基于 MapReduce 的 PageRank 算法优化研究

张 永, 尹传晔, 吴崇正
(兰州理工大学 计算机与通信学院, 兰州 730050)

摘 要: 为了提高 PageRank 算法的计算效率,提出了基于块结构划分的方法,将网页之间的链接关系转换成网络块间的关系,减少了 map 和 reduce 操作的调用次数,降低了 I/O 传输造成的开销,提高计算的效率。实验证明,该方法具有一定的优越性。

关键词: MapReduce; PageRank 算法; 块结构划分; Hadoop
中图分类号: TP391 **文献标志码**: A **文章编号**: 1001-3695(2014)02-0431-04
doi:10.3969/j.issn.1001-3695.2014.02.026

Research on PageRank algorithm optimization based on MapReduce

ZHANG Yong, YIN Chuan-ye, WU Chong-zheng
(*School of Computer & Communication, Lanzhou University of Technology, Lanzhou* 730050, *China*)

Abstract: In order to improve the computational efficiency of PageRank algorithm, this paper proposed a method based on block structure partition, converting the link relation of Web nodes into the relation of net blocks, which would decrease the times of calls about the map and reduce operations, reduced the consumption caused by I/O transmission and improved the computation efficiency. The experiment shows that the proposed method has certain advantages.

Key words: MapReduce; PageRank algorithm; block structure partition; Hadoop

PageRank 算法是 Google 的创始人 Page 等人于 1998 年提出的。该算法是基于 Web 超链接结构分析的算法,并通过链接结构计算网页的重要程度,是评鉴网页排名的重要工具。

PageRank 算法[1,2]利用网络结构中的反向链接信息对网页进行排序,是一种与主题无关的排序算法。其主要思想是基于网页链接结构的分析,即一个网页的质量和重要性是通过指向它的网页数量来衡量,数量越多越重要;另一方面,指向它的网页的重要性(PageRank 值)越高,该网页越重要。

PageRank 算法如式(1)所示:

$$PR(A) = (1-d) + d\sum_{i\in S(A)} PR(T_i)/N_{T_i} \quad (1)$$

其中:$PR(A)$是指网页 A 的 PageRank 值;$d(0<d<1)$为阻尼系数,一般设为 0.85;T_i 为指向网页 A 的所有网页集,$i=1,2,\cdots,n$;$PR(T_i)$为网页 T_i 的 PageRank 值;N_{T_i}为网页 T_i 所链出的所有网页数和。

Hadoop 平台是原 Yahoo 的 Doug Cutting 根据 Google 发布的学术论文研究而来[3]。根据 Google 的三个核心组件 Google file system(GFS)、Map/Reduce、Big Table 提出了开源实现的 Hadoop distributed file system(HDFS)、Map/Reduce 和 HBase。

MapReduce 是在超大集群下进行海量数据的分布式编程模式[4,5],是一种简化的分布式编程模式,包括 map(映射)与 reduce(规约)两项核心操作。在模型中首先对输入的数据进行分割,将分割后的数据分配给 map 函数,而 map 把分配到的数据(一组〈key,vaule〉对)映射为另外的一组〈key2,vaule2〉型中间数据,其映射的规则由具体函数确定,在数据独立性的基础上,该函数具有较大的灵活性和高度并行性[6];Reduce(规约)主要的任务是中间数据〈key2,key2〉进行处理,大规模运算也相对独立,然后通过设定规则整合最终结果。

1 相关研究

文献[7]在计算微博排名 PageRank 算法中,按照 MapReduce 模型进行设计,并引入一个状态转移矩阵实现用户重要性的迭代运算,从而得到较为精确的量化结果。该量化结果不仅合理反映了用户粉丝的数量,而且有效兼顾了用户粉丝的质量,改善了搜索排名。文献[8]采取矩阵分块的方式来降低 PageRank 算法并行迭代次数,从而提高其实用度,但矩阵分块大大降低了并行时间,但是邻接矩阵的存储空间代价大。文献[9]分析 PageRank 算法的迭代计算,迭代之间不变数据重新加载造成了网络带宽以及 CPU 资源的浪费,为此提出了新系统 Hadoop 来解决该问题。文献[10]提出怎样在分布式环境下应用 PageRank 算法,实验结果表明其优于集中式的 PageRank 算法。文献[11]研究针对动态图中的 PageRank 计算问题,在 Web 改变的情况下,更快速更新 PageRank 值。文献[12]综合考虑了元信息和人类监督信息,提出了半监督 PageRank 算法,能够更精确 Web 图的链接排名,并在 MapReduce 框架下实现。

