Privacy Preservation of User History Graph

Shinsaku Kiyomoto[1], Kazuhide Fukushima[1], and Yutaka Miyake[1]

KDDI R & D Laboratories Inc.
2-1-15 Ohara, Fujimino, Saitama, 356-8502, Japan
kiyomoto@kddilabs.jp

Abstract. In this paper, we propose new ideas to protect user privacy while allowing the use of a user history graph. We define new privacy notions for user history graphs and consider algorithms to generate a privacy-preserving digraph from the original graph.

1 Introduction

Graph-structured data are useful for the analysis of relationships and trends of service use [6, 4], and has been considered an analysis tool for a commercial services [10]. Graph-structured data sets are collections of user history data, which are anonymous, but contain sensitive information with implications for user privacy. A privacy concern for leakages of user history data has been reported [11]. Privacy-preserving analysis of such graphs remains a continuing concern as interest in the areas of privacy and data mining increases.

Current research focuses mainly on social network graphs without directed edges. In a social network graph, the anonymity of each node is important for privacy protection. How to obfuscate the relationships between users and nodes is the main issue for existing privacy protection mechanisms for graphs. A method [13] proposed by Yuan *et al.* constructs a k-degree anonymous graph by adding nodes and edges. In the k-degree anonymous graph, the probability that an attacker cannot identify a node of the target user is at most $\frac{1}{k}$. Several studies have examined [5, 7, 3] methods that use secret graph data and respond to queries with approximate results. Blaustein *et al.* formalized the problem of creating a protected account G' of a graph G and provided an algorithm [2] to create a maximally useful protected account of a sensitive social network graph. In the article [15], k-automorphism is proposed: k-automorphism requires that each node has at least $k-1$ other nodes with the same structure in a published graph. Some other techniques for protecting privacy in social networks have been proposed [12, 14]. Privacy preserving data mining methods for link analysis of directed graphs was presented in [8, 1].

In this paper, we propose new privacy notions for user history graphs and consider algorithms to generate a privacy-preserving digraph from the original graph. We assume that the user history graph is published and used for several analyses using non-interactive schemes. The notions are similar in approach to k-anonymity [9], and they are configured by two parameters k and v.

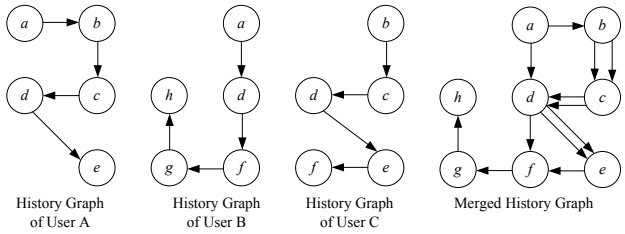

Fig. 1. Example of Merged History Graph

The rest of the paper is organized as follows; user history graphs are explained in section 2. Section 3 provides privacy notions for user history graphs and section 4 presents algorithms for privacy-preserving graph data. Evaluation results of the algorithms are shown in section 5. We conclude this paper in section 6.

2 User History Graph

A history graph as shown in figure 1 is an efficient method to visualize the history of a user. We define a label for each edge to merge the history graphs of all users. A user history graph showing that users perform action a and then b is transformed into a labeled digraph $(a \rightarrow b)$: the label $L(a \rightarrow b)$ denotes the number of users who perform the action from a to b. For simplicity in the definition, we assume that the user does not repeat his previous actions, even though our discussion in this paper can be applied to that case. First, a history for each user is transformed into a labeled digraph such that the label is $L(a \rightarrow b) = 1$. Then, digraphs of all users are merged as follows;

- all nodes that denote the same action are merged into one node.
- The labels are updated to $L(a \rightarrow b) = n$, where digraphs of n users have the same label $L(a \rightarrow b) = 1$.

Note that the digraph $(a \rightarrow b)$ is different from the digraph $(b \rightarrow a)$, so the label is defined for each digraph.

