Journal of Computers Vol. 28, No. 5, 2017, pp. 51-61
doi:10.3966/199115992017102805005

D-GSPerturb: A Distributed Social Privacy Protection Algorithm based on Graph Structure Perturbation

Xiao-lin Zhang[1*], Wen-chao Zhang[1], Chen Zhang[1],
Li-Xin Liu[1] and Xiao-Yu He[1]

[1] School of Information Engineering, Inner Mongolia University of Science and Technology
Baotou, Inner Mongolia, 014010, China
zhangxl@imust.cn

Received 18 June 2016; Revised 19 February 2017; Accepted 19 February 2017

Abstract. The traditional privacy protection algorithm does not meet actual application requirements of processing large-scale graph data in terms of efficiency or capability. D-GSPerturb is a distributed social privacy protection algorithm based on graph structure perturbation; it is proposed to solve link privacy issues in social networks. The present vertex-centric algorithm can search large-scale social networks for reachable vertexes, transfer reachable information, and randomly perturb edges through between-vertex messaging, vertex value updating, and multi-iteration in programming. The experimental results show that D-GSPerturb not only improves the processing speed of large-scale graph data but also ensures the privacy protection effect and availability of data published.

Keywords: big data, D-GSPerturb, edge random perturbation, privacy protection, social network

1 Introduction

With the development of Internet technology, social networks are characterized by large-scale and diverse data. However, traditional privacy protection algorithms with poor effectiveness and low efficiency cannot satisfactorily process large data sets. Thus, parallel computing has become a trend for social network privacy protection in the future [1].

In some cases, the information associated with the edges in the social network may be sensitive, identifying whether the two nodes have a certain relationship or identifying the two endpoint of a link can be considered as a privacy disclosure [2-4]. Therefore, it is necessary to protect link privacy in the social network release.

Fard and Wang propose a neighborhood randomization method to address the problem of link privacy [5]. The basic concept of neighborhood randomization (*NR*) is defined as follows. An edge is hidden by concealing the target or source vertexes of the edge; the pseudo-target vertex w is selected from local neighbors that are structurally nearer to the source vertex u. The initial boundary is kept in the published graph and relies on the probability p; the target vertexes of the initial boundary are substituted in the probability $1-p$, and the pseudo-target vertex w is randomly selected from local neighbors. However, this method is a privacy protection method in a stand-alone environment, which needs to be processed by links one by one, and the poor effectiveness and low efficiency cannot satisfactorily process large data sets.

This paper focuses on the efficient and rapid privacy protection of large-scale data in distributed environment while maintaining the availability of data. This paper makes the following contributions.

(1) A distributed social network privacy protection algorithm (D-GSPerturb) based on Graph Structure Perturbation is designed to improve the efficiency of processing large-scale graph data.

(2) D-GSPerturb based on Pregel-like model can search for reachable vertexes, transfer reachable information, and randomly perturb edges in the distributed environment.

(3) The high efficiency of the D-GSPerturb method and the data availability of the published data are

[*] Corresponding Author

verified by a large number of experimental tests and analyzes on the real data set.

The rest of the paper is organized as follows. We review related work in Section 2, define preliminaries in section 3, present the distributing algorithm in Section 4, and evaluate the processing efficiency and utility of sanitized graphs in Section 5, we conclude the paper.

2 Related Work

Various algorithms have been presented for social network privacy protection [6-10]. Bhattacharya and Mani propose an iterative algorithm to generate k-anonymize vertex degree sequence of a given social network graph in order to protect the graph against passive attack [6]. Wang and Zheng propose two k-anonymization algorithms to protect a social network against the CFP attacks [7]. One algorithm is based on adding dummy vertices, the other algorithm is based on edge modification. Zhang, Liu and Lin design a hierarchical authorization and capability delegation (HACD) model [8]. Based on this model, it propose a novel utility-based popularity anonymization (UPA) scheme, which integrates proxy re-encryption with keyword search techniques, to tackle authentic popularity disclosure in online social networks. Qian, Li and Zhang introduce knowledge graphs to explicitly express arbitrary prior belief of the attacker for any individual user [9]. The processes of de-anonymization and privacy inference are accordingly formulated based on knowledge graphs.

