基于 Twitter Storm 平台并行挖掘最稠密子图

王金明 王远方
(东南大学计算机科学与工程学院 南京 211189)

摘 要 在大规模图结构数据中发现最稠密子图具有极其广泛的应用,如社区发现、垃圾邮件检测和论文引用关系抽取等。基于带标签的无向图,提出了查询标签集的概念,设计了一个可以快速发现最稠密子图的近似算法 DSFLC (Densest Subgraph Finding based on Labelset Constraint):用户提交自定义的查询标签集,算法便可保证在用户可以接受的时间内返回满足查询标签集约束的最稠密子图。对于任何参数 $\varepsilon(\varepsilon>0)$,DSFLC 算法只需扫描大规模数据集 $O(\log_{1+\varepsilon}n)$次,同时可保证算法的近似因子是 $2(1+\varepsilon)$。对 DSFLC 算法进行分析后,发现该算法在预处理阶段易于并行化,因此选择 Twitter Storm 平台,并行化地实现了 DSFLC 算法。最后对从 DBLP 数据库中抽取的合作关系图进行测试,一方面研究 Storm 平台对算法的加速程度;另一方面分析挖掘出的子图的稠密度与参数 ε 之间的关系,最终验证了 DSFLC 算法的实用性和可扩展性。

关键词 最稠密子图发现,查询标签集,DSFLC 算法,Twitter Storm 平台
中图法分类号 TP392 **文献标识码** A

Parallel Mining of Densest Subgraph Based on Twitter Storm

WANG Jin-ming WANG Yuan-fang
(School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

Abstract In large scale graph, finding densest subgraph has a wide range of applications, such as community discovery, spam detection and reference relation extraction. Based on tagged undirected graph, we introduced the concept of QLS and designed an approximation algorithm DSFLC which can quickly find the densest subgraph: users submit a QLS and the algorithm will return the densest subgraph under QLS within the time that user can accept. For any $\varepsilon>0$, DSFLC only needs to scan large-scale data sets $O(\log_{1+\varepsilon}n)$ times, and can ensure the approximation algorithm is a $2(1+\varepsilon)$-approximation algorithm. After analyzing DSFLC, we found this algorithm is easy to parallelize, so we chose Twitter Storm platform to parallel DSFLC algorithm. Finally, the test data sets extracted from the DBLP database verify DSFLC' practicality and scalability.

Keywords Densest subgraph finding, QLS, DSFLC algorithm, Twitter storm platform

1 引言

在大规模图结构数据中发现最稠密子图具有众多应用,涉及的领域包括:在社交网络中发现通过相似关系最紧密联系起来的社区[1,2];在万维网中,发现通过超链接紧密相关的一系列网页[3],这些网页从逻辑上可视为内聚性较高的 Web 社区等。

本文所提出的算法基于带标签的无向图,所谓的标签,是指图中顶点是通过何种关系联系起来的。如图 1 所示,顶点 1 和顶点 2 是通过合作关系联系起来的。由于顶点间的关系往往不止一种,因此连接顶点间的边就具有一个标签集,该标签集记录的是顶点间通过何种关系进行联系,如图 1 所示,顶点 2 和顶点 8 之间既有朋友关系,也有同事关系,因此连接顶点 2 和顶点 8 的边具有的标签集就是{朋友,同事}。本文引入了查询标签集 QLS(Query Label Set)的概念,用户提交自己感兴趣的标签集合,这些标签集合就构成了查询标签集。在带标签的无向图中,通过用户提供的查询标签集,发现的最稠密子图不仅具有最高的稠密度(本文采用的稠密度定义是平均度数:稠密部分所包含的边的总数除以顶点的总数),而且查询标签集应该包含在稠密部分的任意一条边的标签集中。由图 1 知,当查询标签集是 QLS={朋友,同事}时,发现的最稠密子图为图中粗线连通的图,其稠密度为 7/5=1.4。

