SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

August 2017, Vol. 60 082104:1–082104:15
doi: 10.1007/s11432-016-0442-1

Embedding differential privacy in decision tree algorithm with different depths

Xuanyu BAI[1], Jianguo YAO[1*], Mingxuan YUAN[2], Ke DENG[3], Xike XIE[4] & Haibing GUAN[1*]

[1]*Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai* 200240*, China;*
[2]*Huawei Noah's Ark Lab, Hong Kong* 999077*, China;*
[3]*School of Science, RMIT University, Melbourne VIC* 3001*, Australia;*
[4]*Department of Computer Science, Aalborg University, Aalborg East* DK-9220*, Denmark*

Received October 13, 2016; accepted November 14, 2016; published online July 6, 2017

Abstract Differential privacy (DP) has become one of the most important solutions for privacy protection in recent years. Previous studies have shown that prediction accuracy usually increases as more data mining (DM) logic is considered in the DP implementation. However, although one-step DM computation for decision tree (DT) model has been investigated, existing research has not studied the scenarios when the DP is embedded in two-step DM computation, three-step DM computation until the whole model DM computation. It is very challenging to embed DP in more than two steps of DM computation since the solution space exponentially increases with the increase of computational complexity. In this work, we propose algorithms by making use of Markov Chain Monte Carlo (MCMC) method, which can efficiently search a computationally infeasible space to embed DP into DT generation algorithm. We compare the performance when embedding DP in DT with different depths, i.e., one-step DM computation (previous work), two-step, three-step and the whole model. We find that the deep combination of DP and DT does help to increase the prediction accuracy. However, when the privacy budget is very large (e.g., $\epsilon = 10$), this may overwhelm the complexity of DT model, and the increasing trend is not obvious. We also find that the prediction accuracy decreases with the increase of model complexity.

Keywords differential privacy, decision tree, exponential mechanism, exhaustive search, MCMC

Citation Bai X Y, Yao J G, Yuan M X, et al. Embedding differential privacy in decision tree algorithm with different depths. Sci China Inf Sci, 2017, 60(8): 082104, doi: 10.1007/s11432-016-0442-1

1 Introduction

With the increasing development of big data, data preserving techniques are becoming more and more important. Exposure risks are being emphasized, especially for sensitive data. Data privacy protection technologies mainly contain three aspects: data distortion, data encryption and limited data publication. Many privacy protection methods combine different technologies. *K-anonymity* and *L-diversity* are the two representative data publication based privacy protection methods. However, *K-anonymity* is

* Corresponding author (email: jianguo.yao@sjtu.edu.cn, hbguan@sjtu.edu.cn)

vulnerable to homogeneity attack and background knowledge attack and *L-diversity* is vulnerable to similarity attack. Neither of these technologies strictly define an attack model nor give a quantified definition of the background knowledge that attackers have. In order to solve these problems, Dwork et al. [1] put forward Differential Privacy (DP) protection technique.

DP has become the state-of-the-art privacy protection technique. Compared with other data publication based privacy protection solutions [2, 3], it can mathematically provide a very strong privacy protection guarantee. DP can ensure that the output of any authenticated query/calculation is insensitive to any individual record's adding/removing in the database (DB). The previous work [4] has provided many interesting observations on the performance of DP with decision tree (DT) based data mining (DM) algorithms in a large industrial telecommunication big data platform. One important observation is that the prediction accuracy of DT would increase when more DM computations are considered in DP implementation. However, the deepest combination of DP and DT in [4] is to implement DP with one-step DM computation (i.e., nodes are split one layer during one computation in a decision tree). It is very important and challenging to study more complex scenarios when DP is embedded in two-step DM computation, three-step computation until the whole model computation.

In DP implementation, the *Laplace mechanism* [1] is used for computations with an output that is a number and the *Exponential mechanism* [5] is used for computations with an output that is a category value or a structure. DP can be used in many different data mining algorithms. In this paper, we discuss the usage of DP in the decision tree algorithm. When generating a decision tree, the output of the major computation is to select the best splitting structure. Thus, the Exponential Mechanism should be used to select a structure by searching all possible structure space (solution space). With the increase of computational complexity (i.e., different steps of computation are implemented in DT), the solution space exponentially increases. For example, if the training data has 16 attributes and each attribute has 2 values, the number of solutions of subtree with 2 layers (one-step computation), 3 layers (two-step computation), 4 layers (three-step computation) and 5 layers are 16, $16 \times 15^2 = 3600$, $16 \times 15^2 \times 14^4 = 138297600$ and $16 \times 15^2 \times 14^4 \times 13^8 = 1.12813601 \times 10^{17}$ respectively. Thus, it is very challenging to search the entire solution space when DP needs to be embedded in more than two steps of DM computation. Proper sampling solutions must be developed to implement deep embedding of DP in DT. In this paper, we propose a solution that we use exhaustive search in one-step and two-step embedding algorithms and Markov Chain Monte Carlo (MCMC) method [6] in deeper embedding algorithms. MCMC method is a random process, which can efficiently search an exponential space using a sequence of limited number instances. With this design, we can freely embed DP in DT with different depths.

This paper aims to study the effects of different embedding depths on DT model performance when applying DP in DM. In summary, the following contributions are made:

• We propose a novel idea focused on deep embedding of DP in DT based DM applications with different depths in order to improve DM performance.

• We propose algorithms by making use of MCMC method, which is time-efficient to embed DP deeply into the decision tree generation algorithm.

• We conduct extensive experiments to study how different embedding depths influence the DT model performance. Experimental results show that deep combining DP and DT can significantly increase the prediction accuracy.

The rest paper is organized as follows. Section 2 provides a brief review of the state of the art of Differential Privacy. Section 3 presents our algorithm design. Section 4 reports the extensive experimental results. We conclude this work in Section 5.

2 State of the art of differential privacy

This section provides a brief review of the latest development of Differential Privacy (DP) and the preliminaries of DP embedding algorithms.

2.1 Typical algorithms of differential privacy

Recent research work has been conducted on DP solutions due to DP's strong mathematical control of privacy leak [7], though some work still concentrates on previous privacy-preserving methods [8, 9]. Application of DP to the histogram method plays an important role in DP research. Dwork et al. [10] highlighted that the advantages of applying DP to histogram method are that calculation of sensitivity is independent of dimension and privacy protection can be realized by adding only a little noise for contingency table and covariance matrix analysis with a high dimensional output. The study [11] provided a generalization about histogram method with DP completely. Another important application of DP is with the *K-means* clustering method. Blum et al. [12] applied DP in *K-means* algorithm and produced a formula to calculate the sensitivity of the query function. Research on classifier models with DP is also a popular research area. Chaudhuri et al. [13] applied DP in regular logistic regression and proposed a DP method that is independent of sensitivity. The authors of [14] studied adapting DP for a decision tree. In addition, several research work applies DP in other scenarios. Xiao et al. [6] designed a new data publication model using graphs, which are tree structures. DP is used to protect this structure. Erlingsson et al. [15] focused on using DP in crowdsourcing statistics, which come from end-user client software. Wang et al. [16] explored the hardness for query answering mechanism on DP beyond the stateless restriction. Additionally, DP has been used in other different situations, such as item set mining [17], crowdsourcing [18] and geospatial data queries [19]. Machanavajjhala et al. [20] considered DP to be a privacy measurement and studied the relationship between accuracy and privacy of social recommendations.