The basic idea of graph structure perturbation is to protect the data privacy of a social network by perturbing and modifying the graph data randomly. Perturbing a graph is mainly based on randomly adding/deleting *m* edges, switching edge endpoints, and so on. In Reference [10-12], graph structure perturbation is presented. Ying and Wu propose relying on equiprobability; social privacy protection could be achieved by randomly adding, deleting, or switching edges [10]. Zhang and Zhang propose an algorithm to compute the boundary recognition probability based on the connection density between two equivalent classes [11]. The greedy algorithm is applied to minimize the number of edges to be deleted or switched. Thus, this algorithm keeps the possibility of boundary recognition below the given threshold. Fard and Wang propose an algorithm based on perturbing the subgraph edge, particularly after the initial graph is split into several subgraphs, to add/delete *m* edges to/from the subgraph randomly [12]. The aforementioned algorithms for perturbing graph structures obtained randomized and even useless initial graphs. Considering problems of this kind, Fard and Wang introduce an algorithm to randomly perturb the edges in a local neighborhood [5].

In Reference [13-14], big graph privacy is presented. Zakerzadeh, Aggarwal and Barker [13] leveraged the MapReduce framework to create a scalable algorithm that focuses on the sensitive content at the vertexes rather than on the structure. However, MapReduce framework has the potential disadvantages (e.g. large scheduling and disk access overhead) to process the large-scale graph data. Zhang, Guo and Wang [14] proposed a personalized social network privacy protection method based on the Pregel-like system. The method provides different types of privacy protection for different vertex, and adopts the "Think Like a Vertex" concept in the Pregel-like system. But the method solves vertex privacy issues in social networks. Many distribute privacy models [15-16] have recently been proposed for private release of network data. Yuan and Chen [16] followed the SMC approach and designed a secure protocol SP to generate a S-Clustering graph to achieve k-anonymization of bipartite graph. Jurcayk and Li [16] proposed a protocol, namely Distributed Anonymization that is built on top of the k-anonymity and l-diversity principles and the greedy topdown Mondrian multidimensional k-anonymization algorithm. However, the two methods are both completed in the distributed environment rather than leverage distributed compute.

BSP (Bulk Synchronous Parallel model) is a typical computing model to process large-scale graph data in a cloud computing environment [17]. In Reference [17], the BSP model is a message-based parallel execution designed for iterative synchronization in large-scale graph computing. Google Pregel system [18] is the most sophisticated graph processing system based on the BSP model; this system drives the development of a series of Pregel-like systems [19], such as Apache Hama, Yahoo! Giraph, Spark Graphx, and so on.

The existing social network privacy protection method is mainly based on the stand-alone environment, is not suitable for dealing with massive social network data. Considering the limitations of those traditional algorithms in processing large-scale graph data, D-GSPerturb is proposed for social network privacy protection.

3 Preliminaries

3.1 Definition 1: Digraph of Social Network.

It is defined as $G = \{V, E\}$. V denotes a set of user vertexes; each vertex corresponds to a real user in a social network. E denotes a set of social connections. Every edge $<u, v>$ is mapped to a directed social connection between users; these connections indicate the direction from user u to user v; and u is called the source vertex, v is called the target vertex. A simple digraph of a social network G_1 is shown in Fig. 1 as an example.

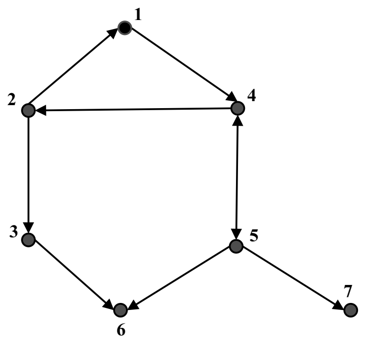


Fig.1. A directed social network graph G_1

3.2 Definition 2: Link Steal

In the directed graph G, it is assumed that the vertexes$\{u, v\} \in V$, $<u, v> \in E$，$<u, w> \notin E$. The operation removes edge $<u, v>$ and adds edge $<u, w>$ to E, where w is called false target vertex of vertex u.

For example, in the digraph of social network G_1, edge $<1, 4> \in E$, $<1, 5> \notin E$, removes edge <1, 4> and add edge <1, 5> to G_1 are called link steal, where vertex 5 is called false destination vertex.

3.3 Definition 3: Neighbor-r

Given that integer $r \geq 0$, and the source vertexes of the graph are indicated by u, the set of all reachable target vertexes in digraph G corresponds to the source vertexes u within r JumpSteps as indicated by $N_r(u)$.

For example, in the digraph of social network G_1, when $u = 1$, $N_1(1) = \{1, 4\}$, $N_2(1) = \{1, 4, 2, 5\}$, $N_3(1) = \{1, 4, 2, 5, 6, 3, 7\}$.