Let *start* be a group of nodes that the number of outgoing edges for a node is larger than the number of incoming edges for the node and *end* be a group of nodes the number of incoming edges for a node is larger than the number of outgoing edges for the node. The symbol $start \rightarrow x$ denotes the routes from the nodes *start* to x, and the symbol $x \rightarrow end$ denotes the routes from x to the *end* nodes. The symbols $\mathcal{N}_{start \rightarrow t}$ and $\mathcal{N}_{t \rightarrow end}$ respectively denote all nodes on all possible routes from $start \rightarrow t$ and all nodes on all possible routes from $t \rightarrow end$, where the symbol $\mathcal{N}_x$ denotes a group of nodes that satisfy the condition x. If $start = \emptyset$, $\mathcal{N}_{start \rightarrow t}$ consists of all nodes that can be reached from node t to trace the digraph backwards. For $end = \emptyset$, $\mathcal{N}_{t \rightarrow end}$ consists of all nodes that can be reached from node t to trace the digraph forwards. The symbol $\emptyset$ denotes an empty set. For the merged history graph in figure 1, $\mathcal{N}_{start \rightarrow d}$ is the set of nodes $\{a, b\}$, and $\mathcal{N}_{d \rightarrow end}$ is the set of nodes $\{e, f, h\}$.

3 Privacy Notion

3.1 Adversary Model

Privacy leakage from a merged history graph is the disclosure of the actions of a particular person from the graph. Attacks against user history graphs are intended to obtain private information on a particular user from the graph. We assume that the merge process is executed on a trusted domain and only the merged history graph is published. We summarize an adversary model as follows:

Adversary against a Merged History Graph. It is assumed that an adversary knows a victim A executed an action t. The objective of the adversary is to obtain actions that A executed before or after the action t. Thus, the adversary searches the merged history graph that includes actions of other people and finds actions of A using the knowledge that action t was executed.

3.2 Notions for Untraceability of Graph

We consider two levels of privacy notions: partial k-untraceability, and complete k-untraceability. Partial k-traceability accepts leakage of some partial actions of a user, but prevents all actions of the user from being revealed. The definition of complete k-untraceability involves meeting the reqirement that no action of the user is leaked. The symbol $Act^A_{N_{x\to y}}$ for user A denotes the sequence of all actions of user A from action x to action y. For example, the sequence of actions from the first action to action x and the sequence of actions from action x to the final action are denoted $Act^A_{\mathcal{N}_{start\to x}}$ and $Act^A_{\mathcal{N}_{x\to end}}$, respectively.

Generally, there are many trivial actions, performed by many users. It is not important for privacy purposes where we keep the information about such actions. Thus, we relax the above definitions to produce an anonymized graph that includes a lot of information for analyses of user history. Let v be the threshold value for the number of performing users that establishes that an action is trivial, that is, we judge the actions $x \to y$ to be trivial if the label $L(x \to y) \geq v$. Both definitions are denoted as follows:

Definition 1. *Partial (k, v)-untraceability.* We assume that an adversary knows an action t of a user A, and we consider all possible adversaries defined for any t in the merged graph. If at least k sequences of actions that are potentially associated with the user A and $k-1$ other users exist as candidates respectively for all actions $Act^A_{\mathcal{N}_{start\to t}}$ and $Act^A_{\mathcal{N}_{t\to end}}$ except trivial actions $x \to y$ that have a label $L(x \to y) \geq v$, it is said that the digraph satisfies partial (k, v)-untraceability for A. If the digraph satisfies the above condition for all users, the digraph is said to satisfy partial (k, v)-untraceability.

Definition 2. *Complete (k, v)-untraceability.* We assume that an adversary knows an action t of a user A, and we consider all possible adversaries defined for any t in the merged graph. If at least k actions that are potentially associated with the user A and $k-1$ other users exist as candidates for each action in $Act^A_{\mathcal{N}_{start\to t}}$ and $Act^A_{\mathcal{N}_{t\to end}}$ except trivial actions $x \to y$ that have a label

$L(x \to y) \geq v$, it is said that the digraph satisfies complete (k, v)-untraceability for A. If the digraph satisfies the above condition for all users, the digraph is said to satisfy complete (k, v)-untraceability.

