A Framework for Optimization in Big Data: Privacy-preserving Multi-agent Greedy Algorithm

Taeho Jung[1], Xiang-Yang Li[1 2] *, and Junze Han[1]

[1] Department of Computer Science, Illinois Institute of Technology
[2] Department of Computer Science and Technology, TNLIST, Tsinghua University

Abstract. Due to the variety of the data source and the veracity of their trustworthiness, it is challenging to solve the distributed optimization problems in the big data applications owing to the privacy concerns. We propose a framework for distributed multi-agent greedy algorithms whereby any greedy algorithm that fits our requirement can be converted to a privacy-preserving one. After the conversion, the private information associated with each agent will not be disclosed to anyone else but the owner, and the same output as the plain greedy algorithm is computed by the converted one. Our theoretic analysis shows the security of the framework, and the implementation also shows good performance.

1 Introduction

Many optimization problems in real world are very challenging, but they are of great usefulness at the same time due to the data-driven decision making in the Business Intelligence (BI). Job scheduling problems in any network or operating system, profit maximization problems in any resource-bounded environment or cost minimization problems in deadline-constrained cases are good examples. These problems are usually modeled as classic optimization problems (*e.g.*, Knapsack problem, Minimum Spanning Tree problem, Weighed Set Cover problem, Travelling Salesman problem) whose solutions are widely known, and the optimum solutions are derived correspondingly. Because data comes from multitudes of parties in the big data, the problem to be solved is often a distributed optimization, and a distributed greedy algorithm is desired, whose input comes from different users. However, to have it working correctly, various information needs to be gathered to make decisions at each iteration, and such information is related to various private data in many real-life problems, which makes users reluctant to provide the necessary input data [1–3]. Naturally privacy implication emerges and the following non-trivial job is expected: same solution needs to be derived without having access to raw sensitive information.

In this paper, we start from the observation that there exists a huge class of problems that can be solved or approximated by multi-agent greedy algorithms, which we denote as *greedy-class problems*. In such algorithms, each

* The research is partially supported by NSF ECCS-1247944 and NSF CMMI 1436786.

agent possesses an instance which may or may not be selected in the optimum/approximated solution (*e.g.,* the bundle in WDCA), and the algorithms determine the final solution based on all the data (description of the instance, relevant information about the instance *etc*) provided by multiple agents. Instead of presenting privacy-preserving mechanism for every greedy-class problem, we propose a general framework to convert multi-agent greedy algorithms to privacy-preserving ones such that agents' privacy leakage is properly protected.

The main challenge comes from the local decision making at each iteration in greedy algorithms. Decision must be made based on certain *weight* information associated with agents' instances, but the algorithm should not have access to those information due to privacy concerns. We present three novel techniques to solve this challenge, and as long as the greedy algorithm fulfills our requirements, it can be converted to a privacy-preserving one via our framework. Finally, our contributions are summarized below:

1. We propose a general solution for multi-agent greedy algorithms which keeps agents' private information secret. Three novel techniques are presented to realize the privacy-preserving computation.
2. The privacy-preserving greedy algorithm generated by our framework achieves the same result as the original greedy algorithm.
3. Based on the framework, we give a uniform definition of the privacy for multi-agent greedy algorithms, and we prove that the framework does not breach the privacy.
4. Our framework enables various useful greedy algorithms in distributed optimization problems where users are reluctant to disclose their private information.

2 Related Work

2.1 DCOP, DCS and Our Greedy-Class Problem

Distributed Constraint Optimization Problem (DCOP, [4]) is similar to our optimization problem. The only difference is that DCOP's objective function is the sum of each agent's private cost (i.e., weight in this paper) while our objective function is any function of it. Universal solutions for any Distributed Constraint Optimization Problem (DCOP) or any Distributed Constraint Satisfaction (DCS) have long been a hot research topic [5,6], but much less attention is paid to the privacy concerns when compared to other aspects. Yokoo *et al.* [7] presented approaches to the DCS problem with cryptographic techniques, but their methods rely on external servers which may not be always available. Numerous works [8–10] discussed the DCS with privacy enforcement, and finally Modi *et al.* proposed ADOPT [4], which is a complete solver. However, those works suffer from high communication complexity because they all rely on a DFS tree which depicts the constraints relationship between agents, and the total number of messages per agent grows exponentially as the number of agents grows. To the best of our knowledge, [11] is the only work which proposes a general solution to the DCOP within a polynomial time based on a BFS tree, but they assume each agent is not aware of the system's topology, otherwise privacy is not preservable.

The contribution of this paper is prominent compared to above works. First of all, the DCOP is the special case of the greedy-class problem investigated in this paper. Secondly, our framework converts any distributed greedy algorithm such that the converted algorithm returns the final solution within a polynomial time regarding the number of agents. Finally, although we also assume a special organization of the agents in advance (according to [12]), awareness of this organization does not breach privacy in our work.