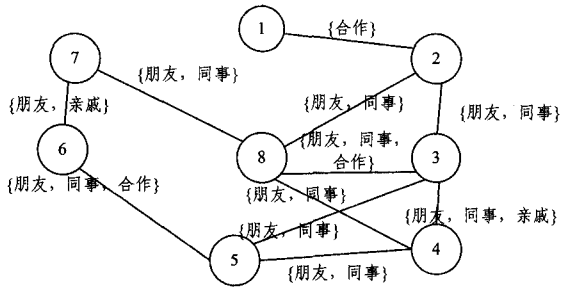

图 1 无向带标签图 G,QLS={朋友,同事}

先前的发现最稠密子图的算法[6,7],一方面没有考虑标

到稿日期:2013-03-10 返修日期:2013-04-28
王金明(1988—),男,硕士生,主要研究领域为图结构数据管理;王远方(1988—),女,硕士生,主要研究领域为图结构数据挖掘。

签的因素，另一方面并不适合处理大规模的数据集。因此有必要提出一种具有性能保障的近似算法，使得算法能在用户可接受的时间内返回近似解。近似算法相比于精确算法具有时间性能上的优势；相比于同样具有时间性能优势的启发式算法，近似算法可以将结果控制在一定的范围内，而启发式算法一般不能保证结果的有效性。本文提出一种基于查询标签集约束的、在大规模图数据集中发现最稠密子图的近似算法DSFLC。因为大规模数据集不可以一次性放入内存中，一般的处理方法是将数据集组织成数据流的形式，使其依次流入内存。针对数据集的这种特性，本文引入了Twitter公司处理流式数据的开源框架Twitter Storm[4]，Storm克服了Hadoop框架在处理海量数据时batch-to-batch的缺点，它具有较高的实时性，使得我们可以实时地发现当前最稠密子图；同时Storm框架的简洁编程模式，易于将DSFLC算法进行并行化部署，可以大大提高算法的执行效率。实验所需的测试数据集是通过在DBLP数据库[5]所提供的dblp.xml(约1G)中抽取合作关系所形成的合作关系图。

本文的主要贡献：

1. 基于无向带标签图和查询标签集，本文提出了一个满足查询标签集约束的、在大规模图数据集中快速发现最稠密子图的近似算法DSFLC。

2. 分析DSFLC算法，引入Twitter Storm流式数据处理框架，将DSFLC算法进行并行化处理，大大提高了算法的执行效率和可扩展性。

本文第2节介绍一些相关工作；第3节给出了在图中发现稠密子图问题的详细定义；第4节介绍了处理稠密子图发现问题的近似算法DSFLC，并予以分析；第5节通过在搭建的Twitter Storm平台上测试从DBLP数据库中抽取的合作关系图，分析并验证算法的可扩展性和有效性；最后总结自己的研究工作。

2 相关工作

Goldberg[6]第一个在无向图中提出发现稠密子图问题，他同时[6]给出了一个需要$O(\log n)$次迭代计算的算法。但是基于最大流方法的算法[6,8]不适合处理大规模数据集，因此Gibson[9]提出在大规模数据集中有效发现稠密子图的算法，该算法基于shingling方法，对原始图进行分割，使其分割后的结果类似于二部图，算法利用shingling方法在该二部图基础上发现稠密子图。Charikar[7]针对稠密子图发现问题，提出了一个简单的近似算法，即通过每次删除度数最小的顶点，比较删除前与删除后稠密度的变化，来发现最稠密子图，同时文献[7]证明了该近似算法是一个2-approximation算法，针对该算法的理论分析为近似算法提供了一个理论基石。文献[10]将文献[7]中的算法进行了优化，其一次不是仅删除一个顶点，而是一次删除一批满足条件的顶点，加快了算法的执行效率，文献[10]还引入了MapReduce框架，将算法进行并行化处理。但是文献[10]仅仅针对不含标签属性的图结构，参考文献[11]中提出的属性图以及属性结构相关模式的概念，基于此本文提出了允许用户自定义的查询标签集的概念。