In digraph G, the set of target vertexes reachable from the source vertexes u via any arbitrary vertex is indicated by $N^*(u)$. The set of neighbors reachable from the source vertexes u within one JumpStep is indicated by $Dst(u)$.

For example, in the digraph of social network G_1, $N^*(1) = N_3(1) = \{1, 4, 2, 5, 6, 3, 7\}$, $Dst(1) = \{4\}$.

The candidate set of pseudo-target vertex w of the source vertexes u is indicated by $PTS(u, r, s)$. r denotes the radius of Neighbor r of the source vertexes u $(r > 1)$. s denotes the number of vertexes contained in $PTS(u, r, s)$; $s \geq |Dst(u)|$. Based on the assumption that $s_1 = |N_r(u)| - |N_1(u)|$, $s_2 = |N^*(u)| - |N_1(u)|$. $s_2 \geq s_1 \geq s$. The selected set $PTS(u, r, s)$ meets any of the following three conditions:

(1) $s_1 > s$, indicating that s vertexes are available in $N_r(u) - N_1(u)$, as well as s vertexes in $PTS(u, r, s)$ which are randomly selected from $N_r(u) - N_1(u)$.

(2) $s_1 < s \leq s_2$, indicating that the number of vertexes in $N_r(u) - N_1(u)$ is less than s, all vertexes of $N_r(u) - N_1(u)$ are included in $PTS(u, r, s)$, and s-s_1 vertexes are randomly selected from $N^*(u) - N_r(u)$.

(3) $s_2 < s$, indicating that the number of vertexes in $N^*(u)- N_1(u)$ is less than s, all vertexes of $N^*(u) - N_1(u)$ are included in $PTS(u, r, s)$, and $s - s_2$ vertexes are randomly selected from $V- N^*(u)$.

For example, in the digraph of social network G_1, $V = \{1, 2, 3, 4, 5, 6, 7\}$. Assuming that $r = 2$, $s = 2$, for the source vertexe $u = 1$, $s_1 = 2 \geq s$, and $PTS(1, 2, 2) = \{2, 5\}$.

3.4 Definition 4: Reachable Information and Reachable Information List

The assemblage of the target vertexes and the distance from the source vertexes are the reachable information. The assemblage of all the reachable information of the source vertexes is called the reachable information list. This list is saved in the data form of the map; the key indicates the target vertex identifier, and the value indicates the distance between the source vertexes and the target vertexes in this list.

For example, in the digraph of social network G_1,for the source vertex $u = 2$, the reachable information list of vertex 2 is {(4, 2),(6,2),(5,3),(7,4)}, and every element is the reachable information of vertex 2.

4 D-GSPerturb for Privacy Protection

4.1 Principle

The Pregel system is vertex-centric; it requires every vertex to conduct parallel computing in the "comp-comm-sync" model. Before the computing process, all graph vertexes are set as either *active* or *inactive*. A pregel job consists of supersteps executed in several sequences. In every superstep, each graph vertex receives messages from neighboring vertexes of the preceding superstep. Once the messages are received, the vertex checks whether to update values, send messages, and change the current status. In the present study, D-GSPerturb is proposed for privacy protection based on the Pregel-like system. Its system architecture is shown in Fig. 2.

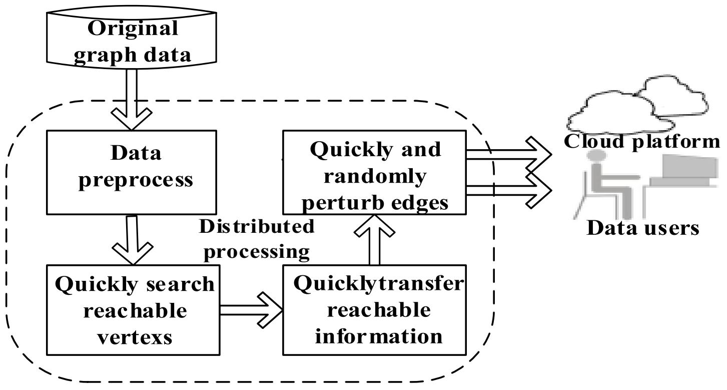

Original graph data
Data preprocess
Quickly and randomly perturb edges
Distributed processing
Quickly search reachable vertexs
Quicklytransfer reachable information
Cloud platform
Data users

