* Step 1:Reduce matching space */
Filter out next-level subtrees that have equal Hash values.
/* Step 2:compute editing distance for(T1→T2) */
DO {
    For every node x in N1
      For every node y in N2
        {
          Compute Dist(x,y);
          Save Dist(x,y) in DT and Oper(x,y) in OT;
        }
  N1={parent nodes of previous nodes in N1};
  N2={parent nodes of previous nodes in N2}.
} While (both N1 and N2 are not empty).
```

图5 分层 DL 算法的基本过程

3.4 产生最小代价编辑脚本

在这一步中，要根据上一步 DL 算法的结果，产生 $T_1 \to T_2$ 的最小代价编辑脚本。DL 算法会产生 DT 和 OT 两个结果集，而我们根据这两个结果就可以求出 $T_1 \to T_2$ 的最小编辑距离($Dist(T_1,T_2)$)和最小编辑脚本(动作序列)。

3.5 算法分析

首先设 $|T_1|$ 和 $|T_2|$ 表示树和结点数目。

1. 在解析和预处理阶段，对树 T_1 和 T_2 的解析处理时间为 $O(|T_1|+|T_2|)$；在解析的时候我们要计算结点 Hash 的值，因为结点的 Hash 值是由结点本身的值和其孩子结点的 Hash 值决定，所以这一步复杂度的上限为 $O(|T_1|\times\log(|T_1|)+|T_2|\times\log(|T_2|))$。

2. 在 DL 算法求解阶段，首先我们考虑最后一层，若计算两个结点最小编辑距离的代价为 $O(1)$，则对所有最后一层结点匹配的计算代价上限为 $O(|T_1|\times|T_2|)$。其次我们考虑倒数第二层，计算这一层任意结点对 $(x,y)(x\in T_1, y\in T_2)$ 之间的最小编辑距离，也就是要计算它们孩子结点的最小编辑距离和它们本身的最小编辑距离。设 $\deg(x)$ 代表 x 的出度，则计算 $Dist(x,y)$ 的代价上限为：

$$O(\deg(x)\times\deg(y)\times\max(\deg(x),\deg(y))\times\log_2(\max(\deg(x),\deg(y))$$ [14]

设 T_1 和 T_2 的最大层数为 L，M_i 表示 T_1 第 i 层的结点数目，N_j 表示 T_2 第 j 层的结点数目，则计算所有 $1,\cdots,L-1$ 层的结点对的代价上限为：

$$\sum_{l=1}^{L-1}\sum_{p=1}^{M_l}\sum_{q=1}^{N_l}O(\deg(x_{lp})\times\deg(y_{lq})\times\max(\deg(x_{lq}),\deg(y_{lq}))\times\log_2(\max(\deg(x_{lp}),\deg(y_{lq})))$$

(x_{lp} 为 T_1 中的 l 层的第个 p 结点，y_{lq} 为 T_2 中的 l 层的第个 q 结点)，设 $\deg(T_i)$ 为树 T_i 的最大出度，则上式：

$$\leqslant O(|T_1|\times|T_2|\times\max(\deg(T_1),\deg(T_2))\times\log_2(\max(\deg(T_1),\deg(T_2)))$$

因为 $\sum_{l=1}^{L-1}\sum_{p=1}^{M_l}\deg(x_{lp})<|T_1|$，$\sum_{l=1}^{L-1}\sum_{q=1}^{N_l}\deg(y_{lq})<|T_2|$

所以整个这一步算法复杂度的上限为：

$$O(|T_1|\times|T_2|\times\max(\deg(T_1),\deg(T_2))\times\log_2(\max(\deg(T_1),\deg(T_2)))$$

4 算法性能测试

4.1 系统配置

我们利用 VC++来执行 DL-Diff，与 X-Diff，GA-Diff 算法一样，程序使用的分析器为(XercesC++V1.4)。X-Diff 原来是 Linux 平台上的程序，为了方便对比，将它移植到 Windows 平台。试验的硬件配置为：Celeron600，128MB 内存，Win2000 操作系统。

4.2 结果分析

在下面的图形中，给出了算法的结果分析，这里我们主要和 X-Diff 算法以及 GA-Diff 算法比较。

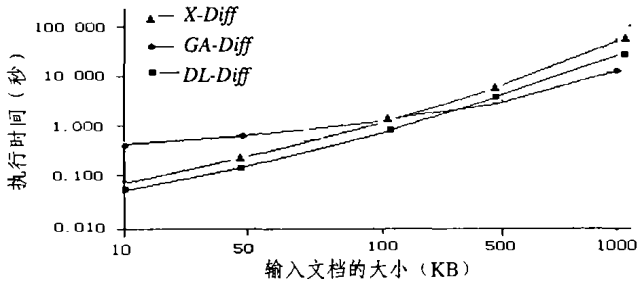


图6 变化率为1%时的执行时间

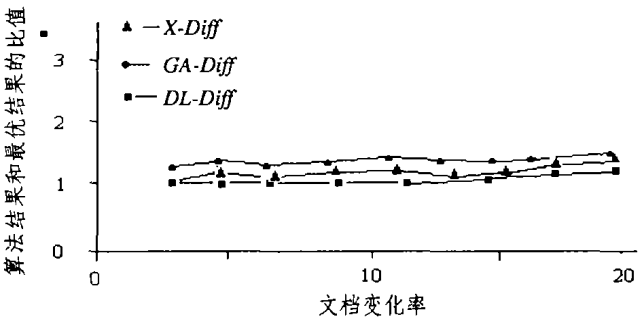


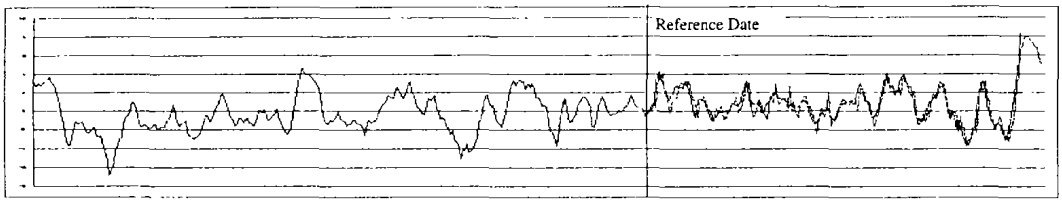
图7 检测结果的质量

图6表示因为两种算法执行时间的比较，图7表示两种算法所得的结果比较。

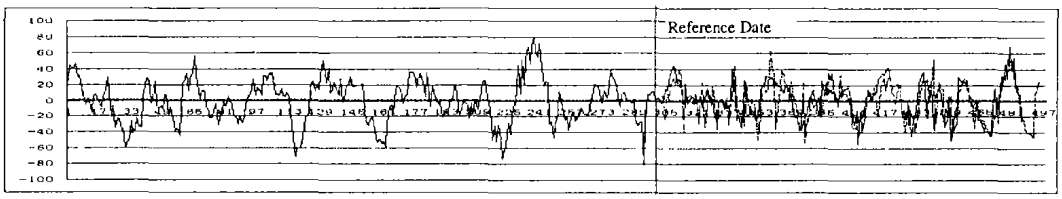
结束语 由上面的结果我们可以看出，与 X-Diff 比较，DL-Diff 的效率一直比它高，因为在对每一层进行计算的时候都尽量减少搜索空间；与 GA-Diff 比较，DL-Diff 算法的效率对于相对比较小的 XML 文档有比较好的效果，而对于比较大的 XMl 文档，GA-Diff 的效率高一些。所以，我们在实际应用中要结合使用这两种检测算法。

(下转第106页)

建立时序预测的CBR模型。我们统一描述时序相似性模型，定义了数据表示和抽象、具体范例，并给出了构造时序范例的一般过程，定义了范例多层概化。最后我们在期货预测中应用了这个思想，取得了较好的效果。



月线 ΔTm 预测结果