MapReduce[12]体系结构提供了在大规模分布式环境下的简洁编程模式，受到了学术界和业界的广泛关注和研究。但是它的批处理方式成为支持流数据处理方式的主要障碍之一。目前，有众多研究和实验性系统从不同层次尝试对其进行改进、扩展，以适应新的应用需求，如具有迭代循环特性的MapReduce[13,14]、具有在线功能的MapReduce[15]。但是这些系统却都不能很好地满足实时性的要求。因此Twitter公司开发了号称“实时版”的Hadoop系统“Twitter Storm”[4]，本文所采用的实验平台就是Storm流数据处理框架。

3 问题定义

定义1 带边标签的无向图$G=(V,E,\sum,\phi)$，其中V代表图G的顶点集合；E代表图G的边集合；$\sum$代表图G边上标签的集合，即$\sum=\{a_1,a_2,a_3,\cdots,a_n\}$，其中$a_i(1\leqslant i\leqslant n)$表示的是边上的标签；$\phi$是一个映射函数，满足$\forall e\in E,\phi(e)\subseteq\sum$，即$\phi(e)$返回的结果是边$e$上的标签集。

如图1所示，图G的边标签的集合$\sum$为：$\sum=\{$合作，朋友，同事，亲戚$\}$，并且$\phi((1,2))=\{$合作$\}\subseteq\sum$。

定义2(查询标签集QLS) 对于带边标签的无向图$G=(V,E,\sum,\phi)$，用户提交的查询标签集(Query Label Set) $QLS=\{a_m,\cdots a_r\}(1\leqslant m\leqslant n,m\leqslant r\leqslant n)$，其中QLS中的标签是按照字典顺序排好序的，满足(1) $\forall a_i\in QLS,a_i\in\sum(m\leqslant i\leqslant r)$ (2)$QLS\subseteq\sum$。

定义3 图G中若存在顶点集合$S(S\subseteq V)$，由顶点集S诱导出的子图$H=(V_H,E_H,\sum_H,\phi)$，若满足$\forall e\in E_H,QLS\subseteq\phi(e)$，那么就称子图$H$满足查询标签集约束，表示为$QLS\vdash H$。

易知，图G中的任意一个顶点i的度数$deg(i)=\{j|(i,j)\in E\}$。

定义4(无向图稠密度) 对于带边标签的无向图$G=(V,E,\sum,\phi)$和查询标签集QLS，若存在顶点集合$S(S\subseteq V)$，并且由顶点集S诱导出的子图H满足查询标签集约束($QLS\vdash H$)，则子图H的稠密度$Den(S)=|E_H|/|V_H|$。满足查询标签集约束条件的图G的最大稠密度：$Den_{\max}(G)=\max_{S\subseteq V\wedge QLS\vdash H}\{Den(S)\}$。

问题1 发现最稠密子图问题：给定带边标签的无向图$G=(V,E,\sum,\phi)$和查询标签集QLS，希望在图G中发现一个顶点集S，满足：

1. $S\subseteq V$；

2. 由S诱导的子图$H=(V_H,E_H,\sum_H,\phi)$，满足QLS$\vdash$H；

3. $Den(S)=Den_{\max}(G)$。

问题1所求的结果是最优解，解决问题1一般需要设计一个精确算法，但是精确算法一般仅适合小规模的图，当将其应用于大规模图数据集时，时间过长就成为精确算法的瓶颈。一些学者针对类似问题1提出了一些启发式算法[6,8]，虽然这些启发式算法相比于精确算法在时间性能上大大提高，但是这些算法返回的结果往往不能确保正确性。考虑精确算法和启发式算法的上述弊端，一些学者提出了若干近似算法[7,9,10]。近似算法不仅可以在用户可接受的时间内返回结果，而且返回的结果相对于最优解也能在一定程度上保持正确性。

定义5(α-approximation算法) $\alpha\geqslant 1$，若存在一个算法A使得针对问题1返回的结果S1满足：$S1\subseteq V$，并且由S1诱导的子图$H=(V_H,E_H,\sum_H,\phi)$，满足$QLS\vdash H$和$Den(S1)\geqslant Den_{\max}(G)/\alpha$，则称算法A是$\alpha$-approximation算法。

