文章编号：0253-374X(2013)10-1589-07

DOI:10.3969/j.issn.0253-374x.2013.10.024

不同时间戳的地图差异匹配改进算法

杜庆峰，赵亚男

（同济大学 软件学院，上海 201804）

摘要：在分析了现有的可伸缩矢量图形(SVG)格式地图差异算法的研究现状和不足的基础上提出了一种改进的 SVG 格式地图差异匹配算法 I-DiffS(improved difference of SVG maps). 该改进算法定义了节点集元素，即节点集元素可能包含 1 个或多个元素节点、属性节点和值节点构成的一个路径节点集合，定义了 SVG 格式解析结构树的标号规则，减少了结构树对应数组的元素个数，也减少了差异脚本中操作类型的数目，缩短了匹配过程. 匹配结果为差异脚本，该脚本记录了前一个时间戳到后一个时间戳的更新操作. I-DiffS 算法相比于现有的最新 DiffS 算法，时间复杂度更低. 应用验证证明了该算法有效.

关键词：可伸缩矢量图形(SVG)；时间戳；地图；差异匹配算法；标号规则

中图分类号：TP311.5　　**文献标志码**：A

Improved Matching Algorithm to Different Time Stamp Maps

DU Qingfeng, ZHAO Ya' nan

(College of Software Engineering, Tongji University, Shanghai 201804, China)

Abstract: Based on an analysis of the existing research about differences matching algorithms based SVG maps and their insufficiencies, this paper proposes an improved differences matching algorithm improved different of SVG maps(I-DiffS). The algorithm defines a node set element, which is a set of path node composed by one or several element node, attribute node and value node. It also defines a labeling rule of structured trees based scalable vector graphics(SVG). Both reduce the number of elements in the arrays related to structured trees and the sum of types of operations in an generated difference script are reduced, which shortens the process of matching. The result of matching is a generated difference script, which records an update operation from the first time stamp map to the second one. Compared with the existing latest DiffS algorithm, this algorithm has lower time complexity. Application validates that DiffS algorithm is effective.

Key words: scalable vector graphics(SVG); time stamp; map; differences matching algorithms; labeling rule

可升伸缩矢量图形(SVG)是一种使用 XML 来描述二维图形及其应用的语言. 一个 SVG 文档可以被解析成一棵倒立的树形结构，与 XML 文档的差异匹配算法类似. XML 文档的差异匹配算法可以分为 3 类：第 1 类是针对有序节点树的差异匹配算法，第 2 类是针对无序节点树的差异匹配算法，第 3 类算法不是将 XML 结构树中所有节点看作有序节点或者全部看作无序节点.

针对有序节点树，Cobena 等[1]提出了一种检测 XML 文档变化算法 XyDiff. 该算法首先采用自底向上的方式计算每个结点的签名(hash 值)和权重(子树的大小)，然后从 2 个文档的根结点开始比较 2 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匹配检测. 该算法的时间复杂度为 $O(nh)$，其中 n 为节点个数，h 为树的高度.

针对无序节点树，Wang 等[2]提出了 X-Diff 算法，该算法采用自顶向下的匹配机制，对 XML 结构树进行“最小修改代价”搜索匹配，该算法是针对无序节点树的匹配检测最快的算法. 该算法的时间复杂度为 $O(n^2 d \lg d)$，其中 d 为最大深度.

第 3 种算法是由 Al-Ekram 等[3]提出的 DiffX 算法，该算法根据节点的不同类型将其分为元素节点、属性节点和文本节点，DiffX 算法将元素节点看成有序节点而将属性节点看成无序节点，这样的数

收稿日期：2012-11-10
基金项目：国家自然科学基金(41171303)
第一作者：杜庆峰(1968—)，男，教授，工学博士，主要研究方向为地理信息系统、软件工程、软件质量管理与软件测试. E-mail：du_cloud@sohu.com
通讯作者：赵亚男(1989—)，女，工学硕士，主要研究方向为地理信息系统. E-mail：yanan_zhao@163.com

据模型更符合 XML 文档的性质. 该算法的时间复杂度为 $O(n^2)$.

现有的 DiffS[4]算法是在 DiffX 算法的基础上定义宏元素节点,以元素节点或宏元素节点作为最小匹配元素,创建 2 个以{id: String, et: String, e: Element}为元素的数组 E1,E2. 该算法的时间复杂度降低至 $O(n \lg n)$.

在现有的 DiffS 算法中,数组 E1,E2 中的元素个数太多,导致数组排序和查找的时间太长. 另外,对于数组中的每个元素节点,它们的属性字符串太长,导致属性排序的时间也太长.

