第 43 卷 第 7 期 计 算 机 学 报 Vol. 43 No. 7
2020 年 7 月 CHINESE JOURNAL OF COMPUTERS July 2020

基于数据流的大图中频繁模式挖掘算法研究

汤小春 樊雪枫 周佳文 李战怀
（西北工业大学计算机学院 西安 710129）

摘 要 随着单个图数据规模的扩大以及应用领域的扩展，大规模单图的频繁模式挖掘的需求越来越强烈.传统的单机环境已经无法满足大规模图数据挖掘的要求，而现有的并行或者分布式环境下的挖掘方法，普遍受到并行性以及数据倾斜等问题的限制，论文在分析了现有的频繁模式挖掘算法后，提出了一种基于数据流的单个大图频繁模式挖掘方法.首先，建立基于数据流的频繁模式挖掘模型，将 MapReduce 模型中的“批”数据变成“微批”数据，提高了数据处理的并行度，并且其迭代方式也满足频繁子图挖掘的反单调性；其二，设计了数据流模型中的频繁模式检查、子图实例扩展以及正规编码计算等操作，实现了基于数据流模型的频繁模式挖掘算法；其三，为解决正规编码计算中的复杂性问题，提出了基于不变关系的正规编码计算策略以及基于编码树的优化策略，优化正规编码比未优化编码的计算性能提升了 30%，基于编码树的优化策略比原始编码计算策略在性能上提升了 10%；最后，对涉及的相关算法进行了实验测试，实验证明，算法提高了频繁模式挖掘的并行性，大幅度减少了大图的搜索空间，降低了正规编码的计算时间，相比于传统算法大规模单图中频繁模式挖掘的效率提升了 30%.

关键词 图挖掘；频繁模式；数据流模型；并行算法；编码树
中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2020.01293

An Algorithm Based on Dataflow Model for Mining Frequent Patterns from a Large Graph

TANG Xiao-Chun FAN Xue-Feng ZHOU Jia-Wen LI Zhan-Huai
(*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710129)

Abstract Big graph data mining has been highly motivated not only by the tremendously increasing size of graphs but also by its large number of applications, such as bioinformatics, chemoinformatics, and social networks. One of the most challenging tasks in big graph mining is pattern mining. These tasks consist on using data mining algorithms to discover interesting, unexpected and useful patterns in large amounts of graph data. Several algorithms exist for frequent pattern mining, but they are mainly used on centralized computing systems and evaluated on relatively small datasets. While modern graphs are growing dramatically, several parallel and distributed solutions have been proposed to solve this problem. However, those methods do not have better performance in scalability and balancing. So that we propose an algorithm based on dataflow model for mining frequent patterns in a large single graph. We construct a dataflow model for Mining frequent patterns, which include three operators: IsFrequent, Expand and Code. At first, the frequent pattern mining method based on dataflow model separates large graph into many micro graphs and has following advantages. These micro graphs can be expanded and calculated simultaneously, because they are independent of each other. At the same time, since each iteration is based on the subgraph instance generated in the previous iteration, only one vertex or one edge needs to be

收稿日期：2018-03-28；在线发布日期：2019-05-30.本课题得到科技部云计算与大数据重点专项（2018YFB1003403）资助.**汤小春**，博士，副教授，主要研究方向为图数据管理、分布式计算等.E-mail：tangxc@nwpu.edu.cn.**樊雪枫**，研究生，主要研究方向为大数据计算.**周佳文**，研究生，主要研究方向为大数据计算.**李战怀**，博士，教授，中国计算机学会(CCF)会员，主要研究领域为海量数据管理、大数据计算等.

extended, it decreases the generation of redundant subgraph in Expand operator. Secondly, we propose a regular code computing strategy based on invariant relation and an optimization strategy based on coding tree. These two approaches solve the problem that it is difficult to calculate the regular code. The results show that our regular code computing strategy improves performance by 30% over the original approach and our optimization strategy improves performance by 10% over the original strategy. Thirdly, we design operators of checking frequent pattern using micro batch data. After the large batch data is decomposed into multiple micro batch data, each micro batch data can be regarded as a single processing unit, a lot of tasks can be generated concurrently, which reduce data skew. These micro batch data can be iteratively computed more easily in parallel computing. And its iterative approach also satisfies the anti-monotonicity of frequent patterns mining. At last, the algorithm of frequent pattern mining is implemented. The experiments on our cluster show that the algorithm can effectively process a variety of large graphs with millions of vertices and tens of millions of frequent pattern mining, and scales well with the degree of available parallelism.

Keywords graph mining; frequent pattern; dataflow model; parallel algorithm; coding tree

《计算机学报》

1 引 言

频繁模式挖掘是从图集或者单图中找到支持度大于某个阈值的所有子图的过程,是图数据挖掘算法研究中的一个主要分支,广泛应用于图的分类[1]、用户兴趣建模[2]、图的聚类[3]、数据库设计[4]、索引的选择[5]以及生物医学[6]等领域.依据输入数据类型可分为单图和图集上的频繁模式挖掘[7],两者的区别在于支持度的计算方法的不同,如果输入数据是图集[8-11],当某个候选模式至少与图集中 τ 个以上的图存在子图同构,则该候选模式是一个频繁模式;如果输入数据是单个的连通图[12-14],当某个候选模式在输入图中至少存在 τ 个不同的同构子图,也称包含 τ 个子图实例时,该候选模式被认定为频繁模式.

在频繁模式挖掘算法中,传统方法是先计算候选模式的可能空间,再针对每种子图模式查找其在输入图中的所有子图实例.最后依据真假值判断该模式是否为频繁模式.由于查找子图模式需要判断子图同构,而判断子图同构是 NP 完全问题[15],因此算法的计算代价非常大.早期频繁模式挖掘中的图数据规模较小,采用单机处理基本能够满足要求.随着图数据规模的扩大,单个计算节点已无法满足挖掘过程对计算能力的要求,也无法满足挖掘过程的内存需求,因此频繁模式的挖掘进入分布式和并行处理阶段.

现有的分布式或并行算法基本采用两种方式.一种方式与单机环境下的频繁模式挖掘思想类似,即先产生模式,并行计算每个候选模式的频繁次数[16-18].其过程包括两个阶段,(1)模式产生阶段[18].首先针对一个输入图数据,在其模式的空间 X 中枚举可能的候选模式,为下一步的迭代处理提供依据;候选模式的产生主要使用基于 Apriori[5]方法和模式增长[9]的方法.另外,为了防止冗余模式的产生,需要使用图的同构算法以保证模式不重复出现;(2)频繁次数的计算阶段[16-17].针对每个模式,采用并行方法检查其在输入图数据中是否存在同构的子图实例.即,扫描整个图数据,找出模式在图数据中满足条件的全部子图实例,检查子图实例的出现次数是否满足支持度阈值条件.上述方法中,在从大小为 k 的频繁模式向大小为 $k+1$ 的候选模式扩展的过程中,为了防止自同构和冗余候选模式的出现,算法需要进行串行计算,因此难以实现并行化.另外,在阶段(2)计算中,对于单个大图,其并行计算能力也较弱.

第二种方式是枚举-归纳方法[13,19],其输入是图数据和指定的模式,即对于一个指定的模式 P,在图数据中枚举可能的子图实例 X,针对 $x\in X$,使用布尔函数 q,如果存在 $q(x)=\text{true}$,即当子图实例满足限制条件 q 时,其结果为真,说明模式在图数据中存在一个子图实例与模式同构.由于子图枚举的代价非常大,所以这种方法无法利用频繁子图挖掘的反单调性的特点.故文献[13]只支持指定子图大小的

挖掘，即需要输入子图模式；文献[19]采用两阶段方法，第一个阶段通过概率方法，得到可能的模式空间；第二阶段针对每个模式，在输入图数据中找到所有大小为 k 的连通子图，进行归并，计算支持度，最后得到所有大小为 k 的频繁模式. 这种方式的优点是并行性非常好，适合分布式计算，但是缺点是精度问题以及冗余计算多.

在计算框架上，一部分文献中采用了 MPI[19-21] 或者 MapReduce、Pregel 等大数据计算框架[22-25]，另外一部分使用自定义的图数据计算框架[26]. 前者存在以下问题：(1) 模式产生过程的并行性差. 产生子图模式的过程中，为了保证子图模式的唯一性，只能以串行的方式执行；(2) 子图查询代价大. 针对每个子图模式，需要遍历一次图数据，找到所有可能的子图实例，并检查这些子图实例是否与模式同构. 高昂的同构计算代价再加上输入的图数据规模较大，因此，计算出正确的结果是非常困难的；(3) 通信代价较大. Map 任务与 Reduce 任务之间的数据传输开销大，而使用 BSP 编程模式时会受到节点内存的限制；(4) 数据倾斜问题. Map 任务或者 Reduce 任务大小不同，导致任务结束时间严重的不一致. 上述问题影响了现有的大数据处理框架在大规模图上进行频繁模式挖掘的效率，所以一部分研究者将 Pregel 的“Think Like a Vertex”变成“Think Like a Pattern”的计算方式，虽然解决了部分问题，但是存在平台专用的限制，通用性不足.

本论文主要研究大规模单图中的频繁子模式的挖掘，提出了基于 dataflow 计算模型[27-28] 的挖掘方法，将 MapReduce 编程模型中的“批”变成“微批”，提高了并发性，降低了数据倾斜程度. 该方法采用数据流方式，将频繁模式挖掘过程分解为三个不同的操作算子，操作算子之间的数据为“小图流”的集合. 并行处理每个“小图”，对这些“小图”实施三个操作算子，得到新的“小图流”，然后对新的“小图”继续实施操作算子，循环执行直到达到要求为止. 由于后一次迭代使用前一次的结果，因此该算法能够满足频繁子图挖掘的反单调性要求.

基于数据流模型的频繁模式挖掘算法，利用“小图”数据本身具有的并发性，采用流水方式[29]，不但有效地提高了处理的并发度，而且极大地提升了能够处理的图数据的规模. 算法可以使用 Spark[30] 或者 Flink[27] 等通用的大数据处理框架，能够很好地实现大规模单图中的频繁模式挖掘.

基于数据流模型的频繁模式挖掘方法有以下 5 个优点：(1) 并行计算正规编码. 因为“小图”之间不存在关联关系，所以可以采用并行的方式进行正规编码计算和归纳操作；(2) 并行扩展新的子图实例. 频繁模式中的每个“小图”之间不存在关联关系，所以可以并行地扩展大小为 $k+1$ 的小图；(3) 减少图数据的遍历. 由于每次迭代都是在上一次迭代产生的子图实例的基础上进行，因此只需要扩展一个顶点或者一条边，就可以得到此次迭代的全部子图实例，从而减少了遍历的代价；(4) 提高了数据的并行性. 大图分解为多个的小图后，每个小图可以当作一个处理单元，数据自身的并行性得到充分的体现，有利于采用并行计算框架进行计算；(5) 流水处理模式使得大规模图数据的频繁模式挖掘成为可能.

但是，本文提出的挖掘算法存在的最大问题是子图实例的存储，如何保存此次迭代产生的子图实例使之用于下一次的迭代是一个极大的挑战. 从输入的图数据中组合各种子图实例，其规模是非常庞大的，例如，对于大小为 k 的子图，最坏情况下连通子图数量可以达到 $C_n^k(1-2n2^{-n}+o(2^{-n}))$，因此存储量非常巨大. 但是，随着分布式存储系统，如 Kafka、HDFS 等的出现，海量数据的快速缓存成为可能，可以有效缓解现有的存储问题.

论文的主要工作有：

(1) 提出了基于数据流模型的频繁模式挖掘方法，利用微批模式，提高了处理的并行性和能够处理的图数据的规模；

(2) 分析了现有的图的正规编码计算方法，对子图的正规编码计算方法进行了优化；

(3) 提出了子图实例的数据结构以及频繁模式挖掘中的流水操作算法；

(4) 采用编码树的思想，大大降低了正规编码计算的规模；

(5) 可以使用现有的基于 dataflow 模型的大数据处理平台(Spark 等)进行分析处理.

本文第 2 节介绍频繁模式挖掘的研究现状；第 3 节介绍论文中使用的相关概念；第 4 节介绍基于数据流的频繁模式挖掘模型；第 5 节介绍正规编码计算的优化策略；第 6 节描述实验性能测试；第 7 节是结论.