问题 2　给定带边标签的无向图 $G=(V,E,\sum,\phi)$和查询标签集 QLS,设计一个 α-approximation 算法,即在图 G 中发现一个顶点集 S,使其满足:

1. $S\subseteq V$;

2. 由 S 诱导的子图 $H=(V_H,E_H,\sum_H,\phi)$,$QLS \vdash H$;

3. $Den(S)=Den_{max}(G)/\alpha$。

其中,$Den_{max}(G)$是通过精确算法求出的最优解。

本文提出的在大规模图中发现最稠密子图的问题就是上述问题 2,所以第 4 节设计的算法就是针对问题 2 所提出的近似算法。

4　DSFLC 算法设计及分析

针对无向带标签图,提出了一个贪婪近似算法 DSFLC,用于解决在大规模图数据集中发现最稠密子图的问题。令 $G=(V,E,\sum,\phi)$为无向带标签图,QLS 为用户提交的查询标签集,ε 为近似算法所需的参数($\varepsilon>0$)。算法每一轮会删除一批不符合条件的顶点,算法的返回值是最优解的 $2(1+\varepsilon)$-approximation 近似解。

原始数据首先需要进行预处理,通过扫描数据集中的每条边,判断该边是否满足 QLS 约束。如果满足 QLS 约束就存储,否则删除。当原始数据经过预处理后,剩余的数据信息便构成子图 $H=(V_H,E_H,\sum_H,\phi)$,图 H 符合查询标签集约束,即:$QLS \vdash H$。初始化两个顶点集合 V 和 VS,用预处理后的子图的顶点集对它们进行初始化赋值。同时定义一个保存每轮要删除的顶点的集合 $TempVS$。每一轮要删除的顶点度数小于 $2(1+\varepsilon)*Den(V)$,其中 $Den(V)$代表由当前顶点集 V 构成的子图的稠密度。根据 TempVS 顶点集中的顶点,删除这些顶点和与其相邻的边,更新数据集。然后判断删除前的数据集 VS 和删除后的数据集 V 的稠密度,若删除后的数据集 V 的稠密度大于删除前的数据集 VS 的稠密度,则对数据集 VS 进行更新,使其等于删除后的数据集 V。当数据集 V 为空时算法结束,返回数据集 VS。具体算法见表 1。

表 1　DSFLC 算法

```
DSFLC 算法
输入:G=(V,E,∑,φ),QLS,
输出:VS
1. H=(VH,EH,∑H,φ)←数据预处理;
2.   V,VS←VH;
3.   while V 不空时
4.     TempVS←{V∈V | deg(v)≤2(1+ε) * Den(V) };
5.     根据 TempVS 顶点集中的顶点删除这些顶点和与其相邻的边;
6.     V←(V−TempVS);
7.       if Den(V) > Den(VS)
8.         VS←V;
9.       end if
10.  end while
11. return VS;
```

若想在图中发现 Top-k 最稠密子图,则只需对 DSFLC 算法进行稍微修改,添加一个存储稠密子图数据的集合 DSS。在顶点数据集 VS 发生变化的前 K 轮,只需要将 VS 中的结果依次放入 DSS 集合中,DSS 集合(小顶堆)会自动根据放入的数据集 VS 诱导的子图的稠密度按照从小到大的顺序进行排序。当 VS 第 $K+1$ 次发生变化时,只需要将此时由 VS 诱导的子图的稠密度和 DSS 中最小稠密度进行比较,若大于 DSS 中最小稠密度,则将此时的 VS 集合添入 DSS 集中,并对其顺序进行更新。当 V 集合为空时,返回 DSS 集合,其代表的是 $Top\text{-}K$ 最稠密子图。算法见表 2。