In a complete (k, v)-untraceable graph, each action t except trivial actions has k outgoing edges and incoming edges; thus, an action of the user A which connects to the action t cannot be identified from k candidates. Thus, the graph satisfies untraceability for the adversary who knows the action t of the user. It is trivial that a complete (k, v)-untraceable graph satisfies partial (k, v)-untraceability; all actions except trivial actions are connected to k potential actions in a complete (k, v)-untraceable graph. A graph that satisfies partial (k, v)-untraceability generally produces much information than a complete (k, v)-untraceable graph, where the partial (k, v)-untraceable graph and the complete (k, v)-untraceable graph are generated from a user history graph. However, the (k, v)-untraceable graph may reveal partial actions of users due to the relaxed definition of the privacy notion; an attack is successful where an adversary obtains all actions of a user. To trace all actions of the user, the adversary has to select a sequence of actions from k sequences of actions; thus, all actions of the user is untreaceable, even though some actions are treaceable for the adversary. The parameter k means that an action (or a sequence of actions) is potentially associated with a user and $k - 1$ other users in the untraceable graph, and the parameter v means that v users do the same action in the graph. Generally, we should select the parameter $v = k$ with regards to a privacy requirement for a merged graph. Actions of a user are hidden in actions of a group that consists of k members including the user. A privacy notion for the graph should be selected from the above two notions according to a use case of the graph and its privacy requirements. Tradeoffs between privacy and the amount of information that is provided by the untraceable graph will be evaluated in section 6.

4 Algorithm

4.1 Algorithm generating partial (k, v)-untraceable history graph

Details of the algorithm are denoted as **Algorithm 1**, where oe_t and ie_t are respectively defined as the number of outgoing edges and incoming edges of a node t. The algorithm to generate a partial (k, v)-untraceable history graph is as follows.

1. This step consists of a part of the detailed algorithm, from line 1 to line 3. For input of a user history graph **G**, the algorithm adds a virtual incoming edge $(s_r \to r)$ to each node $r \in start$ until the number of the incoming edges is the same as the number of outgoing edges. Then, the the algorithm adds a virtual outgoing edge $(q \to u_q)$ to each node $q \in end$ until the number of the outgoing edges is the same as the number of incoming edges. A label of a virtual incoming edge $L(s_x \to x)$ denotes the number of users who first do the action, and a label of a virtual outgoing edge $L(y \to u_y)$ denotes the number of users who do the action at the end.

2. This step consists of a part of the detailed algorithm, from line 4 to line 12. The algorithm searches for a node t that has fewer outgoing edges than k and for which all lower nodes $\mathcal{N}_{t \to end \setminus t}$ have fewer outgoing edges than k. Then, the algorithm removes all outgoing edges $(t \to *)$ that satisfy $L(t \to *) < v$. Next, the algorithm searches for a node t' that receives incoming edges numbering less than k and all upper nodes $\mathcal{N}_{start \to t' \setminus t'}$ that receive fewer incoming edges than k. Then the algorithm removes all incoming edges $(* \to t')$ that satisfy $L(* \to t') < v$. The algorithm repeats this step until no node that meets the conditions has found.
3. This step is the same as line 13, line 14 and line 15 in the detailed algorithm. The algorithm removes virtual incoming and outgoing edges, removes nodes that has no edges, and outputs the modified graph.

Algorithm 1 Generation of a Partial (k, v)-Untraceable History Graph

Input: User History Graph G, parameters k and v
Output: Anonymized Graph $G^{\alpha}(G, k, v)$
1: $G^{\alpha}(G, k, v) \leftarrow G$
2: Add virtual incoming edges to *start* nodes
3: Add virtual outgoing edges to *end* nodes.
4: $T \leftarrow$ all nodes t, where $oe_{\mathcal{N}_{t \to end}} < k$ and its all edges are not $L(t_i \to *) \geq v$
5: $T' \leftarrow$ all nodes t', where $ie_{\mathcal{N}_{start \to t'}} < k$ and its all edges are not $L(* \to t'_j) \geq v$
6: **while** $T \neq \emptyset$ or $T' \neq \emptyset$ **do**
7: Choose t_i from T
8: Remove all outgoing edges of t_i, where $L(t_i \to *) < v$ from $G^{\alpha}(G, k, v)$
9: Choose t'_j from T'
10: Remove all incoming edges of t'_j, where $L(* \to t'_j) < v$ from $G^{\alpha}(G, k, v)$
11: Update T and T'
12: **end while**
13: Remove virtual edges
14: Remove all node t'' where $oe_{t''} = 0$ and $ie_{t''} = 0$ from $G^{\alpha}(G, k, v)$
15: **return** $G^{\alpha}(G, k, v)$