2.2 Privacy-Preserving Computation

Secure Multi-party Computation (SMC, [13]) is a generic solution for the privacy-preserving computation, in which n parties jointly and privately compute any function $f_i(x_1, x_2, \cdots, x_n) = y_i$, where x_i is the input of the i-th party and y_i is the output returned only to him. Each party i knows nothing but y_i. Since SMC evaluates any function in a privacy-preserving manner, it can be directly used to solve the distributed greedy-class problem in theory. However, it suffers from a high computation and communication complexities because of the garbled circuits [14] and the oblivious transfer [15]. Both complexities are exponential to the input length with large hidden constant factors.

Homomorphic Encryption (HE) is another common solution to the privacy-preserving computation actively used in academia [16–19]. It allows direct additions and multiplications on ciphertexts while preserving their decryptability. For example, $E(m_1) * E(m_2) = E(m_1 \hat{*} m_2)$ where $E(m)$ is the ciphertext of m and $*, \hat{*}$ stand for various operations (addition, multiplication etc).

3 Backgrounds & Problem Formulation

3.1 Big Data Greedy-Class Problems

We discuss the optimization problems which we denote as *greedy-class problems* in this paper because they are solved or approximated by a greedy algorithm. A big data greedy-class problem $P = (I, D, d(\cdot), f(\cdot), l(\cdot))$ is a problem which:

1. has a set of instances $I = \{i_1, \cdots, i_n\}$, and the final solution set $\hat{S} \subseteq I$.
2. has an information set D to be associated with the instances.
3. has a mapping $d(i)$ associates private information to an instance i.
4. has a real-value objective function $f(S)$ to be optimized.
5. has a feasibility function $l(S)$ to check whether a set of instances $S \subset I$ is feasible, *i.e.,* satisfying the constraint of the problem.

3.2 Adversary & System Model

We consider two models in this paper: **agent-authority** model and **all-agents** model. In the agent-authority model, two entities participate in the problem solving: a central authority and a group of **non-cooperative** agents. Each agent $a_j \in \{a_0, \cdots, a_{n-1}\}$ holds his instance i_j and the corresponding private information $d(i_j)$. If an agent has more than one instance, we assume the agent controls a virtual agent for each of his instances ([20]). In the agent-authority model, the central authority is supposed to receive only the global solution set $\hat{S}$ of the problem, and agents will not learn $\hat{S}$. In the all-agents model, each a_j will receive his local solution set $\hat{S}_j$ indicating whether his instance is contained in the global one $\hat{S}$, and no one in the system learn $\hat{S}$.

We assume semi-honest adversaries in this work. That is, the *honest-but-curious* agents and the central authority follow the protocol specification in general, but they are interested in others' information and try to harvest them. That is, agents try to infer the final solution set S as well as other agents' private information, and the central authority tries to infer each agent's private information associated with the instances. Also, we assume that it is computationally intractable to compute discrete logarithm in large integers as in other similar research works [16–18, 21–25], .

3.3 Greedy Algorithm Analysis

Algorithm 1 is an example of a common greedy algorithm, where the definition of weight $w(i, S)$ is decided by the problem and its greedy solution. For example, in the greedy algorithm for the Knapsack problem, the weight is each item's value per weight; in the Early Deadline First (EDF) algorithm for the Job Scheduling problem, the weight is each job's end time; in the weighted set cover problem (WSCP), the weight defined in its common greedy algorithm is marginal gain per cost of the chosen set.

Algorithm 1 Generic Greedy Algorithm

```
1: $S := \emptyset$, and define the weight function $w(i, S)$.
2: Given $S$, compute $w(i, S)$ for each instance $i$.
3: Find the $i = \mathbf{argmax}_i w(i, S)$.
4: If $l(S \cup \{i\}) =$ True, $S := S \cup \{i\}$.
5: Repeat 2-4 until the termination condition is satisfied.
6: Return $S$ as the final solution set $\hat{S}$.
```

Different formats of greedy algorithms exist for different types of problems (covering problem, packing problem, static weight etc.). In covering problems, the feasibility of current set S is false until the termination condition is satisfied (e.g., travelling salesman problem, vertex cover), while it is true until the termination condition is satisfied in packing problems (e.g., winner determination, knapsack). Also, in some problems, weights are constants irrelevant to the current set S (travelling salesman problem, job scheduling), and therefore the weight does not need update at every iteration. However, most of them are accepted in our framework with slight conversion. For example, the weighted set cover problem is a covering problem in which a given set S is feasible if the union of all instances is the universe set U. In the common greedy algorithm for this problem [26], the if condition in Step 4 should be 'False'. In such case, we can add a negation in front of the feasibility function and use the same algorithm. Therefore, *w.l.o.g.* we discuss this specific example.

We mainly focus on the following three privacy concerns in this paper. First of all, the **computation of** $w(S, i)$ may leak information about S as well as the private information associated with the instance since each instance's weight often directly or indirectly discloses the private information of the instance. Secondly, **finding the** $\mathbf{argmax}_i w(i, S)$ may also breach the confidentiality of private information related to it. Thirdly, the **feasibility function** $l(S \cup \{i\})$ leaks various sensitive information in two aspects. On one hand, information about the final solution set $\hat{S}$ may be leaked to agents since the sub-solution S should be merged with someone's instance i in each iteration. On the other hand, the constraint associated with the feasibility may be relevant to each instance's private information. For example, weight of items in a 0-1 knapsack problem, start time and finish time in a job scheduling problem, and elements contained in each set in the set cover problem should be checked in $l(\cdot)$. Besides, the algorithm usually terminates when the loop iterates over all instances (*e.g.*, Knapsack, Job Scheduling), but sometimes it needs to terminate at the first time the $l(S \cup \{i\})$ returns 'False'. Then, such **termination condition's evaluation** also raises privacy concerns.

