引用格式:牛健,崔焕庆,成曦,等. 求解大规模稀疏有向图回路的多线程并行算法[J]. 山东科技大学学报(自然科学版),2018, 37(2):32-38.

NIU Jian, CUI Huanqing, CHENG Xi, et al. Multithreading parallel algorithm for solving circuits of large-scale sparse directed graphs[J]. Journal of Shandong University of Science and Technology (Natural Science), 2018, 37(2):32-38.

求解大规模稀疏有向图回路的多线程并行算法

牛　健,崔焕庆,成　曦,傅　游

(山东科技大学 计算机科学与工程学院,山东 青岛 266590)

摘　要:传统的基于深度优先遍历的回路求解算法限于计算机内存无法对大规模图进行求解,而已有的分布式图计算系统需要借助计算机集群,成本较高。针对此问题,给出一种可在普通计算机上求解大规模有向图所有回路的多线程并行算法。该算法根据顶点的出度,首先删除出度为 0 的顶点,然后采用多线程并行求解包含出度较大的顶点的回路,最后使用串行算法求出图剩余部分的回路。实验表明,此算法能够在普通计算机上求得大规模有向稀疏图的所有回路。

关键词:大规模有向稀疏图;有向回路;多线程;并行

中图分类号:TP30　　**文献标志码**:A　　**文章编号**:1672-3767(2018)02-0032-07

DOI: 10.16452/j.cnki.sdkjzk.2018.02.005

Multithreading Parallel Algorithm for Solving Circuits of Large-scale Sparse Directed Graphs

NIU Jian, CUI Huanqing, CHENG Xi, FU You

(College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, Shandong 266590, China)

Abstract: To solve the incapability of traditional circuit algorithms based on depth-first traversal in solving the large-scale graph due to the limited memory and the high cost of the existing distributed graph computing platforms with the help of computer cluster, this paper presented a multithreading parallel algorithm for solving circuits of large-scale sparse directed graphs which could be conducted on ordinary personal computers. According to the out-degree of vertices, the algorithm first deleted the vertices with the out-degree of 0, then solved the circuits containing the vertices with larger out-degree by using multithreading paralell algorithm, and finally computed the residual graph's circuits by using serial algorithm. Experiment results show that the proposed algorithm can solve all circuits of a large-scale sparse graph on an ordinary computer.

Key words: large-scale sparse directed graph; directed circuits; multithreading; parallel

图在知识图谱、社交网络等很多领域都具有广泛应用,其中求解图的回路是一个重要问题[1]。基于

收稿日期:2017-07-24

基金项目:国家重点研发计划课题(2017YFC0804406,2017YFB0202001);山东省计算机网络重点实验室开放课题(SDKLCN-2015-03)

作者简介:牛　健(1993—),男,山东枣庄人,硕士研究生,主要从事大规模图数据处理的研究.

崔焕庆(1979—),男,山东泰安人,副教授,博士,主要从事无线传感网络,高性能计算的研究,本文通信作者.

E-mail:cuihq79@163.com

DFS(deep-first search,深度优先遍历)的求解算法是目前最常用的算法[2],但随着图的规模日益增大,传统的串行求解算法由于存储能力的限制,无法求解该问题。

本文面向大规模稀疏有向图,给出了一种可在普通 PC(personal computer,个人计算机)上基于多线程的并行求解算法 MPCF(multithreading parallel circuit finding)。该算法首先删除图中出度为 0 的所有顶点,然后找到图中出度较大的顶点,采用多线程方法并行求解含有这些顶点的回路,接着删除这些顶点,再采用单线程方式求解剩余图的回路。

1　相关工作

求解图的回路是一个被广泛关注的问题,针对不同的回路,已提出了多种求解算法,如求解 Hamilton 回路的非递归算法[1]、求解所有顶点的最短回路的算法[3]、求解最长回路的算法[4]等。这些算法基本上都是基于 DFS 实现的,其核心过程是:从某个顶点出发,找出刚访问顶点的第一个未被访问的邻接点,然后再从此邻接点出发,继续找它的下一个新的邻接点进行访问,重复此步骤,直到遇到无法继续访问或者遇到一个已经被访问的顶点。如果是后者,则表示发现了一个回路。由此可见,DFS 算法的主要过程就是递归搜索,而这也正是它的不足之处:当图数据集规模较小时,递归工作栈浅,内存开销小,单机运算足以胜任;但对于大规模图,单个顶点的邻接点数目众多,频繁的递归操作势必会造成内存不足,最终因堆栈溢出而出错。王玉英等[5]提出的求解有向图的全部简单回路的算法,基于矩阵运算,避免了递归调用,但对于稀疏图而言,邻接矩阵的存储方式占用了过多的存储空间,也容易造成内存溢出。