2　相关的工作介绍

2.1　单机环境的频繁模式挖掘

单机频繁模式挖掘包含两个步骤，候选模式

《计算机学报》

产生和支持度计算.候选模式的产生主要有两种策略,一是水平方向的连接,二是最右路径扩展.前者的策略是通过合并两个大小为k的频繁模式来形成大小为$k+1$的候选模式,如AGM(Apriori-based Graph Mining)算法[10]、FSG算法[8].后者采用的方法是将一个新的顶点添加到大小为k的模式的最右路径上,形成大小为$k+1$的候选模式,如gSpan(graph-based Substructure pattern mining)算法[12]、CloseGraph(Closed Graph patern mining)算法[31]、FFSM算法[23].其它的一些策略包括扩展合并、从右到左的树合并以及基于扩展的等价类等,也出现在一些文献中,但它们不够具有典型性.

待处理的输入数据是由许多小的图组成的集合时,每个图是连通的,但图之间不连通.频繁模式支持度的计算方法非常简单,即对图集中的每一个图,如果与模式存在子图同构,则计数值增加1,检查完整个图集后,根据计数值确定模式是否频繁.

在单图的频繁模式挖掘中,待处理的输入数据是一个大的连通图,频繁模式的计算方法与图集大相径庭.为了保证频繁模式大小的反单调性,单个图中的频繁模式支持度计算方法主要采用MNI[32]、MIS[33]以及HO[34]等方法.SIGRAM[35]使用MIS方法来计算频繁模式的支持度,为了提高支持度计算的效率,SIGRAM需要存储中间数据,即需要存储模式在输入图中的全部实例,由于一个模式的实例非常多,因此存储中间数据需要使用大量的内存空间.GRAMI[36]采用CSP方法,一定程序上减少了存储空间,但是增加了计算代价.

随着多核CPU的使用,采用多核CPU进行频繁模式挖掘的方法开始出现.在文献[37]中,对传统的数据挖掘算法SUBDUE进行了并行化处理,输出了近似的结果.文献[31]对SIGRAM算法进行了并行化,该系统继承了SIGRAM中的限制,即只支持小的稀疏图.虽然这样的系统能充分利用CPU资源,但是,单个机器存在内存限制问题,使其无法应用在大规模图数据上.

2.2 分布式环境的频繁模式挖掘

随着图集规模的扩大,单个机器环境下的频繁模式挖掘逐渐被分布式环境取代.对于图集,挖掘方法基本上都是借助MapReduce编程模型,先将图集分散到各个节点,之后进行本地挖掘,最后进行全局的归并计算.文献[22]采用过滤和提炼的方法.首先将图集分散到集群节点上,然后在各个节点上挖掘候选子图模式,最后对候选子图模式进行聚合,得到最终的频繁模式.文献[23]采用迭代MapReduce方式进行频繁模式的挖掘.首先将图集分散到各个节点上,每个节点上的Map任务读取HDFS上的大小为$k-1$的子图模式;扩展成大小为k的候选模式后,检查本地图集中候选模式的出现次数;最后将候选模式以及出现次数发送给Reduce任务,计算全局的支持度,并将频繁模式写入HDFS;不断递归,直到挖掘出全部的频繁模式.

对于单个大图,MapReduce编程模型存在一定的局限性[13].文献[26]实现了专用的处理框架Arabesque,并且提出了以模式为中心的编程模型,目的在于解决频繁模式挖掘中的迭代问题.它将频繁模式挖掘分为两个阶段:过滤和处理.过滤过程确定一个子图的出现是否需要处理,而处理过程则处理子图的出现并产生输出.Arabesque与Pregel非常相似,但是它不是以顶点为中心,而是以模式为中心.它存在的问题是在进行频繁模式挖掘时,aggregate计算和process计算之间子图实例的网络通信开销过大,并且数据存储代价太大.文献[13]提出了MRSUB方法来挖掘频繁模式,它也是基于MapReduce编程模型.每个Map任务根据分配到的边,在输入图数据中查找包含该边的大小为k的子图实例,以每个子图实例的正规标签作为key,将子图发送给Reduce任务,Reduce任务中计算支持度.这种方法的缺点有两个,一是Map和Reduce之间的Reduce copy过程造成磁盘I/O开销大;另外一个是仅支持指定大小的子图模式的挖掘,无法迭代.文献[38]采用BSP模型,利用Pregel大数据处理框架实现了频繁模式挖掘算法Pegi.它使用粗粒度和细粒度相结合的方式,master节点控制挖掘进程,slave节点负责子图实例的发现.Pegi利用聚集器来同步master机器和slave机器的信息流.其缺点是需要查找全部的子图实例才能计算支持度和子图的扩展部分.子图实例会随着图数据的增长而呈指数增长,会增加每台机器的内存开销,由于Pregel无法扩展内存因此无法提高数据处理的规模.另外,由于无法确保运行中顶点的分布,会产生系统负荷不平衡的问题.文献[19]采用MPI编程模型实现了单图的频繁模式挖掘.该算法包含2个步骤:(1)利用概率的方法确定候选模式;(2)频繁次数计算.由于采用概率模型,正确性和精度受到一定的限制.

以上单个大图的频繁模式挖掘中,有使用MPI、MapReduce及BSP模型,也有使用自定义模型.使用MPI模型,候选模式的生成是瓶颈.采用MapReduce

《计算机学报》

计算模型，Map 任务中会出现数据倾斜问题，会影响系统的总体性能，另外由于 Map 和 Reduce 之间的通信数据量太大，也会影响系统加速值. 采用 BSP 计算模型，由于每个 Worker 节点内存的限制，使得所能处理图数据的规模无法更大. 采用自定义模型，如过滤-处理计算模式，由于大规模子图实例存储问题以及过滤和处理之间的数据传输代价导致无法进一步扩大计算规模，另外其框架的通用性较差.

3 问题描述

从单图中挖掘频繁模式时，子图同构是一个 NP 问题. 另外，由于子图实例之间存在重叠，不同的支持度计算方法精确度不一样，因此频繁模式支持度的计算是另外一个难点. 下面先给出这两个方面的描述，然后说明频繁模式挖掘问题.

3.1 图同构的检测

定义 1. 假设存在标签图 G，其表示为五元组 $G=(V,E,\Sigma_V,\Sigma_E,l)$，$V$ 是顶点的集合，E 是边的集合，Σ_V 和 Σ_E 分别代表顶点标签和边的标签. 标签函数 l 定义了映射 $V\to\Sigma_V$ 以及 $E\to\Sigma_E$. 不失一般性，我们认为 Σ_V 和 Σ_E 上存在偏序关系 $\geq$.

判断两个图 G 和 G' 是否同构，有两种判别方法. 第一种方式是一一映射，如果两个图的顶点之间存在一一映射，说明它们同构；第二种方式是正规编码计算，如果两个图的最大（最小）编码一样，说明它们同构.

定义 2. 假设存在标签图 $G=(V,E,\Sigma_V,\Sigma_E,l)$ 和 $G'=(V',E',\Sigma'_V,\Sigma'_E,l')$. 当且仅当存在映射 f 满足条件：(1) $\forall u\in V,(l(u)=l'(f(u)))$；(2) $\forall(u,v)\in V,((u,v)\in E\Leftrightarrow(f(u),f(v))\in E'$；(3) $\forall(u,v)\in E,(l(u,v)=l'(f(u),f(v)))$；则称 G 和 G' 是同构的.

如果 G 和 G' 是同构的，并且 G 和 G' 是同一个图，即 $G=G'$，那么称 G 是自同构.

定义 3. 标签图 G' 的编码. 对于标签图 G'，其顶点和边都有标签，标签值按字典序排序. 对图中的任意两个顶点 v_i 和 v_j，标签值存在关系 $num(l(v_i))\leq num(l(v_j))$，$num(l(v_i))\geq num(l(v_j))$. 图 G' 的邻接矩阵表示如下：

$$\boldsymbol{M}_{G'}=\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,k} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{k,1} & x_{k,2} & x_{k,3} & \cdots & x_{k,k} \end{pmatrix}.$$

$\boldsymbol{M}_{G'}$ 中的 $x_{1,1},x_{2,2},\cdots,x_{k,k}$ 代表顶点的标签值，而 $x_{1,2},x_{1,3},\cdots,x_{k-1,k}$，则代表边的标签值. 根据图 G' 的邻接矩阵，可以得到图 G' 的编码 $c(\boldsymbol{M}_{G'})=x_{1,1}x_{2,2}\cdots x_{k,k}x_{1,2}x_{1,3}x_{2,3}x_{1,4}\cdots x_{k-1,k}$，其值是通过扫描邻接矩阵的上三角元素的值而组成. 对于有向图，需要将 $x_{j,i}$ 添加到 $x_{i,j}$ 之后，其编码表示为 $c(\boldsymbol{M}_{G'})=x_{1,1}x_{2,2}\cdots x_{k,k}x_{1,2}x_{2,1}x_{1,3}x_{3,1}x_{2,3}\cdots x_{k-1,k}$.

定义 4. 正规编码. 给定一个图 G，通过交换顶点的次序可以得到一系列的邻接矩阵 $\boldsymbol{NM}_G=\{\boldsymbol{M}_G\}$，每个邻接矩阵都可以得到一个编码 $c(\boldsymbol{M}_G)$，选择字典序最大或者最小的 $c(\boldsymbol{M}_G)$ 来可以代表图 G，称 $c(\boldsymbol{M}_G)$ 是图 G 的正规编码.

$$cl(G)=\max_{\boldsymbol{M}_G\in\boldsymbol{NM}(G)}c(\boldsymbol{M}_G).$$

引理 1. 当图 G 和图 G' 中的标签按照相同的规则设定偏序关系，例如字符的字典序，计算可得图 G 和 G' 的正规编码分别为 $cl(\boldsymbol{M}_G)$ 和 $cl(\boldsymbol{M}_{G'})$. 图 G 和图 G' 是同构的，当且仅当 $cl(G')=cl(G)$ 成立.

证明. 若 G 和 G' 的正规编码相同，则其顶点的标签满足一一对应关系. 另外，其后续的边的标签反映出顶点之间存在边而且边的标签相同，其满足定义 2 中的一一映射，所以是同构的.

反过来，如果两个图是同构的，那么必然存在顶点的一一对应，而且边也存在着一一对应，因此，总可以找到两个完全一样的序列，使得其编码相同. 对编码排序后，其最小（最大）编码也一定相同. 证毕.

本文利用正规编码，比较两个图之间的关系，为图建立一个唯一标识码. 不考虑图的原始顶点次序和边的次序，我们通过两个图的正规编码判断两个图是否同构. 图的正规编码在频繁模式挖掘中非常重要，但是它的计算过程复杂，几乎和子图同构一样，也是一个 NP 完全问题[9].

推论 1. 对于一个图 $G(V,E)$，即使图中顶点的次序和边的次序发生变化，$cl(G)$ 不变.

证明. 假设图中顶点或者边的次序发生变化，得到一个比 $cl(G)$ 还大的编码，此时显然与正规编码的最大编码定义矛盾，所以不存在这样的比 $cl(G)$ 大的编码. 证毕.

3.2 支持度计算

定义 5. 假设存在标签图 $G=(V,E,\Sigma_V,\Sigma_E,l)$ 和 $G'=(V',E',\Sigma'_V,\Sigma'_E,l')$，$G$ 是 G' 的子图，即满足条件：(1) $V\subseteq V'$；(2) $\forall u\in V,(l(u)=l'(u))$；(3) $E\subseteq E'$；(4) $\forall(u,v)\in E,(l(u,v)=l'(u,v))$. 如果有一任意的标签图 P，且 P 和 G 是同构的，则称 P 是 G' 的子图模式，记为 $P\subseteq G'$，称 G 为 G' 的子图

《计算机学报》

实例.例如图1中,图(g)是输入的图数据,子图(g1)和(g2)代表两种模式,子图实例是指定模式在输入图数据中的所有子图同构情况,子图实例的个数表示频繁次数.对于子图(g2),在图(g)中有两个子图实例,一个是由顶点1、2和4组成,另一个由顶点1、3和4组成.