One interesting observation in work [4] is that the prediction accuracy of DT usually increases when more DM computations are considered in DP implementation. However, the combination of DP and DT in [4] only implements DP with one-step DM computation [14]. In this work, we study algorithms that provide deeper embedding of DP in DT by making use of exhaustive search and MCMC method. We compare the performance when embedding DP in DT with different depths and reveal interesting observations.

3 DP embedding algorithm design

This section formally presents the proposed algorithms that embed DP in DT with different depths. We firstly introduce the common DT generation algorithm with DP (i.e., the previous work) and propose our new ideas. We then design a new quality function used to evaluate the effectiveness of the splitting structure (subtree). Next we talk about the details of our new embedding algorithms. Finally, we give the time complexity analysis of the proposed algorithms.

3.1 The skeleton of decision tree generation algorithm with DP

There are three major calculations performed during decision tree generation: inner splitting, leaf node label generation and stopping condition checking. Therefore, in order to embed DP, we need to implement DP for these three calculations. It is obvious that the Laplace Mechanism can be used to apply DP for the leaf node label generation and the Exponential Mechanism can be used to apply DP for splitting a node. Stopping condition checking normally checks whether the tree has reached the pre-defined maximal depth, or there are some attributes left to split, or all instances in the leaf node have the same label or the number of instances in a leaf node is lower than a threshold. Only calculation of the number of instances in a leaf node needs DP protection with the Laplace Mechanism. If these three calculations are protected by DP, the decision tree generation is thereby protected. Since the calculations are independent for different branches, according to DP's Composability and Maximum properties, we only need to guarantee that the sum of privacy budgets consumed by calculations on the same branch does not exceed the privacy budget limit ϵ. A branch represents the path from the root node to a leaf node and each leaf node has a different

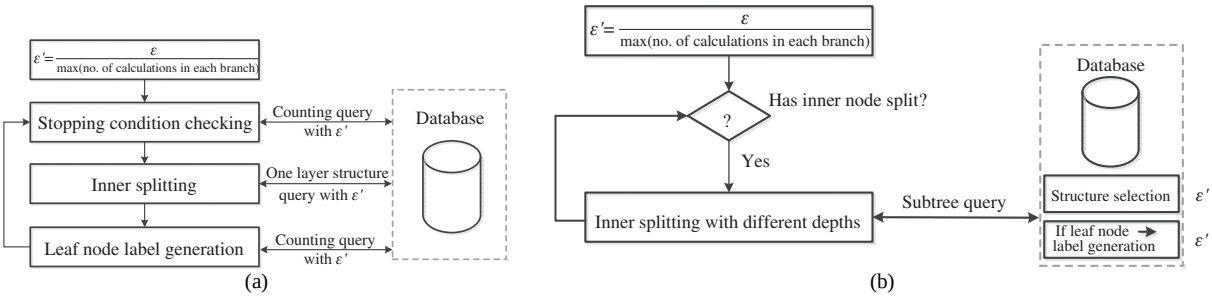

Figure 1 DP implementation skeleton of decision tree. (a) Previous work; (b) this work.

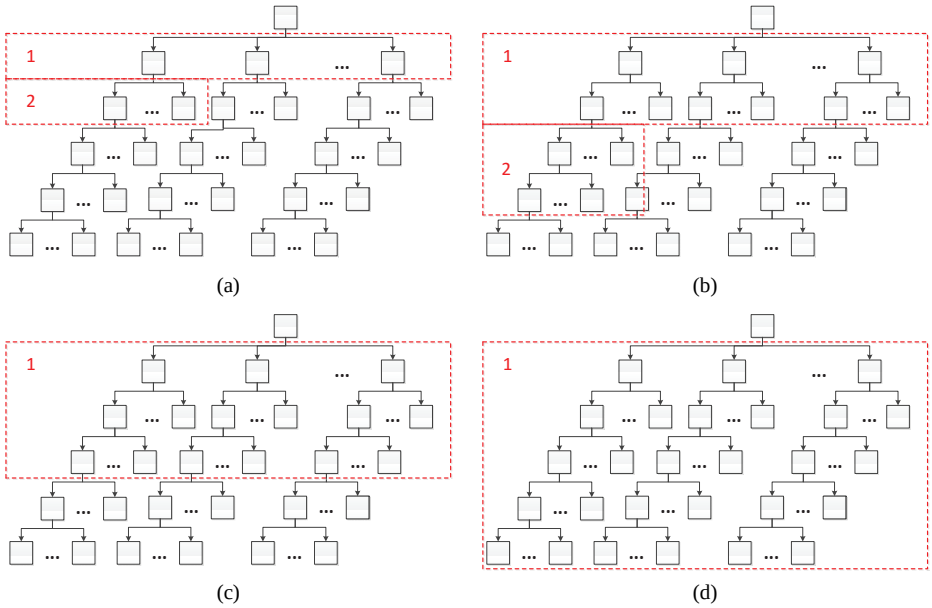

Figure 2 (Color online) Embedding DP in DT with different depths. (a) One-step computation (previous work); (b) two-step computation; (c) three-step computation; (d) whole tree computation.

branch. Thus, each calculation uses $\frac{\epsilon}{\max(\text{number of calculations in each branch})}$ budget. This is how previous work [4, 14] implements DP for decision tree. This implementation is demonstrated in Figure 1(a).

In this paper, we design a more advanced solution. When we split an inner node, we can have different selections to apply DP. We can apply DP for just splitting only one inner node (one-step computation), i.e., the previous work. We can also consider DP when we generate a three-layer subtree under this inner node (the tree generation algorithm splits two layers each time). We can further generate a four-layer subtree under this inner node with DP protection. If a tree is directly generated with the maximum depth under the root node, DP can be implemented for the whole model generation (all inner splitting computations are regarded as one computation). These new ideas are shown in Figure 2. These new algorithms are compared with the existing method [14], the one-step computation algorithm, which were not considered in previous work. All calculations, including the inner splitting and leaf node manipulating, are combined together as one calculation behind the interface of database. The design skeleton is shown in Figure 1(b). Since this work is focused on studying how to combine DP and DT with different depths, different from previous work [4, 14], we do not prune the tree. In the following sections, we will introduce our detailed design.

3.2 Quality function

In DT generation algorithms, quality function is used to select the splitting attribute, which is based on several impurity measurements, such as Information Gain, Information Gain Ratio or Gini Index.

In $CART$ algorithm, Gini Index has been commonly used to evaluate the effectiveness of the splitting. Smaller value of Gini index indicates better splitting when generating a decision tree.

