Chapter 9
Differentially Private Social Network Data Publishing

9.1 Introduction

With the significant growth of Online Social Networks (OSNs), the increasing volumes of data collected in those OSNs have become a rich source of insight into fundamental societal phenomena, such as epidemiology, information dissemination, marketing, etc. Much of this OSN data is in the form of graphs, which represent information such as the relationships between individuals. Releasing those graph data has enormous potential social benefits. However, the graph data infer sensitive information about a particular individual [13], which has raised concern among social network participants.

If the differential privacy mechanism is adopted in graph data, the research problem is then to design efficient algorithms to publish statistics about the graph while satisfying the definition of either node differential privacy or edge differential privacy. The former protects the node of the graph and the latter protects the edge in the graph.

Previous works have successfully achieved either node differential privacy and edge differential privacy when the number of queries is limited. For example, Hay et al. [95] implemented node differential privacy, and pointed out the difficulties to achieve the node differential privacy. Paper [229] proposed to publish graph dataset using a *dK-graph* model. Chen et al. [41] considered the correlation between nodes and proposed a correlated release method for sparse graphes. However, these works suffer from a serious problem: when the number of queries is increasing, a large volume of noise will be introduced. This problem can be tackled by iteration mechanism, which will be presented in this chapter.

This chapter focuses on both node and edge differential privacy. First, the chapter present several basic methods on node and edge differential privacy, then proposed an iteration method to publish a synthetic graph. Specifically, given a set of queries, an iteration method is used to generate a synthetic graph to answer these queries accurately. The iteration process can be considered as a training procedure, in

Table 9.1 Application settings

Application	Social network data publishing
Input data	Graph
Output data	Query answers and synthetic graph data
Publishing setting	Non-interactive
Challenges	Large query set
Solutions	Iterative based method
Selected mechanism	Iteration
Utility measurement	Noise measurement
Utility analysis	Laplace properties
Privacy analysis	Sequential composition
Experimental evaluation	Graph distance

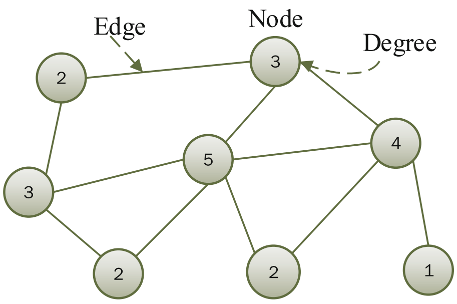

Fig. 9.1 Graph

which queries are training samples and the synthetic graph is an output learning model. A synthetic graph is finally generated by iterative update to answer the set of queries. As the training process consumes less privacy budget than the state-of-the-art methods, the total noise will be diminished. Table 9.1 shows the basic setting of the application.

9.2 Preliminaries

We denote a simple undirected graph as $G\langle V, E\rangle$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices (or nodes) representing individuals in the social network and $E \subseteq \{(u, v)|u, v \in V\}$ is a set of edges representing relationships between individuals. Figure 9.1 shows an example of a graph, in which the nodes are represented by circles and edges are represented by lines. The degree of a node refers to the number of its neighbourhoods. Formally, we define degree as follows:

Neighbourhood:

$$N(v) = \{u|(u, v) \in E, u \neq v\}; \qquad (9.1)$$

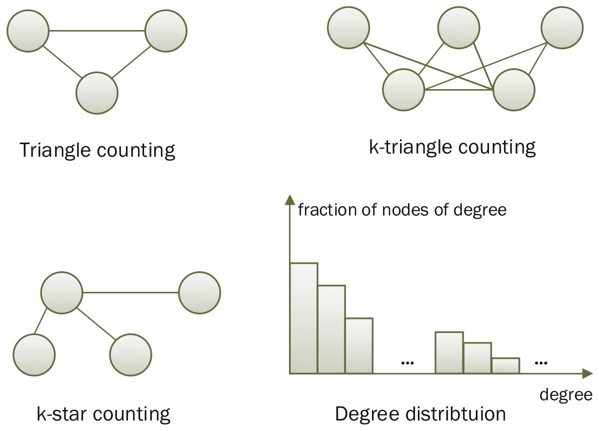

Fig. 9.2 Queries on graphs

Degree:

$$D(v) = |N(v)|. \tag{9.2}$$

Besides the count of nodes and edges in a graph, Fig. 9.2 shows several popular subgraph queries, including triangle counting, *k-triangle* counting, *k-stars* counting, and degree distribution.