表 2　发现 Top-k 最稠密子图算法

```
发现 Top-k 最稠密子图算法
输入:G=( V,E,∑,φ),QLS,k
输出:DSS
1. H=(VH,EH,∑H,φ)←数据预处理;
2. 初始化顶点集合 VS,V←VH;
3. while VS 不空 do
4.   del_VS={i∈VS | deg(i)≤2(1+ε) * Den(VS)} ;
5.   VS=VS−del_VS;
6.   if (Den(VS)>Den(S)) then
7.     if (DSS,size<k) then
8.       S←VS;
9.         DSS←DSS∪S;
10.    else
11.      用当前稠密子图将 DSS 集合中最小稠密度的集合替换;
12.      end if
13.    end if
14. end while
15. return DSS
```

定理 1　DSFLC 算法的解是最优解的 $2(1+\varepsilon)$近似解。

证明:假设顶点子集 opt_V 为给定标签集 QLS 约束条件下最稠密子图发现问题的最优解,由稠密度定义知,若 opt_V 为最优解,则删除最优解中的任何一个顶点,其稠密度值下降。因此最优解中的任何一个顶点的度数均高于平均度数,即 $\forall i\in opt\_V, deg_{opt\_v}(i)\geq Den(opt\_V)$。在算法执行过程中,当 opt_V 中的顶点 i 第一次被删除时,i 一定也同时属于 $TempVS$。由于 $opt\_V\subseteq V$,因此存在 $deg_{opt\_v}(i)\leq 2(1+\varepsilon)*Den(V)$,根据不等式的关系:$Den(opt\_V)\leq 2(1+\varepsilon)*Den(V)$,易知 $Den(V)\geq Den(opt\_V)/2(1+\varepsilon)$。

发现 Top-k 最稠密子图算法只需在上述分析中考虑 DSS 集合的添加元素(稠密子图)和元素的替换策略。因此发现 Top-k 最稠密子图算法是 $2(1+\varepsilon)$-approximation 算法。

定理 2　DSFLC 算法执行 $O(\log_{1+\varepsilon}n)$轮后结束。

证明:$2|E(V)|=\sum_{i\in TempVS}deg(i)+\sum_{i\in(V-TempVS)}deg(i)$,由于 $\sum_{i\in TempVS}deg(i)>0$,因此 $2(1+\varepsilon)(|V|-|TempVS|)*|E(V)|/|V|$。

化简上式可知:

$1/(1+\varepsilon)>(|V|-|TempVS|)/|V|$,即 $(1+\varepsilon)<|V|/(|V|-|TempVS|)$,存在 $|TempVS|>\varepsilon*(|V|/(1+\varepsilon))$。$V$ 的规模每轮至少减少 $1/(1+\varepsilon)$,易知经过 $O(\log_{1+\varepsilon}n)$轮,算法结束。

同理可以证明发现 Top-k 最稠密子图算法也只需执行 $O(\log_{1+\varepsilon})$轮。

5　实验

5.1　数据集描述

本文采用的数据集是从 DBLP 数据库[5] 提供的 dblp.xml 中抽取出来的。dblp.xml 数据集约有 1G 大小,数据采用的存储形式是 xml 格式,dblp.xml 中包含的子节点类型有:article, inproceeding, proceeding, book, incollection, phdthesis, mastersthesis. www。本实验暂时只研究 article 类型的信息,其余 7 种类型的信息暂时不进行抽取。

由于本实验需要构造一个合作关系图 G,其中顶点 V 代表的是作者(通过作者名字表示),边 E 上的信息是两个作者合作写的文章名字(title)。通过自定义停用词词库,对 title

中的单词去除停用词，title 中剩余的单词就可以作为两个作者间的关系。因此经过 XML 文件的提取过程，以及使用停用词词库削减边上标签集的规模等过程后，便可以构造出本实验所需的测试数据集。