As in previous work [4,14], on the basis of Gini Index, the quality function is q:

$$q(n) = -\sum_{j=1}^{t} \mathcal{T}_j \left(1 - \sum_{c \in C} \left(\frac{\mathcal{T}_{j,c}}{\mathcal{T}_j}\right)^2\right), \tag{1}$$

where n represents a node and t is the number of children belonging to n. We define $child_j$ as the jth child node of n. $\mathcal{T}_j$ represents the number of instances of $child_j$ and $\mathcal{T}_{j,c}$ represents the number of instances of $child_j$, whose value of class label C is c. The sensitivity of the function is $\triangle q = 2$ [14] and larger value of q indicates better splitting.

In this paper, in order to make use of the Exponential Mechanism to select a splitting structure, we define a new quality function and estimate its global sensitivity. Since a subtree is generated in each computation instead of one layer, the score of the subtree's root node is used as the quality function. When generating a subtree st from node n (node n is the root node of st), the score calculation formula is

$$q'(n) = \sum_{j=1}^{t} \begin{cases} q'(child_j), & (child_j) \in \mathcal{S}_i, \\ -\mathcal{T}_j \left(1 - \sum_{c \in C} \left(\frac{\mathcal{T}_{j,c}}{\mathcal{T}_j}\right)^2\right), & (child_j) \in \mathcal{S}_l, \end{cases} \tag{2}$$

where $\mathcal{S}_i$ and $\mathcal{S}_l$ are the set of inner nodes and the set of leaf nodes in st respectively.

It can be observed that (2) is an extension of (1), and it is easy to see they have the same form. Thus, the global sensitivity of the new quality function in this paper is $\triangle q' = 2$.

3.3 DP embedding algorithm

The detailed implementation of DP embedding algorithm is shown in Algorithm 1. As well as the training instance set $\mathcal{T}$, attribute set A and class label C, the maximal tree depth D, embedding depth d and the differential privacy budget ϵ are also input parameters. Privacy budget ϵ controls the privacy protection strength. D represents the maximum depth of the tree without breaking privacy. d represents how many steps of computation we want to combine with DP. In this algorithm, there are a maximum of $\lceil \frac{D}{d} \rceil + 1$ computations in each branch. The budget is equally allocated to each computation, so each single major computation gets a privacy budget $\frac{\epsilon}{\lceil \frac{D}{d} \rceil + 1}$. Since the inner splitting and leaf node manipulating calculations are all moved into the database, in our framework, only the subtree structure and leaf node labels are returned. Stopping condition checking becomes a hidden middle calculation step. Thus when DP is embedded in one-step DM computation, our privacy budget for each calculation is $\frac{\epsilon}{D+1}$ instead of $\frac{\epsilon}{2(D+1)}$ in previous work [4,14], in which an extra counting query needs to output its result at each layer in stopping condition checking. Each time, a $(d+1)$-layer subtree extension is completed. If there are less than d layers left to reach the maximum depth, the tree is extended with D-(current depth) layers. Each node has a tag $stopFlag$ set in Algorithm 2 or Algorithm 3, representing whether this node meets the terminating condition or not, i.e., whether it can be split any further.

When we want to embed DP in one-step or two-step computation, we can enumerate all the possible subtree (splitting) selections and directly use the Exponential Mechanism to search the subtree space. We call this implementation *the Embedding Algorithm with Exhaustive Search*. However, when DP is embedded in more than two steps of computation, it is computationally infeasible to search all the subtree selections and MCMC method is used to simulate the Exponential Mechanism. We call this implementation *the Embedding Algorithm with MCMC*.

3.4 Exhaustively search the splitting selections

When embedding DP with one-step or two-step DM computation ($d = 1$ or $d = 2$), the subtree space is not too large and all possibilities can be enumerated. The algorithm is shown in Algorithm 2. When splitting a node, the attributes that require checking cannot overlap with the splitting attributes used

Algorithm 1 DP Embedding algorithm—DPEA($\mathcal{T}$, A, C, D, d, ϵ)

Require: $\mathcal{T}$—training dataset, A—attribute set, C—class label, D—maximal tree depth, d—steps of computation to embed DP, ϵ—privacy budget;
Ensure: dt—a decision tree;
$\epsilon' = \frac{\epsilon}{\lceil \frac{D}{d} \rceil + 1}$
dt = root
$innerNodes$ = {[root, stopFlag=False]}
while $|innerNodes| > 0$ **do**
  $innerNode$ = pop one node from $innerNodes$
  $left = D -$ (current depth)
  if $left < d$ **then**
    if $left \leqslant 2$ **then**
      subTree st = exhaustive_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, $left$, ϵ')
    else
      subTree st = mcmc_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, $left$, ϵ')
    end if
  else
    if $d \leqslant 2$ **then**
      subTree st = exhaustive_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')
    else
      subTree st = mcmc_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')
    end if
  end if
  for each leaf node ln in st **do**
    if $ln.stopFlag ==$ False **then**
      $innerNodes = innerNodes \cup \{ln\}$
    end if
  end for
  extend dt with st from $innerNode$
end while
return dt

Algorithm 2 Exhaustively search the subtree space—exhaustive_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')

Require: $innerNode$—the node to split, $\mathcal{T}$—training dataset, A—attribute set, C—class label, D—maximal tree depth, d—steps of computation to embed DP, ϵ'—privacy budget;
Ensure: st—a subtree;
$ST[]$ = all (d+1)-layer subtrees rooted at $innerNode$ whose attributes do not appear in $innerNode$'s ancestors
for each $ST[i] \in ST$ **do**
  calculate $ST[i].score$
  $P[i] = \exp(\frac{\epsilon' * ST[i].score}{2 * \triangle q'})$
end for
$sum = \sum_{i=0}^{|P|-1} P[i]$
for each $P[i]$ **do**
  $P[i] = P[i]/sum$
end for
$r = random(0, 1)$
$st = ST[i]$|where r falls in P[i]
for each leaf node ln in st **do**
  if ln reach the stopping condition **then**
    $\forall c \in C : N_c = NoiseCount(\text{c in } ln, \epsilon')$
    $ln.label = argmax_c(N_c)$
    $ln.stopFlag =$ True
  else
    $ln.stopFlag =$ False
  end if
end for
return st

Algorithm 3 Search the subtree space with MCMC—mcmc_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')

Require: $innerNode$—the node to split, $\mathcal{T}$—training dataset, A—attribute set, C—class label, D—maximal tree depth, d—steps of computation to embed DP, ϵ'—privacy budget;
Ensure: st—a subtree;
1: st = a random (d+1)-layer subtree rooted at $innerNode$ whose attributes do not appear in $innerNode$'s ancestors
2: $t = tag = 0$
3: **while** $buffer.varience > threshold$ and $t < stepLimit$ **do**
4: compute $st.score$
5: $buffer[(tag{+}{+}) \ \% \ buffer.size\ ] = st.score$
6: st' = randomly replace one inner node in st
7: compute $st'.score$
8: $st = st'$ with probability

$$\min\left(1, \frac{\exp(\frac{\epsilon' * st'.score}{2*\triangle q'})}{\exp(\frac{\epsilon' * st.score}{2*\triangle q'})}\right)$$