9.3 Basic Differentially Private Social Network Data Publishing Methods

9.3.1 *Node Differential Privacy*

Node differential privacy ensures the privacy of a query over two neighbouring graphs where two neighbouring graphs differ one node and all edges connected to the node. Figure 9.3 shows the neighboring graphes in node differential privacy. Hay et al. [95] first proposed the notion of node differential privacy and pointed out the difficulties to achieve it. They showed that the result of query was highly inaccurate for analyzing graph due to the large noise.

Let us use Fig. 9.3 to illustrate the problem. When answering query f_1: how many nodes are there in the graph? the $\triangle f_1$ equals to 1, and the noise adding to f_1 is scaled to $Lap(1/\epsilon)$, which is quite low. However, when answering query f_2: how many edges are there in the graph? the sensitivity of f_2 equals to 5 as the maximum degree of all nodes is 5. The noise adding to the f_2 result is quite large comparing with the f_1.

The high sensitivity of node differential privacy derives from the degree of a node. When deleting a node, the maximum changing is determined by the largest degree of node in a graph. Theoretically, the sensitivity of a graph G will be

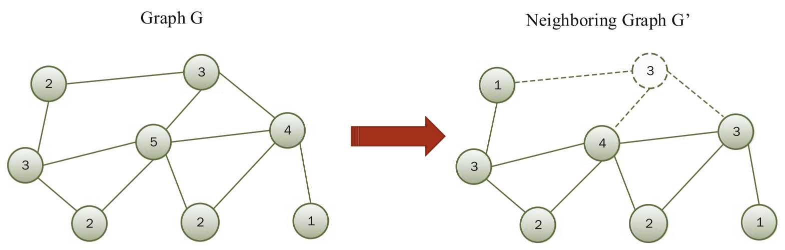

Fig. 9.3 Neighboring graph in node differential privacy

maximum to $n - 1$. How to decrease the sensitivity of f_2 is a challenge. One of the key ideas to achieve a better utility of node differential privacy is to transform the original graph to a new graph with lower sensitivity. Several methods have been proposed to achieve the goal. These methods roughly can be grouped into three categories: truncation, Lipschitz extension and iterative based mechanism.

9.3.1.1 Truncation and Smooth Sensitivity

Truncation method transforms the input graphes into graphes with maximum degree below a certain threshold θ [117]. The graph G is truncated to G_θ by discarding nodes with degree $> \theta$. Figure 9.4 shows a truncated graph G_3 for original graph G, in which one node with degree 5 is discarded to make sure all nodes are equal or below the degree of 3. By this way, the sensitivity of edge counting query will be decreased from sensitivity 5 to sensitivity of 3.

Algorithm 1 ϵ-Node-Private Algorithm for Publishing Degree Distributions

```
Require: $G$, $\epsilon$, $\theta$, degree distributions query $f$
Ensure: $\widehat{p}$.
    1. determine the randomized truncated parameter: select $\widehat{\theta} \in \{D + \frac{\log n}{\beta} + 1, \dots, 2D + \frac{\log n}{\beta}\}$;
    2. computer $G_{\widehat{\theta}}$ and smooth bound $S(G_\theta)$ with $\beta = \frac{\epsilon}{\sqrt{2}(\widehat{\theta}+1)}$;
    3. output $\widehat{f} = f(G_{\widehat{\theta}}) + Cauchy(\frac{2\sqrt{2}}{\epsilon}\widehat{\theta}S(G_\theta))^{\widehat{\theta}+1}$.
```

Kasiviswanathan et al. [117] showed that given a query f defined on the trunked graph G_θ, a smooth bound S is necessary for the number of nodes whose degrees may change due to the truncation. They applied Nissim et al.'s [172] β-smooth bound $S(G)$ for local sensitivity, which has been discussion in Definition 2.4 in Chap. 2. One can add noise proportional to smooth bounds on the local sensitivity using a variety of distributions. Kasiviswanathan et al. used the Cauchy distribution $Cauchy(\sqrt{2}S(G)/\epsilon)$. Algorithm 1 shows a typical algorithm to publish degree distributions for a G.

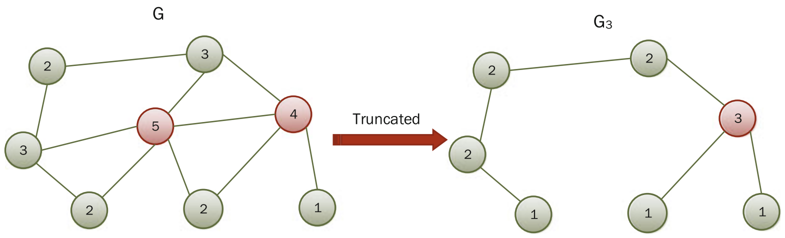

Fig. 9.4 Truncation method on graph

There are three major steps in the algorithm. The first step determines the truncated parameter θ. As we may not know the maximum degree in the graph, or the maximum degree may be very large, θ is normally approximated by $\widehat{\theta}$ Therefore, the algorithm randomized the cutoff to obtain a better bound. Given a target parameter θ, the algorithm picks a random parameter in a range of bounded constant multiple of θ. The second step creates the truncated graph by discarding the node with degree greater than θ. The final step add Cauchy distributed noise to each entry of the degree distribution.