考虑到现有研究存在的不足,可以重新定义宏元素节点,提出一种新的元素节点:节点集元素,达到减少数组元素个数、降低数组排序和查找时间的目的. 另外,通过一种标号定义规则来重新标识 SVG 结构树,使得属性字符串的长度大大缩短,达到减少属性排序时间的目的.

1 改进算法 I-DiffS 的思想及实现

1.1 I-DiffS 算法思想

1.1.1 相关定义

(1)节点集元素. 在某时间戳的 SVG 格式地图对应的倒状解析[5]结构树中,从根节点开始将某特定的分支中的元素节点、属性节点和值节点看成一个整体,称为节点集元素. 如某 id 为 c2 的节点集元素,它的元素节点集合为{I,J,J,C},属性节点为"SY",值节点为"rgb(0,255,255)60".

(2)倒状解析结构树标号规则. 在 SVG 格式地图中主要包括基本图形元素集{line, rect, circle, eclipse, polyine, polygon, path, text},常见框架元素集{SVG, g, defs, use, symbol, desc, title, image},常用属性元素集{x, y, r, rx, ry, width, height, stroke/style, fill, tspan}. 共有 26 个主要基本元素,可以用 1 位大写字母标识,在这 26 个主要基本元素之外的元素可用 1 位小写字母标识,如果 1 位字母标识完毕,可采用 2 位或多位字母标识,以此类推.

按照倒装解析结构树的标号定义规则,主要基本元素的标号定义如表 1.

1.1.2 算法思路和实现

(1)优化后的解析结构树. 将原 SVG 文件通过节点集元素的定义来得到一棵 SVG[6]节点集元素结构树,之后再根据倒装解析树的标号定义规则将 SVG 节点集元素结构树转化成优化后的标号节点集元素结构树.

表 1 SVG 格式地图主要基本元素的标号定义

Tab.1 Labeling rule of major basic labeling rule of major basic element on SVG map

元素	标号	元素	标号	元素	标号
line	A	g	J	fill	S
rect	B	defs	K	rx	T
circle	C	use	L	width	V
eclipse	D	symbol	M	ry	U
polyine	E	desc	N	height	W
polygon	F	title	O	stroke/style	X
path	G	image	P	r	Y
text	H	x	Q	tspan	Z
SVG	I	y	R		

(2)生成匹配节点集元素集合 M. 首先对 SVG[7]格式地图文档对应的结构树中的属性元素节点(除 id 属性外)按字典顺序排列,该属性元素节点对应的值节点按照排序好的属性元素节点顺序排列. 如,一个属性元素节点排序前为"SWV",排序后为"SVW",则值元素节点为"none100100".

接着创建 2 个以节点集元素为元素的数组 E1, E2; E1 中数组个数等于第 1 个版本 SVG 格式地图[8]对应的树结构中节点集元素的个数 m_1,同理 E2 中数组个数等于第 2 个版本 SVG 格式地图对应的树结构中节点集元素的个数 m_2.

数组 E1 和 E2 中元素的属性由以下 6 部分构成:第 1 个字符串是节点集元素的 id 值;第 2 个字符串是节点集元素的元素节点所组成的字符串 path;第 3 个字符串是节点集元素的属性节点所组成的字符串 et;第 4 个字符串是节点集元素的值节点所组成的字符串 value;第 5 个指针用来记录所对应的节点集元素的指针 e;第 6 个是用来记录所对应节点集元素的头指针 h,指向一个由元素节点指针所组成的链表;第 7 个是一个数组 flag,用来记录节点集元素的元素节点使用次数,数组 flag 中的元素初始值为零.

如某节点集元素在数组 E1 中的表示为:该节点集元素的 id 值为 c2,元素节点所组成的字符串 path 为"IJJC",属性节点所组成的字符串 et 为"SY",值节点所组成的字符串 value 为"rgb(0,255,255)60",该节点集元素的指针 e 为 A11,指向元素节点指针所组成的链表的头指针 h 为→A1→A4→A8→A11,用来记录各元素节点使用次数的数组 flag 的长度为 4,其中 flag[A1]的值为 9, flag[A4]的值为 2, flag[A8]的值为 2, flag[A11]的值 1.

对数组 E2 中数组元素依据 id 值进行排序,这

样无 id 值的数组元素被全部排在数组元素的前面,接着对前面无 id 值的数组元素依据 path 值进行排序.