3.4 Problem Formulation

Given the analysis on possible information leakage, we define the privacy of our framework as follows.

Definition 1. *Denote a generic multi-agent algorithm as $\mathcal{A}_{gen}$, a converted privacy-preserving multi-agent algorithm as $\mathcal{A}_{priv}$, all the communication strings produced by our framework as $\mathcal{C}(1^\kappa)$, where κ is the security parameter. Then, an adversary's advantage over instance i's private information $d(i)$ is defined as*

$$adv_i = \Big|\mathbf{Pr}\left[d(i)|\mathcal{C}(1^\kappa), \mathsf{Output} \leftarrow \mathcal{A}_{priv}\right] - \mathbf{Pr}\left[d(i)|\mathsf{Output} \leftarrow \mathcal{A}_{gen}\right]\Big|$$

where $\Pr[d(i)]$ *is the probability that a correct $d(i)$ is inferred. Further, an adversary's advantage over the final solution set $\hat{S}$ is defined as*

$$adv_S = \Big|\mathbf{Pr}[\hat{S}|\mathcal{C}(1^\kappa), \mathsf{Output} \leftarrow \mathcal{A}_{priv}] - \mathbf{Pr}[\hat{S}|\mathsf{Output} \leftarrow \mathcal{A}_{gen}]\Big|$$

where $\Pr[\hat{S}]$ *is the probability that any information about $\hat{S}$ is inferred.*

Definition 2. *We say our framework* ***securely*** *converts a generic greedy algorithm to a privacy-preserving one if all polynomially bounded adversaries' advantages are negligible w.r.t. the input size.*

Informally, these definitions say our framework successfully converts a greedy algorithm to a privacy-preserving one if any polynomial-time adversary cannot increase his probability by to guess the correct private information $d(i)$ or the global solution $\hat{S}$ by attacking our framework. Then, our problem to be solved in this paper is: designing a framework which **securely** converts any generic greedy algorithm for a greedy-class problem to a privacy-preserving one which achieves the same solution as the original algorithm. Note that our framework has certain requirements, and only the greedy algorithms fulfilling the requirements can be converted by our framework (summarized in Section 5).

4 Building Blocks For the Framework

4.1 Multivariate Polynomial Evaluation Protocol (MPEP)

Our previous works [12, 27] implemented a multi-party polynomial evaluation protocol in which the following multivariate polynomial is evaluated without disclosing any x_i provided by different entities: $poly(\mathbf{x}) = \sum_{k=1}^{m}(c_k \prod_{i=1}^{n} x_i^{d_{i,k}})$.

where p is a large prime number. Then, each party reports $R_i m_i$ instead of m_i, and the product $\prod m_i$ can be achieved from $\prod R_i m_i$ without disclosing individual m_i. This product calculation requires that every m_i be non-zero. Secondly, in order to implement the privacy-preserving sum calculation, we used the following *binomial property* to calculate a sum via a product: $\prod(1+x)^{m_i} = (1+x)^{\sum m_i}$ which is equivalent to $1 + x\sum m_i \mod x^2$. With this property, sum is indirectly computed by the product, and the above privacy-preserving product calculation can be used. Finally, the product and sum calculations are combined to evaluate the aforementioned polynomial in a privacy-preserving manner.

Two models are proposed in the protocol: One Aggregater model and Participants Only model. In the former one, only a third-party authority receives the evaluation result while all the participants receive it in the latter one.

4.2 Secure Computation of $w(i, S)$

The current solution set S, which should be kept secret to agents, is usually related in the weight computation. For example, in a common greedy algorithm of the WSCP, the weight is defined as (i is a set of items): $w(i,S) = \frac{|\bigcup_{i' \in S\cup\{i\}} i'| - |\bigcup_{i' \in S} i'|}{d(i)}$, where $d(i)$ is the cost of the selected set i. In such problems, each agent needs to compute the weight without knowing S. We use an n-dimensional binary vector $\mathbf{S}$ to represent it, where its k-th bit $s_k = 1$ if a_k's instance $i_k \in S$ and 0 otherwise. Then, $w(i_k, S)$ is a function: $f(s_0, \cdots, s_n)$, and we can find an equivalent polynomial to compute it, which can be conducted securely via MPEP. For WSCP, another m-dimensional vector $\mathbf{C}_S$ can be defined to indicate whether m items are included in currently chosen sets S, where the k-th bit $c_{k,S} = 1$ if k-th item is included and 0 otherwise. Then, we have $c_{k,S} = 1 - \prod_{j=1}^{n}(1 - c_{j,k,S})$ where $c_{j,k,S}$ is 1 if k-th item is in a_j's instance and his instance is in S, and 0 otherwise. Then, the final weight can be computed as:

$$\begin{aligned} w(i,S) &= \frac{\#\text{ of 1's in } \mathbf{C}_{S\cup\{i\}} - \#\text{ of 1's in } \mathbf{C}_S}{d(i)} \\ &= \frac{\sum_{j=1}^{m} c_{j,S\cup\{i\}} - \sum_{j=1}^{m} c_{j,S}}{d(i)} \\ &= \frac{\sum_{k=1}^{m}(1 - \prod_{j=1}^{n}(1 - c_{j,k,S\cup\{i\}})) - \sum_{k=1}^{m}(1 - \prod_{j=1}^{n}(1 - c_{j,k,S}))}{d(i)} \end{aligned} \tag{1}$$