Truncating G to G_θ is not easy, as deleting all nodes with degree greater than θ will ultimately delete more nodes and edges than we expected. Blocki et al. [27] solved this problem by selecting an arbitrary order among the edges, traversing the edges and removing each encountered edge that is connected to a node that has degree is greater than θ. Day [53] used an reverse way to create truncated graph. They first deleted all edges and add edges in a pre-defined order to achieve G_θ.

9.3.1.2 Lipschitz Extension

A more efficient method to achieve node differential privacy is based on Lipschitz extension. A function f' is a *Lipschitz extension* of f from G_θ to G if it satisfies with (1) f' agrees with f on G_θ, and (2) the global sensitivity of f' on G is equal to the local sensitivity of f on G_θ. As the sensitivity of f' is lower than that of f, Lipschitz extension of f is considered as an efficient way to decrease the sensitivity.

Figure 9.5 shows two conditions of Lipschitz extension. The large square is used to show all graphes with all possible degrees, and the eclipse inside the square is applied to show G_θ. For a query f, the global sensitivity is denoted by Δf, which is larger than the local sensitivity Δf_θ. If the algorithm can find an efficiently computable Lipschitz extension f' that is defined on all of G, then we can use the Laplace mechanism to release $\widehat{f}(G)$ with relatively small additive noise. Consequently, the target of the algorithm is to find a f' for f.

Kasiviswanathan et al. [117] proposed a flow-based method to implement such extensions for subgraph counts. Figure 9.6 shows the graph flow. Given a graph $G = (V, E)$ and a degree bound θ, the flow-based method first constructs a flow graph by copy two versions of nodes set $V_l = v_l | v \in V$ and $V_r = v_r | v \in V$, which are called left and right copies, respectively. The flow graph of G with a source s and a sink t is a directed graph on nodes $V_l \bigcup V_r \bigcup \{s, t\}$. Edges (s, v_l) and (v_r, t)

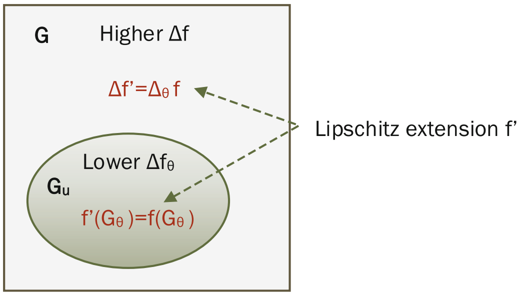

Fig. 9.5 Lipschitz extension

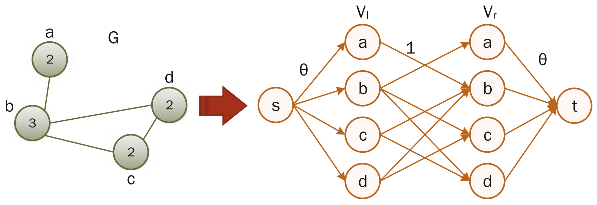

Fig. 9.6 Flow based method to obtain Lipschitz extension

are with capacity θ, while each edge (u, v) in E is added as (u, v') between v_l and v_r with capacity 1. Let $v_{fl}(G)$ denote the value of maximum flow from s to t, $v_{fl}(G)/2$ is a Lipschitz extension of an edge query f. The global node sensitivity $\triangle v_{fl}(G) \leq 2\theta$. Accordingly, Kasiviswanathan et al. [117] published the number of edges by Algorithm 2.

Algorithm 2 ϵ-Node-Private Algorithm for Publishing Edge Numbers

```
Require: $G$, $\epsilon$, $\theta$, number of edge query $f$
Ensure: $\widehat{f}$.
    1.$\widehat{f} = f(G) + Lap(2n/\epsilon)$ and threshold $\tau = \frac{nlogn}{\epsilon}$;
    if $\widehat{f} \geq 3\tau$ then
        2. output $\widehat{f}$.
    else
        3. compute $v_{fl}(G)$ with $\theta$;
    end if
    4.$\widehat{f} = v_{fl}(G)/2 + Lap(2\theta/\epsilon)$.
```

Blocki et al. [27] proceeded with a similar intuition. They showed that Lipschitz extensions exist for all real-valued functions, and give a specific projection from any graph to a particular degree-bounded graph, along with smooth upper bound on its local sensitivity.