针对 PageRank 所要处理的网页数量较大,计算过程中需要经过多步的迭代,同时消耗大量的存储空间和计算资源,本文提出基于开源的 Hadoop 平台,将 PageRank 算法与 MapReduce 编程模式相结合,并采取基于块结构划分的方式,来减少

收稿日期: 2013-05-21; **修回日期**: 2013-06-30

作者简介:张永(1963-),男, 教授,硕导,主要研究方向为智能信息处理、数据挖掘、图像识别与处理等;尹传晔(1986-)男,硕士研究生,主要研究方向为智能信息处理、分布式搜索引擎(yinchuanye720@163.com);吴崇正(1987-),男, 硕士研究生,主要研究方向为智能信息处理、分布式搜索引擎.

计算量,提高 PageRank 算法的计算效率。

2 基于 Map/Reduce 算法优化的研究

2.1 基于块结构划分方法的计算流程

块结构算法是一种分而治之的思想。本文采用块结构算法划分网络的方法,将整个 Web 图中的节点按照一定的规则划分为若干个小的网络节点块,而每一块都是一个超级节点,这样每次迭代节点权值的计算量则大大减少,同时超级节点内网页的权值计算可以同时进行。

一般节点的 URL 可以分为四个等级,分别是 domain、host、directory、page,按照不同的域名等级对 Web 图进行划分,则形成不同的分块策略;理想的网络分块结果是超级节点内的链接尽量多,而超级节点之间的链接应该尽量少,这样也符合超级节点内高内聚,超级节点间低耦合的特点。文献[13]在实验中按照 host 等级划分 Web 图时算法能取得最好的效果。文献[14]指出,按照域名的分类方法能够很好地进行网络分块,节点链接的分布呈现出超级节点内紧密、超级节点间稀疏的特点。域名的四个等级,如表 1 所示。

表 1 域名的四个等级

等级	示例:http://www.jitong.gsut.edu.cn/institution/yjfx.asp
domain	gsut.edu.cn
host	jitong.gsut.edu.cn
directory	jitong.gsut.edu.cn/institution
page	jitong.gsut.edu.cn/institution/yjfx.asp

本文为了获得更好的 Web 分块效果,采用 host 等级来划分网络节点。基于块结构算法的网络划分流程为:首先将 Web 图中每个节点 URL 进行分词,用该 URL 的主机名标志该节点;其次,将主机名相同的节点划分在同一子图中;最后将 Web 图转换为若干个子图。下面定义两个指标量来衡量分块的效果:块内链接紧密度(percent)和网络分块平均紧密度(average-percent)分别如式(2)和(5)所示。根据域名分块的方法,若提取不同 host 的数据为 Y,获得 Y 个分块,每个块表示为 $B_i(i=1,2,\cdots,Y)$,则

$$\text{percent}=\frac{\sum_{j\in B_i}\text{inside}_j}{\sum_{j\in B_i}\text{all}_j}\quad i=1,2,\cdots,Y \tag{2}$$

其中:inside_j、all_j 分别如式(3)和(4)所示。

$$\text{inside}_j=\begin{cases}1 & \text{if page}_j\rightarrow\text{page}_k,\text{for all } k\in B_i,k\neq j\\ 0 & \text{otherwise}\end{cases} \tag{3}$$

$$\text{all}_j=\begin{cases}1 & \text{if page}_j\rightarrow\text{page}_k,\text{for all } k\in \overset{Y}{\underset{i=1}{YB_i}},k\neq j\\ 0 & \text{otherwise}\end{cases} \tag{4}$$

$$\text{average}-\text{percent}=\frac{\sum_{i=1}^{Y}\sum_{j\in B_i}\text{inside}_j}{\sum_{i=1}^{Y}\sum_{j\in B_i}\text{all}_j} \tag{5}$$