Fig. 2. System architecture

Almost importantly, the initial graph data is processed into data types supported by the Pregel-like system. Then, all vertexes reachable from those sources are searched in the graph of the social network. Based on the found vertexes, pseudo edges are created from the target vertexes to the source vertexes; these created pseudo edges enable the sending of reachable information to the source vertexes. Lastly, *PTS*(u, r, s) is computed at every source vertex according to the list of reachable information. Based on the obtained result of *PTS*(u, r, s), every edge of the source vertexes is perturbed randomly. The Pregel-like model is used to search for reachable vertexes, transfer reachable information, and randomly perturb edges; supersteps are executed in several sequences for large-scale social privacy protection, where in secure data are available for users or cloud platforms.

4.2 Description

Quick Search for Reachable vertexes. In a distributed environment, graph vertexes are assigned to different computing vertexes; each of the vertexes is initialized. In every superstep, active vertexes receive messages and check whether every vertex identifier given in messages exists in the vertexes value. This vertexes value is saved in the data form of a Map with a key indicating the identifier of source vertexes and a value indicating the number of supersteps. If the vertex identifier does not exist, then the vertex value is updated with messaging. If this value fails to update, then the corresponding vertexes are set to *inactive*. Every vertex is searched for reachable vertexes with the specific procedures of Algorithm 1.

Searching for vertexes reachable from the source vertexes mainly involves the following two cases:

① When s*uperstep*=0, vertexes with out-degree neighbors are *active*, whereas the remaining vertexes are set to *inactive*; the active vertexes send their identifiers to all their respective neighbors (1-10).

② When s*uperstep* ≠0 the vertex statuses are checked according to the received message. Going through message list of active vertexes, the vertexes identifier not given in the vertexes value are searched using the updated the vertexes value and then sent to the neighboring vertexes. If all vertex identifiers are found, then the vertexes are *inactive* (11-25). Case ② is repeated until all vertexes are *inactive*; then, the program stops.

Algorithm.1 search reachable vertexes.

```
Input：Message transfer between supersteps messages
Output: The list of reachable vertexs reachMapList
1  reachMapList←φ;
2  long step = getSuperstep();
3  if step == 0 then
4     setValue(reachMapList.put(VertexId,0));
5     if getEdges == null then
6          voteToHalt();  return;
7     else
8         msgList.add(VertexId);
9         sendMessToNeighbors (msgList);
10    end if
11 else
12    if messages == null then
13         voteToHalt();  return;
14    else
15        reachMapList = getValue();
16        for each messList in messages do
17             for each VertexId in messList do
18                  if isNotExistInReachMapList (VertexId) then
19                       setValue(reachMapList.put(VertexId, step);
20                       msgList.add(VertexId);
21                 end if
22        if msgList.size == 0 then
23            voteToHalt();  return;
24        sendMessToNeighbors (msgList);
25 end if
26 return reachMapList;
```

For example, in the simple graph of a social network in Fig. 1, when s*uperstep*=0, the first algorithm prevails; vertex 5 sends identifier to vertexes 4 and 7, as shown in Fig. 3(a). When s*uperstep*=1, the second algorithm prevails, as shown in Fig.3(b); the vertexes value after being updated consists of the contents in brackets in Fig. 3(b), e.g., the list of vertexes reachable to vertex 4 are shown as {(4,0), (1,1), (5,1)}. The case of s*uperstep*=1 occurs repeatedly until all vertexes are *inactive*; then, the program stops. As shown in Fig. 1, the program stops after six supersteps. The results of Algorithm 1 are shown in Table 1. For example, in vertex 1, source vertexes 1, 2, 4, and 5 can reach vertex 1 and have distances of 0, 1, 2, and 3, respectively.