Above works can only efficiently compute Lipschitz extensions for one-dimensional functions, in which the output is a single value. Raskhodnikova et al. [189] developed Lipschitz extensions for degree distribution queries with multidimensional vector outputs, via convex programming. Specifically, they designed Lipschitz extensions with small stretch for the sorted degree list and for the degree distribution of a graph.

9.3.1.3 Iterative Based Mechanism

Chen et al. [45] proposed an iterative based method to achieve node differential privacy. Given graph G and any real-valued function f, they defined a sequence of real-valued functions $0 = f_0(G) \leq f_1(G) \leq \cdots \leq f_m(G) = f(G)$ with the *recursive monotonicity* property that: $f_i(G') \leq f_i(G) \leq f_{i+1}(G')$ for all neighbors G and G' and $\forall i \in \{0, 1, \cdots, m\}$. They then defined quantity X to approximate the true answer of f, and the global sensitivity of X is Δ. $X_\Delta(G) = \min_{i \in [0,1]}(f_i(G) + (n - i)\Delta)$, where $X_\delta(G) \leq f(G)$ but close to $f(G)$ for larger Δ. For a carefully chosen Δ, they output the $X_\Delta(G)$ via Laplace mechanism in the global sensitivity framework, as an approximation of the real-valued function $f(G)$. This recursive approach can potentially return more accurate subgraph counting for any kinds of subgraphs with node differential privacy. However, constructing the sequence of functions $f_i(G)$ is usually NP-hard, and how to efficiently implement it remains an open problem.

9.3.2 Edge Differential Privacy

Edge differential privacy means adding or deleting a single edge between two nodes in the graph makes negligible difference to the result of the query. The first research over edge differential privacy was conducted by Nissim et al. [172], who showed how to evaluate the number of triangles in a social network with edge differential privacy. They showed how to efficiently calibrate noise for subgraph counts in terms of the smooth sensitivity. The results of this technique are investigated by Karwa et al. [113] to release counts on k-triangles and k-stars.

Rastogi et al. [190] studied the release of more general subgraph counts under a much weaker version of differential privacy, edge adversarial privacy, which considers a Bayesian adversary whose prior knowledge is drawn from a specified family of distributions. By assuming that the presence of an edge does not make the presence of other edges, they computed a high probability upper bound on the local sensitivity, and then added noise proportional to that bound. Rastogi et al.'s method can release more general graph statistics, but their privacy guarantee protects only against a specific class of adversaries, and the magnitude of noise grows exponentially with the number of edges in the subgraph.

Hay et al. [95] considered releasing a different statistics about graph, the degree distributions. They showed that the global sensitivity approach can still be useful when combined with post-processing of the released output to remove some added noise, and constructed an algorithm for releasing the degree distribution of a graph, with the edge differential privacy.

Based on the local sensitivity, Zhang [249] adopted exponential mechanism to sample the most suitable answer for subgraph query. The score function is designed carefully to make sure it reflects the distribution of query outputs. A different approach was proposed by Xiao [234], who inferred the networks structure via connection probabilities. They encoded the structure information of the social network by the connection probabilities between nodes instead of the presence or absence of the edges, which reduced the impact of a single edge.

Zhu et al. [260] proposed an iterative graph published method to achieve edge differential privacy. They proposed a graph update method that transfers the query publishing problem to an iteration learning process. The details are presented in the following section.

9.4 Graph Update Method

9.4.1 Overview of Graph Update

The proposed method is called *Graph Update* method as the key idea is to update a synthetic graph until all queries have been answered [260]. For a social network graph G and a set of queries $F = \{f_1, \ldots, f_m\}$, the publishing goal is to release a set of query results $\widehat{F}$ and a synthetic graph $\widehat{G}$ to the public. The general idea is to define an initial graph $\widehat{G}_0$ and update it to $\widehat{G}_{m-1}$ in m round according to m queries in F. Release answers $\widehat{F}$ and the synthetic graph $\widehat{G}$ are generated during the iteration. During the process, four different types of query answer involve in the iteration:

- *True answer a_t*: this is the real answer that a graph response to a query. True answer can not be published directly as it will arise privacy concern. The true answer is normally used as the baseline to measure the utility loss of a privacy-preserving algorithm. The symbol a_t is used to represent the true answer for a single query f, and $A_t = F(G) = \{a_{t1}, \ldots, a_{tm}\}$ is applied to represent an answer set for a query set F.
- *Noise answer a_n*: when we add Laplace noise to a true answer, the result will be the noise answer. Traditional Laplace method will release the noise answer directly. However, as we mentioned in Sect. 9.1, it will introduce large amount of noise to the release result. A single query answer is represented by $a_n = \widehat{f}(G) = f(G) + Lap(s/\epsilon)$ and an answer set is represented by $A_n = \widehat{F}(G) = \{a_{n1}, \ldots, a_{nm}\}$.
- *Synthetic answer a_s*: this is the answer generated by a synthetic graph $\widehat{G}$. A single query is presented by $a_s = f(\widehat{G})$ and $A_s = F(\widehat{G}) = \{a_{s1}, \ldots, a_{sm}\}$ is applied to represent an answer set.
- *Release answer a_r*: this is the answer finally released after the iteration. In Graph Update method, the release answer set will consist of noise answers and synthetic answers. The algorithm applied $a_r = \widehat{f}$ and $A_r = \widehat{F} = \{a_{r1}, \ldots, a_{rm}\}$ to represent the single answer of a query and the answer set, respectively.

These four different query answers will control the graph update process. The overview of the method is presented in Fig. 9.7. On the left side of the figure, the

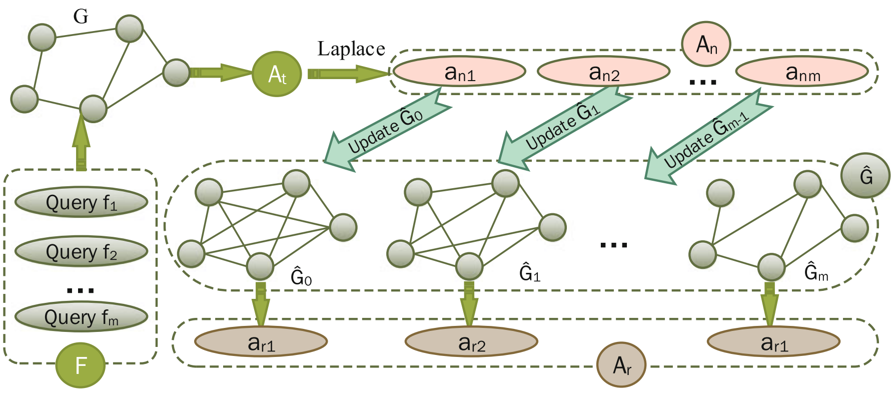

Fig. 9.7 Overview of *Graph Update* method

query set F performs on the G to get true answer set A_t. Laplace noise is then added to A_t to get a set of noise answer $A_s = \{a_{s1}, \dots a_{sm}\}$. Each noise answer a_{si} helps to update the initial $\widehat{G}_0$ and produce a release answer a_{ri}. The method eventually outputs $A_r = \{a_{r1}, \dots, a_{rm}\}$ and the $\widehat{G}_m$ as final results.

Comparing with the traditional Laplace method, the proposed *Graph Update* method adds less noise. As some queries are answered by the synthetic graph, these query answers will not consume any privacy budget. Moreover, the synthetic graph can be applied to predict new queries without any privacy budget. Eventually, the *Graph Update* method can outperform the tractional Laplace method.

Algorithm 3 Graph Update Method

```
Require: $G, F = \{f_1, \dots, f_m\}, \epsilon, \eta_0$
Ensure: $A_r = \{a_{r1}, \dots, a_{rm}\}$.
    1. $\epsilon' = \epsilon/m$
    2. initial graph $\widehat{G}_0$;
    for each query $f_i \in F$ do
        3. Compute true answer $a_{ti}$;
        4. Add Laplace Noise to true answer $a_{ni} = \widehat{f}_i = f_i(G) + Lap(S/\epsilon')$;
        5. Compute synthetic answer $a_{si} = f_i(\widehat{G})$;
        6. $\eta_i = a_{ni} - a_{si}$;
        if $|ta_i| > \eta_0$ then
            7. $a_{ri} = a_{ni}$;
            8. update the $\widehat{G}_{i-1}$ to $\widehat{G}_i$;
        else
            9. $a_{ri} = a_{si}$;
            10. $\widehat{G}_m = \widehat{G}_{m-1}$
        end if
    end for
    11. Make all degrees in $G$ round numbers.
    12. Output $A_r = \{a_{r1}, \dots, a_{rm}\}$, and $\widehat{G}$;
```

9.4.2 Graph Update Method

At a high level, the *Graph Update* method works in three steps:

- *initial the synthetic graph*: As the method only preserves the edge privacy, it assumes that the number and the labels of nodes are fixed. The synthetic graph is initialed as a fully connected graph with fixed nodes.
- *update the synthetic graph*: the initial graph will be updated according to result of each query in F, until all queries in F have been used.
- *release query answers and synthetic graph*: Two types of answers, noise answers and synthetic answers that have potential to be released. Synthetic graph is also released to the public.