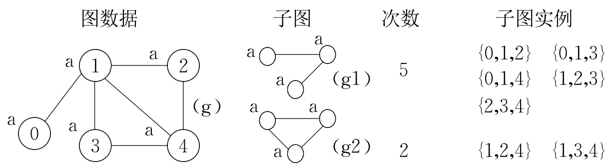

图1 子图模式及实例

如果图$G(V,E)$中的任意一对顶点之间都存在一条路径,那么称G是连通的.如果$G'(V',E')$是$G(V,E)$的子图,并且对于$\forall u,v\in V'$,当存在$(u,v)\in E$时,也存在$(u,v)\in E'$,那么就称G是G'的一个导出子图.

如果存在两个图P和G',G'上的任意子图与P是同构的,则存在一个映射f,P在G'上的全部子图同构记为$F=\{f_1,f_2,\cdots,f_k\}$,F称为全部的子图实例,$|F|$称为子图实例的个数.如图1中的(g1),在图(g)中存在5个映射,$\{0,1,2\}$,$\{0,1,3\}$等都是子图(g1)的子图实例.

数据挖掘中,反单调性是非常重要的原则之一,它可以用来有效地进行剪枝,提高挖掘的效率.如果不能保证反单调性,就必须采用穷举搜索.当计算子图模式的支持度时,最直接的方法是计算子图模式在图数据中子图实例的个数,但是这种计算方法有可能破坏反单调性,例如,在图2中,当子图是单个顶点db时,它的子图实例个数为1,即图2(a)中的顶点2.当我们给出的子图为图2(b)时,它的子图实例个数为2,即图2(a)中$\{1,2\}$和$\{2,3\}$.随着子图模式规模的增加,出现了子图实例个数也增加的现象,所以它不满足反单调性.目前,有三种保证反单调性的支持度计算方法,基于最小像集(MNI)方法、有害重叠(HO)方法以及最大独立集(MIS)方法[8].三种方法的区别在于两个不同的子图实例中允许顶点或者边重叠的程度.计算方法不同,计算的复杂度也不一样.本论文采用MNI方法,因其计算是最简单的,而HO和MIS方法是NP完全的.

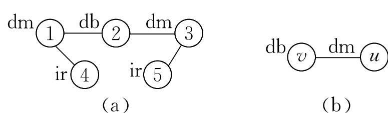

图2 支持度计算的例子

定义6. 基于像集的支持度.存在子图P在图G的全部子图同构映射$F=\{f_1,f_2,\cdots,f_k\}$,假设$F(v)=\{f_1(v),f_2(v),\cdots,f_k(v)\}$是一个像集,它包含图$P$中不同的且存在于图$G$中的顶点$v$.子图$P$在图$G$中的支持度表示为$s_G(P)$,其定义为

$$s_G(P)=\min\{t|t=|F(v)|\ \text{for all}\ v\in V_P\}.$$

图2(b)中的顶点v,对应于图2(a)中的2,即$F(v)=\{2\}$,而图2(b)中的顶点u,对应于图2(a)中的1和3,即$F(u)=\{1,3\}$,此时$s_G(P)$就为$|F(v)|$,即支持度为1.后文中不出现混淆的情况下,基于像集的支持度都简称为支持度.

3.3 频繁模式挖掘问题

给定一个标签图G以及最小支持度δ,频繁模式挖掘就是找出所有的模式P,使得模式P在图G中的映射数量超过δ,记为$s_G(P)\geq\delta$.由于每个子图模式中可能存在相同标签的顶点或者边,因此本文增加了系统的建模能力.若一个图中所有顶点和边具有唯一的标签,则非常容易能够将一组顶点或者边聚集形成一个模式,此时可以使用传统的基于Apriori的算法来挖掘频繁模式.但是,如果图中存在多个顶点或者边具有相同的标签,就无法使用基于Apriori的算法,只能通过图的同构方法解决问题.

问题1. 频繁连通子图挖掘问题.

如果只关注子图的连通性,则挖掘出的频繁模式是由某些相关联的边组成,每条边有两个顶点.依据此原则,本文不需要考虑非连通子图的挖掘.

问题2. 频繁导出子图挖掘问题.

如果要求挖掘出的子图是导出子图,则挖掘出的频繁模式由某些顶点组成,对于任何一对顶点,如果在原始图数据中存在边,频繁模式之间也必须存在边.

由于连通子图挖掘和导出子图挖掘的方法基本一样,唯一区别是前者针对边,后者针对顶点,因此本文只讨论导出子图的挖掘问题.

4 基于数据流的频繁模式挖掘模型

在频繁模式挖掘过程中,当挖掘大小为k的频繁模式时,需要检查图中大小为k的所有子图.按照这种思想,我们将大图分解为多个大小为k的子图,之后并行检查这些子图是否同构,对同构的子图进行聚集并计算其支持度,最后对这些子图进行扩展得到大小为$k+1$的子图,接着进行同构计算、聚集计算支持度,通过不断迭代,从而得到全部的频繁模

式. 但是，上述方法存在一个问题，随着 k 的增加，子图个数成几何级数增长，为解决该问题我们计算大小为 k 的模式时，只对大小为 $k-1$ 的频繁模式进行扩展. 由于对每个子图进行计算的过程中不需要其它子图，因此我们可以将这些子图看作数据流，子图同构、扩展、支持度计算看作变换操作，故本文提出的基于数据流的频繁模式挖掘模型由以下步骤组成：

(1) 初始数据流计算. 每一条边作为一个子图实例，挖掘模式大小为 2 的全部频繁模式；

(2) 扩展阶段. 扩展每一个大小为 $k-1$ 的模式的子图实例，得到全部的大小为 k 的子图实例；

(3) 编码计算阶段. 使用正规编码计算方法，计算各个子图实例的正规编码；

(4) 归并阶段. 将正规码相同的子图实例聚集；

(5) 频繁模式检查阶段. 检查每一类的子图实例构成的子图模式的支持度是否满足最小支持度的要求，将满足支持度的子图模式的全部子图实例保存，转到步骤(2)继续迭代；如果不存在满足支持度的子图模式，则计算结束.

4.1　子图的产生及存储

子图实例是模式在图数据上映射的像集，其包含顶点信息、边的信息以及顶点和边的标签信息，可抽象为 $inst^k=(cl,V_{inst})$，其中 cl 表子图实例对应的模式的正规编码，V_{inst} 代表子图实例中包含的顶点编号的集合，顶点编号即输入图数据 G 中顶点的编号，即 $V_{inst}\subset V$，$k=|V_{inst}|$ 代表子图实例对应的模式的大小. 大小为 k 的子图实例集合表示为 $INST^k=\{inst_1^k,inst_2^k,\cdots,inst_n^k\}$.

子图模式作为从输入图中挖掘出的结果，它是对子图实例的抽象，记为 $p^k=(cl,\Sigma_p)$，其中 cl 代表子图模式的正规编码，Σ_p 代表子图模式中包含的顶点标签，即 $\Sigma_p\subseteq\Sigma_V$，$k$ 代表子图模式的大小，其中 $k=|\Sigma_p|$. 大小为 k 的模式的集合表示为 $P^k=\{p_1^k,p_2^k,\cdots,p_n^k\}$.

若子图模式不是导出子图，将子图实例以及子图模式中的顶点集合 V_{inst} 变成边的集合 E_{inst}，即代表边的编号和边的标签信息. 后文中不作特别说明的情况下，只关注导出子图.

定义 7.　假设 $P=(V_P,E_P,\Sigma_P,\Sigma_P,l)$ 是图 $G=(V,E,\Sigma_V,\Sigma_E,l)$ 中的一个子模式，P 在图 G 中的像集可以表示为 $I_p=(cl(p),V_P,E_P,X,D)$ 表示，其中 $cl(p)$ 代表子图模式 P 的编码，而 X 代表顶点 V_P 的域，即 V_P 在图 G 中的顶点编号的取值范围，D 代表边 E_P 的域，即 E_P 在图 G 中的边的编号的取值范围. 子图实例的集合表示为表 1 的格式. 表 1 中顶点对应的列表示子图模式中的第 i 个顶点在图 G 中的像集，表 1 中包含两个子图模式 x 和 y，正规编码为 $cl(x)$ 和 $cl(y)$. 每一行代表一个子图实例，第一列表示模式 x 和 y 中的顶点在图 G 中对应的顶点标识的集合，同样 v_k 表示模式的另外一个顶点在图 G 中对应的顶点标识的集合. 如果用 $D(v_1)$ 表示子图模式 P 中顶点 v_1 的不重复的可能取值，那么 $D(v_1)$ 就是顶点 v_1 的值域，同理 $D(v_k)$ 表示顶点 v_k 的值域；列 $e_1,\cdots,e_m$ 代表子图模式中的边，用 $D(e_1)$ 表示子图模式 P 中边 e_1 的值域，$D(e_m)$ 表示图模式 P 中第 m 条边 e_m 的值域. 表 1 中的分量 $id_1,id_l,id_m,id_n,\cdots$ 等代表图 G 中的顶点标识，即顶点的唯一 id，$(id_l,id_k),\cdots$ 等代表图 G 中的边.

表 1　子图实例表

顶点			边			模式正规编码
v_1	…	v_k	e_1	…	e_m	
id_1	…	id_n	(id_1,id_k)	…	(id_k,id_l)	$cl(x)$
id_l	…	id_m	(id_l,id_k)	…	(id_k,id_j)	$cl(y)$

在频繁模式挖掘算法中，随着图数据规模的增大，子图模式的规模随之扩大，每个模式对应的子图实例数量更是呈指数级别增长，因而导致计算量以及中间数据的急剧膨胀. 例如，对于任何一个有 n 个顶点的图 G 的一个子图模式 P，假设子图模式的大小为 k，最坏情况下其子图实例的存储规模为 $n^k\times(n(n-1)/2)^k$，若只存储子图实例的边，其存储规模为 $(n(n-1)/2)^k$，不能够只存储子图实例的顶点，因为其会造成边的信息不足而无法满足挖掘要求. 例如图数据 CiteSeer，其顶点数为 3312，边数为 4591，标签数为 6，平均度为 2.8，当模式的大小为 6 时，其子图实例个数约为 10^9.

虽然子图实例的规模非常大，但是我们必须将其保存下来，原因有两点：第一，计算支持度时需要子图实例. 单个大图数据中计算支持度比较困难，因为各个子图实例之间存在着顶点的重叠，因此在计算支持度时，必须要访问所有子图实例，否则无法计算. 第二，缩小子图实例的扩展范围. 本论文提出的算法采用扩展方法得到更大的子图模式，因此需要将上次迭代产生的频繁模式全部保存，作为下次迭代的输入数据. 这种方式虽然会带来子图实例存储

《计算机学报》

的压力，但是能够有效地减少子图实例的模式空间，从而提高算法的效率. 本文采用 HDFS 来存储计算过程中产生的所有子图实例.

4.2 数据操作

基于数据流模型的频繁模式挖掘算法包括一个数据输入操作、一个增量更新操作以及三个流水操作，三个流水操作分别是判断操作、将大小为 k 的子图实例扩展成大小为 $k+1$ 的子图实例的操作以及计算大小为 $k+1$ 的子图实例的正规编码操作. 本文在上述五个操作的基础上，实现了频繁模式的流水挖掘模型.

(1) 判断操作

判断操作对扩展操作的结果进行检查，即计算子图模式的支持度，去掉非频繁的子图模式；如果判断结果为不存在频繁模式，则迭代停止，否则执行扩展操作. IsFrequent 方法根据输入的子图模式及其对应的子图实例，检查条件 $s_G(P)\geq\tau$. 即对模式中的每一个标签，检查全部子图实例中无重复出现的顶点编号的次数，操作 1 代码的第 3 行，第 4 行对模式中的所有标签，找到对应的顶点数最小的值，若该最小值不小于设定的阈值 τ，则被检查模式为频繁模式，其对应的所有子图实例需要保存.

过程 1. 支持度计算.