为了更清晰地阐述基于块结构划分的方法,在此规定该方法为 SGPR(segment PageRank),首先来以图 1 为例描述 SGPR 的计算流程。图 1 中有 9 个节点,分别被命名为 1~9。为了减少计算过程中的网络传输和 I/O 开销,这 9 个网页的 URL 采用哈希表来构建 URL 与数字的映射关系,其中 1→2 表示网页 1 指向网页 2。

首先规定一些参数,在此沿用 NumPR 方法中的规定,预处理阶段包括预处理输入和预处理输出两部分。其中预处理输入格式为 key→value,如 1→2,表示网页 1 指向网页 2;预处理输出的格式为 key 节点号→(newPR,oldPR;key 网页指向链接的集合),如 1→ (0.5,0.5;2,3,4)。为了简化计算,将 PageRank 算法中的阻尼系数 d 规定为 0.5,其中 newPR 指该 key 网页在本轮迭代时 PageRank 值,oldPR 指的是上一轮迭代时的 PageRank 值。下面就开始描述 SGPR 方法的计算流程。

a)分块划分。分块划分要符合超级节点内紧密、超级节点间稀疏的特点。本文划分原则按照主机名划分的方法,如图 2 所示,假若节点 1、2、3 具有相同的主机名,建立二级哈希映射,首先将节点 1、2、3 的 URL 映射为数字 1、2、3,然后将这三个节点划分在同一个子图中,命名为 G_1,然后再采用哈希表来存储子图与各网页节点的映射关系,等迭代结束后,再重新建立各网页节点间的对应关系。为了更为简明地阐述本文的方法,在此将图 1 中九个节点分为三块:$G_1(1,2,3)$、$G_2(4,5,6)$、$G_3(7,8,9)$。

b)预处理阶段。按照分块结构算法划分,图 1 中 Web 节点形成了 3 组键值对,经过既定规则的划分,网络图由 9 个节点变成了 3 个大的超级节点,NumPR 算法中网页之间的链接关系变成了子图与子图之间的链接关系,有效地压缩了节点的数量,减少了迭代的次数。预处理阶段的输入如图 2 所示。

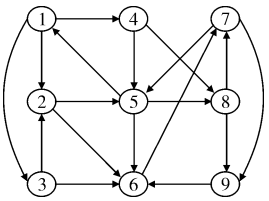

图 1 九个节点的示例图

$G_1:(1,2,3)\rightarrow\begin{cases}0.5,0.5:2,3,4\\0.5,0.5:5,6\\0.5,0.5:2,6\end{cases}$

$G_2:(4,5,6)\rightarrow\begin{cases}0.5,0.5:5,8\\0.5,0.5:1,6,8\\0.5,0.5:7\end{cases}$

$G_3:(7,8,9)\rightarrow\begin{cases}0.5,0.5:5,9\\0.5,0.5:7,9\\0.5,0.5:6\end{cases}$

图 2 块划分图(G_1,G_2,G_3)

在此如 G_1 代表 1、2、3,这样就可以实现 key 值的压缩计算,减少了调用 map 和 reduce 操作的次数,降低了系统的开销。预处理阶段的输出,如表 2 所示。

表 2 SGPR 方法预处理的输出

阶段	key_{in}	value_{in}
map_1	G_1	G_1 rank_1:1→2,3;rank_3:3→2
map_2	G_1	G_2 rank_1:1→4;rank_2:2→5,6; rank_3:3→6
map_3	G_2	G_1 rank_5:5→1
map_4	G_2	G_2 rank_4:4→5;rank_5:5→6
map_5	G_2	G_3 rank_4:4→8;rank_5:5→8;rank_6:6→7
map_6	G_3	G_2 rank_7:7→5;rank_9:9→6
map_7	G_3	G_3 rank_7:7→9;rank_8:8→7,9