依次遍历 E1 数组元素,如果当前数组元素 s 对应的树结构的节点集元素已经在匹配节点集元素集合 M 当中(即 s. e∈Array(M. X)),则结束本次循环进入下一次循环,否则继续;当 s 的 id 不为空时根据 id 对数组 E2 进行折半查找得到数组元素 d,其中元素 d 和 s 的 id 值、path 值、et 值和 value 值均相等;当 s 的 id 为空时根据 path 折半查找得到元素,其中元素 d 和 s 的 path 值、et 值和 value 值均相等;同时遍历 d. h 和 s. h 所对应的 2 个链表,如果 2 个链表中对应的每一项都相同,则将(s. e, d. e)合并入匹配节点集元素集合 M. 该子算法的具体实现(伪代码)如下,其中序号为数值标号.

```
1   Procedure I-SVG-Match (SVG1,SVG2)
2     Input SVG1,SVG2: Tree
3     Output M: MatchSet
4   Begin
5     sort attribute nodes(except id attribute) in each
6   node set element of SVG1,SVG2
7     E1,E2: Array of { id: String, path: String, et:
8   String, value: String, e: Element, h: HeadNode,
9   flag: ArrayList<Integer> }
10    traverse SVG1 in depth
11      let e be the current unit
12      add {id value of e, path nodes of e, attribute
13  nodes of e, value nodes of e, e, h, flag} to E1
14    End traverse
15    traverse SVG2 in depth
16      let e be the current unit
17      add {id value of e, path nodes of e, attribute
18  nodes of e, value nodes of e,e,h, flag} to E2
19    End traverse
20    sort elements having id attribute of E2 according to
21  the first string 'id'
22    sort elements of E2 according to the second string
23  'path', which have no id attribute
24    traverse E1 in sequence
25      let s be the current unit
26      if (s. e, *) belong to M then
27        skip current unit
28      End if
29      if s. e has id attribute
30        find element d of E2 which d. id equals to s. id,
31  d. path equals to s. path,d. et equals to s. et,d. value
32  equals to s. value
33      else
34        find element d of E2 which d. path equals to s.
35  path,d. et equals to s. et,d. value equals to s. value
36      let h1 be s. h and h2 be d. h
37      while h1 and h2 is not equal to null
38        do
39        If h1. value is not equal to h2. value
40          go to the end of the program
41        h1=h1→next and h2=h2→next
42      M = M ∪ (s. e, d. e)
43    End traverse
44  End
```

(3)依据匹配节点集元素集合 M 生成差异脚本. 该算法是以子算法 I-SVG-Match 得到的匹配节点集元素集合 M 为基础,首先,对 SVG1 对应的结构树进行遍历,如果被遍历的节点集元素在 SVG2 中不能找到对应的节点集元素对在匹配节点集元素集合 M 中,则对于 SVG1 中被遍历的节点集元素 s,顺序遍历节点集元素头指针 s. h 所指向的链表,每当遍历到链表中一个指针对应的元素节点时,如果该元素节点的 flag 值为 1,则在 DiffXml 中加入 delete 操作,同时对 SVG1 的副本 SVG0 所对应的结构树执行 delete 操作,并对链表中的每个元素节点的 flag 值做减 1 操作;否则,不执行任何操作. 不断循环此过程,直到遍历到 null 停止. 其中对于与属性节点相邻的元素节点执行 delete 操作时,除了删除该元素节点之外,还要将与其相邻的属性节点和值节点也一并删除.

接着,对 SVG2 对应的结构树进行遍历,如果被遍历的节点集元素在 SVG1 没有对应的节点集元素构成的节点集元素对在匹配节点集元素集合 M 中,则对于 SVG2 中被遍历的节点集元素 d,顺序遍历节点集元素头指针 d. h 所指向的链表,每当遍历到链表中一个指针对应的元素节点时,如果该元素节点的 flag 值为零,则在 DiffXml 中加入 insert 操作,同时对 SVG1 的副本 SVG0 所对应的结构树执行 insert 操作,并对链表中的每个元素节点的 flag 值做加 1 操作. 不断循环此过程,直到遍历到 null 停止. 其中对于与属性节点相邻的元素节点执行 insert 操作时,除了添加该元素节点之外,还要将与其相邻的属性节点和值节点也一起添加进去.