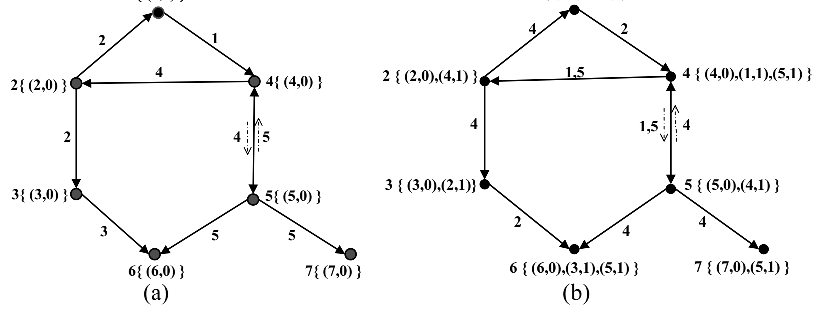


Fig. 3. Search for reachable vertexes

Table 1. Search results of reachable vertexes

Vertex Id	Vertex Value
1	(1,0)(2,1)(4,2)(5,3)
2	(2,0)(4,1)(1,2)(5,2)
3	(3,0)(2,1)(4,2)(1,3)(5,3)
4	(4,0)(1,1)(5,1)(2,2)
5	(5,0)(4,1)(1,2)(2,3)
6	(6,0)(3,1)(5,1)(4,2)(2,2)(1,3)
7	(7,0)(5,1)(4,2)(1,3)(2,4)

Quick random edge perturbation. The vertexes value is ergodic to every vertex. First, the source vertex identifiers are searched within the range of jumpstep greater than or equal to two; pseudo edges (such as those in Figs. 4a and 4b) are obtained from the target vertexes to the source vertexes. Then, the target vertexes are sent in the Pregel-like system reachable information to the source vertexes via the pseudo edges. Finally, the source vertexes compute the set of $PTS(u,r,s)$ according to the reachable information list and complete the edge random perturbation.

For Algorithm 2, transferring reachable information mainly involves the following cases:

① When *superstep*=0, all source vertexes are *active*, whereas the remaining vertexes are *inactive*. The source vertexes send reachable information to the target vertexes (3–11) through the pseudo edges.

② When s*uperstep*=1, vertexes without received messages are *inactive*, whereas those with received messages are *active*. The target vertexes receive messages and save all reachable information in the vertex value; then, the program stops (12–20).

Algorithm.2 transfer reachable information.

```
Input：Message transfer between SuperSteps messages
Output: The list of reachable information ReachInforList
1 ReachInforList←φ;
2 long step = getSuperstep();
3 if step = 0 then
4    if getEdges == null then
5         voteToHalt();  return;
6     reachMapList = getValue();
7     for each value in reachMapList do
8       // Reachable information is sent via pseudo edges to the
source vertexes
9          msgMap.put(vertexId,distance);
10         sendMess(targetId, msgMap);
11     end for
12 else if step =1 then
13     if messages == null then
14         voteToHalt();  return;
15     for each mess in messages do
16         ReachInforList.put(destId,distance);
17     setValue(ReachInforList);
18     voteToHalt();
19     end if
20 end if
21 return ReachInforList;
```

As shown in Fig. 4(a), Algorithm 2 only requires two supersteps to send all reachable information. For example, in vertex7, when s*uperstep*=0, the reachable information (7,3) is sent to vertex 1, the reachable information (7,4) is sent to vertex 2, and the reachable information (7,2) is sent to vertex 4, as shown in Figs. 4(a) and 4(b). When s*uperstep*=1, vertexes 1, 2, and 4 receive reachable information that is saved in the list of reachable information. The list of reachable information of every vertex consists of the contents in brackets, as shown in Fig. 4(b).

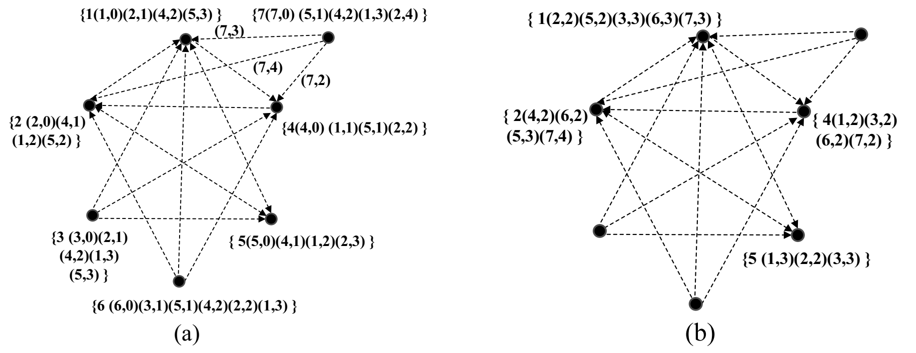


Fig. 4. Transmission of reachable information

The random perturbation algorithm first obtains a reachable information list for each source vertex. Then, according to the local neighbor radius *r*, *PTS* (*u*) size *s*, calculate the *PTS* (*u*, *r*, *s*). Finally, for each edge of the source vertex, the edge random perturbation is performed according to the random probability *p*. The random perturbation of each vertex is shown in Algorithm 3.