5.2 Twitter Storm 介绍

MapReduce，Hadoop 等相关技术可以处理海量数据，但是这些数据处理技术均不是针对实时情况的，它们的设计目的也不是进行实时计算。但是随着大规模实时数据处理已经越来越成为一种业务需求，Twitter Storm 公司开发了一个号称“实时版”的 Hadoop 流式数据实时处理框架。Storm 如同 Hadoop 一样，既可以简化并行批处理数据处理的过程，也提供了简化并行实时数据处理的原语。Storm 的高伸缩性体现在每秒可处理的消息量很大[4]。为了扩展一个实时计算任务，只需要添加机器到集群，并提高这个计算任务的并行度即可。Storm 使用 Zookeeper 来协调集群内的各种配置，一个实时计算应用程序的逻辑在 Storm 里面被封装到 topology 对象中，Storm 里面的 topology 相当于 Hadoop 中的一个 MapReduce Job。Storm 中的信息流是一个没有边界的 tuple 序列，数据信息被封装到 tuple 中，在系统中进行传递。本实验的逻辑拓扑如图 2 所示。

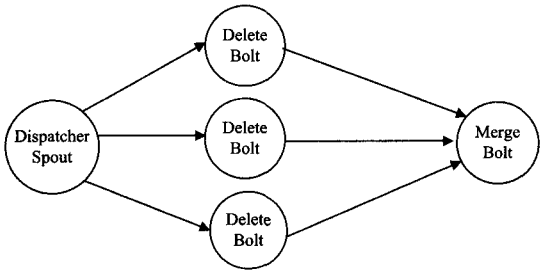

图 2　Storm 逻辑拓扑图

其中 DispatcherSpout 从抽取的合作关系数据集中读取数据，然后转发给 DeleteBolt，DeleteBolt 删除不包含在 QLS 的边。由于这个操作是琐碎的，而且是耗时的，因此采用 Storm 流式处理框架就可以显现出 Storm 的优越性。MergeBolt 搜集符合条件的数据，然后执行算法，并返回实验结果。

5.3 实验结果分析

实验编写语言为 JAVA，并在 Ubuntu 10.04LTS 上测试，测试用的主机 CPU 为：Intel® Core™2 CPU 4300@2.80 GHz，内存为 4GB。实验所需的平台是 Twitter 公司提供的开源 Storm 框架。本实验中，Storm 的伪分布模式搭建是使用一台上述单机，使其既作为 Spout 节点分发数据，又作为 Bolt 节点进行逻辑处理；Storm 的完全分布模式是由多台上述单机搭建而成的，选取其中一台作为 Spout 节点，剩下的机器作为 Bolt 节点。

本实验测试了 4 组查询标签集，对于每一组查询标签集，首先统计符合查询标签集约束的边的频数。在这里令 $Q1=\{\text{parallel, graph}\}$，$Q2=\{\text{structure, index}\}$，$Q3=\{\text{dense, graph}\}$，$Q4=\{\text{prototype, system}\}$。之后分析引入 Storm 框架，将算法并行化处理后的执行时间与单机情况下未进行并行化处理的执行时间进行比较。然后研究完全分布模式下增加机器的台数是否可以大幅度提高算法的执行时间，并分析进一步加快算法执行时间的原因。最后通过调整参数 ε 值来观察稠密度的相应变化。