Algorithm 3 is a detailed description of the *Graph Update* method. In step 1, the privacy budget ϵ is divided by m and will be arranged to each query in the set. Step 2 initializes the graph to $\widehat{G}_0$ as a full connected one. Then for each query f_i in the query set F, the algorithm computes the true answer $f_i(G)$ at Step 3. After that, the noise answer and the synthetic answer of f_i are computed at Step 4 and 5, respectively. Step 6 measures the distance between the true answer and the synthetic answer. If the distance is larger than a threshold η_0, the Step 7 will release the noisy answer. Otherwise, the synthetic graph will be updated by an *Updated Function* in Step 8 and Step 9 will release the synthetic answer. This means the synthetic graph is applicable for answering question, so in Step 10, the algorithm puts the current synthetic graph to the next round. This process is iterated until all queries in F are preceded. Finally, As the number of edges should be a integer, the algorithm round the number of degrees in Step 11. the algorithm generates A_r and $\widehat{G}$ as the output in Step 12.

Algorithm 4 Update Function

```
Require: $\widehat{G}, f, \eta, \theta, (0 < \theta < 1)$
Ensure: $\widehat{G}'$.
    1. Identify related nodes $V_f$ that $f$ involved;
    if $\eta > 0$ then
        2. $D(V_f) = (1 + \theta) * D(V_f)$;
    else
        3. $D(V_f) = \theta * D(V_f)$;
    end if
    4. $\widehat{G}' = G \cup D(V_f)$.
    5. Output $\widehat{G}'$.
```

The parameter η_0 is a threshold controlling the distance between A_n and A_s. A larger η_0 means less update of the graph and most of the answer in A_r are synthetic answers. It leads to less privacy budget consuming, however, when the synthetic graph is far away from the original graph, the performance may not optimal. A smaller η_0 means the algorithm has more updates of the graph and most of the answer in A_r are noise answers. More privacy budgets will be consumes in this configuration. Consequently, the choice of η_0 will have impact on different scenarios.

9.4.3 Update Function

Step 8 in Algorithm 3 involves with an *Update Function*, which updates the synthetic graph $\widehat{G}$ to graph $\widehat{G}'$ according to query answers. Specifically, *Update Function* is controlled by the distance η between the a_n and a_s of f. If a_n is smaller than a_s, it means that the synthetic graph has more edges than the original graph in the related nodes. *Update Function* has to delete some edges between the related nodes. Otherwise, *Update Function* will add some edges in the synthetic graph.

These related nodes is defined in the follow Definition 9.1:

Definition 9.1 (Related Node) For a query f and a graph G, *related nodes* V_f are all nodes that response to the query f, $D(V_f)$ is used to denote degrees of those nodes.

The number of edges for a node should be a integer. However, to adjust degree of those related nodes, we arrange weight θ ($0 \leq \theta \leq 1$) for each edge. After the updating, these weights will be rounded to represent node edges.

Algorithm 4 illustrates the detail of *Update Function*. In the first step, the function identifies related nodes. If $\eta > 0$, which means the synthetic graph has less edges than the original one, the function will enhance the θ in Step 2. If $\eta \leq 0$, which means the synthetic graph has too many edges, the function will diminish those edges by θ in Step 3. Step 4 merges the edges to the original graph. Step 5 outputs the $\widehat{G}'$.

9.4.4 Privacy and Utility Analysis

9.4.4.1 Privacy Analysis

To analyze the privacy level of the proposed method, the sequential composition is applied. For the traditional Laplace method, when answering F with m queries, ϵ will be divided into m pieces and arranged to each query $f_i \in F$. Specifically, we have $\epsilon' = \epsilon/m$ and for each query, the noise answer will be $a_{ni} = f_i + Lap(s/\epsilon')$. According to the *sequential composition*, the Laplace method preserve $(\epsilon' * m)$-differential privacy, which is equal to ϵ-differential privacy.

In *Graph Update* method, the release answer set A_r are the combination of noise answers A_n and synthetic answers A_s. Only A_n consume privacy budget, while A_s do not. In Algorithm 4, even Step 4 adds Laplace noise to the true answer, the noise result does not release directly. Only when the algorithm processed to Step 7, in which a_n is released, the algorithm consumes the privacy budget. Suppose there are $j(0 \leq j \leq m)$ queries in F is released by synthetic answers, the algorithm preserves $((m-j) * \epsilon')$-differential privacy. As $(m-j) * \epsilon' \leq m * \epsilon'$, the *Graph Update* method preserve more strict privacy than tractional Laplace method.