输入：候选子图模式以及子图实例

输出：频繁模式对应的子图实例

```
IsFrequent(P^k, INST^k, τ) {
1. FOR(l=0; l<k; l++) {
2.    //统计每个属性列中无重复的顶点编号出现次数
3.    m[l]=distinct count(INST^k[i].V[l]);
4.     IF(min_l(m[l])≥τ)
5.       RETURN TRUE;
6. }
7. RETURN FALSE;
8. }
```

(2) 扩展操作

扩展操作的任务是扩大子图的大小，即对前一次(如大小为 k 的子图)模式中的子图实例，通过扩展顶点得到更大的子图实例(如大小为 $k+1$ 的子图). 对于图 $G(V,E)$ 中的一个顶点 v，其邻接顶点的集合表示为 $\Gamma(v)=\{u\mid(v,u)\in E\}$，对于一个顶点集合 $V'\subseteq V$，集合 V' 的邻接顶点集合表示为 $\Gamma(V')=\bigcup_{v\in V'}\Gamma(v)\backslash V'$.

对于扩展操作，若按照邻点关系扩展子图实例，可能出现大量的重复的子图实例. 例如，对图 1 中的图数据按照邻点关系扩展子图实例，会产生冗余子图实例，如图 3 中虚线框中的子图实例.

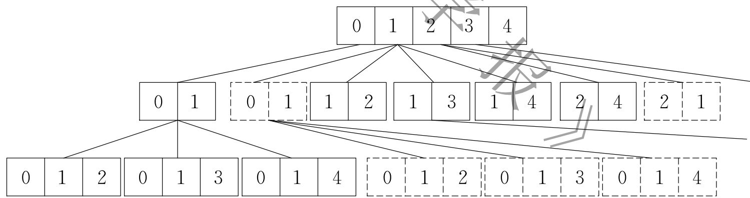

图 3 子图实例扩展

假设图 $G(V,E)$ 中包含 n 个顶点，每个顶点具有唯一编号，且顶点编号与顶点编号之间存在偏序关系 $v_1\prec v_2\prec\cdots\prec v_n$. 此时，假设一个子图实例所包含的顶点集合为 V'，其在上一次迭代中最后被扩展的顶点表示为 $last$，顶点 v 的可扩展邻接顶点集合表示为 $N(v)=\{u\mid(v,u)\in E\wedge v\succ last\}$，由此可得顶点集合 V' 的可扩展邻接顶点集合，将其表示为 $N(V')=\bigcup_{v\in V'}N(v)\backslash V'$. 通过偏序关系判断某个顶点是否为可扩展顶点，可以避免子图实例的冗余扩展问题，如图 3 中，不会扩展得到冗余的子图实例 $\{0,1\}$ 以及 $\{2,1\}$，此外，也不会得到大小为 3 的冗余的子图实例 $\{0,1,2\}$，$\{0,1,3\}$ 以及 $\{0,1,4\}$.

扩展操作代码中，第 2 行执行得到一个子图实例的全部候选可扩展邻接顶点集合. 第 3 行至第 8 行，在该子图实例的基础上扩展得到新的子图实例集合. 其中第 4 行和第 5 行，针对每一个候选顶点，按照其顶点标签以及边的信息，建立新的子图实例；第 7 行，将每个新的子图实例加入集合中，得到大小为 $k+1$ 的子图实例的集合. 第 9 行将结果返回.

过程 2. 子图实例扩展.

输入：大小为 k 的子图实例

输出：大小为 $k+1$ 的候选子图实例

```
Expand(k, inst, G){
1.  g', P^k, G^{k+1}, CAND;
2.  CAND={u|u∈N(inst.V)};
3.   FOR(w∈CAND){
4.      g'.V=inst.V∪{w};
5.      g'.E=G.E∪{(v,w)∈G.E∧v∈G.V};
6.      last=w;
```

7. 　$G^{k+1}=\cup\{g'\}$；

8. 　}

9. 　RETURN G^{k+1}；

10. }

考虑图1中的输入图数据，假如我们得到包含两个顶点的子图实例(0,1)，即一条边，它属于某个频繁模式，我们需要对它进行扩展，形成大小为3的子图实例，此时顶点0的邻接顶点是{1}，而顶点1的邻接顶点是{2,3,4}，所以经过扩展操作后，大小为3的子图实例包含{0,1,2}，{0,1,3}以及{0,1,4}.

(3)正规编码生成操作

因为在操作1的支持度计算过程中，需要将正规编码作为key，即将具有相同正规编码的子图实例全部聚集，以便计算支持度.所以，需要计算新的子图实例的正规编码.

过程3.　计算正规编码.

输入：大小为k的子图实例

输出：子图实例的正规编码

Code(g'){

1. $n=|g'.V|$；
2. FOR($i=1$ to n)
3. 　$s_i=i$；
4. $MaxCode=l(v[s_1])l(v[s_2])\cdots l(v[n])$ $l(v[s_1],v[s_2])\cdots l(v[n-1],v[n])$
5. FOR($i=2$ to $n!$){
6. 　$m=n-1$；
7. 　WHILE($s_m>s_{m+1}$)
8. 　　$m=m-1$；//从右向左找到第一个值减小的元素
9. 　$k=n$；
10. 　WHILE($s_m>s_k$)
11. 　　$k=k-1$；//从右向左找到第一个值超过s_m的元素
12. 　$swap(s_m,s_k)$；
13. 　$p=m+1$；
14. 　$q=n$；
15. 　WHILE($p<q$){
16. 　　$swap(s_p,s_q)$；
17. 　　$p=p+1$；
18. 　　$q=q-1$；
19. 　}
20. 　$Code=l(v[s_1])l(v[s_2])\cdots l(v[n])$ $l(v[s_1],v[s_2])\cdots l(v[n-1],v[n])$
21. 　IF($Code>MaxCode$)　$MaxCode=Code$；
22. }
23. RETURN $MaxCode$；
24. }

《计算机学报》

对于输入的子图实例g'，第2、3行将顶点的下标存入变量s_i中；第4行根据顶点的标签排列，得到顶点的编码；第5行至21行，计算出顶点的全部排列，并计算每种排列的编码，最后得到最大编码作为子图实例的正规编码.

(4) ExtractPattern操作

为了支持流水操作的触发事件，ExtractPattern操作为入口，子图模式的正规编码作为Key，从持久存储中获得Key相同的子图实例，触发一次新的迭代.

(5) SaveAsPattern操作

为了支持流水操作结果的增量更新模型，新扩展的子图实例按照模式的正规编码进行增量更新，将增量数据更新到以Key为标识的数据中.

4.3　基于数据流模型的频繁模式基本挖掘算法

基于数据流模型的频繁模式挖掘方法中，通过扩展操作得到新的子图模式及其对应的子图实例；通过子图实例判断操作判断子图模式是否频繁.对于频繁模式，对其子图实例进行扩展，否则就抛弃该子图模式及其所有实例.通过不断地迭代，直到没有新的频繁模式出现为止.图4给出了基于数据流的频繁模式挖掘模型.G为输入的图数据，执行IsFrequnet操作得到频繁的标签(第一次迭代时，也是最小的子图模式)，然后执行Expand操作对频繁模式的子图实例进行扩展，得到新的子图实例，最后对新的子图实例执行Code操作，计算其正规编码，并存储到分布式文件系统中.一次计算完成后，进行下一次的迭代，直到不存在频繁模式为止.

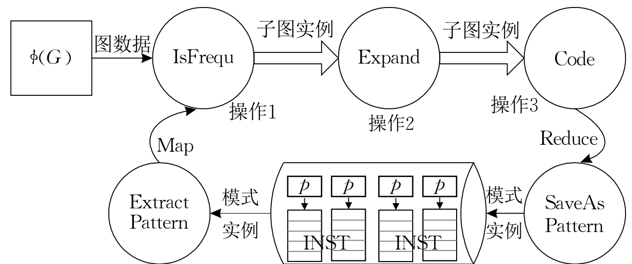

图4　基于数据流的频繁模式挖掘模型

上述挖掘模型中，输入图数据被分解为多个子图G_u^0，构成子图集合用符号$\phi(G)$表示.原始图数据$\phi(G)=\{G_u^0|u\in V(G)\}$，其中$V(G_u^0)=\{u\}\cup\Gamma(u)$，$E(G_u^0)=\{(u,v)|v\in\Gamma(u)\}\cup\{(v,w)|v,w\in\Gamma(u)\}$.

算法1实现了对小规模图数据进行基于数据流模型的频繁模式挖掘算法.算法1的第2行，检查顶点和边，如果其标签值出现次数低于阈值，将其从图G中删除，得到图G'；第4至6行实现了创建模式

大小为 2 的子图实例；第 7 和 8 行，调用计算支持度方法，即判断操作，得到频繁模式及其全部子图实例；第 9 行检查新的候选子图实例集合是否为空，若为空则退出；第 10 和 11 行执行扩展操作，得到扩大后的新的子图实例的集合；第 12 和第 13 行，聚集新的候选子图实例，得到全部的候选子图模式，然后转到第 7 行，继续迭代执行.

算法 1. 导出频繁模式挖掘.

输入：图数据 G，支持度 τ

输出：频繁导出子图

freqSubGMine($G(V,E)$){

1. $G', Temp^k, INST^k, P^k$
2. $G'=Init(G,\delta)$//删除 G 中不频繁标签对应的顶点和边
3. $k=2$；
4. WHILE $e\in G'.E$
5. WHILE $P[i]^k.cl=cl(e)$
6. $INST[i]^k=\bigcup(cl(e),e.v_1,e.v_2)$；
7. FOREACH $P[i]^k$ IN P^k
8. $Temp^k=\bigcup IsFrequ(P^k[i],INST^k[i],\delta)$；
9. IF $Temp^k=\varnothing$ GOTO 15；
10. FOREACH $inst\in Temp^k$
11. $INST^{k+1}=\bigcup Expand(k+1,inst,G')$；
12. FOREACH $inst\in INST^{k+1}$
13. $\langle P^{k+1},INST^{k+1}\rangle=\bigcup Code(k+1,inst,G')$；
14. GOTO 7；
15. RETURN；
16. }

4.4 基于数据流模型的频繁模式挖掘过程

由于采用数据流模型，所以对于大规模的图数据，可以利用现有的数据流处理引擎，如 Hadoop、Spark 等计算框架来进行处理. 下面主要介绍依据 Spark 计算框架设计的大规模图数据的频繁模式挖掘方法.

计算每个候选子图模式的支持度时，需要使用其对应的全部子图实例，若采用 Map/Reduce 编程模型，Map 任务执行完毕到 Reduce 任务开始执行的过程中，会执行 Reduce Copy 操作，该过程的磁盘 I/O 开销巨大；若仅仅使用 Map 任务进行计算，不执行 Reduce 操作，可能引起数据倾斜问题，某些子图模式对应的子图实例非常多，某些子图模式对应的子图实例非常少，从而导致计算结束时间延迟，影响下一次的迭代的开始，经分析我们采用 Spark 作为计算平台，其扩展操作产生的结果直接存储在 HDFS 分布式文件系统中，按照子图实例的正规编码进行分割，将正规编码作为 key 来分割文件内容.

挖掘大小为 $k+1$ 的频繁模式时，我们将上次迭代存储在 HDFS 中的所有大小为 k 的子图实例作为输入数据，通过判断操作计算子图模式的支持度，得到大小为 k 频繁模式并对其进行扩展，从而得到大小为 $k+1$ 的全部子图实例，将其存储在 HDFS 分布式文件中. 大小为 k 的子图实例的存储，其数据格式如表 1 所示. $v_1,v_2,\cdots,v_k$ 是模式的顶点，这些顶点组成的图是模式，$u_1,u_2,\cdots,u_k$ 是数据图中的顶点，是模式图中的顶点在数据图中的映射，即存在 $f_1(v_1v_2\cdots v_k)\rightarrow(u_1u_2\cdots u_k)$，$u_1,u_2,\cdots,u_k$ 是输入图的子图.

表 1 中，模式相同的子图实例具有相同的正规编码，每个子图模式包含多个子图实例，每个子图实例由 k 个顶点的标号组成. 因此，我们将模式的正规编码作为 rowkey，其包含的子图实例作为列存储.