9: $t{+}{+}$;
10: **end while**
11: **for** each leaf node ln in st **do**
12: **if** ln reach the stopping condition **then**
13: $\forall c \in C : N_c = NoiseCount(\text{c in } ln, \epsilon')$
14: $ln.label = argmax_c(N_c)$
15: $ln.stopFlag$ = True
16: **else**
17: $ln.stopFlag$ = False
18: **end if**
19: **end for**
20: return st

by its ancestors. We exhaustively find all the subtrees satisfying the above condition. We assign a score to each possible splitting selection and generate the probability according to the Exponential Mechanism formula. The score is calculated by (2). When the probability of selecting each splitting solution is obtained, a normalized probability array is generated to implement the Exponential Mechanism. A random number is used to select a subtree st according to the normalized probability array P. Here we give an example. We assume that there are three subtrees in total, i.e., the lengths of ST and P are both 3. The initial elements of the array P are assumed to be 0.3, 0.5 and 0.4. After normalizing, P becomes $[\frac{0.3}{1.2}, \frac{0.5}{1.2}, \frac{0.4}{1.2}](0.3 + 0.5 + 0.4 = 1.2)$. After accumulating, P becomes $[\frac{0.3}{1.2}, \frac{0.3+0.5}{1.2}, 1]$. A number r is then generated randomly and its value is assumed to be 0.73. As a result, we will select the third subtree to split since $\frac{0.3+0.5}{1.2} < 0.73 < 1$. All the leaf nodes in st have a flag to represent whether it can be split or not. If we get a leaf node, we use noised counting through the Laplace Mechanism to assign label to it. The function $NoiseCount()$ [14] adds noise to the counting results of nodes through the Laplace Mechanism.

3.5 Search the splitting selections with MCMC method

Exhaustive search is not efficient when $d > 2$, since the subtree space becomes too large to enumerate. In this case, we make use of MCMC method to handle it. The details can be found in Algorithm 3. Firstly, a (d+1)-layer subtree st is generated in a random manner. MCMC simulates the Exponential Mechanism using a sequence of local transitions in st. It is checked whether st meets either of the terminating conditions: the number of operations has reached its limit or st has been convergent. Convergence means that variance of latest specified number of scores is less than the threshold value. $Buffer$ is an array that stores the latest specified number of scores. If the terminating conditions are not achieved, we randomly replace one inner node in st to get st' and update the current state in the following way:

$$st = \begin{cases} st', & \text{with probability } \alpha, \\ st, & \text{with probability } 1-\alpha, \end{cases} \tag{3}$$

where

$$\alpha = \min\left(1, \frac{\exp(\frac{\epsilon' * st'.score}{2 * \triangle q'})}{\exp(\frac{\epsilon' * st.score}{2 * \triangle q'})}\right).$$

Then we check whether st meets the terminating conditions again. If not, the algorithm repeats the replacing process. When a terminating condition is achieved, the algorithm obtains st. After checking and assigning labels to leaf nodes, the algorithm returns st. Since the number of attribute combinations in the subtree is not infinite and the subtree can change from one state to any other state through finite replacements with specific probabilities, the process of MCMC will become convergent eventually.

When the embedding depth plus one equals the maximum tree depth ($d + 1 = D$), the whole DM model computations are regarded as one computation to embed DP in. The privacy budget assignment becomes $\epsilon' = \frac{\epsilon}{2}$ and there is only one element in $innerNodes$ during the algorithm, i.e., the root node. A D-layer subtree is randomly generated to start the MCMC method.

3.6 Time complexity analysis

Finally in this section, we would like to provide the computational complexity analysis of the proposed algorithms. Firstly, some parameters are defined: N—number of attributes that do not appear in $innerNode$'s ancestors, M—number of instances in $innerNode$, K—max(number of values of each attribute that does not appear in $innerNode$'s ancestors) and $stepLimit$—number limit of operations. For Algorithm 2, when $d = 1$, the computational complexity is $\mathrm{O}(MN)$. When $d = 2$, the maximum computational complexity is $\mathrm{O}(MN \times (N-1)^K)$. For Algorithm 3, the computational complexity is $\mathrm{O}(dMstepLimit)$. K' represents the maximum number of values of each attribute in A. For d-step computation embedding, Algorithm 1 invokes Algorithm 2 or Algorithm 3 a maximum of $\frac{1-K'^{\lceil \frac{D}{d} \rceil}}{1-K'}$ times. This is a polynomial time algorithm. Note that classification using DT is an offline data mining task and the classification accuracy is the key optimization objective.

4 Experimental results

4.1 The dataset and evaluation strategy

Our algorithms are tested on three public datasets from UCI Machine Learning Repository, namely *Congressional Voting Records* (*vote*)[1)], *Mushroom* (*mushroom*)[2)] and *Tic-Tac-Toe Endgame* (*tic-tac-toe*)[3)]. These three datasets all have one class label and contain 16, 22 and 9 attributes respectively. The average prediction accuracy of 10-*fold cross-validation* is used to evaluate the decision trees. 10-*fold cross-validation* is a widely used method to represent the prediction effect of a classifier. In 10-*fold cross-validation*, the initial dataset is equally divided into 10 partitions, with which 9 of them are used as training dataset and 1 is used as test dataset. The process repeats 10 times by making each partition work as test data one time. The definition of prediction accuracy is $\frac{T'}{T}$, where T is the number of instances in the test dataset and T' is the number of instances whose class label is correctly predicted by the decision tree.

In our experiments, the performance of four algorithms is evaluated with different embedding depths. We choose $d = 1, 2, 3, D-1$ and call the four algorithms *one-step embedding algorithm* (previous work) [14], *two-step embedding algorithm*, *three-step embedding algorithm* and *the whole tree embedding algorithm* respectively. The performance of these four algorithms is tested under different privacy budgets and ϵ is varied from 0.01 to 10. Note that the privacy budget should be typically smaller than 1 and most

1) UCI Machine Learning Repository dataset, Congressional Voting Records. http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records.

2) UCI Machine Learning Repository dataset, Mushroom. http://archive.ics.uci.edu/ml/datasets/Mushroom.

3) UCI Machine Learning Repository dataset, Tic-Tac-Toe Endgame. http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame.

studies have a preference for using values such as 0.1 and 1 [1, 14, 21]. In order to remove randomness, each algorithm is run 300 times to obtain the average performance.

For the first dataset *vote*, one-step and two-step embedding algorithms both use the function [exhaustive_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')]. The three-step and whole tree embedding algorithms use the function [mcmc_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')]. In the three-step embedding algorithm, the terminating conditions of MCMC method are a buffer size of 50 with the number of operations limited to 1000. This means that if the variance of latest 50 scores is less than the threshold value or the number of operations reaches 1000, the subtree generation process is complete. For the whole tree embedding algorithm, since the number of attribute combinations is much larger than that for the three-step embedding algorithm, more old nodes need to be replaced. The buffer size is set as 500 and the number of operations is limited to 10000. On the other side, when the number of instances in a node is less than 20, this node cannot be split any further.