在图 3—图 8 中，SQ 代表单机情况下对应的标签集，FQ 代表(伪)分布模式下对应的标签集。

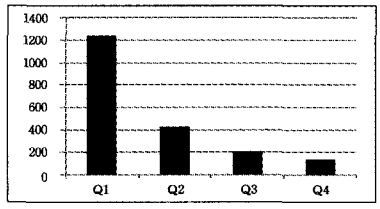

图 3　4 组查询标签集对应的频数

通过对比图 4、图 5 的结果，易知仅仅通过单机伪分布模式就可以将算法的执行时间大幅度降低。图 6、图 7 表明算法在完全分布模式下的执行时间可以进一步降低。

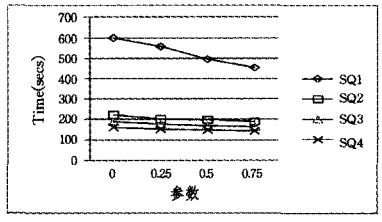

图 4　单机未并行化算法的执行时间

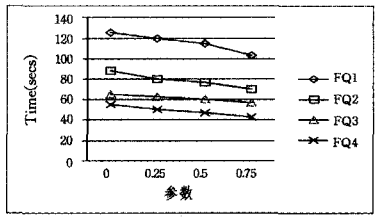

图 5　单机伪分布模式下算法的执行时间

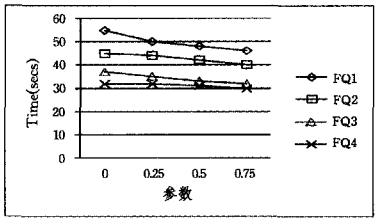

图 6　算法在 5 台单机搭建的 Storm 平台上的执行时间

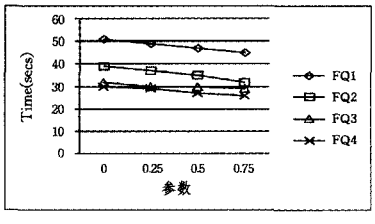

图 7　算法在 7 台单机搭建的 Storm 平台上的执行时间

由图 6 和图 7 可知，当 Storm 集群中的机器达到一定数量时，单纯增加集群中机器的台数并不能大幅度地降低算法的执行时间。要想进一步降低算法的执行时间，应该考虑将算法进行完全并行化处理，而不仅仅局限于数据的预处理阶段。这也是今后亟待解决的一个难题。

最后在由 7 台单机搭建的 Storm 完全分布模式平台下，通过为参数 ε 取不同的值，观察稠密度的变化情况。如图 8 所示，当 ε 值增大时，返回的结果集的范围可能会增大，也就是说返回的解的正确性保证有所降低。但是通过实验结果可知，DSFLC 算法针对大规模数据集时，对 ε 值的变化不会太敏感($0<\varepsilon<1$)，返回的解也不会随着 ε 值的变化而产生剧烈

的变化，在这个方面可以体现 DSFLC 算法具有较好的可扩展性。

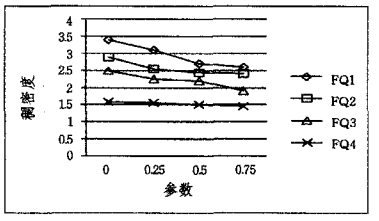

图 8　随着 ε 变化，稠密度的变化

结束语　本文研究了在大规模图数据集中发现最稠密子图问题，提出了一个近似算法 DSFLC：对于任何参数 $\varepsilon(\varepsilon>0)$，DSFLC 算法只需要扫描大规模数据集 $O(\log_{1+\varepsilon}n)$ 次，便可以获得最稠密子图问题最优解的 $2(1+\varepsilon)$ 近似解。基于 Twitter Storm 流式框架，将 DSFLC 算法进行并行化处理，大大提高了算法的执行效率。通过实验测试，验证了算法的有效性和可扩展性，确保了近似算法的解具有较好的理论保证。综上所述，基于 DSFLC 算法，配合使用 Storm 框架可以很好地在大规模流式图数据集中发现最稠密子图。将算法完全进行并行化处理是亟待解决的一个难题，也是我们今后的研究方向。

参 考 文 献

[1] Yon D, Filippo G, Marco P. Extraction and classification of dense communities in the Web[C]//WWW 2007. 2007: 461-470

[2] Newman M E J. Modularity and community structure in networks[C]//PNAS 2006. 2006: 8577-8582

[3] William F G, Steve L, Lee G C. Efficient identification of Web communities[C]//KDD 2000. 2000: 150-160

[4] https://github.com/nathanmarz/storm

[5] http://www.dblp.org/db/

[6] Goldberg A V. Finding a maximum density Subgraph[R]. UCB/CSD-84-171. EECS Department, University of California, Berkeley, 1984

[7] Charikar M. Greedy approximation algorithms for finding dense components in a graph[C]//APPROX 2000. 2000: 84-90

[8] Lawler F. Combinatorial Optimization: Networks and Matroids [M]. Holt, Rinehart, and Winston, 1976

