Adaptive Anonymization of Data using b-Edge Cover

Arif Khan[1], Krzysztof Choromanski[2], Alex Pothen[3], S M Ferdous[3], Mahantesh Halappanavar[1], and Antonino Tumeo[1]
[1]Pacific Northwest National Laboratory, Richland WA 99354.
{ariful.khan, mahantesh.halappanavar, antonino.tumeo}@pnnl.gov
[2] Google Brain Robotics, New York, 76 Ninth Avenue, New York NY 10011. choromanski1@gmail.com
[3] Computer Science Department, Purdue University, West Lafayette IN 47907. {apothen, sferdou}@purdue.edu

***Abstract*—We explore the problem of sharing data that pertains to individuals with anonymity guarantees, where each user requires a desired level of privacy. We propose the first shared-memory as well as distributed memory parallel algorithms for the *adaptive anonymity* problem that achieves this goal, and produces high quality anonymized datasets.**

The new algorithm is based on an optimization procedure that iteratively computes weights on the edges of a dissimilarity matrix, and at each iteration computes a minimum weighted b-*Edge Cover* in the graph. We describe how a 2-approximation algorithm for computing the b-*Edge Cover* can be used to solve the adaptive anonymity problem in parallel.

We are able to solve adaptive anonymity problems with hundreds of thousands of instances and hundreds of features on a supercomputer in under five minutes. Our algorithm scales up to $8K$ cores on a distributed memory supercomputer, while also providing good speedups on shared memory multiprocessors. On smaller problems where an a Belief Propagation algorithm is feasible, our algorithm is two orders of magnitude faster.

I. Introduction

Research agencies in the U.S., Europe, and Canada (e.g., NIH; Science Europe, an association of several research funding and research performing organizations; Canadian Institutes of Health Research) require research data to be made publicly, permanently, and freely available. However such data often contains sensitive information about individuals, and therefore the engine responsible for data-sharing must be equipped with mechanisms for providing privacy guarantees of that data. (The European Union has begun to enforce the General Data Protection Regulation (GDPR) policies in 2018.) Simultaneously, some notion of utility must be maintained, i.e., a significant fraction of the data should be publishable. Disclosures of sensitive data with insufficient anonymization have a history of being re-identified when joined with other publicly available data. For example, a sanitized data set from a group insurance commission which manages health insurance for Massachusetts state employees was matched against the voter list of Cambridge MA, and the health record of the then Governor of the state was identified. Six such occurrences are listed in [1]. To address this issue, we describe a parallel algorithm for solving a privacy problem called *adaptive anonymity*, using a variational optimization algorithm and a b-*Edge Cover* formulation, implemented on both shared-memory and distributed-memory multiprocessors.

An example of this problem is shown in Figure 1. The matrix on the left is the user data, the matrix on the right is the anonymized data with some values masked with stars. Privacy requirements of the users are specified in the caption. A bipartite graph between the users and the keys illustrates how each user's data is confused with that of others. In the Figure, the anonymized data of y_0 is the same as that of y_1 and y_4 if we treat a star as a wild card that matches any value (0, 1, or $*$). We will consider this example in more detail in Section III.

A real-life example considered in Section VI uses an Open Payments data set managed by the U.S. Centers for Medicare & Medicaid Services (CMS), which is a national disclosure program created by the Affordable Care Act to help consumers understand the financial relationships between the pharmaceutical and medical device industries on the one hand, and physicians and teaching hospitals on the other [2]. Each record of the CMS dataset contains a transaction made to a physician or a teaching hospital. There are different types of payments, and we consider only general payments that include identifying information for the applicable manufacturer or applicable Group Purchasing Organizations (GPO) who made the payment, and identifying information for the recipient. For example, among other features, each record contains: teaching hospital name, physician's name, physician's medical license number, physician medical specialty, physician's address, provider's address, type of payment (e.g., drug, medical device, consultancy fee), amount of payment, method of payment, date of payment, etc. Much useful information can be mined from this dataset that could help develop new medical technologies, prevent wasteful health-care spending and inappropriate influences on clinical decision making, etc. However, it is also possible to misuse this Open Payment information in conjunction with other dataset(s). For example, it is possible to identify certain type of medical needs of a particular area and tie it to the socio-economic profile of that area using publicly available census data and then misuse the information for false advertising, insurance manipulation, etc. Anonymizing some features of this dataset before making it publicly available could prevent such abuse. It is not possible to anonymize the CMS dataset using state of the art anonymization techniques that run on serial computers, since its memory requirement is more than

SC18, November 11-16, 2018, Dallas, Texas, USA
978 -1-5386-8384-2/18/$31.00

2.5 TB. Using our new distributed memory parallel algorithm for *adaptive anonymity*, we are able to anonymize the CMS dataset with high utility (Section VI).

The main drawback of earlier approaches for privacy is that they assume a uniform distribution of the desired privacy levels (value of k) across all the users. However, this condition does not hold true for real-life applications, since users vary in their desired level of privacy. Since these methods are not designed for models where different users may need different levels of anonymity, applications using these models lead to poor utility. Recently, some of the first results regarding this heterogeneous scenario were given [3]. An efficient algorithm creating the so-called *b-matching graph*, a core ingredient of the entire data sharing mechanism, was proposed. Unfortunately, this algorithm is too slow in practice because it computes an optimal b-matching at each iteration (requiring $O(kmn \log n)$ running time, where n is the number of nodes, m is the number of edges in a similarity graph used to produce obfuscated data, and k is the maximum level of privacy desired). The algorithm does not have much parallelism. Other well known exact algorithms for solving the problem stated in [3] share the same weaknesses.

We describe the first shared-memory and distributed-memory parallel data sharing algorithms satisfying adaptive anonymity requirements, giving privacy guarantees adapted to different needs of different users. Our contributions in this paper are as follows:

1) An approximation algorithm that employs variational optimization and a *b-Edge Cover* formulation to solve the *adaptive anonymity* problem.
2) A shared memory implementation of the *adaptive anonymity* algorithm.
3) A linear memory formulation of the algorithm, which enables the solution of problems two orders of magnitude *larger* than previous algorithms on a shared memory machine.
4) A distributed memory implementation of the *adaptive anonymity* algorithm, which can solve large-scale problems three orders of magnitude *faster* than shared-memory implementations.

Our work employs several concepts including a formulation of the adaptive anonymity problem, a variational optimization algorithm to solve the problem, a *b-Edge Cover* computation to solve a grouping step within each iteration of the optimization, the reduction of the *b-Edge Cover* problem to a *b-Matching* problem, and the b-SUITOR algorithm for the last problem. We illustrate the relationships among these problems and algorithms in Fig. 2 to aid in understanding the rest of the paper, which is organized as follows. In Section II, we discuss previous work on data privacy and the *b-Edge Cover* problem. We discuss a general framework for the *adaptive anonymity* problem and identify the most compute-intensive step, called the *Grouping* step in Section III. We show that a 2-approximate *b-Edge Cover* algorithm can efficiently compute the *Grouping* step in Section IV. In Section V, we discuss shared memory and distributed memory implementations of our algorithm. We report performance results in Section VI, and provide concluding remarks in Section VII.

II. Related Work

Here we discuss previous work on data privacy as well as approximation algorithms for the *b-Edge Cover* problem.