For the second dataset *mushroom*, due to much larger number of instances and attributes, some parameters are slightly changed. The large number of attributes results in too many subtree selections during each split even for the two-step embedding algorithm, which is computationally infeasible. Therefore for *mushroom*, the two-step embedding algorithm uses the function [mcmc_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')] instead of [exhaustive_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')]. More operations in MCMC are also required. The algorithm parameters are set as follows. In the two-step algorithm, the number of operations is limited to 500 and the buffer size is set as 50. In the three-step algorithm, the number of operation is limited to 3000 and the buffer size is set as 75. In the whole tree algorithm, the number of operation is limited to 30000 and the buffer size is set as 500. The threshold for the number of instances in a node is set as 50.

For the third dataset *tic-tac-toe*, due to its size being between *vote* and *mushroom*, we use the function [mcmc_search_Exp($innerNode$, $\mathcal{T}$, A, C, D, d, ϵ')] in two-step, three-step and whole three embedding algorithms. In two-step algorithm, the number of operations is limited to 500 and the buffer size is set as 50. Other parameters are the same as those for *vote*.

4.2 MCMC convergence

To confirm the correctness of using MCMC to realize Exponential Mechanism and the convergence of MCMC, we follow the method used in [6]. The main purpose of MCMC method is to draw a random sample from the desired distribution. Additionally, the essence of Exponential Mechanism is a method to draw a sample x from $\mathcal{X}$ with probability proportional to $\exp(\frac{\epsilon * q(x)}{2 * \triangle q})$, where $q(x)$ is the score function and $\triangle q$ is the sensitivity of this function. Therefore, when the distribution of MCMC matches the target distribution required by Exponential Mechanism, MCMC method can be used to realize Exponential Mechanism.

In this paper, the transition ratio of MCMC is set as

$$\min\left(1, \frac{\exp(\frac{\epsilon' * st'.score}{2 * \triangle q'})}{\exp(\frac{\epsilon' * st.score}{2 * \triangle q'})}\right),$$

where $st'.score$ represents $q'(st')$. Hence when MCMC becomes convergent, we draw a sample st' from the distribution of MCMC with the probability:

$$\Pr(st') = \frac{\exp(\frac{\epsilon' * st'.score}{2 * \triangle q'})}{\sum_{st \in ST} \exp(\frac{\epsilon' * st.score}{2 * \triangle q'})}. \tag{4}$$

This is equivalent to the distribution required by Exponential Mechanism that outputs st' with probability proportional to $\exp(\frac{\epsilon' * st'.score}{2 * \triangle q'})$. Therefore, we can conclude that if MCMC is able to become convergent eventually, the application of MCMC method in our algorithms achieves differentially privacy [6]. Interested readers can learn more from Shen et al. [22], which firstly proposed the idea of applying MCMC method to achieve Exponential Mechanism.

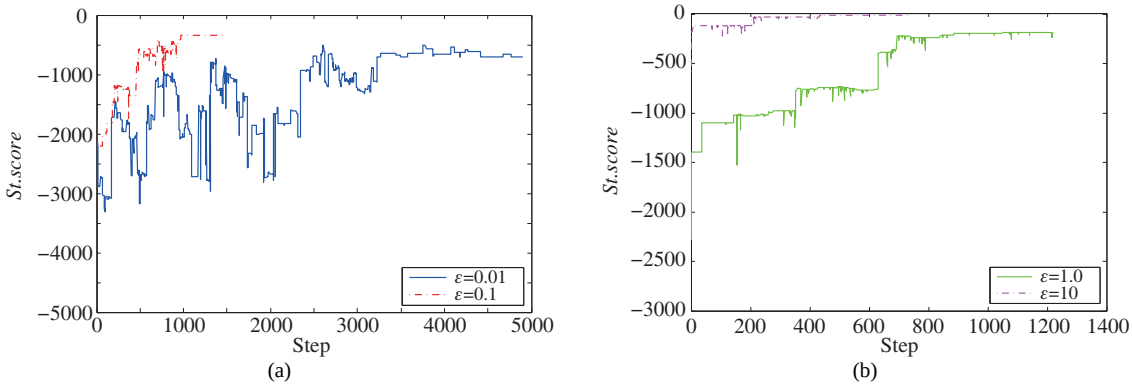

Figure 3 (Color online) Trace of *st.score* for *mushroom*. (a) $\epsilon = 0.01$ or 0.1; (b) $\epsilon = 1.0$ or 10.

Then, we also use the same method used in [6] to discuss the convergence of MCMC. MCMC's convergence is diagnosed by tracing the $q(st)$, i.e., *st.score*, where *st* represents a subtree. We conducted experiments on dataset *mushroom*. For *mushroom*, MCMC method is used in the two-step, three-step and whole tree embedding algorithms. Since the subtree generated in two-step and three-step algorithms is equivalent to a whole tree (depth of tree is 3 and 4 respectively). We only plot the trace of *st.score* in the whole tree embedding algorithm and set the maximal depth of tree as 20. The trace of *st.score* is showed in Figure 3.

For *mushroom*, the buffer size is set as 500 in the whole tree embedding algorithm. That means if the variance of latest 500 scores is less than the threshold value, the subtree generation process is complete. From Figure 3, we can observe that MCMC method is able to become convergent, i.e., *st.score* becomes stable eventually. Figure 3 also shows that if privacy budget ϵ becomes larger, *st.score* will becomes more unstable and MCMC method needs more iterations to become convergent. Therefore, the application of MCMC method to realize Exponential Mechanism in our algorithms is valid and can achieve DP. Additional discussion about MCMC convergence can be found in [6, 23, 24].

4.3 Experimental results

Figure 4 demonstrates the average performance of the four algorithms on *vote*: the one-step embedding algorithm, the two-step embedding algorithm, the three-step embedding algorithm and the whole tree embedding algorithm. Four privacy budgets, 0.01, 0.1, 1 and 10 are tested for each of the algorithms.

Figure 4(a) gives the results when $\epsilon = 0.01$. The x axis is the maximal depth of the decision tree and the y axis is the average prediction accuracy. It can be observed that the whole tree embedding algorithm outperforms the other algorithms. The three-step embedding algorithm performs better than the one-step and the two-step embedding algorithms in most cases. Since the privacy budget is very small, a lot of noise is added. The results fluctuate and the two-step embedding algorithm does not show clear benefits over the one-step embedding algorithm.

Figure 4 (b) and (c) show the results when $\epsilon = 0.1$ and 1 respectively. We can clearly observe that the whole tree embedding algorithm is better than the three-step embedding algorithm, the three-step embedding algorithm is better than the two-step embedding algorithm and the two-step embedding algorithm performs better than the one-step embedding algorithm. This confirms that deeply combining DP and DM does help to increase the prediction accuracy.