Example. We show an example of the execution for the merged history graph in figure 2, where we set $k = v = 2$. The algorithm checks whether each node satisfies the condition for outgoing edges which is described in the step 2, and find that the nodes h and g satisfy the condition. Because, the nodes $\{a, d, e, f\}$ have two outgoing edges, and the nodes b and c have edges $L(b \to c) = 2$, $L(c \to d) = 2$, respectively. Then, the outgoing edges $(g \to h)$ and $(h \to u_h)$ are removed. The algorithm has no node that satisfy the condition for incoming edges which is described in the step 2. Note that the incoming edge $(s_a \to a)$ satisfies $L(s_a \to a) \geq 2$. The algorithm removes virtual edges from the graph, and then the algorithm removes the node h that has no connection with other nodes. Thus, the output of the algorithms for the example graph is the graph including seven nodes as described in figure 2.

Any adversary who knows an action of a user cannot find sequences of all actions of the user from the output of **Algorithm 1**. For example, if an adversary knows that a user do the action g, there exist at least two possible sequences of the actions, $(a \to d \to f \to g)$ and $(a \to d \to e \to f \to g)$.

Computational Cost. We estimate the computational cost of the algorithm 1. Let E and N be the total number of edges in the graph and the total number of nodes in the graph. The cost of line 1 is $O(1)$. The computational cost from line 2 to line 5 is $O(N + E)$. Lines 6 to 12 require $O(N^2 + NE)$ computation and lines 13 and 14 require $O(N)$ computation, respectively; thus the total cost from line 1 to line 14 can be estimated as $O(N^2 + NE)$. Where the graph is a dense graph ($E = O(N^2)$), the computational cost is $O(N^3)$.

4.2 Algorithm generating complete (k, v)-untraceable history graph

Details of the algorithm are denoted as **Algorithm 2**. The algorithm to generate a complete (k, v)-untraceable history graph is as follows;

1. The algorithm first executes **Algorithm 1** except line 13 and line 15.
2. This step consists of a part of the detailed algorithm, from line 3 to line 11. The algorithm searches for a node t that has fewer outgoing edges than k and removes all outgoing edges $(t \rightarrow *)$ that satisfy $L(t \rightarrow *) < v$, until no node is found. Then, the algorithm searches for a node t' that receives fewer incoming edges than k and removes all edges $(* \rightarrow t')$ that satisfy $L(* \rightarrow t') < v$. The algorithm repeats this step until no node that meets the conditions has found.
3. This step consists of line 12, line 13, and line 14 in the detailed program. The algorithm removes virtual edges, removes nodes to which no edge is connected, and outputs the modified graph.

Example. An example of the output by **Algorithm 2** is shown in figure 2. We set $k = v = 2$. After the step 1, the algorithm checks whether each node satisfies the conditions described in the step 2, and find that the node g satisfies the condition for incoming edges, because the node g has a incoming edge of $L(f \rightarrow g) = 1$. Then, the incoming edge $(f \rightarrow g)$ is removed. Next, the algorithm finds the node f satisfies the condition for outgoing edges, and removes the virtual edge $(f \rightarrow u_f)$. In the final step, the virtual edges are removed from the graph, and then the node f that has no edge is removed.

Any adversary who knows an action of a user cannot find any action of the user from the output of **Algorithm 2**. For example, if an adversary knows that a user do the action e, the adversary knows a trivial action d, and possible actions $\{a, b, c\}$ that are potentially associated with other users.

Computational Cost. We can calculate the total computational cost of **Algorithm 2** in the same manner. The total computational cost is estimated as $O(N^3)$.