Algorithm.3 random perturbation.

```
  Input : A directed graph G,neighborhood radius r,the size s of PTS
(u),link retention probability p.
  Output: The sanitized G*.
  1 G*←φ;
  2 if getEdges == null then
  3     voteToHalt();  return;
  4 else
  5     ReachInforList = getValue();
  6     PTSuList = ComputePTSuList(ReachInforList，r，s);
  7     for each edge in edgeList do
  8        EdgeRandom Perturb(p， PTSuList);
  9     voteToHalt();
  10 end if
  11 return G*;
```

4.3 Analysis of Present Algorithm

Analysis of data availability. The D-GSPerturb privacy protection algorithm aims to randomly perturb edges based on local neighborhood; it greatly reduces the information loss of the graph structure. In this study, the data availability is explored by analyzing changes in the graph structure before and after perturbation. The analysis mainly has the following measures: the average shortest path length applied in messaging, search, computing, and so on; the maximum eigenvector that indicates the thresholds for the maximum degree, chromatic number, and viral transmission closely related to the maximum eigenvalue; the degree centrality, which indicates that any vertex with a higher popularity is more popular in the social network; and the closeness centrality, which indicates that the higher the closeness centrality, the quicker the messaging. Changes in these four parameters are enough to explain changes in the graph structure data.

The graph of a social network before and after perturbation is evaluated using *RelativeError* $= u - u^{*} / u$, where u and u^{*} are the measures for the graph before and after perturbation, respectively. The following two aspects are used to evaluate the graph:the average shortest path length and the maximum eigenvalue, which indicates that the smaller the value, the higher the data availability. The variations in the degree and closeness centrality before and after perturbation are measured by analyzing Spearman's correlation similarity between the ranking list L of vertexes of the initial graph and the ranking list L^{*} of vertexes of the graph after perturbation. The greater the values of these two measures, the higher the data availability.

5 Results and Analysis

The present experiment was conducted on a Hadoop cluster consisting of 5 servers with hardware configuration requirements of CPU 1.80 GHz and RAM 16 GB. Apart from using the Pregel-like system model Giraph-1.1.0 and Spark-1.2.1, the experiment also adopted a real life directed social network data that we called LiveJournal, a comprehensive SNS online dating website with a total of 4,847,571 subscribers and 68,993,773 social connections. In the experiment, the probability of substituting target vertexes is expressed as δ; $\delta = 1 - p$.

5.1 Analysis of Processing Time

The D-GSPerturb privacy protection algorithm is designed to enhance the efficiency of processing large-scale data sets of a social network and ensuring the privacy protection of traditional edge random perturbation algorithms (NR). To meet the experimental design, the initial data set was segmented into eight equal parts; the data were reengineered into four different datasets by 1:2:4:8. Based on those four datasets, Giraph and GraphX were used; the digraph data of the social network was processed as the D-GSPerturb algorithm.

As shown in Fig. 5(r = 2, s = 2|$Dst(u)$|, δ = 0.5), insignificant differences exist between the NR algorithm and the D-GSPerturb algorithm in the runtime. However, as the data size (split_3 and split_4) exponentially increased, the processing efficiency of the NR algorithm was significantly lower than that of the D-GSPerturb algorithm.

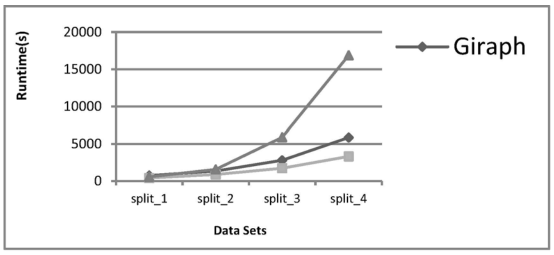


Fig. 5. Comparison chart of processing time

5.2 Analysis of Data Availability

Figs. 6 and Fig. 7 show the effects of different neighborhood radii on the average shortest path length and the maximum eigenvector (Note that the x-axis coordinate denotes the local neighborhood radius; the y-axis coordinate denotes the error rates of the graph structures before and after perturbation; δ = 0.5; and x denotes |$Dst(u)$|.). As shown by the two charts, the average shortest path length and maximum eigenvector increase as the neighborhood radius increases, whereas the data availability gradually decreases.