日线 ΔTd 预测结果

图4 LME copper 03的预测结果

参 考 文 献

1 Harvey C. Andrew: Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge, 1989

2 Bergmann R, Wilke W. On the Role of Abstraction in Case-Based Reasoning. In EWCBR 1996. 28～43

3 Berndt, Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In: KDD Workshop, 1994. 359～370

4 Holt C C. Forecasting Seasonal and Trends by Exponentially Weighted Moving Averages. Carnegie Institute of Technology, Pittsburgh, Pennsylvania, 1957

5 Cole L, et al. Representing Cases for CBR in XML. In: UKCBR Workshop, Dec. 2000

6 von Constantin A. Fuzzy Logic and Neuro Fuzzy Applications Explained. Prentice Hall. Englewood Cliffs, 1995

7 Das G, et al. Finding Similar Time Series. In: Proc. of Principles of Data Mining and Knowledge Discovery, 1st European Symposium. Trondheim, Norway, Jun. 88～100

8 Faloutsos C, Ranganathan M, Manolopoulos Y. Fast Subsequence Matching in Time-Series Database. In: Proc. 1994 ACM SIGMOD Conf. Minneapolis, 1994

9 Finn J V. An Introduction to Bayesian Networks. UCL Press, London, 1996

10 Ge, Smyth. Deformable Markov model templates for time-series pattern matching. KDD 2000. 81～90

11 Hayes C, Cunningham P. Shaping a CBR view with XML. In ICCBR 1999. 468～481

12 Hetland M L. A Survey of Recent Methods for Efficient Retrieval of Similar Time Sequences. In: Mark Last, Abraham Kandel, Horst Bunke, eds. Data Mining in Time Series Databases. World Scientific, 2003

13 Rumellhart, Hinton, Williams. Learning representations by back-propagating errors. Nature, 1986. 323

14 Holger K, Schreiber T. Nonlinear Time Series Analysis. Cambridge University Press, Cambridge, 2000

15 Jaczynski M. A Framework for the Management of Past Experiences with Time-Extended Situations. In: 6th ACM Conf. on Information and Knowledge Management (CIKM'97), Las Vegas, Nov. 1997

16 Jagadish, et al. Similarity-Based Queries. PODS 1995. 36～45

17 Lefteri T H, Robert U E. Fuzzy and Neural Approaches in Engineering. John Wiley, New York, 1997

18 Lotfi Z A, et al. Fuzzy Sets and their Applications to Cognitive and Decision Processes. Academic Press, New York, 1975

19 Keogh, Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. in ACM 2000. 1～58

20 Goldin, Kanellakis. On Similarity Queries for Time-Series Data: Constraint Specification and Implementation. CP 1995. 137～153

21 Leake D B. CBR in Context: The Present and Future. AAAI Press/MIT Press, 1996

22 George B E P, Jenkins G M. Time Series Analysis: Forecasting and Control. Holden Day, San Francisco, 1970

23 Pecar B. Case-based Algorighm for Pattern Recognition and Extrapolation. In: 22nd SGAI Int'l Conf. on Knowledge Based Systems and Applied Artificial Intelligence, Cambridge, Dec. 2002