其中:key_{in}代表子图的编号,value_{in}代表 key_{in}中节点指向网页所在子图的编号。其中 $\text{rank}_n=\{\text{newPR}_n,\text{oldPR}_n,\text{Num}_n\}$,代表编号为 n 的节点指向的链接;n 为 key_{in}子图中编号为 n 的节点,花括号中分别表示节点 n 这次迭代和上次迭代的 PageRank 值,及节点 n 所指向链接的个数。同样预处理的输出也是 Map 阶段的输入数据。

c)SGPR 方法的 map 阶段。若 key_{in}子图中有 m 个节点指向 value_{in}子图的节点 n,那么网页 n 的 m 个被投票值会在 map 输出阶段被提前计算出结果,然后再传输给 reduce 端。map 阶段的输出部分如表 3 所示。

表 3 SGPR 方法 map 阶段的输出

阶段	key_{out}	$\text{value}_{\text{out}}$
Map_1	G_1	2→rank'_1,rank'_3 3→rank'_1
Map_2	G_2	4→rank'_1 5→rank'_2 6→rank'_2,rank'_3
Map_3	G_1	1→rank'_5
Map_4	G_2	5→rank'_4 6→rank'_5
Map_5	G_3	7→rank'_6 8→rank'_4,rank'_5
Map_6	G_2	5→rank'_7 6→rank'_9
Map_7	G_3	7→rank'_8 9→rank'_7,rank'_8

其中:key_{out}对应的是表中的 $value_{in}$中子图编号;$value_{out}$中则是 key_{out}子图中被投票节点的值,如 $m \rightarrow rank'_n$($rank'_n = newRank_n/Num_n$)表示节点编号为 m 的网页被投票的值。其中 $rank'_n$ 值被计算出来,将更多地计算量放在了 map 端实现,传输到 reduce 端的不再是计算列表,而是数值,极大地降低了网络传输消耗。另外可以看出,输出部分共有 7 个键值对(key→value),共 14 个 URL→Rank 对,要是采用 NumPR 方法,则会产生 18 个 URL→Rank 对;随着节点的增多,这种压缩效果会显得更加明显。

d)SGPR 方法的 reduce 阶段。这个阶段的主要任务是计算节点最后的 PageRank 值,reduce 阶段输入的信息则是 map 阶段输出的数据,即所有被投票节点及所获得贡献值,因为在 map 阶段提前计算了节点的被投票值,将数据直接传输给 reduce 端,那么就减少了数据传输和 I/O 的开销。例如节点 2,它被节点 1 和 3 链接,根据相同的 key 值,合并权值,可知节点 2 最后的值为 $PR_2 = (1-d) + d(rank'_1 + rank'_3) = 0.5 + 0.5 \times (0.167 + 0.250) = 0.709$,当全部计算结束后,更新每个节点的 newRank 值,再将信息以 $G_m \rightarrow \{G_n\ Rank_i : i \rightarrow j, \cdots\}$ 形式输出,作为下一轮 map 的输入,继续迭代,直到满足某个收敛条件为止。

2.2　算法设计

基于 MapReduce 的 PageRank 算法伪代码描述如下:

```
Map(key_in, value_in){ //map 操作
//key_in为子图编号
//value_in为 key_in中节点所链接网页所在的子图编号
/*Rank_n = {newPR_n, oldPR_n, Num_n},代表编号为 n 的节点指向的链接。其中 n 为 key_in子图中编号为 n 的节点,花括号中分别表示节点 n 这次迭代和上次迭代的 PageRank 值,及节点 n 的所指向链接的个数*/
input(value_in, Rank_n: n→x, y, z, …); //map 阶段的输入数据
output(key_out, m→rank'_n); //map 阶段的输出数据
/*key_out对应的是表中的 value_in 中的子图编号,value_out 中则是 key_out子图中被投票节点的值,如 m→rank'_n(rank'_n = newRank_n/Num_n)表示节点编号为 m 的网页被投票的值*/
for(each page m in m→rank'_n){
  rank'_n = newRank_n/Num_n;
  output(m, rank'_n); //输出页面 m 的被投票值
}
}
Reduce(key, value){ //reduce 操作
//key 为页面 m
//value 为链接页面 m 的页面 n 的投票值
input(m, rank'_n); //reduce 的输入数据
for(each page n link the page m)
PR_m = (1-d) + d Σ_i^n (rank'_i); //计算网页 m 的权值
Renew(Rank_n);
//更新每一个页面的 Rank_n = {newPR_n, oldPR_n, Num_n}
output(key_in, value_in); //key_in为子图编号 G_m
/*value_in为 key_in中节点所链接网页所在的子图编号 G_n,输出形式 key_in→value_in,信息将以 G_m→{G_n Rank_i: i→j, …}的形式输出*/
}
```