A. Data Privacy Algorithms

The are several classes of algorithms for data privacy, of which one of the most powerful is *differential privacy* [4]. Intuitively, differential privacy is the loss of privacy for an individual when their private data is used in a collective data product that can be queried. However it is not a flexible tool in this setting since it usually requires the particular use of the data to be specified in advance. In many applications the users would like to release the raw data, with some obfuscation, for purposes of general exploration. In order to be consistent in our terminology we refer to users as instances and attributes as features from now onward.

Another kind of privacy can be defined on relational data where individuals can identified using some external knowledge. To handle re-identification attacks, Sweeney [5] proposed the k-anonymity method. A data set is said to be *k-anonymous* if each record is indistinguishable from at least $(k-1)$ other records in the context of their features. Here the privacy assigned to each record is a constant value k.

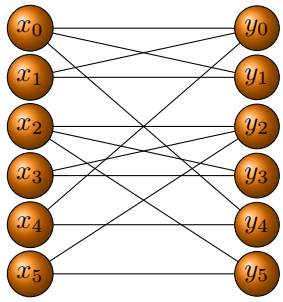
Adaptive Anonymity is a generalization of k-anonymity, where privacy requirements of individuals can vary. Some attempts to address the anonymity issue in the adaptive context were made in [6], [7], [8], [9], but the algorithms were not efficient and exact privacy guarantees were not formulated. The state of the art for the *adaptive anonymity* problem is described in [3]. The algorithm in [3] uses a *Belief Propagation* based formulation, which requires impractically high runtimes for large problems, and is also not guaranteed to converge if the solution is not unique. The algorithm uses a variational optimization formulation, in which a b-matching is computed exactly at each iteration. As stated in the Introduction, this algorithm requires $O(kmn \log n)$ running time.

Several studies discuss ways to strengthen k-anonymity based data privacy, namely l-diversity [10] and t-closeness [11]. The authors of the paper [11] noted that it may be more desirable to use both k-anonymity and t-closeness at the same time. However, both these techniques are based on a procedure for finding equivalence classes, i.e., *grouping* instances based on some criteria. The *grouping* step is also the most compute-intensive step of these algorithms. Hence our work is applicable to such modifications of adaptive anonymity. Our goal is to devise a method to *group* instances that is amenable to parallelization, and we achieve it by computing an approximate *b-Edge Cover* in a graph.

B. b-Edge Cover Algorithms

The *b-Edge Cover* problem is a special case of the *Set Multicover* problem: Here we are given a collection of subsets of a set, each with a cost, and we are required to find a sub-collection of subsets of minimum total cost to cover each element e in the set a specified number $b(e)$ times. If each subset has exactly two elements, then we have the *b-Edge Cover*

username				
Anna (x_0)	1	0	0	0
Robert (x_1)	0	0	0	0
Paul (x_2)	0	0	1	1
Peter (x_3)	1	0	1	1
Carl (x_4)	1	1	0	0
Olaf (x_5)	0	1	1	1



				key
*	*	0	0	172 (y_0)
*	0	0	0	236 (y_1)
*	*	1	1	672 (y_2)
*	0	1	1	229 (y_3)
1	*	0	0	761 (y_4)
0	*	1	1	298 (y_5)

Fig. 1. An example of an *adaptive anonymity* problem. Left: usernames and feature matrix (x, X); Right: the anonymized feature matrix with keys (Y, y); Center: A bipartite graph that matches each user to a set of anonymized keys compatible with the user's data. There are six users and four features, and the privacy requirements are: $k(x_0) = 3, k(x_1) = 2, k(x_2) = 3, k(x_3) = 2, k(x_4) = 2, k(x_5) = 2$. The solution using adaptive anonymity masks eight data items, while k-Anonymity for $k \geq 2$ would mask ten elements.

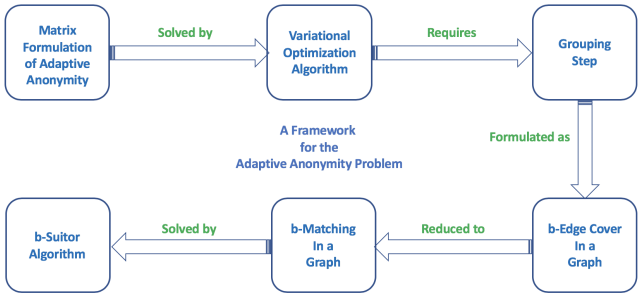


Fig. 2. Our framework for solving the adaptive anonymity problem.

problem. This problem arises in communication or distribution problems on networks where each communication node has to be *covered* several times to increase reliability in the event of a communication link failing [12]. The well-known k-nearest neighbor graph construction to represent noisy and dense data is also related to the *b-Edge Cover* problem. An exact algorithm for the *b-Edge Cover* problem can be obtained using an exact algorithm for computing a *b-Matching* of maximum weight, and the best known exact algorithm for *b-Matching* is due to Anstee, with time complexity $\min\{O(n^2m + nlog(\beta)\ (m + nlogn)), O(n^2log(n)(m + nlog(n)))\}$, where m and n are as defined earlier, and $\beta = \max b(v)$ [13], [14].

However the exact algorithm is not practical for solving large problems, and is also not amenable for parallelization on modern architectures. Recently a number of approximation algorithms have been developed for the minimum weighted *b-Edge Cover* problem in order to solve large problems in parallel. The authors in [15] developed a $3/2-$ approximate *Locally Subdominant Edge* algorithm (LSE) which computes the same edge cover as the GREEDY algorithm but is amenable for parallelization. The authors in [16] have described two 2-approximation algorithms, called *static LSE* (S-LSE) and *Matching Complement Edge cover* (MCE). The MCE algorithm is currently the fastest approximation algorithm and in practice computes a near-optimal *b-Edge Cover*. The algorithm uses a $1/2$-approximate b'*-Matching* algorithm called b-SUITOR to compute a *b-Edge Cover*. The b-SUITOR algorithm scales up to $50\times$ on shared memory machines with 60 cores, and up to $16K$ cores on a distributed memory machines [17], [18]. The parallel efficiency is the main motivation to use MCE algorithm for solving *adaptive anonymity* problem in this paper. One variant of the b-SUITOR algorithm also has the property that it does not require the whole graph in memory at one time in order to compute the solution. We use this property to reduce the memory complexity of the *adaptive anonymity* algorithm on shared memory machines.

III. A GENERALIZED FRAMEWORK

In this section we give a precise mathematical description of the problem and an algorithm that achieves *adaptive anonymity* and high utility of the shared data at the same time. We start with the definition of adaptive anonymity, which is generalized from k-anonymity proposed in [5].

Definition III.1. *A release of data is said to have the adaptive-anonymity property if the information for each individual v contained in the release cannot be distinguished from the information of at least $k(v) - 1$ individuals in the dataset.*

The difference between *adaptive anonymity* and k-anonymity is that the latter uses a uniform value k for all individuals instead of a value $k(v)$ for each individual v. For k-anonymity, the value of k has to be the maximum of $k(v)$ for all v to satisfy the privacy requirements. If there exists a user who would like their record to be confused with all others, in the k-anonymity setting, the obfuscated data will have little utility.