由于图结构的特殊性,顶点与顶点之间具有很强的依赖关系,很难找到一种合适的图划分方式来打破这种依赖性,而且 DFS 算法本身难以有效地实现并行化[6],因而目前尚无有效的基于 DFS 的回路求解并行算法。

近年来逐渐兴起的云计算系统为基于机群的并行图处理系统提供了有力支撑,从而催生了如 Pregel[7]、GraphX[8]、GPS[9]等图处理系统。这些系统都是基于 BSP(bulk synchronous parallel)模型实现的,大致处理流程如下:

1) 将图划分成多个分区,每个分区分配到集群中的多个计算节点上;

2) 将每个计算节点上的顶点均标记成“活跃”状态;

3) 开始运行一个超步,每个顶点均调用自定义的函数,函数功能包括执行计算、消息传递等;

4) 同步结束后,运行下一个超步并重复此过程,直到所有顶点都不再“活跃”并且系统中不会有任何消息在传输,这时执行过程结束。

可见,这类系统能够以批处理方式来处理大规模图数据集问题,但这些计算框架均需要搭建分布式处理系统,还要考虑分布式文件存储和任务调度,大大提高了计算成本。SC-BSP[10]模型在 BSP 的计算框架下引入分离器和组合器,对图划分的边集进行合理分配和管理,提高了水平扩展能力,但图划分对计算效率的影响仍然较大。此外,基于 BSP 模型的求解算法均是以顶点为中心的消息传递方式来处理图问题的,效率较低。

2　问题定义

$G=(V,E)$ 是一个有向图,其中 $V=\{v_1,v_2,\cdots\}$ 是顶点集合,$E=\{\langle v_i,v_j\rangle \mid v_i,v_j\in V, i\neq j\}$ 是有向边(又称为弧)的集合。若 $\langle v_i,v_j\rangle\in E$,则称 v_j 为 v_i 的邻接顶点。记

$$d_i^{\text{out}}=\left|\{v_j \mid \langle v_i,v_j\rangle\in E\}\right|$$

为顶点 v_i 的出度。记

$$d_{\max}^{\text{out}}=\max\{d_i^{\text{out}} \mid v_i\in V\},\ d_{\text{avg}}^{\text{out}}=\frac{1}{|V|}\sum_{i=1}^{|V|}d_i^{\text{out}}$$

分别为所有顶点的最大出度和平均出度。定义

$$\delta=\frac{|E|}{|V|(\,||V|-1|\,)}$$

为图的稠密度,并称 $\delta < 0.01$ 的图为稀疏图。设 G 中所有顶点的出度有 N 个不同的值 $a_1, a_2, \cdots, a_N$,且 $a_i > a_j (0 < i < j < N+1)$,显然 $d_{\max}^{\text{out}} = a_1$。定义

$$\rho_\alpha = \frac{|\{v_i \mid a_{\lceil N\alpha \rceil} \le d_i^{\text{out}} \le a_1\}|}{|V|}, \alpha \in (0,1]$$

表示 N 个出度中,最大的前 $\lceil N\alpha \rceil$ 个出度对应的顶点总数与 G 中所有顶点数量之比。记 ρ_0 为出度为 0 的顶点数占顶点总数的比例,即

$$\rho_0 = \frac{|\{v_i \mid d_i^{\text{out}} = 0\}|}{|V|}$$

本文所要解决的问题是求解给定的一个大规模稀疏有向图的所有简单回路。

3 回路求解的 MPCF 算法

大规模稀疏图是一类非常常见的模型,如服从幂率分布的社交网络[11]。这类图具有两个非常重要的特点:一是图中存在大量出度/入度为 0 的顶点;二是图中仅有少部分顶点的出度/入度非常大。文献[11]的分析表明微博数据便具有上述两个特点。本文对斯坦福大学的 SNAP[12] 数据集中的 7 种不同规模的图进行了分析(表 1),这些数据也满足上述两个特点。基于此,可以设想:

1) 删除出度为 0 的顶点,能够大幅度缩减图的规模;