将迭代计算的过程分为 map 和 reduce 两个阶段。第一个阶段主要生成每个网页的权值向量,而第二个阶段主要是合并每个网页被投票值的和,更新并输出每个子图链接关系的列表,用于下一次迭代计算;而在 map 端将每个网页被投票值计算出来,形成一个个数据,替代了原来的计算列表,这样就大大降低了网络传输的 I/O 消耗,提高了迭代的效率。

3　实验及其结果分析

为了验证改进后算法的效率,本实验将以传统的 PageRank 和 NumPR 算法作对比,从不同数据量大小的迭代效率上分析所提方法的性能。

3.1　实验参数的设定和数据集

采用 10 台 Intel 酷睿 1.86 GHz 双核,内存 1 GB 的 PC 通过 100 Mbps 交换机互连,搭建分布式环境;每台 PC 机都安装上 Windows XP SP3 操作系统、JDK 6.0、Hadoop 0.20.203.0 及 Lucene 3.0 版本;其中选择 1 台 PC 机作为 master 节点,剩余 9 台作为 slave 节点。Node1 作为 NameNode,Node2 ~ Node10 作为 DataNode,负责计算任务。设置数据的分块大小为 64 MB,其中设定迭代收敛的条件为某节点的 PageRank 值之差小于等于 10^{-6}。规定初始的 PageRank 值为 0.5。数据集采用 Stanford University 社会网络研究平台提供的 Web 图数据。

3.2　实验方案设计

本节主要是为了验证改进后的 PageRank 算法能减少中间的迭代计算量,降低任务执行的时间,从而提高了算法的性能,进而提高了系统的运行效率。实验在集群环境不变的情况下,当数据量不断增大时,比较本文改进 PageRank、传统 PageRank 及 NumPR 算法的迭代执行时间,以分析三种算法的执行效率。实验分别测试 Web 图中有 28 万、90 万、85 万、120 万、186 万节点时,各种算法迭代结束时任务消耗的时间。最终执行时间值越大,则说明迭代阶段执行的时间越长,则该算法的效率就越低。

3.3　结果分析

针对不同 Web 图数据集,根据既定的迭代收敛条件,在不同 Web 图中,验证的三种算法迭代计算,从开始计算到迭代结束,执行时间如表 4 所示。

表 4　不同数据集下三种算法迭代执行时间　　/s

Web 图节点数	PageRank	NumPR	SGPR	Web 图节点数	PageRank	NumPR	SGPR
28 万	369	320	243	120 万	831	512	359
90 万	434	383	277	186 万	1267	1034	482
85 万	407	364	263				

由表 4、图 3 所示,SGPR 及 NumPR 算法在不同的数据集上执行时间都远远小于 PageRank 算法。由此可见,传统的 PageRank 算法在迭代计算过程中的时空消耗较大,随着数据块不断增长,迭代执行时间急剧上升,而其他两种算法的迭代执行时间上升的幅度较为平缓,这说明了对 PageRank 的优化算法有一定的效果。下面仅讨论 NumPR 和 SGPR 算法。