The numerator can be evaluated via One Aggregater MPEP where only the i's owner receives the result, and the recipient can divide $d(i)$ to the result to compute his weight $w(i, S)$.

Different problems have different weight functions and thus different polynomials. Even the same problem may have several different equivalent polynomials, thus it is out of this paper's scope to give a general conversion for any type of problems. We assume the participants of the problem (central authority or agent) have agreed on one polynomial in advance.

4.3 Finding the Maximal Weight $w(i, S)$

The goal is to find the instance with maximal weight without disclosing its weight. Our idea is to linearly transform the weight $w(i, S) \rightarrow (w(i, S) + \delta)\delta'$

and sort the instances based on the transformed weights to find the instance with the maximal weight. The challenge is to let agents agree on two global random numbers δ, δ' without knowing them. Here is how we achieve this goal.

Firstly, three agents $A = \{a_p, a_q, a_r\}$ are randomly chosen among all $a_j \in \{a_1, \cdots, a_n\}$. Each $a_j \in A$ individually and independently picks two random numbers $\delta_j, \delta'_j \neq 0$. Then, the following transformation (Algorithm 2) is conducted for all $j \in \{0, \cdots, n-1\}$, where i_j is a_j's instance.

Algorithm 2 Transformation for $w(i_j, S)$

```
1: The following sum is evaluated via One Aggregater MPEP, where a randomly
   chosen agent $a_x \in A$ ($a_x \neq a_j$) is the only recipient who achieves the result, and
   $a_j$ provides $w(i_j, S)$: $sum_j = w(i_j, S) + (\delta_p + \delta_q + \delta_r)$.
2: The following product is calculated via One Aggregater MPEP: $prod_j = (w(i_j, S) +
   \delta_p + \delta_q + \delta_r)\delta'_p\delta'_q\delta'_r$, where $w(i_j, S) + \delta_p + \delta_q + \delta_r$ is provided by the agent $a_x$, who is
   chosen at Step 1, and $\delta'_p, \delta'_q, \delta'_r$ are provided by $a_p, a_q, a_r$ respectively. In the agent-
   authority model, One Aggregater MPEP is used so that only the central authority
   knows the results, while Participants Only MPEP is used to send transformed
   weights to every agent in the all-agent model.
3: The result is the transformed weight of $w(i_j, S)$.
```

In the final transformed weight, $\delta_p + \delta_q + \delta_r$ is the δ, and $\delta'_p\delta'_q\delta'_r$ is the δ' that are used in the linear transformation $w(i, S) \to (w(i, S) + \delta)\delta'$. The result recipient sorts the instances according to the transformed weights that he received, and he learns the rank of the instances and nothing else about the weight $w(i, S)$ due to the random numbers. The reason we pick three random agents is because random numbers can be inferred when $a_j \in \{a_p, a_q, a_r\}$ if we have less than three random numbers. On the other hand, we do not employ more than three to avoid unnecessary performance loss.

We assume some user authentication mechanism is in place so that the central authority (agent-authority model) knows the owner of each transformed weight since he needs to arrange each instance into a solution set. In contrary, we assume the ownership of the instance is hidden by employing an anonymized network (torproject.org) in the all-agents model. This is necessary because disclosing the ownership tells all agents everyone else's rank, and this may give side information about the global solution set to adversaries.

4.4 Feasibility Check

The goal of this function is to check whether a set of instances S is feasible. We have three different methods to check the feasibility: set-based check, algebra-based check, and graph-based check.

Set-based check

Definition 3. *A feasible set S is maximal (minimal) if it is not a superset of any smaller feasible set.*

Then, we use the following *subset-closure property* to check the feasibility of a given set S for the packing problem: $\forall S_1, S_2 \subseteq S_1 : S_1$ is feasible $\to S_2$ is feasible.

Similarly, *superset-closure property*, which is an analogue, can be used to check the feasibility of a given set S in the covering problem.

In the packing (covering) problem, any subset (superset) of a feasible set is also feasible. Then, a given set S is feasible if and only if it is a subset (superset) of some maximal (minimal) feasible set, or it is one of the maximal (minimal) feasible sets itself. Consequently, one only needs to see if $S \subseteq S'$ ($S' \subseteq S$) for all maximal (minimal) feasible sets S' to evaluate $l(\cdot)$. Then, we use the same n-dimensional binary vector $\mathbf{S}$ used in secure weight computation (Section 4.2). Due to the inner product property, $S \subseteq S'$ if and only if $\mathbf{S} \cdot (\{1\}^n - \mathbf{S}') = 0$. Then, given a family of all maximal (minimal) feasible sets $\mathcal{S}^*$, one can evaluate the following term: $\exists S' \in \mathcal{S}^* : \ \mathbf{S} \cdot (\{1\}^n - \mathbf{S}') \Leftrightarrow \prod_{S' \in \mathcal{S}^*} (\sum_{j=1}^{n} s_j \cdot (1 - s'_j)) = 0$.