Our model is illustrated in Figure 1. We are given a dataset $X \in \mathbb{Z}^{n\times f}$, where n is the number of individuals and f is the number of features. Each row $x_v \in \mathbb{Z}^f$ of X is a contribution of the user v to the dataset and consists of f discrete features. A feature might be race, age, height, weight, income bracket, etc. A vector $\mathbf{k}$ of length n, where an element $k(v)$ of $\mathbf{k}$ is a privacy parameter of the v-th user is also given. The value $k(v)$ specifies that the data of the v-th user must be indistinguishable from that of $k(v)-1$ other users. The output of the algorithm is an anonymized dataset $Y \in (\mathbb{Z} \cup \{*\})^{n\times f}$, where the $*$ symbol indicates that a particular feature has been masked. Each feature vector $x_v \in X$ is associated with a username $x_v \in \mathbb{Z}$ and each row y_u of Y is associated with a key $y_u \in \mathbb{Z}$. Keys are output together with a matrix Y.

The sensitive information that needs to be hidden from an *adversary* (one who is trying to discover the identity of the users from the value of their features) is the matching between usernames x_v and corresponding keys y_u. We call this the *canonical matching*. Thus, the goal of the adversary is to reveal as many edges of the canonical matching as possible. The engine must publish data in such a way that instances with larger privacy parameter $k(v)$ have a smaller probability of being matched with the corresponding key by the adversary.

We say that feature vector $x_v \in X$ is *compatible* with vector $y_u \in Y$ if $x_v(l) = y_u(l)$ for every $1 \leq l \leq f$ such that $y_u(l) \neq *$. Hence y_u can be obtained from (confused with) x_v after masking some attributes of x_v. Thus either the feature values agree between the instances or we can match a '*' in the second vector to 0, 1, or * in the first.

In the suppression model analyzed here the goal is to mask as few attributes in X as possible to produce Y (to get as high utility of the published data as possible), but in such a way that each entry x_v of X can be confused with at least $k(v)$ rows y_u in Y. Thus we have the following measure of the utility of the presented scheme.

Definition III.2. *The utility of the suppression model is the ratio of the number of unmasked features and the total number of features, nf. The goal of the database algorithm producing obfuscated data is to minimize the number of masked features such that all privacy constraints are met.*

Given a dataset X, we define a function $\gamma(i, j, l)$ as follows,

$$\gamma(i,j,l) = \begin{cases} 1 & \text{if } X_{il} \neq X_{jl}, \\ 0 & \text{otherwise.} \end{cases}$$

For an undirected graph G with an adjacency matrix $\mathbf{G} \in \{0,1\}^{n \times n}$ and a dataset X, we define the *Hamming distance* $h(G)$ as

$$h(G) = \sum_i \sum_j \mathbf{G}_{ij} \sum_l \gamma(i,j,l).$$

Here $\mathbf{G}_{ij}$ is either zero or one.

Given a graph G, we compute an expression for the number of stars to put in the dataset. The maximum number of stars one can put is nf. Let us consider a node i in the graph G, and a column l in X. Now we find the rows j of column l which correspond to neighbors of the node i, i.e, $\mathbf{G}_{ij} = 1$. If every such element X_{jl} is equal to X_{il}, then we do not have to put a star in Y_{il}; otherwise we need to put a star in the position Y_{il}. Mathematically this can be expressed as follows.

$$s(G) = nf - \sum_i \sum_l \prod_j (1 - \mathbf{G}_{ij}\ \gamma(i,j,l)).$$

The second term in this equation counts the positions in the matrix Y where no stars are needed. If $X_{jl} = X_{il}$ then $\gamma(i,j,l) = 0$; if this is true for all neighbors j of node i, then every term in the product is 1, and then the value of Y_{il} is set to X_{il} and not a star.

Choromanski, Jebara and Tang proposed the following method [3] that finds a good-quality approximation of G. First the algorithm minimizes $h(G)$ over all graphs G satisfying the privacy requirements. Then a variational upper bound [19] on $s(G)$ is iteratively minimized with the use of the weighted version of the Hamming distance. The first phase of their algorithm solved the b-Matching problem exactly.

It is clear from the definition that there are two goals for solving the *adaptive anonymity* problem: group instances to satisfy privacy constraints and hide as little data as possible. Since optimal solution for this problem is NP-complete [3], our approximate solution comes from the observation that if we group similar instances together (w.r.t. their corresponding features) then we need to hide fewer features. Our proposed *adaptive anonymity* algorithm is shown in Algorithm 1.

Algorithm 1 Adaptive Anonymity ($X \in \mathbb{Z}^{n \times f}$, $\mathbf{k} \in \mathbb{Z}^{\mathbf{n}}$, ϵ)

Let $c_\epsilon = \log(\frac{\epsilon}{1+\epsilon})$
Initialize $W \in \mathbb{R}^{n \times f}$ to the all ones matrix
Initialize Y to X
while not converged **do**
    Let G be a weighted graph, where:
    $E_{ij} = \sum_l (W_{il} + W_{jl}) \gamma(i,j,l)$ ▷ Compute Graph
    C = *b-Edge Cover* $(G, E, \mathbf{k} - \mathbf{1})$ ▷ Grouping Step
    for all i, l **do** ▷ Update W
        $W_{il} \leftarrow exp(\sum_j C_{ij} \gamma(i,j,l)\ c_\epsilon)$
for all i, l **do**
    if $C_{ij} = 1$ and $X_{jl} \neq X_{il}$ for any j **then**
        $Y_{il} = *$
$Y_{public} = MY$, where M is a random row permutation of $\mathbb{B}^{n \times n}$

The algorithm is a variational optimization algorithm which iterates until some convergence criterion is met or a maximum number of iterations is reached. First, the algorithm creates a complete graph of n vertices corresponding to instances and a weight multiplier matrix initialized to all ones. Within an iteration, the algorithm assigns the weight of an edge between two vertices based on some dissimilarity measure between the two instances, multiplied by the weight multiplier. Next, the algorithm performs a *grouping* step based on the current weight assignment, and then the weight multipliers are adjusted based on the grouping.

A critical part of Algorithm 1 is how the *grouping* step is done. There are two requirements for the *grouping* step: i) each instance v has to be in a group with $k(v) - 1$ other instances, and ii) "similar" instances should be grouped together in order to minimize number of masked data elements. In order to achieve this goal, we use a *b-Edge Cover* formulation for the *grouping* step.

Definition III.3. *A b-Edge Cover in a graph is a subgraph C such that every vertex v has **at least** $b(v)$ edges incident on it in the subgraph. If the edges are weighted, then a cover that **minimizes** the sum of weights of its edges is a minimum weight b-Edge Cover.*

Given the definition of the *b-Edge Cover*, we group instances together with the following three steps:

1) Create a complete graph G where the instances are the vertices.
2) Calculate the edge weight between a pair of vertices using the dissimilarity measure between the corresponding

instances. The dissimilarity between two instances is the number of the features in which the instances do not agree.

3) Set $b(v) = k(v) - 1$ for each vertex, and solve a *b-Edge Cover* problem with the input (G, b).