依据表 1 的数据模型，图 5 给出了大小为 3 的频繁模式实例以及模式的例子. 按照正规编码，得到两种频繁模式 aaa110 和 aaa111. 对于模式 aaa110，它对应的子图实例都包含 3 个顶点，分别记为 x,y,z，其顶点的标签均为"a"，其对应的子图实例的个数为 5，每个子图实例通过顶点的编号表示，对应于图数据中的顶点{0,1,2}、{0,1,3}、{0,1,4}、{1,2,3}以及{2,3,4}. 而另外一个模式 aaa111，其对应的子图实例个数为 2，对应于图数据中的顶点{1,2,4}以及{1,3,4}.

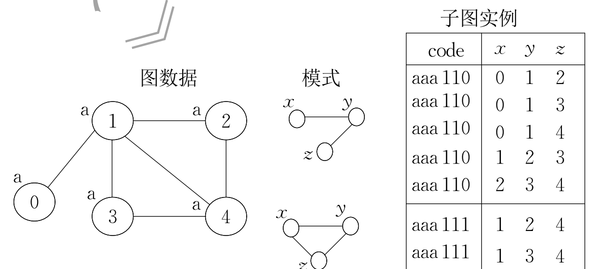

图 5 子图实例的存储模型

利用 Spark 计算框架，频繁模式挖掘过程中，主要的 RDD 和操作包括支持度计算、扩展子图实例以及计算正规编码.

(1) 支持度计算

调用操作 ExtractPattern，获得候选子图模式 $cl(g^k)$ 及其包含的全部子图实例，计算支持度，检查候选子图模式是否频繁.

计算过程中，将子图实例看作二维表，每个元组代表一个子图实例，它由子图实例所包含顶点的唯一编号表示，如 $\{u_3,u_5,\cdots,u_{l+1}\}$. 列属性是顶点的标签值，如模式大小为 k 的顶点标签分别为

《计算机学报》

$l(v_1), l(v_2), \cdots, l(v_k)$.

模式的子图实例的数据表示为 SOURCE $=\{cl(g^k), \{u_1, u_2, \cdots, u_k\}, \{\cdots\}, \{u_3, u_5, \cdots, u_{l+1}\}\}$.

对 SOURCE 数据执行 parallelize 操作，得到 RDD1 $=\{\{u_1, u_2, \cdots, u_k\}, \{\cdots\}, \{u_3, u_5, \cdots, u_{l+1}\}\}$.

对 RDD1 执行 flatMap()操作，将顶点映射成二元组$\{ID, u_i\}$，其中第一个成员是列标识，第二个成员是顶点标识，即 RDD2 $=\{(1, u_1), (2, u_2), \cdots, (k, u_k)\}$.

对 RDD2 执行 IsFrequent$(g^k, INST^k, \delta)$，对获得的结果进行过滤，如果 $\delta \geq \tau$，则模式 g^k 是频繁的，需要对其继续执行后续的计算，否则删除该模式.

（2）子图实例扩展

大小为 k 的频繁模式，依次枚举其子图实例，调用 Expand 操作，得到大小为 $k+1$ 的候选子图实例.

对 RDD1 $=\{\{u_1, u_2, \cdots, u_k\}, \{\cdots\}, \{u_3, u_5, \cdots, u_{l+1}\}\}$，执行 flatMap 操作，其操作算子为扩展操作，即 Expand$(k, inst, G)$，通过执行该操作我们可以得到 RDD2 $=\{\{u_1, u_2, \cdots, u_{k+1}\}, \{\cdots\}, \{u_3, u_5, \cdots, u_{l+1}\}\}$.

（3）正规编码计算

对于扩展操作得到的 RDD2，执行 Map 操作，其操作算子为计算正规编码操作，即 $cl(g^{k+1})$，得到 RDD3 $=\{\{cl(g^{k+1}), u_1, u_2, \cdots, u_{k+1}\}, \cdots\}$，其中子图实例的正规编码作为 Key，Value 为该正规编码对应的每一个子图实例 $u_1, u_2, \cdots, u_{k+1}$. 按照 Key 进行 reduce 操作后，调用 SaveAsPattern 方法将结果写入 HDFS 分布式文件.

以上 3 个过程执行完成后，就得到所有大小为 k 的频繁模式，同时也得到大小为 $k+1$ 的候选子图模式. 不断地迭代执行上述过程，可以得到图数据中的全部频繁模式.

4.5 算法的性能分析

频繁模式的检测（操作 1）中，需要检查每一个子图实例. 假设存在 n 个大小为 k 的子图模式，每个子图模式包含 m 个子图实例，一个大小为 k 的子图模式的支持度计算复杂度为 $O(mk)$，则 n 个大小为 k 的子图模式的支持度计算复杂度为 $O(nmk)$. 假设输入图数据中顶点的不同标签数量为 l，最坏情况下 n 的大小为 l^k，假设顶点数为$|V|$，最坏情况下 m 的大小为 $C_{|V|}^k$，所以频繁模式检测操作的复杂度与输入图数据中顶点的不同标签数量及顶点个数有关，即为 $O(l^k C_{|V|}^k k)$.

子图实例的扩展（操作 2）中，需要从大小为 k 的子图实例增加为大小为 $k+1$ 的子图实例. 对于一个大小为 k 的子图实例，其顶点的度为 d 的话，最坏情况下可以扩展的候选顶点为 kd 个，也就是说扩展一个子图实例的计算复杂度为 $O(kd)$，所以 n 个模式，每个模式 m 个子图实例的话，其计算复杂度为 $O(nmkd)$，所以子图实例扩展的计算复杂度与输入图的顶点个数、顶点的标签数量以及顶点的度有关，表示为 $O(l^k C_{|V|}^k k^2 d)$.

对于大小为 k 的所有子图实例，需要计算其正规编码（操作 3），以便得到其对应的模式. 该过程需要对 k 个顶点计算所有可能的排列，并从中找到编码最大的值，最坏情况下其复杂度为 $O(k!)$.

频繁模式挖掘过程的计算复杂度与顶点的数量$|V|$、输入图数据的稠密程度，即顶点的度 d、顶点不同标签的数量 l 以及频繁模式的大小 k 值有关.

实际中，影响计算复杂度的关键因素包括 k 值以及支持度阈值 τ. 随着 k 的增大，其复杂度按指数数量扩大. 随着支持度阈值 τ 的增大，可以大规模地减少子图模式的数量 n，即减小最坏情况下 l^k 的值，由于 l^k 与 k 是指数关系，因此减少子图模式的数量可以大幅降低计算的时间复杂度.

5 基于数据流的频繁模式挖掘算法的性能优化

5.1 基于标签值的正规编码计算优化

频繁模式挖掘算法中，计算正规编码用于检查子图实例是否是子图模式的映射，还可以用于检查两个子图模式是否相同，因此，设计一个高效的正规编码计算算法能减少频繁模式挖掘的计算代价.

图的正规编码是图的唯一表示，即如果两个图是同构的，则它们一定具有相同的正规编码. 计算图的正规编码包括两个步骤：（1）对图的顶点进行排序后得到其对应的邻接矩阵，将邻接矩阵转换为线性的符号序列，即将邻接矩阵的行按照次序转换为 0 和 1 的序列（这里将所有边的标签设置为“1”）. 如图 6 中的图数据包含 5 个顶点，边转换为 0 和 1 后的序列为“aaaaa1010100111”，其中前 6 个“a”代表顶点的标签，邻接矩阵的上三角按照列分别表示为“1”，“01”，“010”和“0111”. 当边的标签也存在时，用边的标签代替 1，顶点之间不存在边时，仍然记为 0.（2）求最大或者最小的符号序列. 图 6(a)中的序列不能直接作为正规编码，因为随着顶点次序的交换，可以得到很多个不同的字符序列，而且这些不同的

《计算机学报》

序列全部代表同一个图，所以它不是唯一的，违反了正规编码唯一代表图的前提条件. 通过不断交换顶点的次序得到其对应的邻接矩阵，按照字典序选择一个最大的字符序列或者最小的字符序列，将其作为正规编码使用. 对于图6(a)中的图数据，通过多次交换顶点次序可以得到其最大的字符序列，如图6(b)所示，其字符串序列为“aaaaa1111101000”，从字典序的关系看，该序列是最大的序列，同时是能够唯一表示图6中图数据的正规编码.

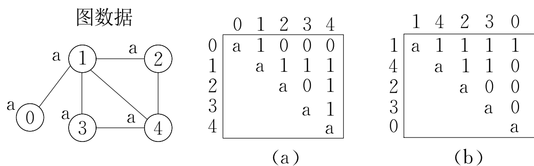

图6　正规编码计算

计算正规编码时，需要遍历所有可能的顶点排列，这个过程的计算复杂度最大，因此我们首先对该过程进行优化，优化的思想是利用顶点标签的不变特性.

对于给定的图数据，其顶点的度以及标签值是不变的，因此，我们利用这些不变特性对图的正规编码的计算进行优化. 根据不变特性对顶点进行分片，具有相同特性的顶点被分到同一个片中. 由于顶点的不变特性不会随着顶点次序而变化，因此无论顶点次序如何变化，都可以得到相同的分片，对分片后的图数据的顶点，利用邻接矩阵的上三角来计算图的正规编码. 对顶点分片之前，计算正规编码时需要对所有顶点进行全排列，顶点分片之后，各个顶点都存在约束，此时只需要对各个分片进行全排列，然后对各个分片的排列进行乘法操作，即可得到图的正规编码. 若两个图同构，它们之间一定存在相同的顶点分片，进行全排列后亦可得到相同的正规编码.

文献[12]采用顶点度和顶点标签值对顶点进行分片，分片后各个片中的顶点不相交. 顶点的度和标签值相同的顶点被分到同一个片中，先按顶点的度再按顶点的标签值进行分片. 优化前，计算正规编码的时间复杂度为$O(|V|!)$. 优化后，假设m是分片的数量，每个分片包含$p_1,p_2,\cdots,p_m$个顶点，计算正规编码的复杂度为$O\left(\prod_{i=1}^{m}(p_i!)\right)$，很显然复杂度得到降低，提高了计算效率.

例如，图7中的图数据，按顶点的度分为三个片p_1、p_2和p_3，其中p_1包含一个顶点1，$p_1=\{1\}$；p_2包含一个顶点2，$p_2=\{2\}$；p_3包含3个顶点0、3和4，$p_3=\{0,3,4\}$. 片p_1和p_2中顶点的度为3，但是片p_1中顶点标签值为“a”，而片p_2中顶点标签值为“b”，片p_3中顶点的度均为2，并且顶点的标签值均为“b”. 按文献[12]中的方法计算图的正规编码，排列各个分片中的顶点序列的次数为$1!\times1!\times3!=6$. 若不进行分片，排列顶点序列的次数为$5!=120$.

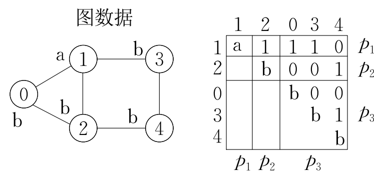

图7　正规编码的分片计算

文献[12]的算法忽略了正规编码计算方法的特点，采用乘法原理计算正规编码，优化效率有限. 本文采用的正规编码优化方法可将复杂度降低到$\sum_{i=1}^{m}(p_i!)$，即加法运算的复杂度.

定义8.　编码的连接. 设图G中包含两个分片，第一个分片包含p_1个顶点，第二个分片包含p_2个顶点，p_1中的顶点集合为V_1，p_2中的顶点集合为V_2，如果顶点集合V_1和V_2的子图字符序列分别为$c(G_{p_1})$和$c(G_{p_2})$，那么V_1和V_2连接后的子图序列表示为

$$c(G_{p_1})\cdot c(G_{p_2}).$$

其计算方法如下，顶点V_1生成了一个邻接矩阵$\boldsymbol{M}_1$，顶点V_2生成了一个邻接矩阵$\boldsymbol{M}_2$，我们依据$E_{p_1\cdot p_2}=\{(v,w)\mid v\in V_1\wedge w\in V_2\}$，将$\boldsymbol{M}_2$的行和列追加到$\boldsymbol{M}_1$的行和列之后，得到新矩阵$\boldsymbol{M}_p$，其上三角阵由三部分元素组成，$X_1=\{x_{ij}\mid x_{ij}\in\boldsymbol{M}_1\}$，其中$1\le i\le p_1$，$1\le j\le p_1$；$X_2=\{x_{ij}\mid x_{ij}\in\boldsymbol{M}_2\}$，其中$p_1+1\le i\le p_1+p_2$，且$p_1+1\le j\le p_1+p_2$；$X_3=\{x_{ij}\mid x_{ij}=l(v_i,w_j)\in E\wedge v_i\in V1\wedge w_j\in V2\}$，其中$x_{ij}$为图$G$中的边的标签，$1\le i\le p_1$，且$p_1+1\le j\le p_1+p_2$. 矩阵$\boldsymbol{M}_p$的字符序列表示为$c(V_1\cup V_2)=c(G_{p_1})\cdot c(G_{p_2})$.