5 Evaluation

We implemented a prototype module for **Algorithm 1** and **Algorithm 2**, and evaluated the transaction time for generating an anonymized history graph on a

Algorithm 2 Generation of a Complete (k, v)-Untraceable History Graph

Input: User History Graph G, parameters k and v
Output: Anonymized Graph $G^{\alpha}(G, k, v)$
1: $G^{\alpha}(G, k, v) \leftarrow G$
2: Execute **Algorithm 1** except line 13 and 15
3: $T \leftarrow$ all nodes t, where $oe_t < k$ and its all edges are not $L(t_i \rightarrow *) \geq v$
4: $T' \leftarrow$ all nodes t', where $ie_{t'} < k$ and its all edges are not $L(* \rightarrow t'_j) \geq v$
5: **while** $T \neq \emptyset$, or $T' \neq \emptyset$ **do**
6: Choose t_i from T
7: Remove all outgoing edges of t_i, where $L(t_i \rightarrow *) < v$ from $G^{\alpha}(G, k, v)$
8: Choose t'_i from T'
9: Remove all incoming edges of t'_j, where $L(* \rightarrow t'_j) < v$ from $G^{\alpha}(G, k, v)$
10: Update T and T'
11: **end while**
12: Remove vertual edges
13: Remove all node t'' where $oe_{t''} = 0$ and $ie_{t''} = 0$ from $G^{\alpha}(G, k, v)$
14: **return** $G^{\alpha}(G, k, v)$

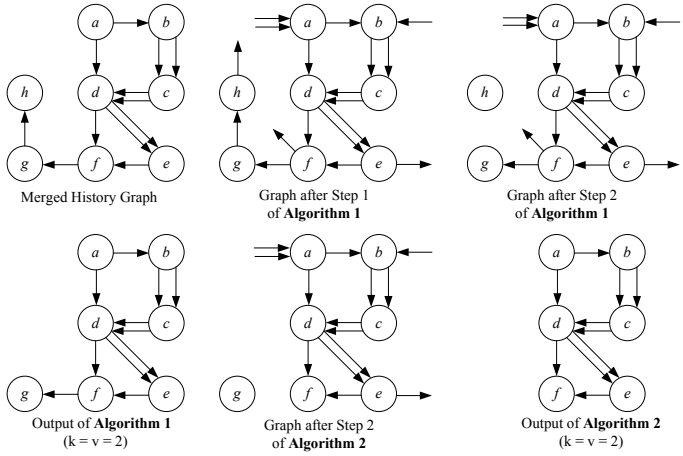

Fig. 2. Outputs of Algorithms

PC (Core i7 2.93GHz, 4.00GB Memory). We randomly generated merged user history graphs in two steps. In the first step, we generated a merged user history graph of n nodes under the condition that the probability of node connection $L(a, b) = 1$ was p; we first obtained an n-node merged graph where all labels are $L(a, b) = 1$. Then, we randomly assigned $n^{\frac{1}{2}}$ actions of $10 \times n^{\frac{1}{2}}$ users to digraphs and updated the labels in the graph. The data format described a user history graph $a \overset{L(a,b)}{\longrightarrow} b$ as a tuple of three data $\{a, b, L(a, b)\}$.

Performance. The evaluation results of the average transaction time for 20 graphs are summarized in table 1 and table 2. The tables show the transaction time of **Algorithm 1** and **Algorithm 2**, respectively. We examined two different probabilities of node connections with five patterns of k. We set $v = k$ for all experiments. The transaction time depends on the number of processed nodes that are input into T and T', and the number of processed nodes is configured

Table 1. Transaction Time of Partial (k, v)-Untraceable History Graph Generation

(second)	100 nodes	150 nodes	200 nodes	300 nodes	400 nodes	500 nodes	700 nodes	1000 nodes	1500 nodes	2000 nodes
p=0.01, k=2	0.01	0.01	0.03	0.07	0.08	0.14	0.21	0.27	0.67	1.04
p=0.01, k=5	0.01	0.04	0.21	0.52	1.03	1.67	2.98	4.32	10.80	19.22
p=0.01, k=10	0.01	0.07	0.36	2.33	6.51	17.85	24.45	30.99	62.06	138.72
p=0.01, k=20	0.01	0.07	0.38	2.34	6.82	14.27	42.79	128.22	425.58	834.59
p=0.01, k=30	0.01	0.07	0.39	2.39	6.37	14.66	41.41	127.87	424.65	931.05
p=0.1, k=2	0.01	0.01	0.01	0.02	0.03	0.06	0.11	0.23	0.55	0.97
p=0.1, k=5	0.01	0.02	0.03	0.06	0.13	0.21	0.52	1.08	3.31	7.53
p=0.1, k=10	0.06	0.12	0.19	0.52	1.16	2.12	5.42	14.27	46.89	109.14
p=0.1, k=20	0.18	0.55	1.41	4.35	9.41	18.11	43.58	113.71	305.10	622.19
p=0.1, k=30	0.18	0.55	1.32	4.32	10.32	19.46	51.31	146.82	472.82	1039.43