9.4.4.2 Utility Analysis

Error measurement is applied to evaluate the utility. The error is defined by *Mean Absolute Error* (MAE). MAE_r of release answer A_r is defined as Eq. (9.3)

$$\begin{aligned} MAE_r &= \frac{1}{m}|\widehat{F}_i(G) - F_i(G)| \\ &= \frac{1}{m}\sum_{f_i \in F} |\widehat{f}_i(G) - f_i(G)| \\ &= \frac{1}{m}\sum_{a_i \in A_r} |a_{ri} - a_{ti}| \\ &= \frac{1}{m}|A_r - A_t|. \end{aligned} \tag{9.3}$$

Similarly, MAE_n of noise answers and MAE_s of synthetic answers are defined as Eqs. (9.4) and (9.5), respectively.

$$MAE_n = \frac{1}{m}|A_n - A_t|; \tag{9.4}$$

$$MAE_s = \frac{1}{m}|A_s - A_t|. \tag{9.5}$$

It is obvious that for true answers A_t, the *MAE* is zero. MAE_n represents the performance of traditional Laplace method. A lower *MAE* implies a better performance.

The target of *Graph Update* method is to achieve a lower MAE_r in a fixed privacy budget. A simulated figure, Fig. 9.8, is applied to illustrate the relationship between *MAE* values and the size of the query set m.

In Fig. 9.8, x axis is the size of the query set and y axis is the value of *MAE*. For noise answer A_n, MAE_n is arising with the increasing of m. A smooth line is applied to represent the MAE_n in this simulated figure. In real case, the line is fluctuated as the noise is derived from Laplace distribution. The MAE_s is decreasing at the beginning with the increasing of m. When it reaches to its lowest point, the MAE_s begins to rise with the enhance of m. This is because with the update of the graph, the synthetic graph is getting more and more accurate, MAE_s is keeping decreasing. However, as the iteration procedure is controlled by the noise answer, it is impossible for synthetic graph to equal to the original graph, no matter how large m is. On the contrary, with the increasing of m, more noise will be introduced to iteration and the synthetic graph will be far away from the original graph.

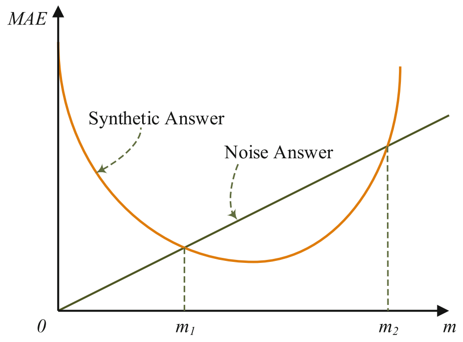

Fig. 9.8 Utility of the query set on a graph

As A_r is the combination of A_n and A_s, MAE_r of release answers can be reflected by synthetic answer MAE_s and noise answer MAE_n. Figure 9.8 shows that MAE_s will below MAE_n when the query size reaches to m_1. After reaching to a lowest point, it begins to increase. After reaching to m_2, the MAE_s is higher than MAE_n. Consequently, when m in the scale of $[0, m_1) \cup (m_2, m]$, the MAE_r is dominated by noise answer MAE_n. When m in the scale of $[m_1, m_2]$, the MAE_r is dominated by synthetic answer MAE_s. By this way, in the scale of $[0, m]$, the MAE_r of release answers is smaller than MAE_n, which means that the performance of the proposed *Graph Update* method is better than the traditional Laplace method.

9.4.5 *Experimental Evaluation*

This section evaluates the performance of the proposed Graph Update method comparing with Laplace mechanism.

9.4.5.1 Datasets and Configuration

The experiment involve with four datasets listed in Table 9.2. These datasets are collected from Stanford Network Analysis Platform (SNAP) [136].

The experiment considers the degree query on nodes, which is similar to the count query on relation dataset. To preserve the edge privacy, the degree query has the sensitivity of 1, which means deleting an edge will have maximum impact of 1 on the query result. The performance of results is measured by *Mean Absolute Error* (MAE) (9.3).

Table 9.2 Graph datasets

Type	Name	Nodes	Edges
Social networks	Ego-Facebook	4039	88,234
Social networks	Wiki-Vote	7115	103,689
Internet peer-to-peer networks	p2p-Gnutella08	6301	20,777
Collaboration networks	ca-GrQc	5242	14,496