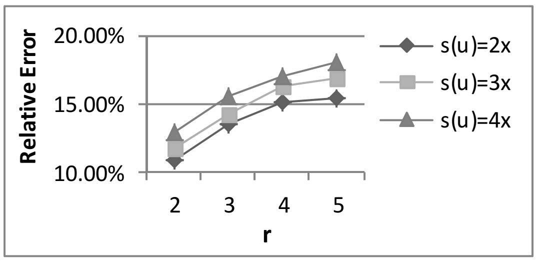


Fig. 6. Error rate in average shortest path length

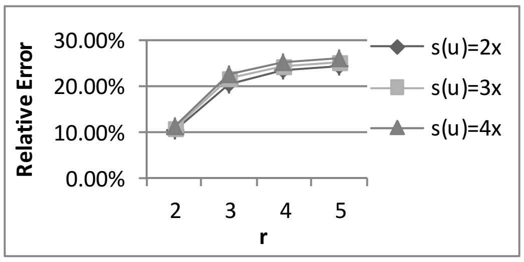


Fig. 7. Maximum eigenvector error rate

Figs. 8 and Fig. 9 show the effects of different neighborhood radii on the degree centrality and the closeness centrality, respectively. (Note that the x-axis coordinate denotes the neighborhood radius; the y-axis coordinate denotes the structural similarity between the graphs before and after perturbation; δ = 0.5, and x denotes |$Dst(u)$|). As shown by these two charts, the similarity between the graphs before and after perturbation in terms of degree centrality and closeness centrality gradually decreases with the neighborhood radius; the data availability shows a downward trend.

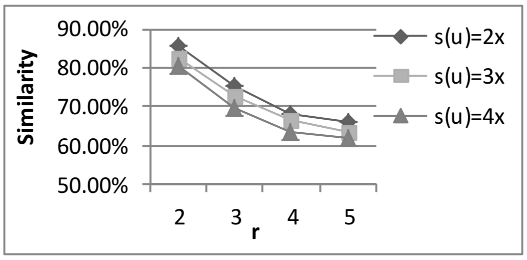


Fig. 8. Similarity in degree centrality

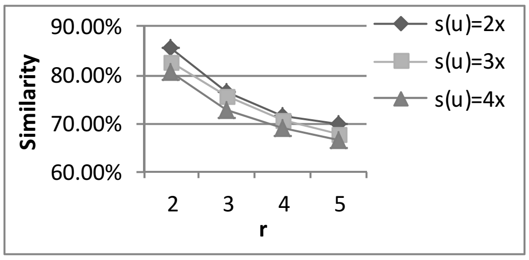
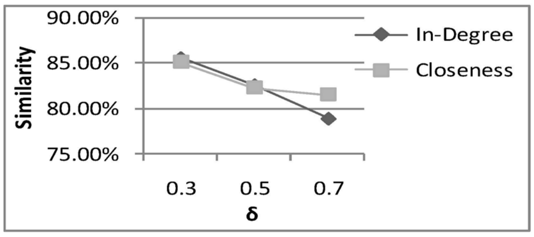
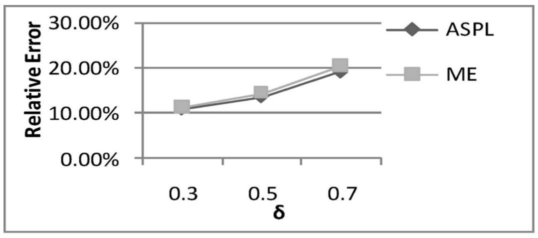


Fig. 9. Similarity in closeness centrality

Figs. 10(a) and Fig. 10(b) show the effects of different edge probabilities δ on the graph structure (Note that the x-axis coordinate denotes the edge probabilities; the y-axis coordinate denotes the structural change between the graphs before and after perturbation; $r = 2$, $s = 2|Dst(u)|$). As δ increases, the similarity in the degree centrality and closeness centrality gradually decreases. However, the relative error rates of the average shortest path length and maximum eigenvector gradually increase; this result indicates that the data availability is low. In Fig. 6 to Fig. 10, the smaller the values of s, r, and δ, the better the protection of the graph structure. Therefore, the thresholds for these values can be set appropriately under conditions that meet privacy protection demands to ensure the availability of the published graph data.