Table 2. Transaction Time of Complete (k, v)-Untraceable History Graph Generation

(second)	100 nodes	150 nodes	200 nodes	300 nodes	400 nodes	500 nodes	700 nodes	1000 nodes	1500 nodes	2000 nodes
p=0.01, k=2	0.01	0.02	0.04	0.09	0.14	0.18	0.27	0.50	1.14	1.80
p=0.01, k=5	0.01	0.06	0.22	0.61	1.10	1.98	4.19	6.25	13.26	23.81
p=0.01, k=10	0.01	0.07	0.43	2.40	6.35	17.27	26.19	32.87	68.87	138.77
p=0.01, k=20	0.01	0.08	0.43	2.43	6.98	14.75	42.81	128.02	427.77	885.70
p=0.01, k=30	0.01	0.08	0.42	2.38	6.48	14.27	41.89	128.13	425.65	936.94
p=0.1, k=2	0.01	0.01	0.02	0.04	0.07	0.11	0.22	0.41	0.97	1.61
p=0.1, k=5	0.02	0.03	0.05	0.10	0.18	0.32	0.67	1.37	4.17	8.32
p=0.1, k=10	0.07	0.13	0.21	0.57	1.18	2.19	5.54	14.56	51.84	109.92
p=0.1, k=20	0.18	0.56	1.39	4.32	9.42	18.01	45.52	114.35	305.09	787.99
p=0.1, k=30	0.17	0.54	1.31	4.31	10.07	19.49	51.21	148.97	473.55	1040.19

by values of k, v and the number of nodes in the merged user history graph; thus, the transaction time was increased according to increases of $k(= v)$ and the number of nodes of the original graphs. A dominant part of the transaction time of **Algorithm 2** is the transaction time of **Algorithm 1**, where k is large; it is expected that many nodes are removed by **Algorithm 1**, and edges of almost all nodes are removed at the first iteration of the *do-while-loop* in the **Algorithm 2**. Example cases of $k = v = 10$ for **Algorithm 1** (denoted P) and **Algorithm 2** (denoted C) are shown in figure 3. The transaction time is approximately proportional to N^3.

Tradeoff. We evaluated the number of nodes that were outputs of algorithms, where we changed parameters k and v. Evaluation results are shown in figure 4. The evaluation results denote reductions of information that is provided by (k, v)-untraceable history graph, where a privacy requirement is tightened. We used random graphs of $n = 1000$ and $p = 0.01$, and evaluated three cases: k was fixed ($k = 10$) and v was changed, k and v were changed ($k = v$), and k was changed and v was fixed ($v = 10$). For both partial (k, v)-untraceable history graphs and complete (k, v)-untraceable history graphs, a result in the case $v = 10$ was almost same as that in the case $k = v$. Changes of the parameter v did not affect partial (k, v)-untraceable history graphs, but strongly affected complete (k, v)-untraceable history graphs. In any of the cases, there are threshold values for parameters k and v; the number of nodes is dramatically reduced, where k and v is increased over the threshold values. Thus, we should choose appropriate values for parameter sets, k and v in an operation of each history graph, even though there is a tradeoff between privacy and the amount of infor-

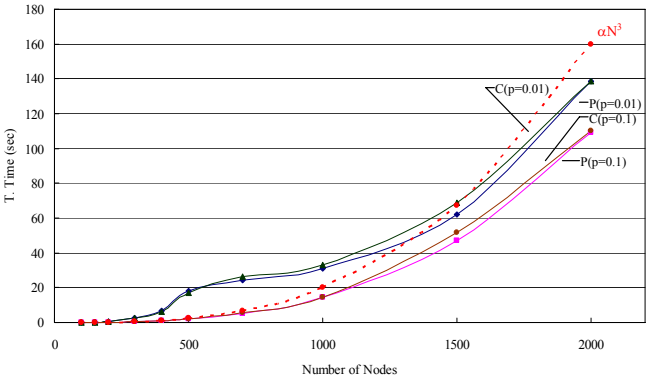

Fig. 3. Transaction Time of Algorithms ($k = 10$)