Since *b-Edge Cover* is a minimization problem, it will group less dissimilar, i.e., more similar vertices together; each vertex v is grouped with $k(v) - 1$ other vertices. We use a 2-approximation algorithm called the MCE algorithm for the *b-Edge Cover* problem [16], [20]. we have compared the anonymizations obtained with the $3/2$-approx and 2-approx *b-Edge Cover* algorithms, LSE and MCE respectively, and found less than 1% difference in utility. The details of the MCE algorithm are explained the next section.

Now we provide more details of our *b-Edge Cover* based formulation of *adaptive anonymity*, in Algorithm 1. The algorithm starts by initializing the weight multiplier matrix $W \in \mathbb{R}^{n \times f}$, to all 1's. The matrix W associates a weight $W_{ij} \geq 1$ to each entry of input dataset X_{ij}, which the algorithm updates at each iteration. The algorithm iterates until the utility measure converges or a maximum number of iterations is reached. At each iteration, we compute edge weights in the graph as the weighted sum of the product of the weight multipliers and the dissimilarity between two instances. Then we group instances using a *b-Edge Cover* C in the graph and the $k(v)$ values. Finally, we update the weight multipliers based on the following rule: We proportionally increase the multiplier value associated with the feature l of the instance i, W_{il}, based on how many times the feature X_{il} differs with X_{jl}, the corresponding feature of other instances j grouped together with i. We increase the weight multiplier of a feature when it has the potential to create more maskings because we compute an approximate minimum weight *b-Edge Cover*. When the algorithm converges, we mask a feature if it does not agree with other values of the same feature in the same group. Then we publish a row-permuted copy of the masked data. The time complexity of Algorithm 1 per iteration is as follows: the *Compute Graph* step has complexity of $O(n^2 f)$; the *Update W* step has complexity of $O(nkf)$, where k is $max(k(v))$; and the *grouping Step* has time complexity $O(n^2 logn)$ if an exact *b-Edge Cover* algorithm is used. If a 2-approximate *b-Edge Cover* algorithm is used, the time complexity of the last step becomes $O(\beta m)$, where $\beta = max_{v \in V} b(v)$.

We discuss theoretical guarantees on the quality of the computed solution below. We will need the following lemma.

Lemma III.1. *For any graph G, the Hamming distance $h(G)$ and the number of stars $s(G)$ satisfy $s(G) \leq h(G) \leq ks(G)$, where $k = max_{v \in V} k(v)$.*

Proof. The first inequality is immediate since we need at most one star for each difference that contributes to the Hamming distance. We consider the contributions that an instance i and a feature l make to $h(G)$ and $s(G)$.

$$\begin{aligned} h(G) &= \sum_i \sum_j \mathbf{G}_{ij} \sum_l \gamma(i,j,l) = \sum_i \sum_l \mathbf{G}_{ij} \sum_j \gamma(i,j,l) \\ &\equiv \sum_i \sum_l h_{il}(G). \\ s(G) &= nf - \sum_i \sum_l \prod_j (1 - \mathbf{G}_{ij}\gamma(i,j,l) \\ &= \sum_i \sum_l (1 - \prod_j (1 - \mathbf{G}_{ij}\gamma(i,j,l))) \\ &\equiv \sum_i \sum_l s_{il}(G). \end{aligned}$$

Now we consider two cases. Case 1: If for every instance j such that $\mathbf{G}_{ij} = 1$, we have $\gamma(i,j,l) = 0$, then $h_{il}(G) = 0$ and $s_{il}(G) = 0$, and hence the inequality holds.
Case 2: There is some j such that $\mathbf{G}_{ij} = 1$ and $\gamma(i,j,l) = 1$. Then, considering the worst-case

$$h_{il}(G) = \sum_j \mathbf{G}_{ij}\gamma(i,j,l) \leq k.$$

Also $s_{il}(G) = 1$ since we need a star here. Hence

$$h_{il}(G) \leq k \leq k \cdot s_{il}(G).$$

Summing over all i and l, we obtain the lemma. □

Theorem III.1. *The first iteration of the while-loop in Algorithm 1 finds a b-Edge Cover C such that*

$$s(C) \leq \alpha k \min_{G \in \mathbb{B}^{n \times n}} s(G),$$

where the minimum in the expression is over all adjacency matrices satisfying privacy requirements, and α is the approximation ratio of the b-Edge Cover algorithm.

Proof. Initially the variational matrix W has all elements set to one, and hence in the first iteration the objective of the *b-Edge Cover* is proportional to the *Hamming distance* of G. Suppose $\hat{G} = \text{argmin } h(G)$; then $h(C) \leq \alpha\ h(\hat{G})$ since we have an approximation algorithm for *b-Edge Cover*. For any graph G, $s(G) \leq h(G)$ from the Lemma. Combining, we have $s(C) \leq h(C) \leq \alpha\ h(\hat{G})$. Now suppose $G^* = \text{argmin } s(G)$. Since $\hat{G}$ minimizes $h(G)$, we have $h(\hat{G}) \leq h(G^*)$. From the Lemma III.1, we have $h(G) \leq k\ s(G)$. Combining all these, $s(C) \leq h(C) \leq \alpha\ h(\hat{G}) \leq \alpha\ h(G^*) \leq \alpha k\ s(G^*)$. □

We illustrate the Grouping step by a *b-Edge Cover* by the example shown in Figure 3, with six instances and six binary features. Each user expects 2-anonymity, i.e., each user wants to be confused with at least one other user. The anonymity algorithm computes a *b-Edge Cover* with $b = 1$ for each node. Given the input data, the anonymity algorithm first constructs a dissimilarity matrix S. Each row of the matrix defines the dissimilarity between that instance and all other instances in the input. For example, the second entry of the first row denotes the dissimilarity between user U_1 and U_2 which is 2, because the instances disagree in features f2 and f4. This dissimilarity matrix acts as the input adjacency matrix

Instances	f1	f2	f3	f4	f5	f6
U_1	1	0	1	0	1	0
U_2	1	1	1	1	1	0
U_3	0	1	0	1	0	1
U_4	0	0	0	0	0	1
U_5	1	1	0	0	0	0
U_6	1	1	0	0	0	1

S	U_1	U_2	U_3	U_4	U_5	U_6
U_1	-	**2**	6	4	3	4
U_2		-	4	6	3	4
U_3			-	**2**	4	2
U_4				-	3	2
U_5					-	**1**
U_6						-

Instances	f1	f2	f3	f4	f5	f6
U_1	1	*	1	*	1	0
U_2	1	*	1	*	1	0
U_3	0	*	0	*	0	1
U_4	0	*	0	*	0	1
U_5	1	1	0	0	0	*
U_6	1	1	0	0	0	*

Fig. 3. An example for *adaptive anonymity*. From top to bottom: original input, dissimilarity matrix (Hamming distances) and anonymized output.

to the *b-Edge Cover* algorithm where each entry refers to the weight of an edge. The bold-font entries are the edges included in the *b-Edge Cover*. We see that grouped pairs are: (U_1, U_2), (U_3, U_4) and (U_5, U_6). Next the anonymity algorithm uses this grouped output to mask entries in the following manner: for each pair, it finds the dissimilar features and mask those features with a $*$. For example, U_5 and U_6 are grouped and the instances do not agree on feature f6, so the algorithm puts $*$ in the corresponding f6 entries. As we can see, there are $6 \times 6 = 36$ entries and after one iteration the algorithm masks 10 entries. Thus the utility at this iteration is $1 - 10/36 = 0.722$, i.e., 72%.