算法执行后生成差异脚本文件 DiffXml 算法对 SVG0 的操作结果就是 SVG2 格式地图对应的结构树,这说明可通过对 SVG1 的结点树施加 DiffXml 脚本中的操作可以再现第 2 版本地图 SVG2. 该子算法的具体实现(伪代码)如下.

```
1   Procedure I-SVG-DiffScript(SVG1,SVG2,M)
2     Input SVG1,SVG2:Tree;M: MatchSet
3     Output DiffXml:XML
```

```
4   Begin
5     let SVG0 be a copy of SVG1
6     traverse SVG1
7       let s be the current node
8       If(s, *) not∈M then
9         let p be a node
10        p is equal to s. h
11        while p is not equal to null
12        do
13         flag[p. value] is equal to 1
14        add to Diffxml
15      "<delete XPath='XPath(p. value,SVG0)'/
    delete>"
16        do delete operation to SVG0
17        End if
18        flag[p. value] is equal to flag[p. value] minus
    one
19        p is equal to p. next
20      End if
21    End traverse
22    traverse SVG2
23      let d be the current unit
24      If ( * ,d) not belong to M then
25        let p be a node
26        p is equal to d. h
27        while p is not equal to null
28        do
29        if flag[p. value] is equal to 0
30        add to DiffXml
31      "<insert XPath='XPath(p. value,SVG2)'>p.
    value</insert>"
32        do insert operation to SVG0
33        End if
34        flag[p. value] is equal to flag[p. value] minus
    one
35        p is equal to p. next
36      End if
37    End traverse
38  End
```

1.2 I-DiffS 算法复杂度

1.2.1 时间复杂度

在算法 I-DiffS 中,假设 SVG1 中的元素个数为 n_1,节点集元素个数为 m_1;SVG2 中的元素个数为 n_2,节点集元素个数为 m_2. 并假设 SVG1 中每个节点集元素的最大属性个数为 t_a,SVG2 中每个节点集元素的最大属性个数为 t_b;SVG1 的最大深度为 d_1,SVG2 的最大深度为 d_2.

在 I-DiffS 的子算法 I-SVG-Match 中的第 5 行,对 SVG1 和 SVG2 中节点集元素的属性节点排序,时间复杂度近似为 $m_1O(t_a\lg t_a)+m_2O(t_b\lg t_b)$. 第 10～19 行,深度遍历 SVG1 和 SVG2,并将 SVG1 和 SVG2 中的节点集元素添加到 SVG1 对应的数组 E1 和 SVG2 中的数组 E2 中,时间复杂度为 $O(n_1)+O(n_2)$. 第 20～21 行,依据 id 值对数组 E2 进行排序,时间复杂度为 $O(m_2\lg m_2)$. 第 24～43 行,遍历 E1,对于 E1 中的每一个元素在 E2 中查找与之匹配的元素,由于 E2 已经排好序,每次查找时间复杂度为 $O(\lg m_2)$,循环的总的时间复杂度为 $O(m_1\lg m_2)$. 因此,子算法 I-SVG-Match 的总的时间复杂度为

$$m_1O(t_a\lg t_a)+m_2O(t_b\lg t_b)+O(n_1)+O(n_2)+O(m_2\lg m_2)+O(m_1\lg m_2)\approx O(n)$$

I-DiffS 的子算法 I-SVG-DiffScript 中,第 6～21 行,遍历 SVG1,如果被遍历的节点集元素在 SVG2 中不能找到对应的节点集元素对在匹配节点集元素集 M 中,再顺序遍历该节点集元素头指针所对应的链表,时间复杂度近似为 $O(m_1d_1)\approx O(m_1)$;第 22～37 行,遍历 SVG2,对于每一个在匹配节点集元素集合 M 中找不到配对的节点集元素,再顺序遍历该节点集元素头指针所对应的链表,时间复杂度近似为 $O(m_2d_2)\approx O(m_2)$. 因此子算法 I-SVG-DiffScript 的总的时间复杂度为 $O(m_1+m_2)$.

基于以上分析,算法 I-DiffS 的总的时间复杂度为 $O(n)+O(m_1+m_2)\approx O(n)$.

1.2.2 空间复杂度

数组 E1,E2 对应 SVG1 和 SVG2,所需的空间复杂度为 $O(m_1+m_2)$. SVG0 为 SVG1 对应的副本,所需空间为 $O(m_1)$. 最坏情况下,E1 中的所有元素能在 E2 中找到匹配节点集元素,因此得到的匹配节点集元素集合 M 所需的空间复杂度为 $O(m_1)$. 因此算法 I-DiffS 最坏情况下的空间复杂度为 $O(m_1+m_2)+O(m_1)+O(m_1)\approx O(n)$.

2 I-DiffS 算法的验证

2.1 算法验证 SVG 原文件

这里验证的是一幅 2 个不同时间戳的 SVG 格式的地图文件,文件名分别为 FirstTimestamp. SVG 和 SecondTimestamp. SVG.