Figure 4(d) shows the results when $\epsilon = 10$. When privacy budget is very large ($\epsilon = 10$) to overwhelm the complexity of DT model, the randomness of algorithms holds the dominant position, causing uncertain results. In this situation, the advantages of more steps of embedding algorithms cannot perform very well. For dataset *vote*, the performance differences between the four algorithms are not as clear as $\epsilon = 0.1$ and $\epsilon = 1$ when the maximal depth of decision tree is less than 8. While when the depth is larger than 8, we can still clearly observe most deeper embedding algorithms perform better than shallow embedding algorithms (i.e., the three-step embedding algorithm performs better than the two-step embedding algorithm and

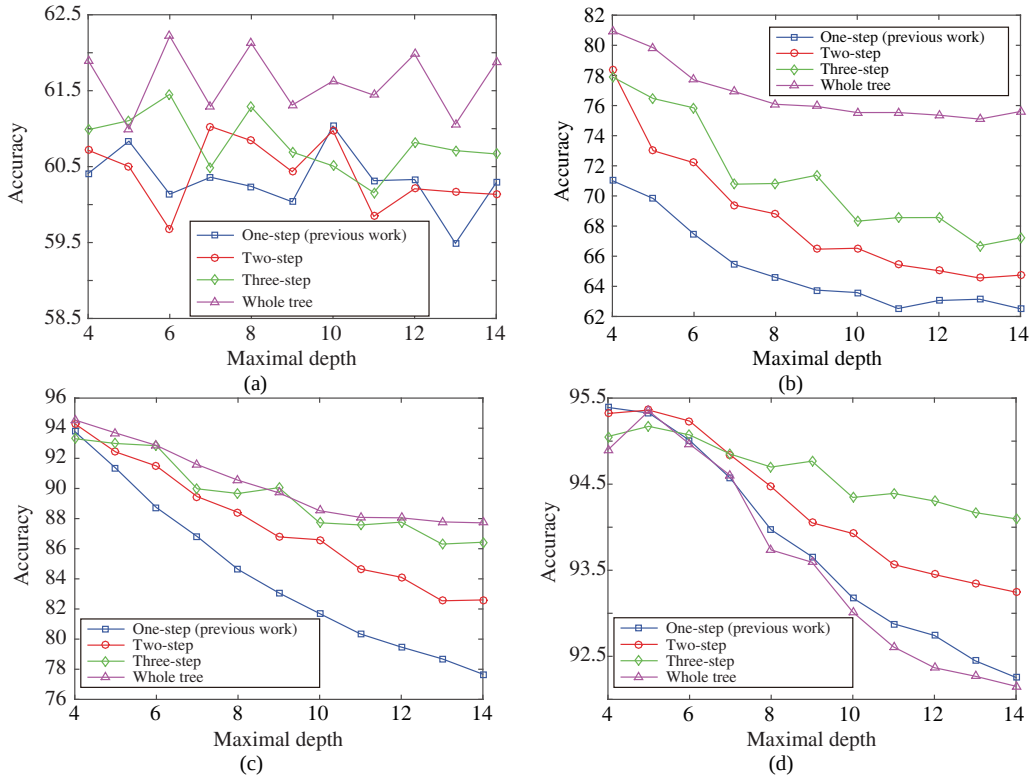

Figure 4 (Color online) Accuracy of algorithms on *vote* with different embedding depths and different privacy budgets. (a) $\epsilon = 0.01$; (b) $\epsilon = 0.1$; (c) $\epsilon = 1.0$; (d) $\epsilon = 10$.

the two-step embedding algorithm performs better than the one-step embedding algorithm). We think that the reason for this is that the complexity of DM model is too small when the depth is less than 8. Thus a big privacy budget such as $\epsilon = 10$, which adds a small amount of noise, does not change the relatively simple structure in the DT model. One exception is the whole tree embedding algorithm, which shows the poorest performance. The experiment is repeated several times with similar results. The reason for this is that when the privacy budget is as large as 10, the DP's influence is very small compared to the real distribution of data. Thus, the randomness in MCMC will introduce extra cost. It should be emphasized that in order to provide protection, the privacy budget should be typically smaller than 1 [1, 14, 21].

Bringing the four figures in Figure 4 together, it can be observed that with the increase of ϵ, the prediction accuracy of decision tree increases a lot. It is obvious that relaxing privacy protection condition helps to maintain the DM effect with DP protection. From Figure 4 (b)–(d), we can also see that with the increase of depth, the prediction accuracy drops. Large depth means more complex model and more attributes are used in the DT model, which means the "variety" of data used in decision tree increases. Similar to Figure 4, Figures 5 and 6 report the results on datasets *mushroom* and *tic-tac-toe* respectively. We can observe the same phenomena as *vote*. Deeper embedding algorithms clearly perform better than shallow embedding algorithms with useful privacy budget settings (smaller than 1 [1,14,21]). In summary, the experimental results of these three datasets verify the effectiveness of our embedding algorithms. The deep combination of DP and DT helps to increase the classification accuracy.

4.4 Comparison with *Private-RDT*

In order to prove the advance of our research work, we compare our method with the existing approach *Private-RDT* proposed by Jagannathan et al [25]. As we are supposed to compare our algorithm with *Private-RDT*, we decide to use the same parameter settings of *Private-RDT* in Jagannathan's paper.

Values of privacy budget ϵ are set as 0.5, 0.75 and 1 respectively. The depth of decision trees generated

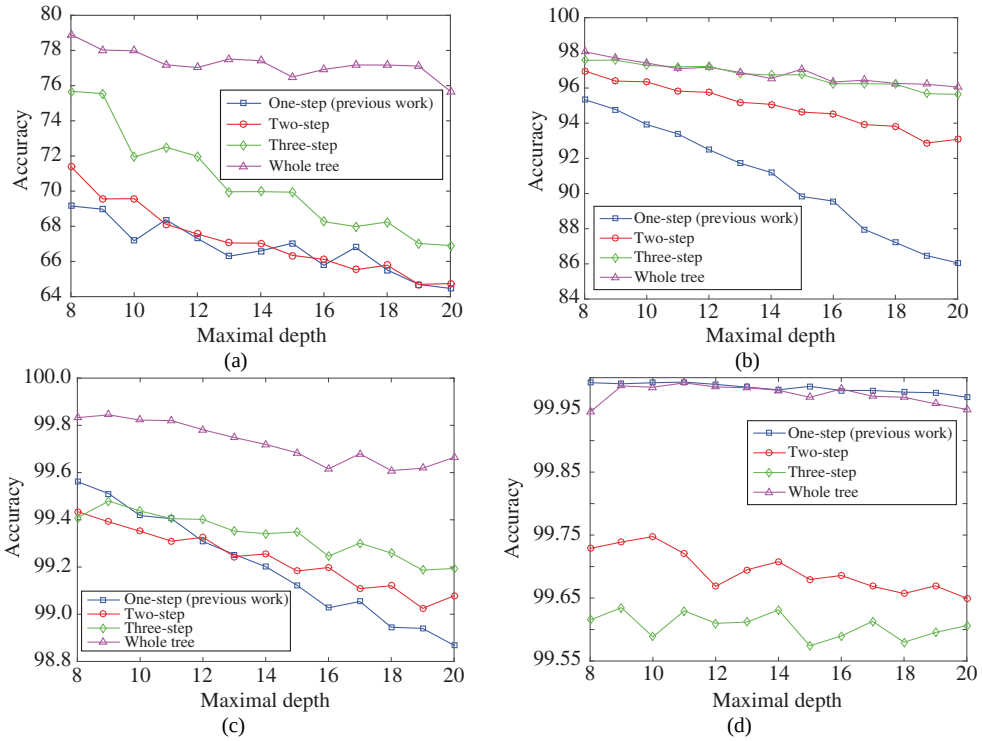

Figure 5 (Color online) Accuracy of algorithms on *mushroom* with different embedding depths and different privacy budgets. (a) $\epsilon = 0.01$; (b) $\epsilon = 0.1$; (c) $\epsilon = 1.0$; (d) $\epsilon = 10$.