2) 采用并行方式求解包含出度较大顶点的回路,删除这些顶点后,再求解剩余图的回路,可有效提高计算效率。

表 1 有向稀疏图出度的分析

Tab. 1 Analysis of out-degree of directed sparse graphs

名称	$\|V\|$	$\|E\|$	$\delta/10^{-4}$	$d_{\text{avg}}^{\text{out}}$	$d_{\max}^{\text{out}}$	$\rho_0/\%$	$\rho_\alpha/\%$		
							$\alpha=0.1$	$\alpha=0.2$	$\alpha=0.3$
p2p-Gnutella04	10 876	39 994	3.38	3.68	100	54.62	0.044	0.074	0.102
p2p-Gnutella25	22 687	54 705	1.06	2.41	64	72.58	0.021	0.042	0.075
p2p-Gnutella30	36 682	88 328	6.56	2.41	54	73.50	0.013	0.037	0.078
p2p-Gnutella31	62 586	147 892	3.78	2.36	78	73.81	0.007	0.025	0.045
email-EuAll	265 214	420 045	5.97	1.58	930	72.44	0.012	0.026	0.041
web-NotreDame	325 729	1 497 134	1.41	4.59	3 445	57.65	0.016	0.032	0.050
wiki-Talk	2 394 385	5 021 410	8.76	2.10	100 022	93.83	0.011	0.023	0.035

MPCF 算法便是基于上述思想设计的。在初始化相关变量(第 1 行)、对顶点出度排序(第 2、3 行)后,从图中删除出度为 0 的顶点(第 4 行),并根据参数 α 将出度排序靠前的顶点放入集合 V_{large} 中(第 5 行)。第 6、7 行启动 n 个线程,其中 $n_{\max}$ 是预先指定的最大线程数。第 8~11 行为并行执行过程,每个线程求包含 V_{large} 中一个顶点的回路集合,同时为保证不同线程求解的是不同顶点,将算法第 9 行中 *nextVertex*()方法做加锁处理。第 12 行将 V_{large} 中的顶点删除,接着求出剩余顶点的导出子图 G'(第 13 行)。第 14~16 行以串行方式求 G' 的回路集合。此处之所以使用串行,是因为图 G' 顶点数目庞大,但由于没有大出度顶点,边数目减少,图变得极其稀疏,在图规模一定的情况下,使用串行算法即可在较快时间内求解,并行算法由于线程启动、管理等方面的时间消耗,反而需要花费更多的时间。该算法中,*nextVertex*() 用于返回 V_{large} 中下一个尚未求解其回路的顶点;*findCircuits*(v) 用于求包含顶点 v 的回路集合,即以顶点 v 作为起点和终点的回路的集合;*induceSubGraph*(V') 用于求顶点集合 V' 的导出子图。

需要指出的是:该算法是在单机上执行多线程的,各个线程共享内存,而且线程在求回路时,对图 G 是只读操作,因此图 G 不需要分割存储,即各线程无需存储 G 的信息。

MPCF 算法

```
    Input:Directed graph G = (V,E), parameter α
    Output:All circuits set CS of G
    procedure MPCF(G, α )
1.  CS ← Φ
2.  Get d_i^out for each v_i ∈ V using Equation (1)
3.  Sort d_i^out in non-descending order {a_1, a_2, ⋯, a_N}
4.  V' ← V − {v_i | d_i^out = 0}
5.  V_large ← {v_i | a_⌈Nα⌉ ≤ d_i^out ≤ a_1}
6.  n ← min{|V_large|, n_max}
7.  Fork n threads
8.  for each thread do in parallel
9.      synchronized u ← nextVertex()
10.     CS ← CS ∪ findCircuits(u)
11. end for
12. V' ← V' − V_large
13. G' ← induceSubGraph(V')
14. for all v ∈ V' do
15.     CS ← CS ∪ findCircuits(v)
16. end for
17. return CS
18. end procedure
```

假定使用邻接表来存储图数据,那么算法的时间复杂性分析如下。第 2 行需 $O(|V|+|E|)$ 来求各个顶点的出度。第 3 行排序需用 $O(N\log N)$ 来排序 N 个不同的出度值。第 4 行需要 $O(\rho_0|V|)$ 来删除出度为 0 的顶点。第 8～11 行并行过程的时间取决于求解包含出度最大顶点 u 的回路的时间。在最坏情况下,若以 u 为根的生成树包含了 V' 的所有顶点,使用 DFS 思想求回路的方法需要时间为 $O((1-\rho_0)|V|+|E|)$。考虑到图的稀疏性,第 12～13 行所求出的导出子图仅包含非常少的边,所以第 14～16 行的时间不会超过 $O((1-\rho_0)|V|)$。因此,该算法的时间复杂性为 $O(2|V|+2|E|+N\log N)$。对于稀疏图,$N<|E|\ll|V|^2$,所以 MPCF 算法的时间复杂度要低于文献[5]所提出算法的时间复杂度 $O(|V|^2)$。