An important feature of our framework, specifically in the shared memory context, is that the memory requirement of Algorithm 1 is linear in the number of instances, whereas a state-of-the-art algorithm [3] requires quadratic memory. This significant reduction comes from an interesting property of the MCE algorithm for solving the *b-Edge Cover* problem.

IV. *b-Edge Cover*: THE MCE ALGORITHM

In this section, we discuss the details of the *Matching Complement Edge cover* (MCE) algorithm.

Definition IV.1. *A b-Matching in a graph is a subgraph M such that every vertex v has **at most** $b(v)$ edges incident on it in the subgraph. If the edges are weighted, then a matching that **maximizes** the sum of weights of its edges is a maximum weight b-Matching.*

An optimal algorithm for the minimum weight *b-Edge Cover* problem can be obtained by computing a maximum weight *b′-Matching*, by the following three step procedure [14]:

1) For each vertex v, compute $b'(v) = \deg(v) - b(v)$, where $\deg(v)$ is the degree of v, the number of edges incident on v.
2) Compute M_{opt}, a maximum weight *b′-Matching*.
3) A min weight *b-Edge Cover* is the complement of the matching: $C_{opt} = E \setminus M_{opt}$.

In this construction, steps 1 and 3 ensure that the computed *b-Edge Cover* is a valid cover, and the optimality of the cover depends on step 2. If we compute an approximate *b′-Matching*, keeping steps 1 and 3 fixed, then the solution to the *b-Edge Cover may not necessarily* be an approximate solution for *b-Edge Cover*. However, the authors in [16] showed that if the *b′-Matching* is computed using the b-SUITOR algorithm then the corresponding *b-Edge Cover* will satisfy 2-approximation bounds.

Algorithm 2 MCE(G, b)

1: $EC = \emptyset$
2: **for** $v \in V$ **in parallel do**
3: $\quad b'(v) = max\{0, \deg(v) - b(v)\}$
4: M=Parallel_b-SUITOR(G, b')
5: **for** $v \in V$ **in parallel do**
6: $\quad EC = EC \cup \{N(v) \setminus M(v)\}$
7: **return** *b-Edge Cover* EC

Since b-SUITOR is an essential part of the MCE algorithm, we briefly describe a variant of it in Algorithm 3. For more details, we refer the reader to the papers [18], [17]. The b-SUITOR algorithm is derived from the SUITOR algorithm for maximum weighted matching [21]. The algorithm is based on vertices making proposals to each other, just as in the Stable Matching problem. Vertices can propose in any order, but each vertex must propose to its current heaviest *eligible* neighbor. A vertex v is an *eligible neighbor* of a vertex u if v does not already have a proposal of higher weight from another neighbor of v. A vertex u can also annul the proposal made by a vertex w to a mutual neighbor v, if the weight of the edge (u, v) is higher than the weight of (v, w). In this case, u proposes to v, and annuls the proposal (v, w); now w must propose to its next heaviest eligible neighbor. An edge is matched when two vertices propose to each other. Since we can annul proposals, any vertex can make proposals thus increasing the parallelism.

The parallel b-SUITOR algorithm is shown in Algorithm 3. The algorithm maintains a queue Q of vertices whose $b(v)$ values are not satisfied yet, for which it tries to find partners during the current iteration of the **while** loop; and also a queue of vertices Q' whose proposals are annulled in this iteration, and will be processed again in the next iteration. (This is what "delayed" means; annulled vertices are not processed in the same iteration. "Partial" means that the adjacency lists are partially sorted to find a subset of heaviest neighbors.) The algorithm then seeks a partner for each vertex u in Q in parallel. It tries to find $b(u)$ proposals for u to make while the adjacency list $N(u)$ has not been exhaustively searched thus far in the course of the algorithm.

Algorithm 3 Parallel_b-SUITOR(G, b)

```
1: Q = V; Q' = ∅;
2: S(v) = ∅, min-priority heap ∀v
3: while Q ≠ ∅ do
4:     for vertices u ∈ Q in parallel do
5:         i = 1;
6:         while i <= b(u) and N(u) ≠ exhausted do
7:             Let p ∈ N(u) be an eligible partner of u;
8:             if p ≠ NULL then
9:                 Lock S(p);
10:                if p is still eligible then
11:                    i = i + 1;
12:                    Add u to S(p);
13:                    if u annuls the proposal of v then
14:                        Add v to Q'; Update db(v);
15:                        Remove v from S(p);
16:                Unlock S(p);
17:            else
18:                N(u) = exhausted;
19:    Update Q using Q'; Update b using db;
20: return S
```

Consider the situation when a vertex u has $i-1 < b(u)$ outstanding proposals. The vertex u can propose to a vertex p in $N(u)$ if it is a heaviest eligible neighbor in the set $N(u)$ and if the weight of the edge (u, p) is greater than the lowest offer that p has. In this case, p would accept the proposal of u rather than its current lowest offer.

If the algorithm finds a partner p for u, then the thread processing the vertex u attempts to acquire the lock for the priority queue $S(p)$ so that other vertices do not concurrently become Suitors of p. This attempt might take some time to succeed since another thread might have the lock for $S(p)$. Once the thread processing u succeeds in acquiring the lock, then it needs to check again if p continues to be an eligible partner, since by this time another thread might have found another Suitor for p, and its lowest offer might have changed. If p is still an eligible partner for u, then we increment the count of the number of proposals made by u, and make u a Suitor of p. If in this process, we dislodge the last Suitor x of p, then we add x to the queue of vertices Q' to be processed in the next iteration. Finally the thread unlocks the queue $S(p)$.

We fail to find an eligible partner p for a vertex u when we have exhaustively searched all neighbors of u in $N(u)$, and none offers a weight greater than the lowest offer u has. In this case u will have fewer than $b(u)$ matched neighbors. After we have considered every vertex $u \in Q$ to be processed, we can update data structures for the next iteration. We update Q to be the set of vertices in Q'; and the vector b to reflect the number of additional partners we need to find for each vertex u using $db(u)$, the number of times u's proposal was annulled.

The time complexity of the b-SUITOR algorithm can be described in the depth-work model [16]. Its parallel depth (number of steps in the parallel algorithm) is $\log \Delta \log b'(V)$, and its work (total number of operations performed by all processors) is $O(\beta' b'(V)) = O(\beta' m)$, which is linear in the number of edges. Here $b'(V) = \sum_{v \in V} b'(v)$, $\beta' = max_{v \in V} b'(v)$, and Δ is the maximum degree of a vertex.

V. PARALLEL IMPLEMENTATION OF *adaptive anonymity*

In this Section we discuss shared memory and distributed memory implementations of the *adaptive anonymity* algorithm. Referring to Algorithm 1, the compute graph (line 6) and the Update W (line 9) steps are pleasingly parallel, and independent in terms of the input instances. These two steps usually take less than 15% of the total execution time, and therefore the main challenge for the scalability of the algorithm stems from the scalability of the *Grouping* step, i.e., the MCE algorithm. In turn, the scalability of the MCE algorithm depends on the performance of the b-SUITOR algorithm. The shared memory as well as distributed memory performance of MCE and b-SUITOR algorithms have been studied earlier in [18], [17], [16]. Hence, the *basic* shared memory and distributed memory implementations of the *adaptive anonymity* algorithm are obtained from these papers. However, we discuss optimizations for shared memory and distributed memory implementations for anonymity computations.