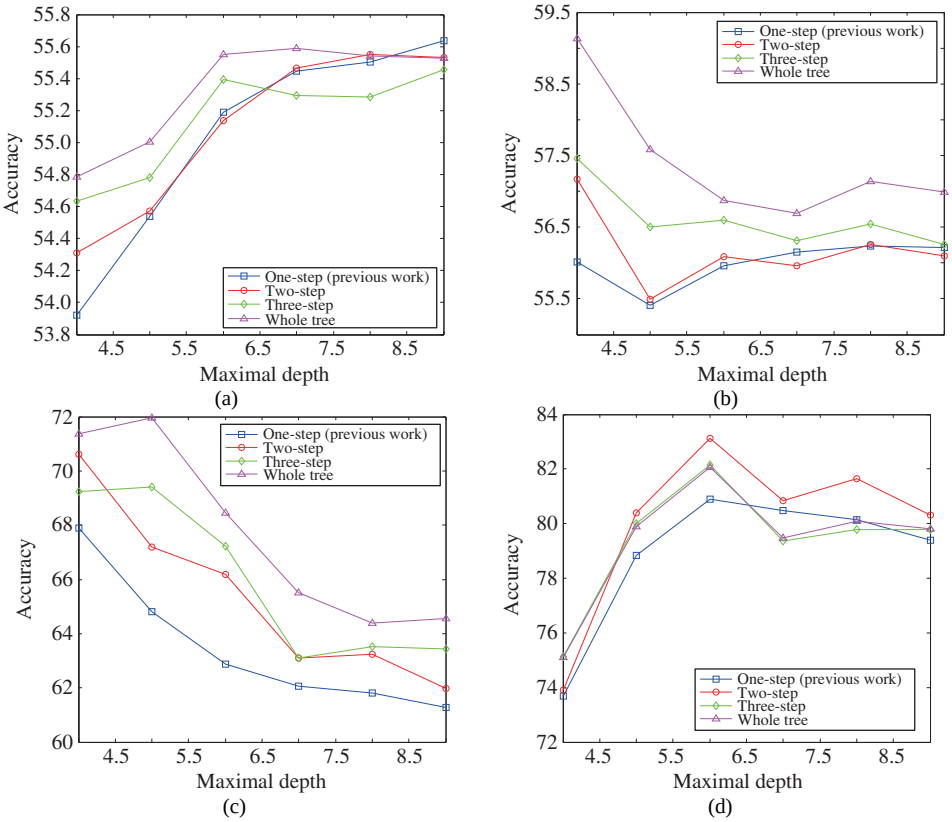

Figure 6 (Color online) Accuracy of algorithms on *tic-tac-toe* with different embedding depths and different privacy budgets. (a) $\epsilon = 0.01$; (b) $\epsilon = 0.1$; (c) $\epsilon = 1.0$; (d) $\epsilon = 10$.

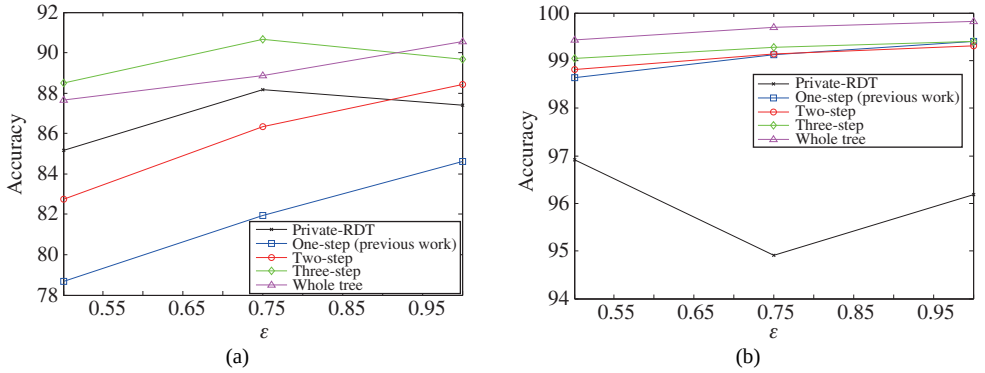

Figure 7 (Color online) Comparison of *Private-RDT* and our algorithms on two datasets. (a) *vote*; (b) *mushroom*.

is $p/2$, where p represents the number of attributes of instances in the dataset. That is to say, the depth of generated trees is 8 for *vote* and 11 for *mushroom*. In *Private-RDT*, the number of trees in the ensemble is not beyond 5 for *vote* and is set as 10 for *mushroom* [25]. In the experiments, our algorithms are run 300 times. We compare these two approaches by using average prediction accuracy, where prediction accuracy is calculated by using cross-validation technique.

Figure 7(a) presents the results of experiments on *vote*. The x axis is the budget ϵ and the y axis is the average prediction accuracy. We can observe that the one-step embedding algorithm (previous work) and the two-step embedding algorithm perform worse than *Private-RDT*. However, the three-step embedding algorithm and the whole tree embedding algorithm perform better than *Private-RDT*. Figure 7(b) presents the results of experiments on *mushroom*. The dataset *mushroom* has much more instances than *vote*. We can obviously observe that our algorithms perform better than *Private-RDT*. Therefore, we can conclude that our method is better than the existing approach *Private-RDT* in [25] and is very meaningful.

4.5 Running time

In this section, we discuss the efficiency (running time) of our algorithms. Our experiments were run on a server with a 3.10 GHz Intel Core i5 processor and 8 GB main memory. Figure 8 shows the average time of generating a single decision tree when the depth is 20 for dataset *mushroom*. In experiments, we use 500, 1000, 2000, 4000 and 8000 instances from *mushroom* respectively and we can see the running time of our algorithms almost increases linearly as the data size scales up.

From Figure 8, we can also observe that more running time is required for deeper embedding algorithms. A DT can be generated within a maximum of 150 s with DP protection. We have analyzed that the proposed algorithms are polynomial time algorithms in Subsection 3.6. Since the classification using DT is an offline data mining task, its efficiency is not a key challenge for our algorithms and the classification accuracy is the primary optimization objective. Deeper embedding of DP in DT increases the accuracy a lot, so it is worthy the cost of a few additional seconds of running time.