In the agent-authority model, this evaluation is conducted locally at the central authority's side. This is possible because the central authority has all the instances, instances' ranks in terms of their weights, the intermediate solution set S during the greedy algorithm, and all maximal feasible sets in $\mathcal{S}^*$. He can create the vectors $\mathbf{S}$ and $\mathbf{S}'$ at every round of the feasibility check and evaluate the above product locally. In the all-agents model, all maximal feasible sets are given to agents, but the instances in the final global solution $\hat{S}$ should be kept secret. Thus no one is allowed to access the intermediate solution set S (otherwise great amount of information about $\hat{S}$ is leaked), and no one has the vector $\mathbf{S}$. That is, each agent a_j has a secret binary value s_j indicating whether his instance is included in the S, and essentially we need to compute the $\sum s_j \cdot (1 - s'_j)$ without disclosing individual s_j. This sum value can be evaluated securely via Participants Only MPEP to let all agents know whether the sum value is 0 without knowing individual s_j.

This idea is intuitive and applicable to any type of greedy-class problem, but it has some limitations. 1) All maximal feasible sets should be given (in MPEP's encrypted format) in advance. 2) Construction of maximal feasible sets requires private information associated with the instances in some problems (e.g., Knapsack and Job Scheduling problem). Therefore, we rely on the following two methods when set-based check is not possible.

Algebra-based check In some greedy-class problems, the feasibility constraints are given by a set of algebraic inequalities which are closely related to the private information. That is, given a set of instances S and its associated information set D, the feasibility constraint is $\{f_i(S, D) \leq \theta_i\}_i$ where each $f_i(S, D)$ is some function of S, D which returns a real value and θ_i is a threshold value depending on the problem. $l(S)$ returns true if all the feasibility constraints are satisfied. For example, in a 0-1 Knapsack problem, there is only one constraint: $f_1(S, D) \leq \theta_1$, where $f_1(S, D)$ is S's total weight and θ_1 is the total capacity, and in the Job scheduling problem, if there are k jobs in the scheduling list, there are $k - 1$ constraints: the finish time of the job J_{i-1} should be less than the start time of the job J_i. Note that an equality can be trivially converted to two inequalities (e.g., $a = b \Leftrightarrow a \geq b, a \leq b$).

Since the feasibility is related to the private information associated with the instances, we need to privately evaluate the inequalities without disclosing private information. It seems the building block [12] can be used to solve this problem, where the input values of $f_1, f_2, \cdots, f_k$ are provided by the owners of various instances in S. However, the protocol proposed in [12] only evaluates a polynomial in an integer domain. Therefore, an equivalent integer polynomials should be found first: $\{poly_i(S, D) \leq \theta'_i\}_i$. Then, we can run MPEP in [12] to evaluate the polynomial values to check the inequalities in a distributed manner without knowing anything about any instances' private information. One Aggregater MPEP is used in the agent-authority model and Participants Only MPEP is used in the all-agents model.

However, this reveals the polynomial values to adversaries, which could be used to infer private information. For example, the constraint inequality in Knapsack problem is chosen items' total weight, and this value can be used to infer individual item's weight. Therefore, we evaluate the following inequalities instead: $\{(poly_i(S, D) - \theta'_1) \prod_{j=0}^{n-1} \delta_{j,i} \leq 0\}_i$, where $\delta_{j,i}$ is a random number independently chosen by a_j for the i-th inequality and $\prod_i \delta_{j,i}$ acts as a global random number as in the weight transformation. By doing so, the polynomial values are masked by the global random number.

Graph-based check The feasibility constraints in some greedy-class problems are given by a graph structure. Given a set of instances S, the set is represented by a graph structure $G_S = (V_S, E_S)$ depending on the problem, and $l(S)$ returns true if some graph constraints are satisfied. Therefore, one needs to convert the set S to a graph G_S first such that the feasibility is equivalent to the graph constraint. The graph constraints fall into one of the following four categories:

1. *Node covering/packing*: the constraint is satisfied if every node is covered at least/most once.
2. *Edge covering/packing*: the constraint is satisfied if every edge is covered at least/most once.

Note that a problem with graph-based constraints may not be a graph-based problem. For example, the WDCA is an auction problem to find the bundle allocation, and it is not a graph-related problem. However, its constraint is an edge packing type: each node represents each bidder and there is an edge between two bidders if one's bundle is not compatible with another one's bundle, and an edge is covered if either incident node's (bidder's) bundle is included in the S. Then, one edge being covered by twice means two incompatible bundles are in S. Its constraint can also be a node packing type: each node represents each good and it is covered if the corresponding good is allocated to a bidder by S. Then, one node being covered twice indicates the good is allocated to two bidders simultaneously.