A. Input Preprocessing

We process the input datasets in accordance with standard practice [3]. As an example, we briefly describe the processing of features for the CMS dataset: First, we replace all the uniquely identifiable features, e.g., physician's name, and medical license number, with unique random strings and we do not use these features further in the algorithm. Second, we group the range of values of a continuous feature, e.g., amount of payment, into deciles, i.e., in groups of $0-10\%$, $10-20\%$ and so on; and then replace each feature value with the median value of the group it belongs to. Finally, we consider all features, i.e., originally categorical features (e.g., medical specialty, method of payment, medical device type), and continuous-valued features that are processed to be categorical (e.g., amount of payment) together, and binarize them using one-hot encoding, a widely used encoding in machine learning. For example, the categorical feature "medical specialty", has 9 categories such as Gynecology, Cardiology, etc. The one-hot encoding of the "medical specialty" feature is a bitstring of 9 bits where each bit represents a particular medical specialty. After binarization, each instance becomes a long concatenated bitstring of all the features as shown in Figure 3. There are a few advantages to the binarized form: i) we compress the processed binarized input by using bitfields to store binary values and store them as a binary file as opposed to text, which saves memory; ii) the bitfield representation speeds up the computation of the dissimilarity matrix by using logical operations instead of comparisons; and iii) after we anonymize the data, it is trivial to map it back from the binarized to the textual/numeric form since there is one-to-one mapping of categories of a feature to one-hot-encoding.

B. Linear Memory Formulation

An important experimental contribution in the context of shared memory machines is the linear memory formulation of the privacy problem. Referring to Algorithm 1, from the

instance-feature matrix X, we can generate the full dissimilarity matrix which requires $O(n^2)$ space. On the other hand, we can save space by generating each element in the dissimilarity matrix when it is needed "*on the fly*" with $O(f)$ computation (recall f is the number of features). Since the dissimilarity represents the input edge weights to MCE algorithm, and MCE has time complexity linear in the number of edges, the total computational complexity of the *adaptive anonymity* algorithm becomes $O(m\Delta f) = O(n^2\Delta f)$ per iteration. That is, the full generation of the dissimilarity matrix requires $O(n^2)$ memory but no additional re-computation of weights. On the other hand, "*on the fly*" does not require any explicit memory allocation but the computation of weights becomes prohibitively expensive. So we need a strategy which is in between these extremes, i.e., one that generates a smaller submatrix of E at any time, and computes the edge cover using these submatrices. The impact of the partial generation of the dissimilarity matrix is two-fold: it allows the user to specify the part size to fit in available memory, and to solve larger problems which would be infeasible otherwise.

The question of partial generation of the dissimilarity matrix boils down to whether the underlying *Grouping* step can work with a partial set of edges and their weights or not. We use the MCE algorithm for this step, which in turn uses the b-SUITOR algorithm. By its design, the "*Delayed Partial*" version of b-SUITOR algorithm can work with a subgraph (Algorithm 3) as we have pointed out in Section IV. (Once the edges in the adjacency set of a vertex are exhausted, if the value of $b'(v)$ is not satisfied yet, more edges will be generated in a subsequent step.) We generate a constant number of elements for each row of the matrix E, and thus the b-SUITOR algorithm reduces the space complexity of *adaptive anonymity* problem from $O(n^2)$ to $O(n)$.

C. Distributed Memory Implementation

There are three major data structures for the *adaptive anonymity* algorithm: input dataset $X \in \mathbb{Z}^{n\times f}$, weight matrix, $W \in \mathbb{Z}^{n\times f}$ and the dissimilarity matrix $E \in \mathbb{Z}^{n\times n}$. In our first implementation, we partitioned all three data structures across the compute nodes, partitioned by instances. Assuming there are p compute nodes, each node i contains: $X_i \in \mathbb{Z}^{n/p\times f}$, $W_i \in \mathbb{Z}^{n/p\times f}$ and $E_i \in \mathbb{Z}^{n/p\times n}$. However, this strategy requires significant communication during the *Compute graph* and the *update W* steps in Algorithm 1. So, we partitioned only E among the compute nodes, since it has $O(n^2)$ memory complexity, but not X and W, because both of them have $O(n)$ space complexity. We partition E by rows, and $\tau(E_i)$ represents the rows of E owned by compute node i. Thus we avoid communication altogether during the *Compute Graph* step. The *Grouping* step, which is handled by the MCE algorithm and in turn by the b-SUITOR algorithm, uses graph weights derived from the distributed dissimilarity matrix E. We follow the strategies described earlier in [17] to implement b-SUITOR on distributed memory multiprocessors. Finally, in the *Update W* step, each compute node i updates the rows r of W, where $r \in \tau(E_i)$, using the output C_i from the *Grouping* step. Each node then broadcasts its updates to all other nodes so that in the next iteration, the *Compute Graph* step has the full matrix W consistent across all nodes. To avoid the communication bottleneck of each node broadcasting all of its local updates, we broadcast the updates in fixed size batches asynchronously, and overlap them with the computation in order to hide latency. The pseudo-code for the *Update W* step in shown in Algorithm 4.

Algorithm 4 Update W

```
1: for all r ∈ τ(E_i) in parallel do
2:     for all l do
3:         W_rl ← exp(Σ_j C_rj γ(i, j, l) c_ε)
4:     Add W_r to send_buffer
5:     if send_buffer is full then
6:         MPI_IBcast(send_buffer)
7:     if Received new update then
8:         MPI_IRecv(recv_buffer)
9:         Update W with recv_buffer
```

VI. EXPERIMENTS AND RESULTS

We conducted our experiments on Cori, a Cray XC40 supercomputer at NERSC, Berkeley. Each node on Cori consists of two 16-core 2.3 GHz Intel E5-2698 (Haswell) processors with 128 GB RAM. Each core in a node has its own 64 KB L1 cache and 256 KB L2 cache, as well as a 40 MB shared L3 cache per socket. Cori nodes are also interconnected with the Cray Aries network using the Dragonfly topology.

We used the *Intel MPI* implementation for inter-node communication and *OpenMP* for intra-node multi-threading, and compiled the code with the built-in compiler wrapper optimized for the system, *CC-2.5.12* with the flags `-O3 -qopenmp`. Our hybrid implementation used the following MPI-openMP settings: one OpenMP process for each of the 32 cores on a node, and one MPI process per node. Hyperthreading did not improve the performance of our code.

We consider eight datasets for *adaptive anonymity* experiments in Table I. We use four small datasets in our experiments to compare an adaptive anonymity algorithm by Choromanski et al [3], which groups individuals using belief propagation and exact b-matching algorithms, with our approximate b-edge cover approach. For each of these problems, these authors picked a specific privacy requirement k that varies with each data item, and we use the same values to be consistent with their work. The values for k ranged from 2 to 10. The earlier belief propagation algorithm is not capable of solving the larger problems in our test set. We consider three larger problems from a Machine Learning Repository at University of California, Irvine [22], and one from the Centers for Medicare and Medicaid Services [2], to demonstrate that the approximate b-edge cover approach can solve them. We generated $b(v)$ values for each problem as the minimum of the degree of a vertex and a uniform random integer between one and the square root of the number of vertices. For each experiment we repeat the computations three times and report the average runtime. The utility for each problem is invariant since the same b-edge cover is computed. Our algorithm terminates if three consecutive iterations do not show any improvement in the

TABLE I
PROBLEM SETS FOR *adaptive anonymity*.