FirstTimestamp. SVG 的具体内容如下.

```
1   <svg width="1000" height="400" >
2     <circle id=c1 r="100" fill=none>
    <rect id=r1 width="150" height="75" fill="
3   rgb(0,0,0)">
4     <g>
5       <path id=p1 fill="rgb(255,255,0)">
```

```
6       <g>
7         <circle id=c2 r="60" fill="rgb(0,255,
        255)">
8         <rect id=r2 width="189" height="52" fill
9       ="rgb(255,255,0)">
10      </g>
11    </g>
12    <g>
13      <eclipse id=e1 rx=101 ry=81 fill="rgb(192,
        192,255)">
14      <g>
15        <path id=p2 fill=none>


16      </g>
17    </g>
18    <rect id=r3 width="209" height="80" fill=
      none>
19  </svg>
```

SecondTimestamp. SVG 的具体内容如下.

```
1   <svg width="1000" height="400" >
2     <g>
3       <rect id=r1 width="150" height="75" fill="
        rgb(0,0,0)">
4     </g>
5     <g>
6       <path id=p1 fill="rgb(255,255,0)">
7       <g>
8         <circle id=c2 r="60" fill="rgb(0,255,
          255)">
9         <rect id=r2 width="189" height="52" fill
10      ="rgb(255,255,0)">
11      </g>
12    </g>
13    <g>
14      <eclipse id=e1 rx=101 ry=81 fill=none>
15      <g>
16        <path id=p2 fill=none>
17        <rect id=r4 width="209" height="80" fill
        =none>
18      </g>
19    </g>
20  </svg>
```

2.2　通过定义节点集元素得到的结构树

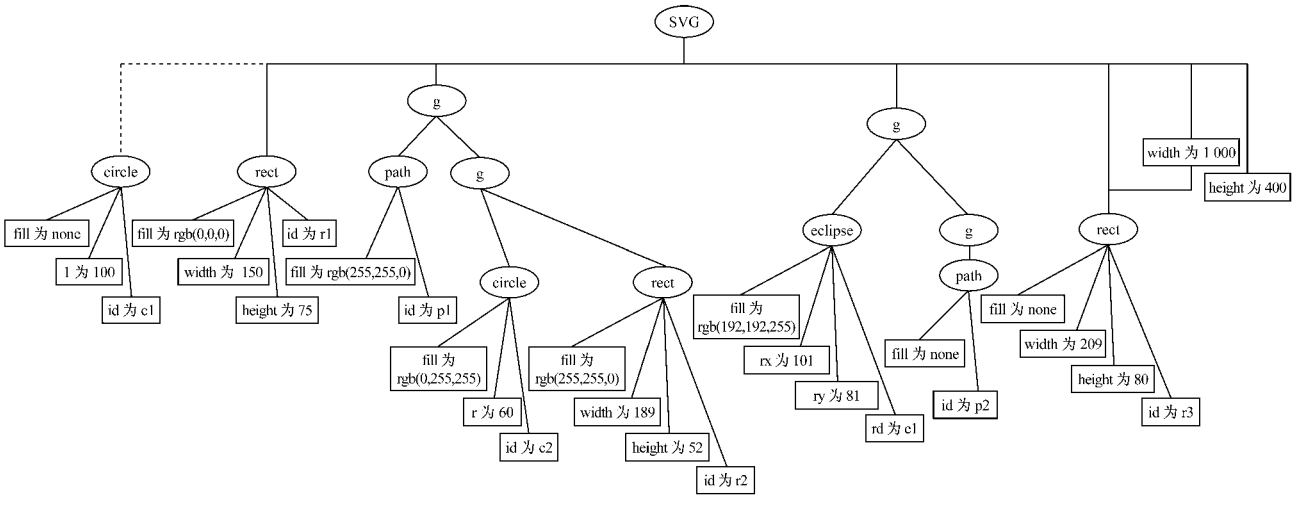
图 1 是文件 SVG1 和 SVG2 在定义了节点集元素之后得到的解析结构树，其中椭圆表示元素节点，矩形表示属性节点和其对应的值节点，虚线标出的是文件 SVG1 和 SVG2 存在的差异.