For an instance i, whether each node in $G_S = (V_S, E_S)$ is covered by it can be represented as a $|V_S|$-dimensional binary vector $\mathbf{i}$ whose k-th bit $i_k = 1$ if the k-th node is covered and 0 otherwise. This is called the *coverage status vector* of i. For the problems of edge types, the coverage status vector is a $|E_S|$-dimensional binary vector. Then, the feasibility function $l(S)$ returns true if and only if $\forall k : \sum_{i \in S} i_k \geq (\leq) 1$ for node/edge covering (packing) type.

For example, in the edge packing type of the feasibility check for the WDCA problem, V_S is the set of all bidders and E_S is the set of edges indicating incompatibility between bidders. The coverage status vector of a bidder's bundle i is a $|E_S|$-dimensional binary vector, where the k-th bit is 1 if the k-th edge is covered (edges are indexed by arbitrary pre-defined order). Then, if any bit's sum over all instances in S is greater than 1, S is not feasible and vice versa.

In the agent-authority model, the above inequalities can be examined locally at the central authority's side since he has all instances and the current solution set S, therefore he can construct the G_S and corresponding coverage status vectors for all instances to examine the inequalities. In the all-agents model, no one is allowed to access S, but we can still use the privacy-preserving sum calculation in [12] to examine the inequalities without disclosing any information about S. Each agent controls the bits i_k's that are relevant to his instance (e.g., the k-th incompatibility edge in WDCA problem).

Feasibility check conclusion In conclusion, for various problems, if the feasibility of a set of instances can be examined via above three methods, one can examine the feasibility without leaking each individual's privacy. Depending on the application requirement, the protocol participants may agree on one of the three types which best protects the privacy. Since the declaration of the feasibility check type does not affect privacy protection, we assume this is declared by any third party.

5 Our General Framework Design

If a greedy-class problem's greedy algorithm fits our framework, the problem can be solved with our framework. That is, if an algorithm's weight function can be represented with polynomials and if its feasibility can be evaluated with one of the aforementioned three types of feasibility check, the original algorithm (Algorithm 1) can be converted to the the privacy-preserving one (Algorithm 3).

Algorithm 3 Converted Privacy-Preserving Greedy Algorithm

```
1: $S := \emptyset$, and define the weight function $w(i, S)$.
2: Given $S$, compute $w(i, S)$ **with secure computation (Section 4.2)**.
3: Find the $i = \text{argmax}_i w(i, S)$ **with transformation (Section 4.3)**.
4: If $l(S \cup \{i\}) = \text{True}$, $S := S \cup \{i\}$, **where** $l(\cdot)$ **is evaluated by one of the three**
   **feasibility checks (Section 4.4)**.
5: Repeat 2-4 until the termination condition is satisfied.
6: Return $S$ as the final solution set $\hat{S}$.
```

Note that some algorithms need to evaluate $l(\cdot)$ for the termination condition while other just need to terminate after the loop is iterated over all instances, and $l(\cdot)$ is evaluated by the privacy preserving feasibility check. Next, we show detailed procedures of our framework in different system models (agent-authority model and all-agents model) and give a running example of it.

5.1 Agent-Authority Model

Firstly, the weight of each agent's instance is computed securely. Only the owner of the instance receives the result by using One Aggregater MPEP. Then, agents and the central authority run the privacy-preserving sorting in Section 4.3. The MPEP ([12]) in the sorting is executed with the One Aggregater Model such that only the central authority learns the polynomial results. When the privacy-preserving sorting is finished, the central authority gets a set of transformed weights of agents' instances as well as a list of the instances in the order of their weights. Secondly, the central authority picks the first instance i in the sorted list and evaluates $l(S \cup \{i\})$. If the problem's feasibility is an algebra-based one, he and the agents are repeatedly engaged in the privacy-preserving feasibility check in Section 4.4, and the MPEP in the check is executed with the One Aggregater Model again so that only the central authority achieves the evaluation result. On the other hand, if the problem's feasibility is a graph-based one or a set-based one, the central authority checks the feasibility at his side locally. If the feasibility check returns true, S and $\{i\}$ are merged to form a new S.

These two steps are repeated until the termination condition is satisfied. When the algorithm terminates, the central authority achieves the global solution set $\hat{S}$ without knowing any agent's private information, and all agents do not gain any information about $\hat{S}$ either.

5.2 All-Agents Model

Firstly, each agent achieves his own weight via privacy-preserving weight computation (Section 4.2). Then, they run the privacy-preserving sorting as well, but the MPEP is executed with the Participants Only Model, where all the participants learn the polynomial results. When the privacy-preserving sorting is finished, the agents gets a set of transformed weights of all instances, and each agent knows the rank of his instance among all instances in terms of the weight. Secondly, the feasibility of $S \cup \{i\}$ should be checked in the order of the instances' weight, therefore the participants jointly and repeatedly run the feasibility check in Section 4.4. If i is the k-th instance in the sorted list, $l(S \cup \{i\})$ is checked at the k-th round of the feasibility check, and S includes all instances who have returned 'True' so far. Then, any one of the three feasibility checks in 4.4 can be used to check $S \cup \{i\}$'s feasibility depending on the problem. At each round, if the $S \cup \{i\}$ is feasible, i is merged in S to form a new intermediate solution set $S := S \cup \{i\}$.