如图8所示，分片p_1中顶点的排序为{12}，$c(G_{p_1})$=“aa1”，分片p_2的顶点排序为{034}，$c(G_{p_2})$=“bbb001”，分片p_1和p_2连接后的邻接矩阵如图8右所示，其$c(G_p)$必须包含边$E_{p_1\cdot p_2}$，因此其连接后的标签序列为“aabbb1101000101”.

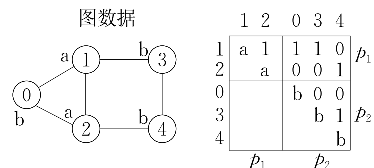

图8　标签的连接

《计算机学报》

定义9. 分片的正规编码. 假设存在标签图$G=(V,E,\Sigma_V,\Sigma_E,l)$，按照顶点的不变特性图$G$被分为大小为$p_1$和$p_2$的2个分片. $cl(G_{p_1})$是分片p_1的正规编码，而且p_1中的顶点标签、度是最大的，我们可以将标签图G的分片正规编码表示为$cl(G)=\max\limits_{c(G_{p_2})}\{cl(G_{p_1})\cdot c(G_{p_2})\}$，即$cl(G)$必须是由$p_1$，$p_2$的2个分片组成的邻接矩阵中的最大字符序列.

依次类推，当图G被分隔为大小为$p_1,\cdots,p_k$的k个片段时，我们可以将其正规编码表示为$cl(G)=(cl(G_{p_1})\cdot c(G_{p_2}))\cdot\cdots\cdot c(G_{p_k})$.

为证明分片正规编码是最大字符序列，需要证明标签连接的正确性，即证明若连接后的序列是最大字符序列，则去掉连接后该标签序列也是最大字符序列.

定理1. 设图$G=(V,E,\Sigma_V,\Sigma_E,l)$的分片连接标签序列是最大字符序列，即可作为正规编码，去掉该序列每一次的最后连接顶点集后，其字符序列仍然是最大的，仍然可作为正规编码.

接下来证明，假设$cl(G)=c(G_{p_1})\cdot c(G_{p_2})$是最大字符序列，去掉$c(G_{p_2})$后其仍然是最大字符序列. 若上述证明成立，则标签的连接一定是最大的，即一定是正规编码.

证明. 利用反证法证明. 假设标签图G的字符序列$cl(G)=c(G_{p_1})\cdot seq(G_{p_2})$是最大字符序列，去掉$c(G_{p_2})$后，$c(G_{p_1})$不是最大字符序列.

按照定义1，可知$cl(G)$的字符序列为

$l_1\cdots l_{r-1}x_{12}\cdots x_{1p_1-1}\cdots x_{p_1-1p_1}y_{1q}\cdots y_{p_1q}\cdots z_{qq+1}\cdots y_{1r}\cdots y_{p_1r}z_{qr}$，其中$q=p_1+p_2+1$，$r=p_1+1$，$l_1\cdots l_{r-1}$是全部顶点的标签的字符序列. $\boldsymbol{x}$，$\boldsymbol{y}$和$\boldsymbol{z}$如图9中所示.

	p_1	p_2	
	x	y	p_1
		z	p_2

图9　标签矩阵元素

去掉$c(G_{p_2})$，我们可以得到标签序列：$cl_1=l_1\cdots l_{p_1}x_{12}x_{13}x_{23}\cdots x_{1p_1-1}\cdots x_{p_1-1p_1}$，假设$cl_1$不是最大字符序列，通过交换其中某两个顶点的顺序，得到一个更大序列：$cl_2=l_1\cdots l_{p_1}x_{12}\cdots x_{mn}\cdots x_{p_1-1p_1}$. 当连接$c(G_{p_2})$时，首先增加顶点标签，再增加矩阵$\boldsymbol{y}$和$\boldsymbol{z}$的值. 由于顶点$p_1$的不变特性大于顶点$p_2$，因此顶点的标签序列不可能发生变化，因此，当我们用$cl_2=l_1\cdots l_{p_1}x_{12}\cdots x_{mn}\cdots x_{p_1-1p_1}$连接$\boldsymbol{y}$和$\boldsymbol{z}$时，无论$\boldsymbol{y}$和$\boldsymbol{z}$矩阵如何变化，$cl_2$的字典序列始终大于$cl_1$，当$cl_1$连接$c(G_{p_2})$后，按照字典序列，$cl_2$仍然大于$cl_1$. 故与假设矛盾，即通过分片连接的标签序列是图的正规编码. 证毕.

《计算机学报》

定理2. 两个图是同构的，当且仅当分片的正规编码相同.

证明. 当两个图同构时，其分片后的不变特性相同，此时通过邻接矩阵得到的编码也相同，当其中一个为最大编码时，另一个也一定是最大编码.

若两个图的分片正规编码相同，则其顶点存在一一对应关系，其顶点之间的边也一一对应，因此两个图必然同构. 证毕.

推论1. 若图G的分片$p_1,\cdots,p_k$存在自同构，同一个正规编码对应多个不同的顶点的排列，其分片组合编码的计算复杂度为$p_1!+\alpha p_2!+\cdots+\beta p_k!$，其中$\alpha$和$\beta$等表示前一次连接时，分片正规编码中包含的不同顶点排序的次数.

证明. 设p_1是自同构，顶点u和v是可交换顶点. 顶点V_1的排列顶点"$\cdots u\cdots v\cdots$"能得到一个正规编码，其邻接矩阵为$\boldsymbol{M}_1$. 交换顶点u和v的次序后，其仍然得到正规编码，但是邻接矩阵为$\boldsymbol{M}_1'$.

对分片p_2，依据$E_{p_1\cdot p_2}=\{(v,w)\mid v\in V_1\wedge w\in V_2\}$进行连接后，其邻接矩阵可能不同，导致字符序列也可能不同. 所以，计算p_1和p_1的分片正规编码时，必须排列$\boldsymbol{M}_1$和$\boldsymbol{M}_1'$. 依此方法类推，若子图存在自同构，需要将其分片的正规编码对应的顶点排序全部保存，新一次的连接操作必须考虑前一次连接操作中形成的子图中的全部自同构顶点排序. 故，按照这种方式，正规编码的计算复杂性就变为$p_1!+\alpha p_2!+\cdots+\beta p_k!$，其中$\alpha$和$\beta$等表示前一次连接后，子图中存在的自同构数量.

5.2 基于编码树的正规编码计算优化

定义10. 编码树. 给定一个图G及其正规编码$cl(G)$，如果图G添加了一个顶点v和该顶点对应的边$e_1,e_2,\cdots,e_k$，得到图G'，则我们可以将图G'的编码树表示为$T(G')$，其格式可以表示为$cl(G_{p_1})\cdot tr(v)\cdot b(e_1)\cdot b(e_2)\cdot\cdots\cdot b(e_k)$.

其中$tr(v)$代表顶点的标签，$b(e_1)$为0或者是边e_1的标签. 若顶点v与图G对应的正规编码中的第一个顶点之间有边，则其值为边的标签值，若无边，则其值为0；同样，$b(e_2)$代表与第二个顶点之间是否存在边，或为边的标签值，以此类推. 图G增加顶点后表示为图G'，图G'的编码树为图G的正规编码$cl(G)$后，追加顶点的标签以及与图G各个顶点之间边的标签.

如图 10 所示，图(a)大小为 $k=3$，其正规编码为“aaa111”，增加顶点“4”后如图(b)所示，顶点“4”分别与顶点“1”和“3”之间存在边，所以其编码树为“aaa111#a101”，其中“#”作为分隔符，可以保留或者去除.

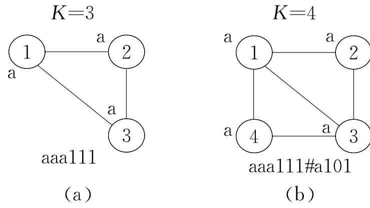

图 10 编码树

定理 3. 若两个图的编码树相同，则两个图同构.

证明. 设存在图 G，图 P 和图 S，其中 $|V_G|+1=|V_S|$，$|V_G|+1=|V_P|$，图 S 和图 P 比图 G 多一个顶点，并且图 G 是图 S 的导出子图，图 G 是图 P 的导出子图. 此时图 G 和图 S 之间必然存在一一映射 f_1，存在 $u_i, u_j \in V_G$ 以及 $v_i, v_j \in V_s$，满足 $l(u_i)=l'(f_1(v_i))$，$(u_i, u_j) \in E \Leftrightarrow (f_1(v_i), f_1(v_j)) \in E'$；图 G 和图 P 之间必然存在一一映射 f_2，存在顶点 u_i，$u_j \in V_G$ 和顶点 $w_i, w_j \in V_P$，满足 $l(u_i)=l'(f_2(w_i))$ 以及 $(u_i, u_j) \in E \Leftrightarrow (f_2(w_i), f_2(w_j)) \in E'$；此时，通过 $u_i, u_j \in V_G$，建立起图 P 和图 S 一个部分的一一映射. 对于最后的一个顶点，由于编码相同，顶点的标签值也相同，而且其对应的边的顺序和标签相同，因此最后一个顶点也存在一一对应关系，故 P 和 S 同构. 证毕.

根据编码树，我们可以设计一种正规编码的优化计算方法.

定理 4. 设存在图 G, P，$|V_G|+1=|V_P|$，G 是 P 导出子图，图 P 的编码树的计算复杂度最大为 $O(|V_P|)$

证明. 若图 G 增加一个顶点 v，得到图 P，增加顶点的位置最多有 $|V_P|$ 个，因此，其计算复杂度最坏情况下为 $O(|V_P|)$. 证毕.

定义 11. 最大(最小)编码树. 给定一个图 G 及其正规编码 $cl(G)$，如果图 G 使用最大(最小)正规编码，且图 G 是自同构的，顶点 v_1, v_2 可以互相交换而不影响正规编码，此时添加了一个新顶点 v 和该顶点对应的边 e_1, e_2，得到图 G'，则可以将图 G' 的编码树表示为 $T(G')$，其格式可以表示为 $cl(G_{p_1}) \cdot tr(v) \cdot b(e_1) \cdot b(e_2) \cdot \cdots \cdot b(e_k)$，此时交换 $b(e_1)$ 和 $b(e_2)$ 会得到不同的编码树，按照字典序可以得到最大(最小)序列，则成为最大(最小)编码树.

如果图 P 包含自同构的话，就需要计算出可以交换的顶点对，然后计算其最大(或者最小)编码树. 由于在 5.1 节中，已经计算了自同构问题，系统这里仅需要交换新加入的边的次序，得到最大(或者最小)编码树.

推论 2. 若存在模式 P 的 k 个子图实例 $p_1, \cdots, p_k$，模式的大小为 m，且存在 α 个自同构顶点交换序列，则计算其正规编码的复杂度为从 $k\Omega(P)$ 变为 $\Omega(P)+\alpha mO(k)$

证明. 对于 k 个子图实例 $p_1, \cdots, p_k$，为了确认其模式为 P，需要计算每个子图实例的正规编码，而每个子图实例的计算复杂度为 $\Omega(P)$，k 个子图实例的计算复杂度可以表示为 $k\Omega(P)$.

由于模式大小为 $m-1$ 的正规编码已经在上一次迭代时计算出来，因此根据编码树定义，k 个子图实例计算编码树的复杂度为 $mO(k)$. 假如子图实例存在子图同构，其中可以交换次序的顶点有 α 对，那么就需要 α 次计算，其复杂度为 $\alpha mO(k)$. 根据定理 3，当 k 个子图实例编码树一样时，只计算其中一个子图实例的正规编码，其计算复杂度为 $\Omega(p_1)$，得到正规编码后，确定新增加顶点 v 在邻接矩阵中的位置，对于剩余的 $k-1$ 个子图实例，只需要按照顶点 v 的位置插入顶点，并复制 p_1 的正规编码，从而得到其正规编码. 所以其计算复杂度为 $\Omega(P)+\alpha mO(k)$. 证毕.