4　实验与结果分析

本文进行实验的图数据即表 1 所示的 7 个有向图。实验用计算机的 CPU 为 Intel® Core™ i5-3210M CPU @ 2.50 GHz,运行内存为 6 GB,操作系统为 Windows10(64 位)。算法采用 JAVA 语言实现,设置 JVM(JAVA 虚拟机)最大堆内存参数"-Xmx"为 4 GB。每个单独实验均做 10 次,每次实验完成后,重启主机以保证每次实验环境一致,并且互不影响。运行时间取 10 次平均值。

4.1　α 的取值

α 是 MPCF 算法的关键参数,决定了进入并行环节的顶点数量,因此首先需要根据图的规模来选取 α 值。此处分别对 α 取 0.1、0.2 和 0.3 进行实验,结果如表 2 所示,表中 t_{para} 和 t_{seq} 分别表示算法 1 中第 8～11 步并行求解的时间、第 14～16 步串行求解的时间,$t_{total}=t_{para}+t_{seq}$。"—"表示串行部分内存溢出,无法求出所有的回路。

表 2 不同 α 值实验结果

Tab. 2 Experimental results of different α values ms

名称	$\alpha=0.1$			$\alpha=0.2$			$\alpha=0.3$		
	t_{para}	t_{seq}	t_{total}	t_{para}	t_{seq}	t_{total}	t_{para}	t_{seq}	t_{total}
p2p-Gnutella04	41.787 9	305.269 5	347.057 4	228.640 1	260.159 7	488.799 8	279.905 7	210.548 4	490.454 1
p2p-Gnutella25	147.704 5	515.024 2	662.728 7	389.790 3	465.002 5	854.792 8	508.382 0	400.258 1	908.640 1
p2p-Gnutella30	214.993 6	706.040 6	921.034 2	434.783 1	683.487 3	1 118.270 4	783.720 0	604.581 9	1 388.301 9
p2p-Gnutella31	333.955 9	904.818 9	1 238.774 8	644.629 3	807.554 9	1 452.184 2	883.467 2	800.597 7	1 684.064 9
email-EuAll	567.472 3	—	—	783.310 3	2 015.635 2	2 798.945 5	1 089.650 3	2 055.481 1	3 145.131 4
web-NotreDame	804.471 1	—	—	1 156.580 6	2 524.007 0	3 680.587 6	1 638.810 0	2 844.600 7	4 483.410 7
wiki-Talk	1 124.478 1	—	—	1 569.218 8	—	—	1 842.056 3	3 065.699 4	4 907.755 7

根据实验结果,可以得出:

1) 当顶点数在 10 万以下时,$\alpha=0.1$ 所需计算时间最少。

2) 当顶点数在 10 万到 100 万之间时,$\alpha=0.1$ 已无法串行求解 G' 的回路。$\alpha=0.2$ 时所需时间最少。

3) 当顶点数超过 100 万时,只有 $\alpha=0.3$ 方可求解回路。

4.2 MPCF 算法性能分析

由于文献[5]的算法是基于矩阵乘法运算的,对于表 1 中的大规模图,仅图的邻接矩阵便需要占用大量的内存空间。如对于 10 万个顶点的图,即使每个矩阵元素仅需 1 B 内存,整个邻接矩阵就需要占用超过 9 GB 内存,在计算过程中,则至少需要 20 GB 内存。因此该算法无法在普通 PC 上实现。

本文将 MPCF 算法与基于 DFS 的串行回路求解算法进行对比分析,执行时间如表 3 所示,占用内存情况如图 1 所示。表 3 中t_{dfs}为串行求解算法所需的时间。t_{para}和t_{seq}是在 4.1 节中确定的 α 值基础上求解算法的时间。