(a) Similarity in degree centrality and closeness centrality



(b) Error rates in ASPL length and ME

Fig. 10. Comparison chart of graph structures with different δ

6 Conclusions

Traditional privacy protection algorithms with poor effect and low efficiency unsatisfactorily process large data sets. Accordingly, parallel computing has become a trend in social network privacy protection. D-GSPerturb is a distributed social privacy protection algorithm based on graph structure perturbation, which is proposed to address link privacy issues in social networks. The present Pregel-like model based on a vertex-centric algorithm can search a large-scale social network for reachable vertexes, transfer reachable information, and randomly perturb edges through between-vertex messaging, vertex value updating, and multi-iteration in programming. The experimental results show that D-GSPerturb not only improves the processing speed of large-scale graph data but also ensures the privacy protection effect and the availability of data published. The parallel computing of large-scale graph data is in the initial stage, there is a lot of room for improvement. Therefore, we should further study the Pregel-like model to improve and merge processing steps of the algorithm, thereby speeding up the efficiency of the algorithm. On the other hand, the algorithm does not consider the influence of random perturbation on the accessibility of vertexes. To improve the data availability of the published graphs, future research work should consider reducing the loss of information on the accessibility of vertexes.

Acknowledgements

This work is partially supported by Natural Science Foundation of China (No.61562065). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

References

[1] XY. Liu, B. Wang, X.C. Yang, Survey on privacy preserving techniques for publishing social network data, Journal of Software 25(3)(2014) 576-590.

[2] C. Sun, P.S. Yu, X. Kong, Y. Fu, Privacy preserving social network publication against mutual friend attacks, Transactions on Data Privacy 7(2)(2015) 71-97.

[3] X. Ying, X. Wu, On link privacy in randomizing social networks, Knowledge and Information Systems 28(3)(2011) 645-663.

[4] A.A. Yarifard, S. Dehnvai, Link perturbation in social network analysis through neighborhood randomization, in: Proc. Fuzzy and Intelligent Systems, 2015.

[5] A.M. Fard, K. Wang, Neighborhood randomization for link privacy in social network analysis, World Wide Web Journal 18 (1)(2015) 9-32.

[6] M. Bhattacharya, P. Mani, Preserving privacy in social network graph with K-anonymize degree sequence generation, in: Proc. 2015 International Conference on Software, Knowledge, Information Management and Application, 2015.

[7] Y.Z. Wang, B.H. Zheng, Preserving privacy in social networks against connection fingerprint attacks, in: Proc. 2015 International Conference on Data Engineering, 2015.

[8] S. Zhang, Q. Liu, Y. Lin, Anonymizing popularity in online social networks with full utility, Future Generation Computer Systems 45(4)(2016) 33-44.

[9] J. Qian, X.Y. Li, C. Zhang, L. Chen, De-anonymizing social networks and inferring private attributes using knowledge graphs, in: Proc. IEEE INFOCOM, 2016.

[10] X.W. Ying, X.T. Wu, Randomizing social networks: a spectrum preserving approach, in: Proc. 2008 International Conference on Data Mining, 2008.

[11] L. Zhang, W. Zhang, Edge anonymity in social graphs, in: Proc. 2009 International Conference on Social Computing, 2009.

[12] A.M. Fard, K.Y. Wang, Limiting link disclosure in social network analysis through subgraph-wise perturbation, in: Proc. 2012 International Conference on Extending DB Technology, 2012.

[13] H. Zakerzadeh, C.C. Aggarwal, K. Barker, Big graph privacy, in: Proc. 2015 International Conference on Extending DB Technology, 2015.

[14] X.L. Zhang, Y.L. Guo, J.Y. Wang, W. Zhang, A personalized social network privacy protection method based on the Pregel-like system, ICIC Express Letters, Part B: Applications 7(4)(2016) 839-845.

[15] M.X. Yuan, L. Chen, P.S. Yu, H. Mei, Privacy preserving graph publication in a distributed environment, Springer Berlin Heidelberg 78(8)(2013) 75-87.

[16] P. Jurczyk, X. Li, Distributed anonymization: achieving privacy for both data subjects and data providers, in: Proc. the 2009 International Conference on Data and Applications Security, 2009.

[17] G. Yu, Y. Gu, Y.B. Bao, Large scale graph data processing on cloud computing environments, Chinese Journal of Computters 34(10)(2011) 753-1767.

[18] G. Malewicz, M.H. Austern, A.J.C. Bik, Pregel: a system for large-scale graph processing, in: Proc. 2011 International Conference on Management of Data, 2011.

[19] S. Salihoglu, J. Widom, Optimizing Graph Algorithms on Pregel-like Systems, in: Proc. the 2014 International Conference on Very Large Data Bases, 2014.