Problem	Instances	Features
Caltech36	768	101
Reed98	962	139
Haverford76	1,446	145
Simmons81	1,518	140
UCI_Adult	32,561	101
USCensus1990	158,285	68
Poker_hands	500,000	95
CMS	745,280	512

utility measure. For smaller problems the algorithm terminates within four iterations and for the larger problems it takes three to nine iterations. However, most of the feature masking occurs in the first iteration. For example, the *CMS* dataset achieves a utility of 79.3% in 6 iterations, whereas the utility after the first iteration is 70.9%.

TABLE II
COMPARING RUN TIMES OF THE *Belief Propagation (BP)* AND MCE ALGORITHMS ON A SINGLE THREAD OF AN INTEL HASWELL.

Problem	BP	MCE	Rel. Perf.	Utility Diff (%)
Caltech36	13m 15s	10s	80	-0.85
Reed98	20m 47s	22s	57	-0.32
Haverford76	1h 07m	55s	73	0.23
Simmons81	59m 29s	45s	79	-0.81

A. Shared Memory Results

In Table II, we compare the run-times of the belief propagation algorithm (BP) [3], with the MCE based algorithm. BP algorithms are well-known in Machine Learning, and have been used to solve a variety of problems including graph matching [23]. We observe that the MCE algorithm is 55 to 80 times faster than the BP algorithm. The last column of Table II shows that *b-Edge Cover* also achieves this improvement without compromising the utility, since the differences are less than 1%. We proceed to describe results from our parallel algorithm that can solve large problems, which are not feasible for the BP algorithm.

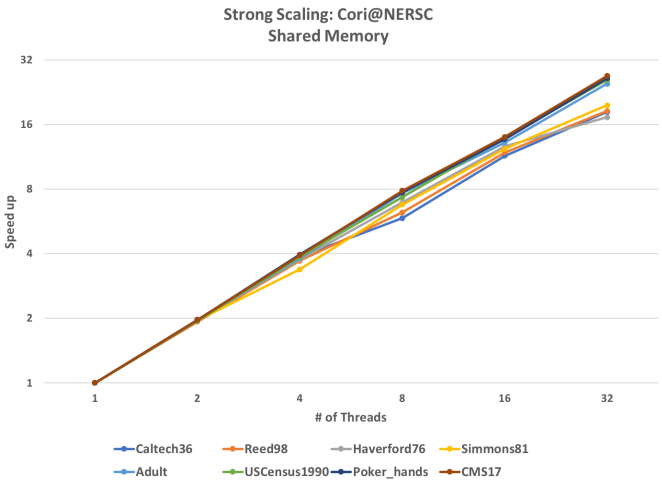


Fig. 4. Strong scaling of *adaptive anonymity* problems on 32 cores of an Intel Haswell processor.

Next we show the impact of the linear memory formulation of the *adaptive anonymity* problem in the shared memory context using 32 cores on one node of Cori. We consider four larger problems to illustrate the effect of trading computation for space. The problems `Poker_hands` and `CMS` have roughly $500K$ and $750K$ instances. Assuming each entry of E is a 4 byte integer, storing the full matrix E would require approximately 1TB and 2TB of memory, respectively, but our machine only has 128GB of memory. Hence we cannot solve these problems with an algorithm that needs the entire dissimilarity matrix for computations. For these problems, we randomly generate k values between $k^l = 5$ and $k^h = 100$ for privacy parameters. We partially generate the dissimilarity matrix using the following strategy: for each row, we generate $t * k^h$ entries, with $t \in \{8, 32, 128\}$, yielding a total memory requirement of $t * k^h n$ for n rows in the dissimilarity matrix E. We summarize the results in Table III. The fastest result for each problem is indicated in bold font. We observe that "*on the fly*" computation, i.e, no storage for the matrix, is significantly slower than using optimal part sizes. We mask less than 25% of the data elements for these problems. Our linear memory formulation also shows the adaptability of our algorithm in terms of memory constraints. A user can easily choose a size factor t dependent on the memory available, and provide it as an input to the algorithm.

Next we show the strong scaling performance of the *adaptive anonymity* algorithm that uses the MCE algorithm on all eight problems in Figure 4. For the larger four problems, we use the best part sizes from Table III. We observe that the algorithm achieves a speedup of $27\times$ on 32 threads for the larger problems. The smaller problems do not scale well beyond 16 threads because the amount of work available per thread is small, and cannot offset the NUMA costs.

B. Distributed Memory Results

We report the strong and weak scaling performance of our algorithm on the three largest problems. The distributed memory implementation does not employ the linear memory formulation, since it is not memory-constrained as the shared memory implementation is. The implementation uses a hybrid strategy, i.e., each compute node is assigned one *MPI* task for inter-node communication and each node uses 32 *OpenMP* threads for parallel computation. In Figure 5, we report the strong scaling performance of our algorithm. An ideal speed-up curve is plotted so that the reader could compare its slope with the observed slopes of the three problems. We observe that the algorithm scales well up to $8,192$ cores, and that initially some problems exhibit super-linear speed-up. This is due to the smaller memory available on fewer processors. For example, `Poker_hands` roughly requires 1 TB of memory for the data and data structures, but 8 compute nodes (256 cores) have only 1 TB memory in total. Since the operating system requires some memory as well, the problem does not fit in the memory. Hence it is likely that many memory swaps occur, slowing the code for fewer processors. The `UCI_Adult` problem strongly scales to $1,024$ cores but not beyond it, due to its small size.

TABLE III
EFFECT OF WORKING SET MEMORY ON RUNTIMES USING THE MCE ALGORITHM ON 32 CORES OF AN INTEL HASWELL PROCESSOR.

Problems	on the fly	$8 * k^h n$	$32 * k^h n$	$128 * k^h n$	Utility
UCI_Adult	1m 48s	18s	**15s**	20s	90.4%
USCensus1990	3h 47m	**13m**	19m	22m	87.1%
Poker_hands	>24h	51m	**46m**	1h 02m	81.3%
CMS	>24h	**6h 11m**	6h 48m	6h 16m	79.3%

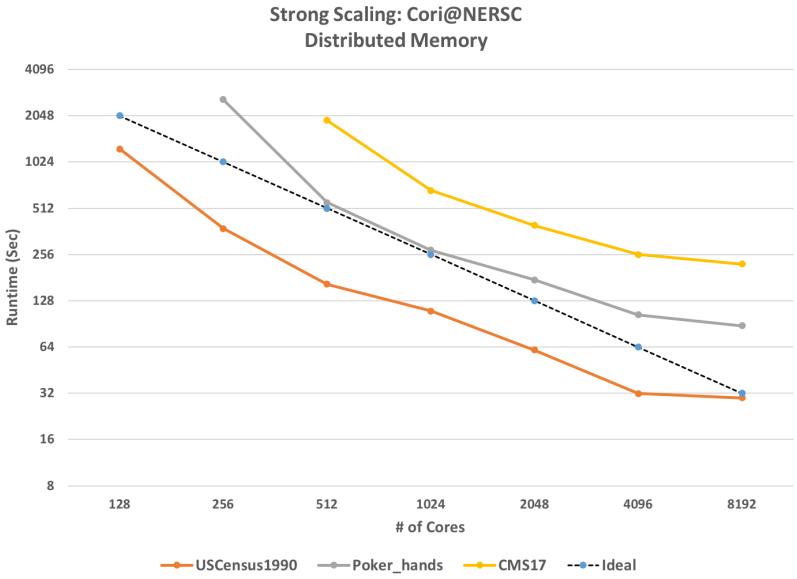


Fig. 5. Strong scaling of *adaptive anonymity* algorithm on Cori using our hybrid MPI-OpenMP code.