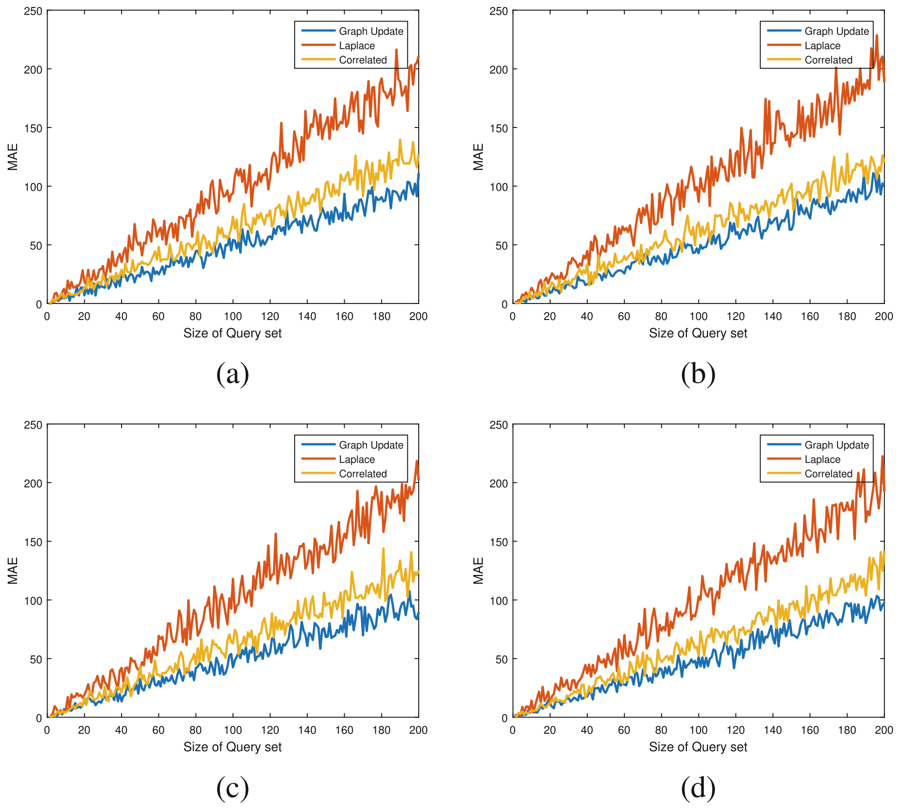

Fig. 9.9 Performance of different methods. (**a**) `ego-Facebook`. (**b**) `Wiki-Vote`. (**c**) `p2p-Gnutella08`. (**d**) `ca-GrQc`

9.4.5.2 Performance Evaluation on Diverse Size of Query Sets

The performance of the *Update Graph* is examined through comparison with the Laplace method [62] and Correlated method [41]. The size of query sets is from 1 to 200, in which each query is independent to each other. Parameters η_0 and θ as optimal ones for each dataset and the ϵ is fixed at 1 for all methods.

According to Fig. 9.9, It is observed that with the increasing of the size of the query sets, *MAE*s of all methods are increasing approximately in linear. This is because the queries are independent to each other and the privacy budget is arranged equally to each query. With the linear increasing of the query number, the noise added to each query answer is enhanced linearly.

Second, Fig. 9.9 shows that *Update Graph* has lower *MAE* comparing with other two methods, especially when the size of the query set is large. As shown in Fig. 9.9a, when the size of query set is 200, the *MAE* of *Graph Update* is 99.8500 while the Laplace method has *MAE* of 210.0020, and the *Correlated* method has *MAE* of 135.2078 which is 52.45 and 26.15% higher than the proposed *Update Graph*. This trend can be observed in Fig. 9.9b–d. The proposed *Graph Update* mechanism has better performance because part of query answers does not consume any privacy budget, while noise is only added in the updated procedure. Other methods, including Laplace method consume the privacy budget when answering every query. The experimental results show the effectiveness of *Graph Update* in answering a large set of queries.

Third, it is worth to mention that when the size of the query set is limited, the proposed *Graph Update* may not necessary outperform the *Correlated* method. Figure 9.9a shows that when the size is less than 20, *MAE*s of *Graph Update* and the *Correlated* method are mixed together. This is because when the query set is limited, the synthetic graph can not be fully updated and may differ from the original graph largely. Therefore, the performance may not necessary outperform other methods significantly. This result shows that *Graph Update* is more suitable in scenarios that need to answer a large amount of queries.

9.5 Summary

Nowadays, the privacy problem have aroused peoples attention. Especially the online social network data, which contains a massive personal information. How to release social network data is a hot topic that attracts lots of attention. This chapter proposes several method to meet with the graph publishing problem. And to overcome the problem of providing accurate results even when releasing large numbers of queries, this chapter then presents an iterative method that transfers the query release problem into an iteration based update process, so as to providing a practical solution for publishing a sequence of queries with high accuracy. In the future, much more complied queries should be investigated, such as cut queries and triangle queries, which can allow researchers to get more information of the dataset while still can guarantee users' privacy.