2.3　通过标号的定义规则得到的优化后的结构树

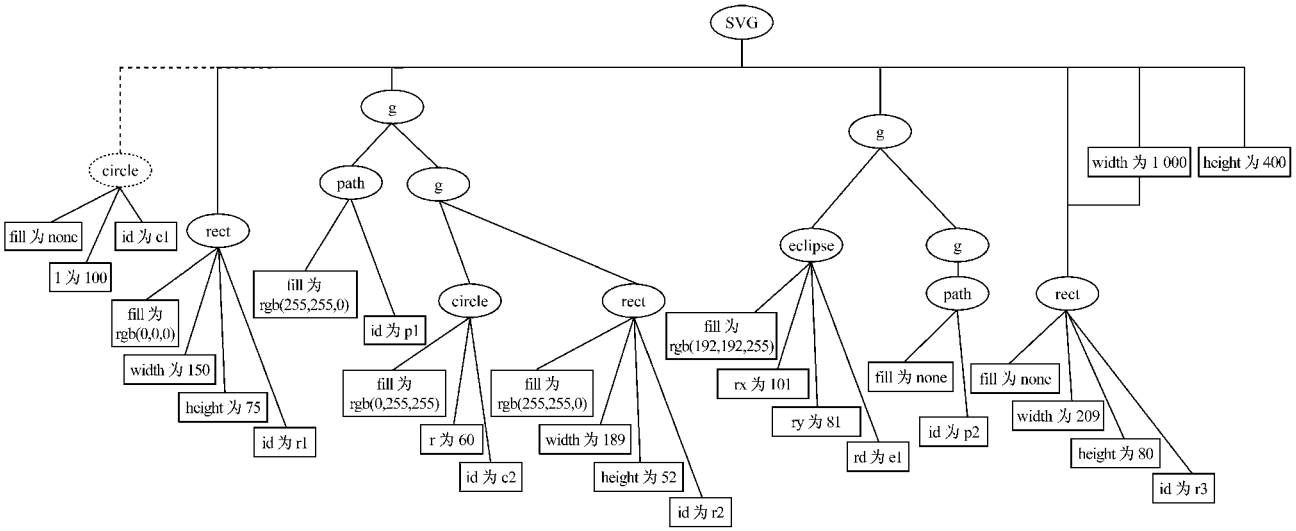
图 2 使用了倒装解析结构树标号定义规则对图 1 的解析结构树进行了优化，并将图 1 中属性节点和值节点组成的矩形进行了分离，使用圆形表示元素节点和属性节点，使用矩形表示值节点，每个与矩形相邻的圆形即为属性节点，其余圆形为元素节点. 其中属性元素节点中属性顺序按字典顺序排序，值节点中值的顺序按属性节点顺序对应排列. 每个元素节点旁边由 $A_n(n=1,2,\cdots,13)$ 和 $B_n(n=1,2,\cdots,13)$ 标出该元素节点的指针（如 A_1，A_2 等和 B_1，B_2 等），用虚线标出文件 SVG1 和 SVG2 的差异.

2.4　生成的节点集元素数组

表 2 和表 3 中的 E1 和 E2 是由优化后的 SVG1 和 SVG2 解析结构树生成的数组，数组的元素为节点集元素，每个元素由 7 种属性构成，分别是节点集元素 id、节点集元素的元素节点所组成的字符串 path、节点集元素的属性节点所组成的字符串 et、节点集元素的值节点所组成的字符串 value、节点集元素的指针 e、节点集元素的元素节点指针所构成的链表的头指针 h 以及记录了节点集元素的各元素节点使用次数的数组 flag. 其中数组 E2 是通过 id 值按字典顺序排序后生成的.



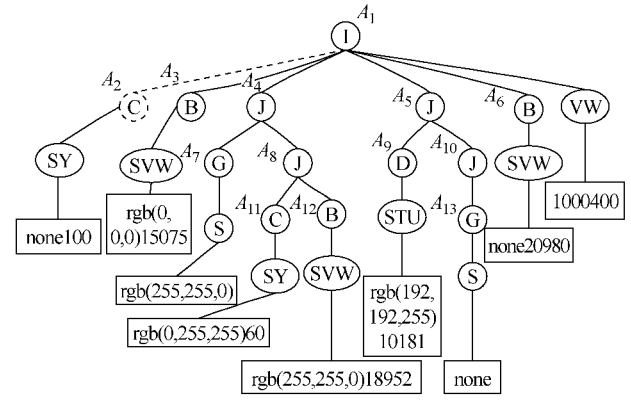
a SVG1



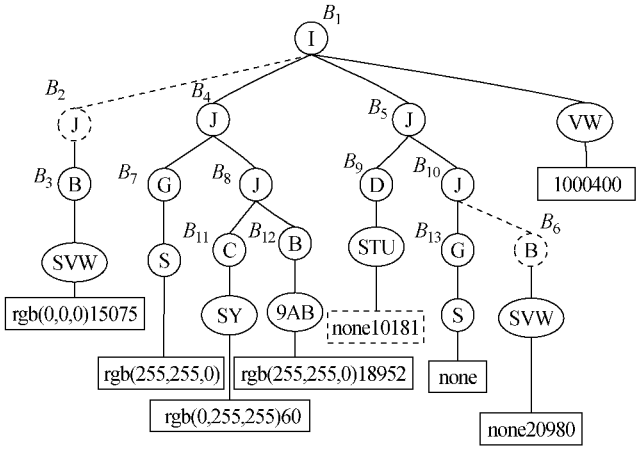
b SVG2

图 1 SVG1 和 SVG2 对应的解析结构树

Fig. 1 Structural tree corresponding to SVG1 and SVG2



a SVG1



b SVG2

图 2 基于 SVG1 和 SVG2 优化后的定义了节点集元素的结构树

Fig. 2 Structural tree corresponding to SVG1 and SVG2 including node set elements