24 Perng, et al. Landmarks: a New Model for Similarity-based Pattern Querying in Time Series Databases. ICDE 2000. 33～42

25 Kalman R E. A new Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering, D. 1960, 82

26 Rafiei, Mendelson. Querying Time Series Data Based on Similarity. In IEEE TRANS. ON KNOWLEDGE AND DATA ENG., 2000, 12(5)

27 Robert B. Statistical Forecasting for Inventory Control, McGraw Hill Book Co., New York, 1958

28 Sacerdoti E D. Planning in a hierarchy of abstraction space. Artificial Intelligence, 1974, 5: 115～135

29 Smyth B, Keane M T, Cunningham P. Hierarchical Case-Based Reasoning Integrating Case-Based and Decompositional Problem-Solving Techniques for Plant-Control Software Design. TKDE, 2000, 13(5): 793～812

30 Wijsen J. Trends in Databases: Reasoning and Mining. IEEE TRANS. ON KNOWLEDGE AND DATA ENG., 2001, 13(3)

31 Goldberg D E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989

(上接第70页)

参 考 文 献

1 Extensible Markup Language (XML). World Wide Web Consortium. http://www.w3.org/XML/

2 Concurrent Versions System (CVS). Free Software Foundation. http://www.gnu.org/manual/cvs-1.9

3 Chawathe S, et al. Change Detection in Hierarchically Structured Information. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data. Montreal, June 1996

4 Douglis F, Ball T, Chen Y F, Koutsofios E. The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web. World Wide Web, 1998, 1(1): 27～44

5 Berk E. HtmlDiff: A Differencing Tool for HTML Documents. Student Project, Princeton University. http://www.htmldiff.com

6 Barnard D T. Clarke G, Duncan N. Tree-to-tree Correction for Document Trees: [Technical Report 95-372]. 1995

7 Isert C. The Editing Distance Between Trees. 1999

8 Curbera D, Epstein A. Fast Difference and Update of XML Documents. XTech'99, San Jose, March 1999

9 Maruyama H, Tamura K, Uramoto R. Digest values for DOM (DOMHash) proposal. IBM Tokyo Research Laboratory. http://www.trl.ibm.co.jp/projects/xml/domhash.htm, 1998

10 Zhang, Shaha D. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. SIAM Journal of Computing, 1989, 18(6): 1245～1262

11 Cobéna G, Abiteboul S, Marian A. Detecting Changes in XML Documents. In: The 18th Intl. Conf. on Data Engineering, San Jose, Feb. 2002

12 Wang Y, et al. X-Diff: An Effective Change Detection Algorithm for XML Documents, 2002

13 Nierman A, Jagadish H V. Evaluating Structural Similarity in XML Documents, 2002

14 Zhang K. A New Editing based Distance between Unordered Labeled Trees. Combinatorial Pattern Matching, 1993, 1: 254～265