These two steps are repeated until the termination condition is satisfied. When the algorithm terminates, every agent knows whether his instance is included in the final solution set $\hat{S}$ but nothing else. In fact, no one in the system has any information about $\hat{S}$ in this model.

5.3 Running Example of the WSCP

We convert the greedy algorithm for WSCP in all-agents model in this section.

At the first iteration, each agent a_j locally computes his weight $w(i_j, S) = \frac{|i_j|}{d(i_j)}$. Then, the agents participate in the instance sorting (Section 4.3) to receive the transformed weights of all instances. Every agent locally sorts the instances based on the transformed weight, and the owner of the $i = \text{argmax}_i(w(i, S) + \delta)\delta'$ knows that his instance is in $\hat{S}$. In the next iteration, weight computation (presented in Section 4.2) is conducted via One Aggregater MPEP to update each

Algorithm 4 Greedy algorithm for the WSCP

```
1: $S := \emptyset$. The weight is defined as Eq. 1 in Section 4.
2: Given $S$, compute $w(i, S)$ for each instance $i \in I$.
3: Sort the instances in the non-increasing order of the weight $w(i, S)$ and find the
   $i = \text{argmax}_i w(i, S)$.
4: If $\neg l(S \cup \{i\}) = \text{True}$, $S := S \cup \{i\}$.
5: Repeat 2-4 until $\neg l(S \cup \{i\}) = \text{False}$.
6: Return $S \cup \{i\}$ as $\hat{S}$.
```

instance's weight, where only the owner of the instance receives the result. At this computation, the owner of the instances in the solution set (i.e., $i \in S$) will set the corresponding $c_{j,k,S} = 1$ in the weight computation. Then, the instance sorting with new random numbers δ, δ' is run again to let all agents know their own rank. They run the feasibility check (Section 4.4) to see if $\neg l(S \cup \{i\}) = \text{True}$ where i is the instance with the maximal weight. If yes, the owner of i knows that his instance is in $\hat{S}$. This is repeated until $\neg l(S \cup \{i\}) = \text{False}$, the owner of the last instance i also knows that his instance is included in the $\hat{S}$. Everyone else learns that his instance is not in the final solution set $\hat{S}$.

Adversaries' Advantage on Private Information Private information might be leaked in the following three parts: weight computation, instance sorting based on transformed weights, and the feasibility check involving private information and $w(i, S)$.

Theorem 1. *Assuming the discrete logarithm is hard, the adversary's advantage* adv_i *is a negligible function.*

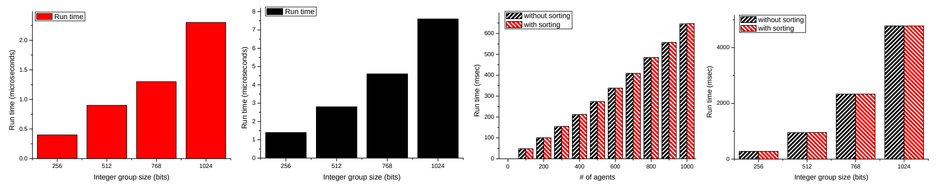
6 Performance Evaluation

6.1 Computation Overhead

We implemented the framework using the GMP library (gmplib.org) based on C in a computer with Intel i3-2365M CPU @ 1.40 GHz $\times 4$, Memory 4GB and SATA Hard Drive 500GB (5400RPM).

Micro-benchmark Since various problems have different $\#_{poly}$ for the weight computation and the feasibility check, we present the computation overhead of a single addition and a single multiplication for them. We measured the average run time of 10,000 additions and 10,000 multiplications of two random numbers in $\mathbb{Z}_p$ respectively, which is shown in Figure 1(a) and 1(b). The integer group size is bit length of p, i.e., the security parameter κ. Note that each operation (either addition or multiplication) is in the order of microseconds, and therefore the overall run time will be of several seconds unless the order of the operations number introduced by the framework is greater than 1 billion, which is unlikely. This shows that our framework is very lightweight.

The Figure 1(c) and 1(d) show the computation overhead of the instance sorting based on their weights. We randomly generated a 20-bit weight for each agent and conducted the weight transformation as well as the final sorting based on



(a) Single multiplication (b) Single addition (c) Sorting, various # of agents (256-bit) (d) Sorting, various bits (500 agents)

Fig. 1. Run time of a multiplication, addition, and sorting

the transformed weights. Quicksort is used in the sorting, and we observed that the sorting's computation overhead is almost negligible. The figures show the run time of the central authority in the agent-authority model. In the all-agents model, all of the agents have the same computation overhead as the central authority in the agent-authority model because everyone needs to compute the final weights based on the received ciphertexts.

Extra Overhead Measurement We measured the run times of the original greedy algorithms and the ones of converted algorithms via our framework respectively for the following four problems: WDCA, Knapsack, Job scheduling and Weighted set cover problem, and they are shown in 1. Network delay and I/O delay are excluded from the measurement. Note that everything is disclosed to the authority in the original algorithm (no privacy consideration), and thus agents do not compute anything. As shown in the table, the extra computation overhead varies greatly for different problems due to different types of corresponding greedy algorithms.