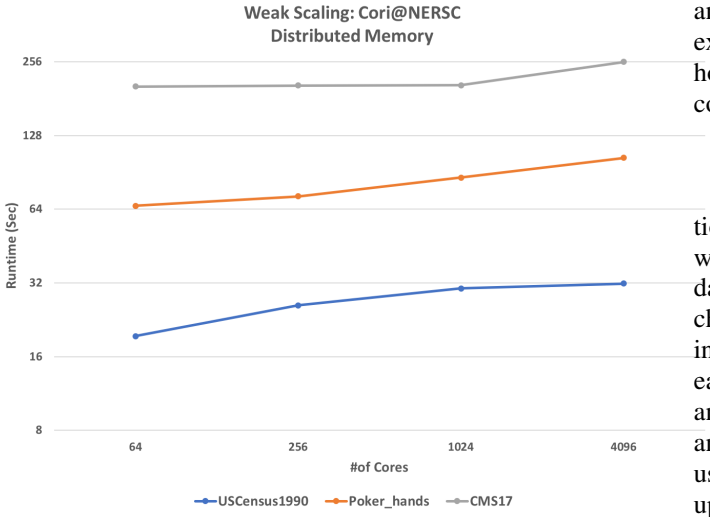


Fig. 6. Weak scaling of *adaptive anonymity* algorithm on Cori.

The *adaptive anonymity* problem has quadratic memory complexity in the number of instances,and hence for each problem we randomly pick 12.5%, 25%, 50% and 100% of all instances and run on 64, 256, 1024 and 4096 cores, respectively. We repeat the process three times for each problem and the report the average run-times in Figure 6. Our algorithm exhibits reasonably good weak scaling as the curves are nearly horizontal, implying that it could potentially scale beyond $8K$ cores with larger problem sizes.

VII. CONCLUSIONS

We have presented a scalable algorithm for the anonymization of large datasets that guarantees privacy requirements while achieving high utility of the published obfuscated datasets. These transformed datasets can be used for machine learning purposes which do not have to be specified in advance. Our technique handles datasets infeasible for earlier methods by significantly reducing the runtime of the anonymization procedure as well as its memory requirements, and it works well in the heterogeneous setting where different users require different levels of privacy. The algorithm scales up to $8K$ cores on a distributed memory machine, and it can solve problems with $700K$ instances and a hundred features in under five minutes instead of days.

ACKNOWLEDGEMENTS

This research was supported by NSF grant CCF-1637534; DOE grant ASCR DE-SC001020; and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. DOE Office of Science and the NNSA.

REFERENCES

[1] K. E. Emam and F. K. Dankar, "Protecting privacy using k-Anonymity," *Journal of the American Medical Informatics Association*, vol. 15, no. 5, 2008.

[2] "Centers for Medicare & Medicaid Services," https://www.cms.gov/OpenPayments/About/Resources.html, Accessed: 2018-02-15.

[3] K. Choromanski, T. Jebara, and K. Tang, "Adaptive anonymity via b-matching," in *27th Annual Conference on Neural Information Processing Systems (NIPS)*, 2013, pp. 3192–3200.

[4] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[5] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[6] X. Xiao and Y. Tao, "Personalized privacy preservation," in *ACM SIGMOD*, 2006, pp. 229–240.

[7] M. Xue, P. Karras, C. Raïssi, J. Vaidya, and K. Tan, "Anonymizing set-valued data by nonreciprocal recoding," in *ACM SIGKDD*, 2012, pp. 1050–1058.

[8] G. Cormode, D. Srivastava, S. Bhagat, and B. Krishnamurthy, "Class-based graph anonymization for social network data," *PVLDB*, vol. 2, no. 1, pp. 766–777, 2009.

[9] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang, "Anonymizing bipartite graph data using safe groupings," *VLDB Journal*, vol. 19, no. 1, pp. 115–139, 2010.

[10] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 2006, pp. 24–24.

[11] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *IEEE 23rd International Conference on Data Engineering (ICDE)*. IEEE, 2007, pp. 106–115.

[12] G. Kortsarz, V. Mirrokni, Z. Nutov, and E. Tsanko, "Approximating minimum-power degree and connectivity problems," in *Proceedings of 8th Latin American Theoretical Informatics Conference (Lecture Notes in Computer Science)*, vol. 4957, 2008, pp. 423–435.

[13] R. P. Anstee, "A polynomial algorithm for b-matchings: An alternative approach," *Inf. Process. Lett.*, vol. 24, no. 3, pp. 153–157, Feb. 1987.

[14] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency. Volume A: Paths, Flows, Matchings*. Springer, 2003.

[15] A. Khan and A. Pothen, "A new 3/2-approximation algorithm for the b-edge cover problem," in *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, 2016, pp. 52–61.

[16] A. Khan, A. Pothen, and S. M. Ferdous, "Parallel algorithms through approximation: b-edge cover," in *Proceedings of IEEE Parallel & Distributed Processing Symposium (IPDPS)*, 2018, pp. 22–33.

[17] A. Khan, A. Pothen, M. M. Patwary, M. Halappanavar, N. Satish, N. Sundaram, and P. Dubey, "Designing scalable b-matching algorithms on distributed memory multiprocessors by approximation," in *Proceedings of ACM/IEEE Supercomputing*, 2016, pp. 773–783.

[18] A. Khan, A. Pothen, M. M. Patwary, N. Satish, N. Sundaram, F. Manne, M. Halappanavar, and P. Dubey, "Efficient approximation algorithms for weighted b-matching," *SIAM Journal on Scientific Computing*, pp. S593–S619, 2016.

[19] M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, no. 2, pp. 183–233, 1999.

[20] S. M. Ferdous, A. Khan, and A. Pothen, "New approximation algorithms for minimum weighted edge cover," in *SIAM Workshop on Combinatorial Scientific Computing (SIAM CSC)*, 2018.

[21] F. Manne and M. Halappanavar, "New effective multithreaded matching algorithms," in *28th IEEE International Parallel and Distributed Processing Symposium*, 2014, pp. 519–528.

[22] K. Bache and M. Lichman, "UCI Machine Learning Repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[23] S. Sanghavi, D. Malioutov, and A. Willsky, "Belief propagation and LP relaxation for weighted matching in general graphs," *IEEE Transactions on Information Theory*, vol. 57, no. 4, pp. 2203–2212, 2011.