5 Conclusion

In this work, we have developed algorithms to flexibly embed differential privacy (DP) in the decision tree model with different depths. Experiments have shown that deeply combining DP and decision tree (DT) does effectively increase the prediction accuracy. This work verifies that one way to apply DP is to embed DP in data mining (DM) algorithms as deep as possible. We have also found that with DP protection, the accuracy of the model is related with privacy budget, the embedding depth and the model complexity. In future work, we would like to study different privacy budget assignment strategies as well as applying deep DP embedding to other DM models.

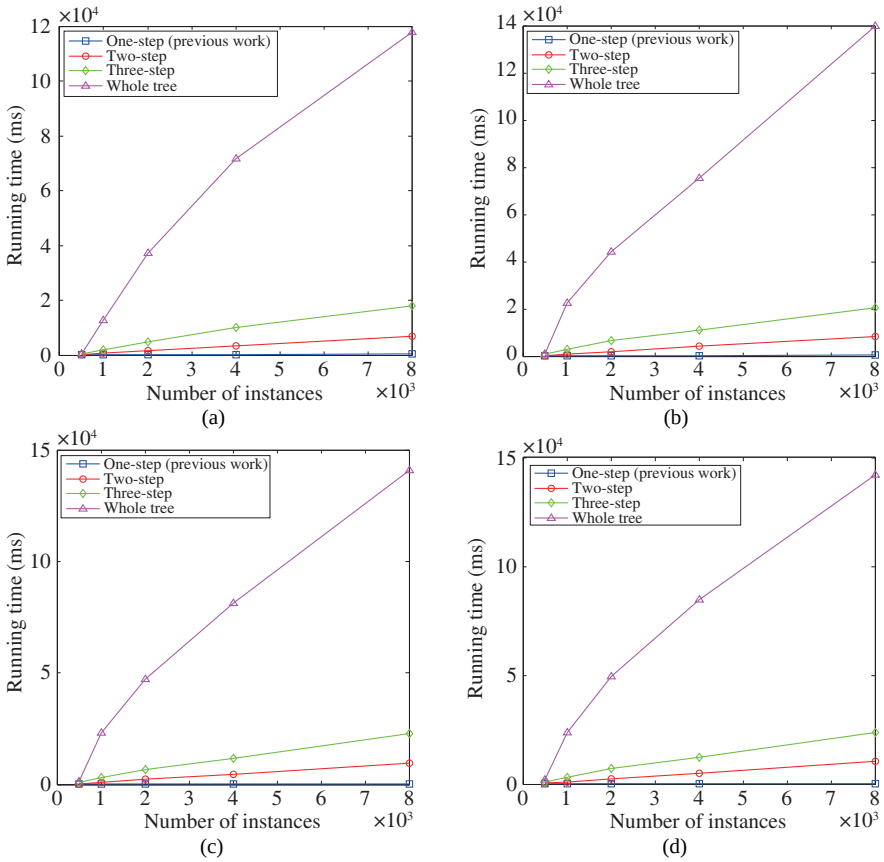

Figure 8 (Color online) Running time of algorithms on *mushroom* with different number of instances and different privacy budgets. (a) $\epsilon = 0.01$; (b) $\epsilon = 0.1$; (c) $\epsilon = 1.0$; (d) $\epsilon = 10$.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 61525204, 61572322), Science and Technology Commission of Shanghai Municipality Project (Grant Nos. 14510722600, 16QA1402200), Aeronautical Science Foundation of China (Grant No. 20145557010), and NRF Singapore CREATE Program E2S2.

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Dwork C. Differential privacy. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, Venice, 2006. 1–12
2 Sweeney L. Achieving k-anonymity privacy protection using generalization and suppression. Int J Uncertain Fuzz, 2002, 10: 571–588
3 Domingo-Ferrer J, Torra V. A critique of k-anonymity and some of its enhancements. In: Proceedings of the 3rd International Conference on Availability, Reliability and Security. Washington, DC: IEEE, 2008. 990–993
4 Hu X Y, Yuan M Y, Yao J G, et al. Differential privacy in telco big data platform. In: Proceedings of the 41st International Conference on Very Large Data Bases Endowment, Kohala Coast, 2015. 1692–1703
5 McSherry F D. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, Rhode Island, 2009. 19–30
6 Xiao Q, Chen R, Tan K-L. Differentially private network data release via structural inference. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, 2014. 911–920
7 Chen R, Xiao Q, Zhang Y, et al. Differentially private high-dimensional data publication via sampling-based inference. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, 2015. 129–138
8 Li H T, Ma J F, Fu S. A privacy-preserving data collection model for digital community. Sci China Inf Sci, 2015, 58: 032101
9 Huang X Z, Liu J Q, Han Z, et al. Privacy beyond sensitive values. Sci China Inf Sci, 2015, 58: 072106
10 Dwork C, Mcsherry F, Nissim K, et al. Calibrating noise to sensitivity in private data analysis. In: Proceedings of the 3rd Conference on Theory of Cryptography, New York, 2006. 265–284
11 Dwork C. A firm foundation for private data analysis. Commun ACM, 2011, 54: 86–95

12 Blum A, Dwork C, McSherry F, et al. Practical privacy: the SuLQ framework. In: Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Baltimore, 2005. 128–138
13 Chaudhuri K, Monteleoni C. Privacy-preserving logistic regression. In: Proceedings of the 22nd Annual Conference on Neural Information Processing Systems, Vancouver, 2008. 289–296
14 Friedman A, Schuster A. Data mining with differential privacy. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington DC, 2010. 493–502
15 Erlingsson U, Pihur V, Korolova A. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, 2014. 1054–1067
16 Wang L W, Zhang J P. On the measurement complexity of differentially private query answering. Sci China Inf Sci, 2015, 58: 092112
17 Li N H, Qardaji W, Su D, et al. PrivBasis: frequent itemset mining with differential privacy. Proc VLDB Endowment, 2012, 5: 1340–1351
18 Hien T, Gabriel G, Cyrus S. A framework for protecting worker location privacy in spatial crowdsourcing. Proc VLDB Endowment, 2014, 7: 919–930
19 Li N H, Yang W N, Qardaji W. Differentially private grids for geospatial data. In: Proceedings of the 2013 IEEE International Conference on Data Engineering. Washington DC: IEEE, 2013. 757–768
20 Machanavajjhala A, Korolova A, Sarma A D. Personalized social recommendations: accurate or private. Proc VLDB Endowment, 2011, 4: 440–450
21 Mohammed N, Chen R, Fung B C M, et al. Differentially private data release for data mining. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, 2011. 493–501
22 Shen E T, Yu T. Mining frequent graph patterns with differential privacy. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, 2013. 545-553
23 Clauset A, Moore C, Newman M E J. Hierarchical structure and the prediction of missing links in networks. Nature, 2008, 453: 98–101
24 Clauset A, Moore C, Newman M E J. Structural inference of hierarchies in networks. In: Proceedings of the 2006 International Conference on Machine Learning on Statistical Network Analysis, Pittsburgh, 2006. 1–13
25 Jagannathan G, Pillaipakkamnatt K, Wright R N. A practical differentially private random decision tree classifier. Trans Data Privacy, 2009, 5: 114–121