[9] Gibson D, Kumar R, Tomkins A. Discovering large dense subgraphs in massive graphs[C]//VLDB 2005. 2005: 721-732

[10] Bahmani B, umar R. Sergei Vassilvitskii: D ensest Subgraph in Streaming and MapReduce[C]//VLDB 2012. 2012: 454-465

[11] Silva A, Meira W Jr, Zaki M J. Mining Attribute-structure Correlated Patterns in Large Attributed Graphs[C]//VLDB. 2012: 466-477

[12] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters[C]//ACM 2008. 2008: 107-113

[13] Bu Ying-yi, Howe B, Balazinska M, et al. HaLoop: Efficient Iterative Data Processing on Large Clusters [C]//VLDB 2010. 2010: 285-296

[14] Abouzeid A, Bajda-Pawlikowski K, Abadi D J, et al. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads[C]//VLDB 2009. 2009: 922-933

[15] Condie T, Conway N, Alvaro P, et al. MapReduce Online[C]// NSDI'10 Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation. USENIX Association Berkeley, CA, USA, 2010: 21

(上接第 253 页)

[2] Xu Bao-wen, Qian Ju, Zhang Xiao-fang, et al. A brief survey of program slicing[J]. SIGSOFT Softw. Eng. Notes, 2005, 30(2): 1-36

[3] Abadi A, Ettinger R, Feldman Y A. Fine slicing: theory and applications for computation extraction[C]// Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering, 2012. Tallinn: Springer-Verlag, 2012: 471-485

[4] Zhang Xiang-yu, Gupta N, Gupta R. A study of effectiveness of dynamic slicing in locating real faults[J]. Empirical Softw. Engg., 2007, 12(2): 143-160

[5] Korel B, Yalamanchili S. Forward computation of dynamic program slices[C]//Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis, 1994. Washington: ACM, 1994: 66-79

[6] Zhang Xiang-yu, Gupta R, Zhang You-tao. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams[C]// Proceedings of the 26th International Conference on Software Engineering, 2004. Edinburgh: IEEE Computer Society, 2004: 502-511

[7] Zhang Xiang-yu, Gupta R, Zhang You-tao. Precise dynamic slicing algorithms[C]//Proceedings of the 25th International Conference on Software Engineering, 2003. Portland: IEEE Computer Society, 2003: 319-329

[8] Nagarajan V, Jeffrey D, Gupta R, et al. A system for debugging via online tracing and dynamic slicing[J]. Software-Practice & Experience, 2012, 42(11): 1431-1431

[9] Alfred V A, Monica S L, Ravi S, et al. Compilers Principles, Techniques and Tools(第 2 版)[M]. 赵建华, 郑滔, 戴新宇, 译. 北京: 机械工业出版社, 2009: 382-402

[10] Do H, Elbaum S, Rothermel G. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact[J]. Empirical Softw. Engg., 2005, 10(4): 405-435

[11] 王雪莲, 赵瑞莲, 李立健. 一种用于测试数据生成的动态程序切片算法[J]. 计算机应用, 2005, 25(6): 1445-1447

[12] Gyimothy T, Beszedes A, Forgacs I. An efficient relevant slicing method for debugging[J]. SIGSOFT Softw. Eng. Notes, 1999, 24(6): 303-321

[13] Beszedes A, Gergely T, Szabo Z M, et al. Dynamic slicing method for maintenance of large C programs[C]//Proceedings of the fifth European Conference on Software Maintenance and Reengineering, 2001. Lisbon: IEEE Computer Society, 2001: 105-113

[14] Masri W, Podgurski A, Leon D. Detecting and Debugging Insecure Information Flows[C]// Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004. Saint-Malo: IEEE Computer Society, 2004: 198-209

[15] Masri W, Nahas N, Podgurski A. Memoized Forward Computation of Dynamic Slices[C]// Proceedings of the 17th International Symposium on Software Reliability Engineering, 2006. Raleigh: IEEE Computer Society, 2006: 23-32