从图 3 可以看出,本文提出的 SGPR 总体上要优于 NumPR 算法,在 Web 图节点较少时,因数据分块较少,数据的并行性较差,所以 SGPR 算法执行效率并不明显;随着节点的不断增加,因所提方法通过压缩节点数据,将较大的数据节点压缩到只有原来的万分之几,每次迭代计算时将大大减少 map 和 reduce 函数的调用次数,产生的中间数据量相对较小,从而在传递网页完整信息过程中所需的系统开销也较小;另外在 map 阶段的操作中将网页被贡献值计算的工作转移到了 map 端进行,传递到 reduce 端的仅仅是一个个数字,而非计算列表,这

样就大大降低了系统的 I/O 开销,平衡了计算资源,提高了系统整体计算性能,从而缩短了迭代计算时间,提高了算法的效率。同时,由图 3 所示,当数据块不断增大时,NumPR 算法迭代执行时间上升的增幅也较大,由此可见,SGPR 算法在性能上有一定的提升。

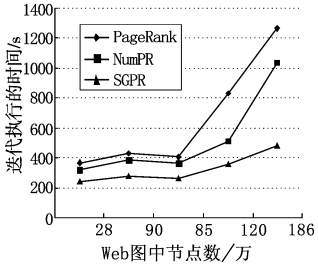

图 3 三种算法在不同数据集下迭代执行时间

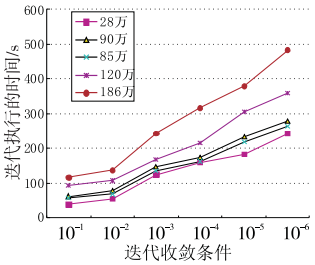

图 4 SGPR 算法在不同收敛条件下迭代执行时间

对比改进后 SGPR 算法在 10^{-1}、10^{-2}、10^{-3}、10^{-4}、10^{-5}、10^{-6}六种情况下执行迭代计算时间,以验证改进后的 SGPR 算法性能是否平稳;由图 4 所示,随着 Web 图中节点数的增大,迭代执行时间的增幅不大,而传统的 PageRank 算法在 Web 图中节点越来越大时,执行时间则急剧增大。由此可见,改进后的 SGPR 算法在迭代计算的执行时间上增长相对较为平稳,迭代收敛的速度较高。

4 结束语

本文主要研究在开源 Hadoop 平台下,结合 MapReduce 模型,实现 PageRank 算法并行化,提出了基于块结构划分的方法,将网页之间的链接关系转换成网络块间的关系,减少了 map 和 reduce 操作的调用次数,降低了 I/O 传输造成的开销,提高了计算的效率。实验证明,本文方法具有一定的优越性。下一步的工作是对 PageRank 算法过程中每次迭代加载不变数据,造成 CPU 资源浪费以及通信开销,以及对于每次迭代结果检测频繁调用 MapReduce 任务的问题继续研究,以期进一步提高 PageRank 算法的并行计算效率。

参考文献:

[1] PASQUINELLI M. Google's PageRank algorithm: a diagram of cognitive capitalism and the rentier of the common intellect[EB/OL]. [2009]. http://matteopasquinelli.com/docs/Pasquinelli_PageRank.pdf.

[2] RIDINGS C, SHISHIGIN M. PageRank uncovered[EB/OL]. [2009-06-18]. http://www.voelspriet2.nl/PageRank.pdf.

[3] 王俊生,施运梅,张仰森. 基于 Hadoop 的分布式搜索引擎关键技术[J]. 北京信息科技大学学报,2011,26(4):53-54.

[4] 李建江,崔健,王聃,等. MapReduce 并行编程模型研究综述[J]. 电子学报,2011,39(11):2635-2642.

[5] Apache MapReduce architecture[EB/OL]. [2012-05-28]. http://hadoop.apache.org./mapreduce/.

[6] JEFFREY D, SANJAY G. MapReduce: a flexible data processing tool [J]. Communications of the ACM, 2010, 53(1): 72-77.

[7] 梁秋实,吴一雷,封磊. 基于 MapReduce 的微博用户搜索排名算法[J]. 计算机应用,2012,32(11):2989-2993.