从推论 2 可以看出，由于 $\Omega(P)$ 计算复杂度为模式大小的阶层，而编码树的计算复杂度是大小的线性时间，因此，采用编码树方法可以显著地减少正规编码的计算时间，其减少接近 $\Omega(p_1)/k$.

6 算法的性能评价

在建立测试用的集群平台时，安装 Hadoop2.6+Spark1.6 运行环境. 平台使用了 20 台中科曙光服务器 620/420，操作系统为 redhat Enterprise server 7.1 X86_64，JAVA 执行环境采用 JDK8u131 64 位版本的虚拟机，每台物理服务器安装 2 个 AMD Opteron(TM) Processor 6212 处理器，每个 CPU 的逻辑核数为 8，最大线程数量为 8，16 GB 内存.

6.1 实验数据

实验数据包含 6 个大图数据，如表 2 所示. 图数据 CiteSeer[26] 将发表的论文作为顶点，AI、Agent

《计算机学报》

等作为顶点标签，论文之间的引用关系作为边. Pattents[①]包含从1963年1月到1999年12月期间美国专利的引用，引用关系作为边，专利授权的专利作为顶点，授权时间作为顶点标签. Youtube[②]列出了抓取的视频id信息，以及从2007年1月到2008年8月之间的每个视频之间的相关信息，每个视频作为顶点，视频的排序以及长度作为顶点标签. Twitter[③]，该图描述了推特用户的信息，每个顶点代表一个推特用户，每条边代表两个用户之间的互动，原始图中没有标签，实验中随机地为顶点添加标签，不同标签的数量设置为100个，标签的分布满足高斯分布. UG100k是一个随机图，该图包含100 000个顶点，边的生成概率为0.0003.

表 2 用于性能评价的图数据

标号	图名称	顶点数量	边的数量	标签数量	平均度
1	CiteSeer	3312	4591	6	2.8
2	Youtube	8350	78743	12	18.9
3	UG10k	10000	493174	25	99.0
4	UG100k	100000	14903216	64	298.0
5	Pattents	2923923	24125824	163	16.5
6	Twitter	8768418	76563426	100	17.5

6.2 基于数据流模型的频繁模式挖掘的性能测试

基于数据流模型的频繁模式挖掘的特点在于扩展上一次迭代产生的子图实例并作为下一次迭代的输入. 传统的基于Apriori方法以及基于模式增长的方法，主要利用扩展前一次的子图模式作为后一次的输入，首先对新扩展的模式进行唯一性检测，防止自同构的出现，之后检查模式是否与输入图存在子图同构，其计算复杂度增加. 论文首先比较单机环境下三种方法的运行时间，模式大小为5.

从表3的数据可以看出，图数据规模较小时，基于数据流的频繁模式挖掘算法的效率较高，其原因在于减少了模式自同构检查的计算量；但是，随着图数据规模增大，其花费的计算时间与Apriori算法以及模式增长算法相差不大，经分析，主要原因在于模式实例的存储导致内存使用率加大，JVM的效率降低. 对于实验数据中的图4、图5和图6，由于顶点和边数量较大，单台计算机处理效率较低，故本实验未进行比较.

表 3 不同图数据的运行时间 （单位：s）

算法	图标号 1		2		3	
	$\tau=100$	$\tau=400$	$\tau=100$	$\tau=400$	$\tau=100$	$\tau=400$
Apriori	2.42	0.57	355.6	354.1	565.6	443.1
模式增长	2.35	0.62	362.3	366.2	562.3	456.2
基于数据流	1.01	0.16	312.1	338.1	467.1	350.4

另外，为了对基于数据流模式的频繁模式挖掘的基本性能进行描述，论文对6个不同的图数据中挖掘出的子图模式的个数以及运行时间进行了测试，表4为前3个图数据的测试结果，表5为后三个图数据的测试结果. 实验中，支持度的阈值分别设置为50、100和800，频繁模式的大小如表4和表5中的$Size$. 从表4和表5中可以看出，随着子图支持度的增加，子图模式数量越来越少，但是计算时间却越长，这说明非频繁模式大量出现，既占用了计算时间，又增加了存储代价.

表 4 不同图数据的运行时间以及频繁模式数量

算法	支持度	图标号 1	2(PPI)	3(Slash)
		$Size=7$	$Size=7$	$Size=6$
时间/s	50	1080	1861	1092
	100	903	1442	708
	800	782	1141	354
子图实例数量	50	14161670898	15292184514	15059680898
	100	8101600898	9261394514	8268671776
	800	981606708	1101394514	1377691694
频繁模式数量	50	1110	61931	898765
	100	753	27854	448657
	800	267	4568	65792

表 5 不同图数据的运行时间以及频繁模式数量

算法	支持度	图标号 4(web)	5	6
		$Size=5$	$Size=4$	$Size=4$
时间/s	50	2461	6602	12289
	100	1982	6297	11094
	800	1439	5693	9841
子图实例数量	50	31244545885	165369545391	316369545391
	100	16134655794	88258747494	155468632199
	800	9046675972	16378534588	86668732773
频繁模式数量	50	1945724	44428437	58639521
	100	988936	21936769	32977899
	800	146857	4308658	4613713

《计算机学报》

6.3 算法的加速比和效率

加速比经常用来衡量串行算法和并行算法的关系，它表示为串行运行时间与并行运行时间的比值：$S(n,p)=T_s(n)/T_p(n,p)$，其中n表示输入数据规模，p表示并行处理器的数量，T_s表示串行计算的时间，T_p表示并行计算的时间. 本论文中的串行挖掘也采用基于数据流模式，由于串行挖掘过程中受到内存资源限制，仅对图数据Youtube和CiteSeer进行了测试，测试时由于Youtube数据受单机挖掘的限制，所以图的模式大小为5时停止计算，最小支持

① http://www.nber.org/patents/
② http://netsg.cs.sfu.ca/youtubedata/
③ http://snap.stanford.edu/data/index.html

度设置为600.并行挖掘过程中,服务器的Executor数量依次从1,2,4增加到40来测试算法的加速比(一台服务器上启动两个Executor).图11(a)和(b)分别展示对图数据Youtube和CiteSeer执行并行算法的加速比.观察实验结果,两个图数据都显示出了较好的加速度,当Executor数量较少时,可以获得近似于线性的加速比.随着Executor数量的增加,加速比的增长趋势变得缓慢,Executor数量约为10时,加速比达到最大值,此后加速比回落,经分析发现,随着Executor的增加,子图实例的数量也大规模增加,HDFS写入数据的开销变大,导致并行执行时间增加.

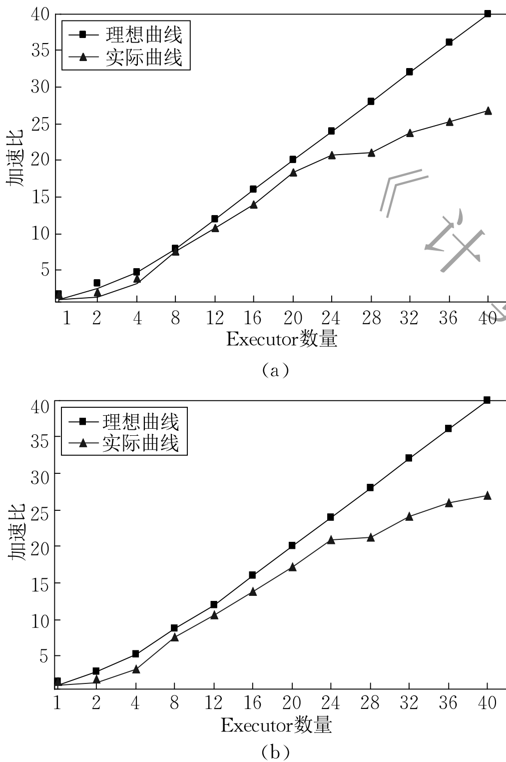

图11 并行算法加速比

另外,为了说明挖掘过程中"每一个Executor"的加速比,论文测试了并行算法的效率,用$E(n,p)$表示,$E(n,p)=S(n,p)/p$,通常采用并行算法挖掘时效率小于1.图12(a)和(b)分别对图数据Youtube和CiteSeer采用并行算法挖掘的效率进行了测试.从图12的曲线可以看出,开始时并行算法的效率下降较缓慢,Executor数量增大到20左右时,效率急剧下降.从图11和图12的结果可以看出,虽然CiteSeer数据的规模比Youtube大,但是CiteSeer数据挖掘出的频繁模式的数量远小于Youtube挖掘出的频繁模式的数量,所以两者加速比区别并非特别明显.

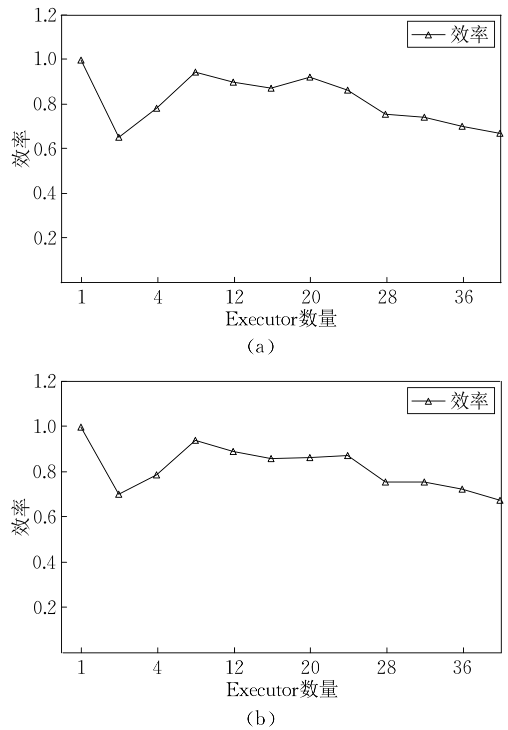

图12 并行算法的效率

6.4 与MapReduce实现方法的比较

论文验证了文献[18]中提出的基于MapReduce的并行频繁模式挖掘算法MRSUB的效率,并且比较了该算法与本文所提出算法的性能.本文验证的相关算法还包括文献[19]中的基于Pregel的算法,但是在使用Pregel计算过程中,由于内存中的数据规模较大导致无法存储到磁盘,因此计算能力较差,在此本文不进行与该算法的比较.Spark使用的环境与MapReduce使用的环境相同,在MapReduce中运行文献[18]提出的算法,在Spark上运行本文提出的基于数据流模型的方法.

从表6中的数据可看出,基于MapReduce的MRSUB算法存在两个不足,一是不同Map任务中包含的候选子图实例的个数差异较大,数据倾斜问

表6 数据流模型与MRSUB的计算时间比较(单位:min)

图号	子图大小	MRSUB算法	数据流模型
1	6	16	11
	7	30	24
2	6	22	16
	7	37	31
3	4	10	5
	5	18.2	11.8
4	4	28	19
	4	39	33
5	3	74	68
	4	114	105
6	3	129	103
	4	233	185

《计算机学报》

题严重，影响整体性能；其二是 Map 任务和 Reduce 任务之间的数据传输时间较长，花费时间较多.