表 2 依据节点集元素的属性节点排序后 SVG1 对应的结构树生成的数组 E1 和 E2

Tab. 2 Array E1 and E2 corresponding to SVG1 and SVG2 including sorted attribute nodes in node set elements

对象	id	path	et	Value	e
E1	无	I	VW	1000400	A_1
	c1	IC	SY	None100	A_2
	r1	IB	SVW	rgb(0,0,0)15075	A_3
	p1	IJG	S	rgb(255,255,0)	A_7
	c2	IJJC	SY	rgb(0,255,255)60	A_{11}
	r2	IJJB	SVW	rgb(255,255,0)18952	A_{12}
	e1	IJD	STU	rgb(192,192,255)10181	A_9
	p2	IJJG	S	None	A_{13}
	r3	IB	SVW	none20980	A_6
排序后的E2	无	I	VW	1000400	B_1
	c2	IJJC	SY	rgb(0,255,255)60	B_{11}
	e1	IJD	STU	none10181	B_9
	p1	IJG	S	rgb(255,255,0)	B_7
	p2	IJJG	S	None	B_{13}
	r1	IJB	SVW	rgb(0,0,0)15075	B_3
	r2	IJJB	SVW	rgb(255,255,0)18952	B_{12}
	r4	IJJB	SVW	none20980	B_6

2.5 生成的匹配节点集元素集合 M

I-SVG-Match 执行匹配过程：当遍历到 $E_1[0]$ 时，$M=\{(A_1,B_1)\}$；当遍历到 $E_1[1]$ 时，$M=\{(A_1,B_1)\}$；当遍历到 $E_1[2]$ 时，$M=\{(A_1,B_1)\}$；当遍历到 $E_1[3]$ 时，$M=\{(A_1,B_1),(A_7,B_7)\}$；当遍历到 $E_1[4]$ 时，$M=\{(A_1,B_1),(A_7,B_7),(A_{11},B_{11})\}$；当遍历到 $E_1[5]$ 时，$M=\{(A_1,B_1),(A_7,B_7),(A_{11},B_{11}),(A_{12},B_{12})\}$；当遍历到 $E_1[6]$ 时，$M=\{(A_1,B_1),(A_7,B_7),$

(A_{11},B_{11})，(A_{12},B_{12})｝；当遍历到 E_1[7]时，M＝｛(A_1,B_1)，(A_7,B_7)，(A_{11},B_{11})，(A_{12},B_{12})，(A_{13},B_{13})｝；当遍历到 $E1$[8]时，M＝｛(A_1,B_1)，(A_7,B_7)，(A_{11},B_{11})，(A_{12},B_{12})，(A_{13},B_{13})｝. 以上是执行子算法 I-SVG-Match 第 24～43 行的循环过程，最终得到匹配节点集元素集合 M.

表 3　依据节点集元素的属性节点排序后 SVG2 对应的结构树生成的数组 E1 和 E2

Tab. 3　Array E1 and E2 corresponding to SVG1 and SVG2 including sorted attribute nodes in node set elements

对象	h	flag
E1	→A_1	flag[1]＝{9}
	→A_1→A_2	flag[2]＝{9,1}
	→A_1→A_3	flag[2]＝{9,1}
	→A_1→A_4→A_7	flag[3]＝{9,3,1}
	→A_1→A_4→A_8→A_11	flag[4]＝{9,3,2,1}
	→A_1→A_4→A_8→A_{12}	flag[4]＝{9,3,2,1}
	→A_1→A_5→A_9	flag[3]＝{9,2,1}
	→A_1→A_5→A_10→A_{13}	flag[4]＝{9,2,1,1}
	→A_1→A_6	flag[2]＝{9,1}
排序后的 E2	→B_1	flag[1]＝{8}
	→B_1→B_4→B_8→B_11	flag[4]＝{8,3,2,1}
	→B_1→B_5→B_9	flag[3]＝{8,3,1}
	→B_1→B_4→B_7	flag[3]＝{8,3,1}
	→B_1→B_5→B_{10}→B_{13}	flag[4]＝{8,3,2,1}
	→B_1→B_2→B_3	flag[3]＝{8,1,1}
	→B_1→B_4→B_8→B_{12}	flag[4]＝{8,3,2,1}
	→B_1→B_5→B_{10}→B_6	flag[4]＝{8,3,2,1}