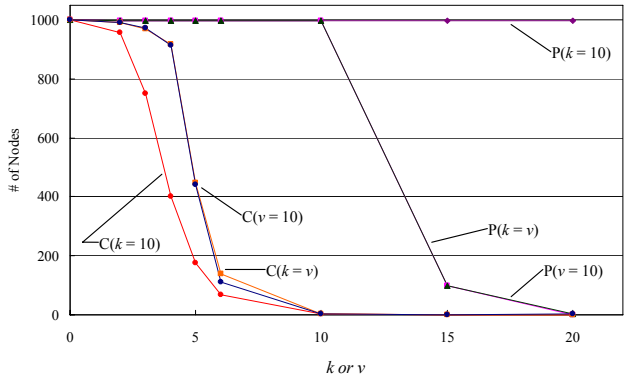

Fig. 4. Number of Nodes for Each Parameters k and v

mation. Especially, the partial (k, v)-untraceable history graph is useful since it holds almost nodes and the graph is close to the original random graph, where k and v are small.

6 Conclusion

In this paper, we have presented concrete algorithms for generating graph data that satisfies the notions. Furthermore, experimental results for performance and tradeoff analyses of the algorithms have been presented.

The extension of the adversary model remains an open issue for our future work. We will consider an adversary who knows an action for a fraction of users in the merged history graph and an adversary who knows some continuous actions of a user in the graph, and effects of the extended adversary models will be analyzed. We will also evaluate the feasibility of graphs using real data.

Acknowledgement. The authors would like to thank anonymous reviewers and program committee members for useful comments on an earlier version of this manuscript.

References

1. Hiromi Arai and Jun Sakuma. Privacy preserving semi-supervised learning for labeled graphs. In *Machine Learning and Knowledge Discovery in Databases*, volume 6911 of *LNCS*, pages 124–139. 2011.
2. B Blaustein, A. Chapman, L. Seligman, M. D. Allen, and A. Rosenthal. Surrogate parenthood: Protected and informative graphs. In *Proc. of the 37th International Conference on Very Large Data Bases*, VLDB '11, pages 518–527, 2011.
3. M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate estimation of the degree distribution of private networks. In *Proc. of the 2009 9th IEEE International Conference on Data Mining*, ICDM '09, pages 169–178, 2009.
4. J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189 –1196, 2008.
5. V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. In *Proc. of the 37th International Conference on Very Large Data Bases*, VLDB '11, pages 1146–1157, 2011.
6. T. Kurashima, K. Bessho, H. Toda, T. Uchiyama, and R. Kataoka. Ranking entities using comparative relations. In *Proc. of 19th International Conference on Database and Expert Systems Applications*, DEXA '08, pages 3124–133, 2008.
7. V. Rastogi, M. Hay, G. Miklau, and D. Suciu. Relationship privacy: output perturbation for queries with joins. In *Proc. of the 28th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '09, pages 107–116, 2009.
8. J. Sakuma and S. Kobayashi. Link analysis for private weighted graphs. In *Proc. of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 235–242. 2009.
9. P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS'98)*, page 188, 1998.
10. TWIMPACT UG. TWIMPACT. `http://twimpact.com/`, 2011.
11. Z. Weinberg, E.Y. Chen, P.R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *2011 IEEE Symposium on Security and Privacy*, pages 147 –161, 2011.
12. X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *Proc. of the 8th SIAM Conference on Data Mining*, SDM'08, pages 739–750, 2008.
13. M. Yuan, L. Chen, and P.S. Yu. Personalized privacy protection in social network. In *Proc. of the 37th International Conference on Very Large Data Bases*, VLDB '11, pages 141–150, 2011.
14. B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *Proc. of the 24th International Conference on Data Engineering*, ICDE'08, pages 506–515, 2008.
15. L. Zou, L. CHen, and M. T. Ö zsu. k-automorphism: a general framework for privacy preserving network publication. In *Proc. of the 35th International Conference on Very Large Data Bases*, VLDB '09, pages 946–957, 2009.