6.5 优化后的结果比较

为测试使用未优化编码方法以及使用优化编码方法，我们计算正规编码的时间差异. 实验中，我们从图数据 Youtube 中遍历大量的大小不同的子图，并计算每个子图的正规编码，待计算全部结束后，统计其全部计算时间. 表 7 给出了子图数量为 1000000 且子图大小分别为 3，4，5，6，7 和 8 时的计算时间，其中 $size=3$ 的子图中，包含两个不同的标签"Agent"和"AI"，"Agent"出现 2 次，"AI"出现 1 次；$size=4$ 的子图中，包含两个不同的标签"Agent"和"AI"，"Agent"出现 2 次，"AI"出现 2 次；$size=5$ 的子图中，包含三个不同的标签"DB"，"Agent"和"AI"，"DB"出现 1 次，"Agent"出现 2 次，"AI"出现 2 次；$size=6$ 的子图中，包含三个不同的标签"DB"，"Agent"和"AI"，"DB"出现 2 次，"Agent"出现 2 次，"AI"出现 2 次；$size=7$ 的子图中，包含四个不同的标签"DM"，"DB"，"Agent"和"AI"，"DM"出现 1 次，"DB"出现 2 次，"Agent"出现 2 次，"AI"出现 2 次；$size=8$ 的子图中，包含两个不同的标签"DM"，"DB"，"Agent"和"AI"，"DM"出现 2 次，"DB"出现 2 次，"Agent"出现 2 次，"AI"出现 2 次. 优化前与优化后的正规编码计算时间如表 7 所示，可看出，特定场景性能提高大约 30%左右.

表 7 优化前后正规编码计算时间对比（单位：s）

算法	k					
	3	4	5	6	7	8
优化前	1.9	7.2	43.9	323.1	2081.9	15607
优化后	1.3	4.2	23.7	89.4	469.2	801.5

为了测试编码树优化的性能，测试采用的方法是，首先选择大小为 k 的子图实例及其正规编码，分别对其扩展一个顶点，使得子图实例大小增大为 $k+1$，分别使用原来的计算方法以及基于编码树的计算方法，计算大小为 $k+1$ 的全部子图实例的正规编码，从而得到计算时间. 测试中，论文以表 7 中的数据为依据，依次将子图大小扩大为 5，6，7，8，9 和 10，再分别计算大小不同的子图实例的正规编码，最后统计计算时间，使用编码树优化后的结果如表 8 所示，从结果可以看出，特定场合下，性能提高大约 10%左右，子图的规模越大，效率提升越显著.

表 8 使用编码树优化前后计算时间对比（单位：s）

算法	k					
	4	5	6	7	8	9
优化前	4.2	23.7	89.4	469.2	801.5	1332.8
优化后	2.8	4.1	13.2	53.9	106.7	308.3

7 结 论

频繁模式的挖掘在许多领域中都有重要的应用. 单机单图的频繁模式挖掘算法得到了广泛而深入的研究，随着单图数据规模的扩大，分布式频繁模式挖掘算法显得越来越重要. 论文通过使用数据流模型，解决了现有算法中 Map 和 Reduce 之间大规模数据的存储以及数据倾斜的问题，通过对包含上百万顶点的图数据进行实验测试，基于数据流模型的挖掘算法能够提高 30%左右的效率. 但是，算法在模式实例的存储方面还有待进一步提高.

参 考 文 献

[1] Deshpande M, Kuramochi M, Karypis G. Frequent substructure-based approaches for classifying chemical compounds //Proceedings of the IEEE International Conference on Data Mining. Burlington, USA, 2003: 35-42

[2] Domshlak C, Brafman R I, Shimony E. Preference-based configuration of Web page content//Proceedings of the International Joint Conference on Artificial Intelligence. Seattle, USA, 2001: 1451-1456

[3] Guralnik V, Karypis G. A scalable algorithm for clustering sequential data//Proceedings of the IEEE International Conference on Data Mining. California, USA, 2001: 179-186

[4] Deutsch A, Fernandez M, Suciu D. Storing semistructured data with STORED. Sigmod Record, 1999, 28(2): 431-442

[5] Yan X, Yu P S, Han J. Graph indexing: A frequent structure-based approach//Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. Paris, France, 2004: 335-346

[6] Cho Y R, Zhang A. Predicting protein function by frequent functional association pattern mining in protein interaction networks. IEEE Transactions on Information Technology in Biomedicine, 2010, 14(1): 30-36

[7] Aridhi S, Nguifo E M. Big graph mining: Frameworks and techniques. Big Data Research, 2016, 6: 1-10

[8] Kuramochi M, Karypis G. Frequent subgraph discovery// Proceedings of the IEEE International Conference on Data Mining. Maebashi City, Japan, 2002: 313-320

[9] Kuramochi M, Karypis G. Grew—A scalable frequent subgraph discovery algorithm//Proceedings of the 4th IEEE International Conference on Data Mining (ICDM'04). Brighton, UK, 2004: 439-442

[10] Inokuchi A, Washio T, Motoda H. Complete mining of frequent patterns from graphs: Mining graph data. Machine Learning, 2003, 50(3): 321-354

《计算机学报》

[11] Yan X, Han J. gSpan: Graph-based substructure pattern mining//Proceedings of the IEEE International Conference on Data Mining. Maebashi City, Japan, 2002: 721

[12] Liu Y, Jiang X, Chen H, et al. MapReduce-based pattern finding algorithm applied in motif detection for prescription compatibility network//Proceedings of the Advanced Parallel Processing Technologies. Rapperswil, Switzerland, 2009: 24-25

[13] Shahrivari S, Jalili S. Distributed discovery of frequent subgraphs of a network using MapReduce. Computing, 2015, 97(11): 1101-1120

[14] Kang U, Tsourakakis C E, Faloutsos C. PEGASUS: Mining peta-scale graphs. Knowledge & Information Systems, 2011, 27(2): 303-325

[15] Wernicke S, Rasche F. FANMOD: A tool for fast network motif detection. Bioinformatics, 2006, 22(9): 1152-1153

[16] Jiang C, Coenen F, Zito M. A survey of frequent subgraph mining algorithms. Knowledge Engineering Review, 2013, 28(1): 75-105

[17] Suryawanshi S J, Kamalapur S M. Algorithms for frequent subgraph mining. International Journal Advanced Research in Computer and Communication Engineering, 2013, 2(3): 1545-1548

[18] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases//Proceedings of the 20th International Conference on Very Large Data Bases. Santiago, Chile, 1994: 487-499

[19] Abdelhamid E, Abdelaziz I, Kalnis P, et al. Scalemine: Scalable parallel frequent subgraph mining in a single large graph//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Salt Lake City, USA, 2016: 61

[20] Di Fatta G, Berthold M R. Dynamic load balancing for the distributed mining of molecular structures. IEEE Transactions on Parallel and Distributed Systems, 2006, 17(8): 773-785

[21] Wang C, Parthasarathy S. Parallel algorithms for mining frequent structural motifs in scientific data//Proceedings of the International Conference on Supercomputing. Heidelberg, Germany, 2004: 31-40

[22] Lin W, Xiao X, Ghinita G. Large-scale frequent subgraph mining in MapReduce//Proceedings of the IEEE International Conference on Data Engineering. Chicago, USA, 2014: 844-855

[23] Bhuiyan M A, Al Hasan M. An iterative MapReduce based frequent subgraph mining algorithm. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(3): 608-620

[24] Hill S, Srichandan B, Sunderraman R. An iterative MapReduce approach to frequent subgraph mining in biological datasets//Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine. Orlando, USA, 2012: 661-666

[25] Lu W, Chen G, Tung A K H, et al. Efficiently extracting frequent subgraphs using MapReduce//Proceedings of the IEEE International Conference on Big Data. Santa Clara, USA, 2013: 639-647

[26] Teixeira C H C, Fonseca A J, Serafini M, et al. Arabesque: A system for distributed graph mining//Proceedings of the Symposium on Operating Systems Principles. Monterey, USA, 2015: 425-440

[27] Carbone P, Ewen S, Haridi S, et al. Apache flink: Unified stream and batch processing in a single engine. Data Engineering, 2015, 36: 28-38

[28] Akidau T, Bradshaw R, Chambers C, et al. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proceedings of the VLDB Endowment, 2015, 8(12): 1792-1803

[29] Chandrasekaran S, Cooper O, Deshpande A, et al. TelegraphCQ: Continuous dataflow processing//Proceedings of the ACM SIGMOD International Conference on Management of Data. California, USA, 2003: 668-668

[30] Xin R S, Gonzalez J E, Franklin M J, et al. GraphX: A resilient distributed graph system on Spark//Proceedings of the International Workshop on Graph Data Management Experiences and Systems. Houston, USA, 2013: 1-6

[31] Reinhardt S, Karypis G. A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph//Proceedings of the IEEE International Parallel and Distributed Processing Symposium. Long Beach, USA, 2007: 1-8

[32] Nijssen S. What is frequent in a single graph?//Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. Osaka, Japan, 2008: 858-863

[33] Vanetik N, Gudes E, Shimony S E. Computing frequent graph patterns from semistructured data//Proceedings of the IEEE International Conference on Data Mining. Maebashi City, Japan, 2002: 458

[34] Fiedler M, Borgelt C. Subgraph support in a single large graph//Proceedings of the IEEE International Conference on Data Mining Workshops. New Orleans, USA, 2007: 399-404

[35] Kuramochi M, Karypis G. Finding frequent patterns in a large sparse graph. Data Mining and Knowledge Discovery, 2005, 11(3): 243-271

[36] Elseidy M, Abdelhamid E, Skiadopoulos S, et al. GraMi: Frequent subgraph and pattern mining in a single large graph. Proceedings of the VLDB Endowment, 2014, 7(7): 517-528

[37] Buehrer G, Parthasarathy S, Chen Y K. Adaptive parallel graph mining for CMP architectures//Proceedings of the International Conference on Data Mining. Hong Kong, China, 2006: 97-106

[38] Zhao X, Chen Y, Xiao C, et al. Frequent subgraph mining based on pregel. Computer Journal, 2018, 59(8): 1113-1128

《计算机学报》

TANG Xiao-Chun, Ph. D., associate professor. His research interests include graph base management and distributed computing.

FAN Xue-Feng, graduated student. Her research interest is big data computing.

ZHOU Jia-Wen, graduated student. Her research interest is big data computing.

LI Zhan-Huai, Ph. D., professor. His research interests include ocean base management and big data computing.

Background

Recently, the research of social network, web analysis, bioinformatics and chemical informatics has received extensive attention. At the same time, a large number of graph modeling data have been produced. Therefore, mining effective information from graph data is a meaningful work. The current popular research includes frequent subgraph mining, graph classification, graph clustering, graph searching, graph indexing and so on. Frequent subgraph mining is widely used in the field of complex network analysis. For example, In biological networks, mining frequent subgraph can reduce the cost of protein structure matching experiments. Another important application is in social network analysis, frequent subgraph mining helps to find stable and unstable relationships and detect frequent patterns. Frequent subgraph mining helps the development of many disciplines. Frequent subgraphs are subgraphs found from a set of graphs or a single large graph with support exceeds the user-defined threshold. Frequent pattern mining has always been a focused theme in graph mining. Many researchers were dedicated to this filed, making tremendous progress, including frequent itemset mining, sequential pattern mining and so forth. With the development of science and technology, researchers will generally face the problem of computing large scale data. The traditional frequent subgraph mining algorithm of single machine is inefficiency, so it is difficult to meet the requirements of large-scale graph data mining. Graph data mining requirements have evolved more sophisticated, most of the traditional graph data mining processes are bounded, ordered data sets, however, there are large scale of unbounded, disorder graph data. At the same time, people, and higher and higher requirements for accuracy. Therefore, we urgently need to optimize and upgrade the method of graph data mining. So that it can better meet the existing data mining needs.

The dataflow model is a general framework that can simply define the representation of parallel computing. It provides a unified abstraction for batch, microbatch and stream processing, so the general framework expressed by the data flow model is independent of the underlying execution engine. The dataflow model is a general framework that can simply define the representation of parallel computing. It provides a unified abstraction for batch, microbatch and stream processing, so the general framework expressed by the data flow model is independent of the underlying execution engine. The data flow model separates the logical representation of data processing from the physical implementation of logic, which makes us pay more attention to the choice of accuracy, delay degree and processing cost instead of worrying about which execution engine to choose. Our approach to large-scale data computing is from choosing and writing programs based on the execution engine to what processes and results we focus on.

Based on above research, a parallel and distributed mining method of frequent subgraph is proposed, which makes the mining time of frequent subgraph shorter and the data result more accurate. And we analyze the exists canonical code calculation methods of graphs, optimize the canonical calculation methods of subgraphs, propose the data structure of subgraph instances and the pipelining algorithm in frequent pattern mining. By using the partial order relation of coding between vertices, the extended scope of subgraph instances is reduced, and the calculation and storage scale is reduced effectively.

《计算机学报》