2.6　生成的差异脚本 I-DiffXml

子算法 I-SVG-DiffScript 依据匹配节点集元素集合 M 对 SVG1 对应的结构树进行遍历，其中节点集元素 A_2，A_3，A_9，A_6 不在集合 M 中，则首先访问 A_2.h 指向的元素节点指针链表，由于 flag[A_2]的值为 1，则在 I-DiffXml 脚本中加入了对 SVG1 中的 A_2 执行 delete 的操作，并对 flag[A_1]和 flag[A_2]做减 1 操作，flag[A_1]的值为 8，flag[A_2]的值为零；接着顺次访问 A_3.h，A_9.h 和 A_6.h 指向的元素节点指针链表，执行的操作步骤同理；此时在 I-DiffXml 中已加入了对 SVG1 中的 A_2，A_3，A_9，A_6 执行 delete 的操作，即在 SVG1 对应的结构树中删除了 A_2，A_3，A_9 和 A_6，并且 flag[A_1]的值为 6，flag[A_2]的值为零，flag[A_3]的值为零，flag[A_5]的值为 1，flag[A_9]的值为零，flag[A_6]的值为零.

接着，对 SVG2 对应的结构树进行遍历，其中节点集元素 B_3，B_9，B_6 不在集合 M 中，则首先访问 B_3.h 指向的元素节点指针链表，由于 flag[B_2]初值为零，则在 I-DiffXml 脚本中加入了 SVG2 中的 B_2，由于 flag[B_3]初值为零，则在 I-DiffXml 脚本中加入了 SVG2 中的 B_3；接着顺次访问 B_9.h 和 B_6.h 指向的元素节点指针链表，执行的操作步骤同理，即在 I-DiffXml 脚本中加入了 SVG2 中的 B_2，B_3，B_9 和 B_6. I-DiffXml 差异脚本的具体实现如下.

＜delete XPath＝'XPath(A_2,SVG1)'/delete＞
＜delete XPath＝'XPath(A_3,SVG1)'/delete＞
＜delete XPath＝'XPath(A_9,SVG1)'/delete＞
＜delete XPath＝'XPath(A_6,SVG1)'/delete＞
＜insert XPath＝'XPath(B_2,SVG2)'＞B_2＜/insert＞
＜insert XPath＝'XPath(B_3,SVG2)'＞B_3＜/insert＞
＜insert XPath＝'XPath(B_9,SVG2)'＞B_9＜/insert＞
＜insert XPath＝'XPath(B_6,SVG2)'＞B_6＜/insert＞

3　结论

在现有的 SVG(XML)差异算法的基础上着重分析了目前最新的 DiffS 算法，并提出了一种改进算法 I-DiffS. 该算法主要对以下几方面进行了改进：①定义了节点集元素，减少了结构树对应数组的元素个数，缩短了匹配过程；②使用了基于结构树的标号定义规则，使得原本的结构树得到了进一步的优化，减少了排序时间；③算法复杂度为 $O(n)$，低于现有的最优匹配算法 DiffS 的时间复杂度 $O(n\lg n)$，适合对大规模 SVG 格式地图进行差异匹配；④由于定义了节点集元素，差异脚本中的操作由原本的 insert，move 和 delete 3 种操作转变成 insert 和 delete 2 种操作. 该算法为今后基于 SVG 的相关研究提供了理论基础.

参考文献：

[1] Cobena G, Abiteboul S, Marian A. Detecting changes in XML documents [C]//Proceedings of the 18th International Conference on Data Engineering. San Jose: IEEE, 2002: 1-3.

[2] Wang Y, DeWitt D, Cai J. X-Diff: an effective change detection algorithm for XML documents[C]//Proceedings of the 19th International Conference on Data Engineering. Bangalore: IEEE, 2003: 2-4.

[3] Al-Ekram R, Adma A, Baysal O. diffX: an algorithm to detect changes in multi-version XML documents[C]//Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative Research. Markham: IBM, 2005: 5-10.

[4] Du Q, Guo ZC, Tang X. DiffSvg-matching algorithm of different timestamp maps based on SVG [C]//IEEE International Conference on Computer Science and Service System, Nanjing: [s.n.], 2012: 1-5.

（下转第 1603 页）