表 3 算法性能分析

Tab. 3 Performance analysis of algorithms ms

名称	t_{dfs}	t_{para}	t_{seq}	t_{total}	回路数
p2p-Gnutella04	355.517 9	41.787 9	305.269 5	347.057 4	62 608
p2p-Gnutella25	615.346 6	77.704 5	515.024 2	592.728 7	97 409
p2p-Gnutella30	—	184.993 6	706.040 6	891.034 2	184 758
p2p-Gnutella31	—	333.955 9	904.818 9	1 238.774 8	447 592
email-EuAll	—	783.310 3	2 015.635 2	2 798.945 5	603 901
web-NotreDame	—	1 156.580 6	2 524.007 0	3 680.587 6	1 078 473
wiki-Talk	—	1 842.056 3	3 065.699 4	4 907.755 7	1 158 291

可以看出,p2p-Gnutella04 和 p2p-Gnutella25 两个图的数据规模较小,串行算法占用内存在 4 GB 以下,且能在数百毫秒内求出所有的有向回路(少于 10 万个)。MPCF 算法的执行时间仅略少于串行算法,约为串行算法时间的 96%~97%,t_{para} 占全部执行时间的 12%~13%。从内存情况看,这两个图的串行求解算法没有因为递归而导致内存溢出,MPCF 算法的并行部分和串行部分占用内存为串行算法的 30%~65%。

对于其余的 4 个图,它们包含的有向回路数在 18 万条以上,此时串行求解算法的递归部分占用内存急剧增加,超过了 3.9 GB,Java 程序出现内存溢出(Out of Memory)异常,从而无法求出所有的有向回路。MPCF 算法的并行阶段仅求解极少量顶点的回路,串行阶段因处理的图 G' 已非常稀疏而递归层次浅,所以 MPCF 算法的内存开销未溢出。但随着图中所包含回路数量的增加,MPCF 算法所需的时间和内存也在增

长。从时间上来看,MPCF 算法的并行部分约为串行部分时间的 20%～37%;从内存上来看,MPCF 算法的并行部分约为串行部分时间的 78%～94%。

因此,MPCF 算法在内存方面能够保证大规模图有向回路的正常求解。

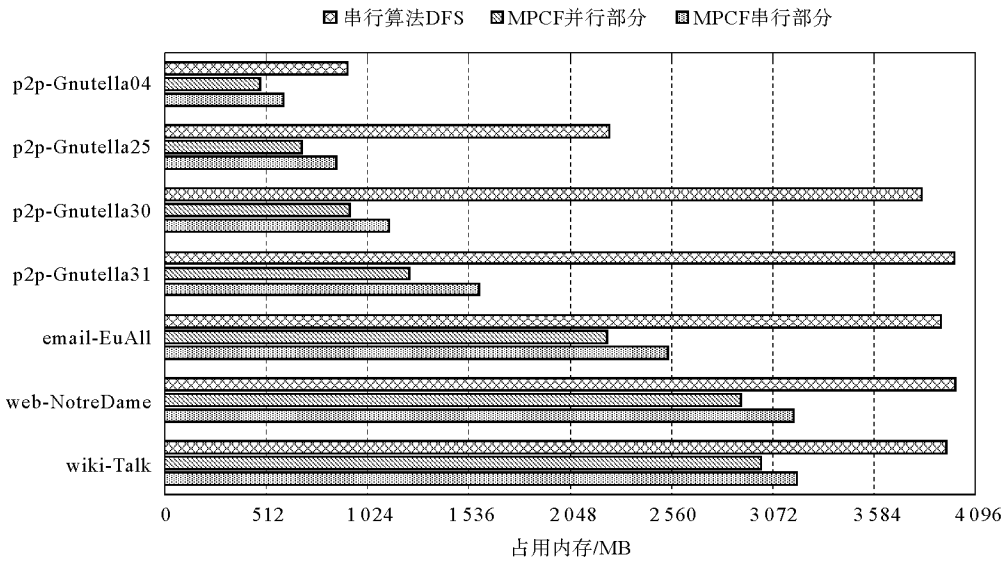

图 1　算法内存分析

Fig. 1　Memory contrast of algorithms

5　总结

针对传统基于 DFS 思想的算法无法在普通 PC 上求解大规模稀疏图所有有向回路的问题,通过对图数据集的分析,提出了一种基于多线程的并行求解算法。同时,为了测试此种求解方式的可行性,本文使用真实社交网络数据进行实验,结果表明此算法可在普通 PC 上求得大规模有向稀疏图的所有回路。本文所提出的算法同样适用于无向图。下一步,我们将进一步研究在多核、众核处理器上处理更大规模图的算法。

参考文献:

[1]SILVA J L C,ROCHA L,SILVA B C H. A new algorithm for finding all tours and Hamiltonian circuits in graphs[J]. IEEE Latin America Transactions,2016,14(2):831-836.