[8] 李远方,邓世昆,闻玉彪,等. MapReduce 下的 PageRank 矩阵分块算法[J]. 计算机技术与发展,2011,21(8):6-9.

[9] BU Y, HOWE B, BALAZINSKA M, *et al*. The HaLoop approach to large-scale iterative data analysis[J]. International Journal on Very Large Data Bases, 2012, 21(2): 169-190.

[10] WANG Yuan, DEWITT D J. Computing PageRank in a distributed Internet search system[C]//Proc of the 30th International Conference on Very large Data Bases. 2004: 420-431.

[11] BAHMANI B, KUMAR R, MAHDIAN M. PageRank on an evolving graph[C]//Proc of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2012: 24-32.

[12] GAO Bin, LIU Tie-yan, WEI Wei, *et al*. Semi-supervised ranking on very large graphs with rich metadata[C]//Proc of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2011: 96-104.

[13] ZHONG Cai-ming, MIAO Duo-qian, WANG Rui-zhi. A graph-theoretical clustering method based on two rounds of minimum spanning trees [J]. Pattern Recognition, 2010, 43(3): 752-766.

[14] SANKARALINGAM K, YALAMANCHI M, SETHUMADHAVAN S, *et al*. PageRank computation and keyword search on distributed systems and P2P networks[J]. Journal of Grid Computing, 2003, 1(3): 291-307.

(上接第 421 页)

5 结束语

e-Learning 学习资源本体常常基于不同规范构建,在资源发现时难以通过语言学本体匹配方法实现,现有结构匹配方法又未能较好解决结构的相似性计算问题,因此本文提出一种基于图同构的本体匹配方法,在计算图结构整体相似性的基础上对本体表示有向图中的点、边进行交替匹配,从而以子图同构判定来实现本体匹配。该方法旨在发现高效的相似子图,提高本体匹配的精准性和效率。

参考文献:

[1] 美国教育部教育技术白皮书[K]. 上海:上海市教科院智力开发研究所,2001.

[2] STUDER R. Knowledge engineering: principles and methods[J]. Data & Knowledge Engineering, 1998, 25(1-2): 199-220.

[3] GANGEMI A, PISANELLI D M, STEVE G. An overview of the ONIONS project: applying ontologies to integration of medical terminologies[J]. Data & Knowledge Engineering, 1999, 31(2): 183-220.

[4] DOAN A H, MADHAVAN J, DOMINGOS P, *et al*. Learning to map between ontologies on the semantic Web[C]//Proc of the 11th International Conference on WWW. New Yrok: ACM Press, 2002: 662-673.

[5] JI Qiu, HAASES P, QI Gui-lin. Combination of similarity measures in ontology matching using the OWA operator[M]//Recent Developments in Orderad Weighted Averaging Operators: Theory and Practice. [S.l.]: Springer, 2011: 281-295.

[6] EUZENAT J, SHVAIKO P. Ontology matching[M]. Berlin: Springer-Verlag, 2007.

[7] COHEN W W, RAVIKUMAR P, FIENBERG S E. A comparison of string distance metrics for name-matching tasks[C]//Proc of IJCAI. 2003: 73-78.

[8] MELNIK S, GARCIA-MOLINA H, RAHM E. Similarity flooding: a versatile graph matching algorithm and its application to schema matching[C]//Proc of the 18th International Conference on Data Engineering. 2002: 117-128.

[9] MADHAVAN J, BERNSTEIN P A, RAHM E. Generic schema matching with CuPid[C]//Proc of the 27th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2001: 49-58.

[10] NOY N, MUSEN M. The PROMPT suite: interactive tools for ontology merging and mapping[J]. International Journal of Human-Computer Studies, 2003, 59(6): 983-1024.

[11] GIUNEHIGLIA F, SHVAIKO P, YATSKEVIEH M. S-Match: an algorithm and an implementation of semantic matching[C]//Proc of ESWC. 2004: 61-75.

[12] 唐杰,梁邦勇. 语义 Web 中的本体自动映射[J]. 计算机学报,2006,29(11):1957-1974.