Table 1. Comparison

Problem	Original	Converted
	Authority	
WDCA	5.87s	603s
Knapsack	4ms	273ms
Scheduling	7ms	11.2s
WSCP	9.31s	135s
	Each agent	
WDCA	n/a	8.1s
Knapsack	n/a	7.9ms
Scheduling	n/a	10ms
WSCP	n/a	350s

7 Conclusion

We designed a framework for multi-agent greedy algorithms in which the final solution comes from multiple agents' input instances. We use our novel secure weight computation, privacy-preserving max finding, and privacy-preserving feasibility check to prevent underlying privacy leakage in the distributed greedy algorithms. We showed that our framework does not leak useful information about the agents' private information, and we also showed that the extra computation overhead introduced by our framework is small. In addition, the communication overhead is much less than that of other general solutions as well. All these are evidence that our framework is a viable option for the business intelligence in the big data context.

References

1. C. Bo, G. Shen, J. Liu, X.-Y. Li, Y. Zhang, and F. Zhao, "Privacy. tag: Privacy concern expressed and respected," in *SenSys*, pp. 163–176, ACM, 2014.
2. X. Chen, X. Wu, X.-Y. Li, Y. He, and Y. Liu, "Privacy-preserving high-quality map generation with participatory sensing," in *INFOCOM*, pp. 2310–2318, IEEE, 2014.

3. J. Zhao, T. Jung, Y. Wang, and X. Li, "Achieving differential privacy of data disclosure in the smart grid," in *INFOCOM*, pp. 504–512, IEEE, 2014.
4. P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," *A.I.*, 2005.
5. S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, 2011.
6. Y. Shi and Y. T. Hou, "A distributed optimization algorithm for multi-hop cognitive radio networks," in *INFOCOM*, IEEE, 2008.
7. M. Yokoo, K. Suzuki, and K. Hirayama, "Secure distributed constraint satisfaction: Reaching agreement without revealing private information," *A.I.*, 2005.
8. M. C. Silaghi and D. Mitra, "Distributed constraint satisfaction and optimization with privacy enforcement," in *IAT*, IEEE, 2004.
9. M. C. Silaghi and M. Yokoo, "Nogood based asynchronous distributed optimization (adopt ng)," in *AAMAS*, ACM, 2006.
10. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, 1998.
11. R. Zivan, "Anytime local search for distributed constraint optimization," in *AAMAS*, 2008.
12. T. Jung, X. Mao, X.-Y. Li, S.-J. Tang, W. Gong, and L. Zhang, "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation," in *INFOCOM*, pp. 2634–2642, IEEE, 2013.
13. A. C. Yao, "Protocols for secure computations," in *FOCS*, 1982.
14. A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, IEEE, 1986.
15. S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, 1985.
16. T. Jung and X.-Y. Li, "Enabling privacy-preserving auctions in big data," in *BigSecurity, INFOCOM Workshops*, IEEE, 2015.
17. L. Zhang, T. Jung, P. Feng, K. Liu, X.-Y. Li, and Y. Liu, "Pic: Enable large-scale privacy-preserving content-based image search on cloud," in *ICPP*, IEEE, 2015.
18. L. Zhang, T. Jung, C. Liu, X. Ding, X.-Y. Li, and Y. Liu, "Pop: Privacy-preserving outsourced photo sharing and searching for mobile devices," in *ICDCS*, IEEE, 2015.
19. L. Zhang, X.-Y. Li, and Y. Liu, "Message in a sealed bottle: Privacy preserving friending in social networks," in *ICDCS*, pp. 327–336, IEEE, 2013.
20. M. Yokoo, T. Ishida, E. H. Durfee, and K. Kuwabara, "Distributed constraint satisfaction for formalizing distributed problem solving," in *ICDCS*, IEEE, 1992.
21. T. Jung, X.-Y. Li, Z. Wan, and M. Wan, "Privacy preserving cloud data access with multi-authorities," in *INFOCOM*, pp. 2625–2633, IEEE, 2013.
22. L. Zhang, X.-Y. Li, Y. Liu, and T. Jung, "Verifiable private multi-party computation: ranging and ranking," in *INFOCOM*, pp. 605–609, IEEE, 2013.
23. T. Jung, X. Li, Z. Wan, and M. Wan, "Control cloud data access privilege and anonymity with fully anonymous attribute based encryption," *TIFS*, vol. 10, no. 1, pp. 190–199, 2014.
24. X.-Y. Li and T. Jung, "Search me if you can: privacy-preserving location query service," in *INFOCOM*, pp. 2760–2768, IEEE, 2013.
25. L. Zhang, X. Li, K. Liu, T. Jung, and Y. Liu, "Message in a sealed bottle: Privacy preserving friending in mobile social networks," *TMC*, 2014.
26. Wikipedia. `http://en.wikipedia.org/wiki/Set_cover_problem`.
27. T. Jung and X.-Y. Li, "Collusion-tolerable privacy-preserving sum and product calculation without secure channel," *TDSC*, vol. 12, no. 1, pp. 45–57, 2014.