[2]LI B,XIONG L,YIN J. Large degree vertices in longest cycles of graphs,I[J]. Discussiones Mathematicae Graph Theory,2016,36(2):363-382.

[3]YUSTER R. A shortest cycle for each vertex of a graph[J]. Information Processing Letters,2011,111(21-22):1057-1061.

[4]PAULUSMA D,YOSHIMOTO K. Cycles through specified vertices in triangle-free graphs[J]. Discussiones Mathematicae Graph Theory,2007,27(1):179-191.

[5]王玉英,陈平,苏旸. 生成有向图中全部简单回路的一种新算法[J]. 山西师范大学学报(自然科学版),2008,36(4):12-15. WANG Yuying,CHEN Ping,SU Yang. A new algorithm to find all elementary circuits of a directed graph[J]. Journal of Shaanxi Normal University(Natural Science),2008,36(4):12-15.

[6]REIF J H. Depth-first search is inherently sequential[J]. Information Processing Letters,1985,20(5):229-234.

[7]MALEWICZ G,AUSTERN M H,BIK A J C,et al. Pregel:A system for large-scale graph processing[C]//ACM SIGMOD International Conference on Management of Data,2010:135-146.

[8]GONZALEZ J E,XIN R S,DAVE A,et al. GraphX:Graph processing in a distributed dataflow framework[C]//Proceedings of 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14),2014:599-613.

[9]SALIHOGLU S,WIDOM J. GPS:A graph processing system[C]//Proceedings of 25th International Conference on Scientif-

ic and Statistical Database Management. Baltimore, Maryland, USA, July 2013: Article No. 22.

[10]赵翔,李博,商海川,等. 一种改进的基于 BSP 的大图计算模型[J]. 计算机学报,2017,40(1):223-235.
ZHAO Xiang, LI Bo, SHANG Haichuan, et al. A revised BSP-based massive graph computation model[J]. Chinese Journal of Computers, 2017, 40(1): 223-235.

[11]罗由平,周召敏,李丽娟,等. 基于幂率分布的社交网络规律分析[J]. 计算机工程,2015,41(7):299-304.
LUO Youping, ZHOU Zhaomin, LI Lijuan, et al. Social network discipline analysis based on power-law distribution[J]. Computer Engineering, 2015, 41(7): 299-304.

[12]LESKOVEC J, KREVL A. SNAP datasets: Stanford large network dataset collection[DB/OL]. (2017-07-24) http://snap.stanford.edu/data, 2014.

(责任编辑:高丽华)

(上接第 31 页)

[7]ADRIANSYAH A. Aligning observed and modeled behavior[D]. Eindhoven: Eindhoven University of Technology, 2014: 129-166.

[8]AALST W M P V D, ADRIANSYAH A, DONGEN B V. Replaying history on process models for conformance checking and performance analysis[J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2012, 2(2): 182-192.

[9]BOENDER C G E. A bayesian analysis of the number of cells of a multinomial distribution[J]. The Statistician, 1983, 32(1/2): 240-248.

[10]DONGEN B F V, CARMONA J, CHATAIN T. A unified approach for measuring precision and generalization based on anti-alignments[M]. Berlin: Springer International Publishing, 2016.

[11]王路,杜玉越. 一种基于校准的模型问题域识别方法[J]. 山东科技大学学报(自然科学版),2015,34(1):42-46.
WANG Lu, DU Yuyue. An alignment-based identifying method of the problem areas within process models[J]. Journal of Shandong University of Science and Technology(Natural Science), 2015, 34(1): 42-46.

[12]DU Y Y, QI L, ZHOU M C. Analysis and application of logical Petri nets to E-commerce systems[J]. IEEE Transactions on Systems Man & Cybernetics Systems, 2014, 44(4): 468-481.

[13]DU Y Y, NING Y. Property analysis of logic Petri nets by marking reachability graphs[J]. Frontiers of Computer Science, 2014, 8(4): 684-692.

[14]吴哲辉. Petri 网导论[M]. 北京:机械工业出版社,2006.

[15] DONGEN B F V. BPI challenge 2012. Dataset [DB]. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

[16]MEDEIROS A K A D, WEIJTERS A J M M, Aalst W M PVD. Genetic process mining: An experimental evaluation[J]. Data Mining and Knowledge Discovery. 2007, 14(2): 245-304.

(责任编辑:傅 游)