Optim Lett (2014) 8:267–277
DOI 10.1007/s11590-012-0572-7

ORIGINAL PAPER

An inference–proof approach to privacy-preserving horizontally partitioned linear programs

Yuan Hong · Jaideep Vaidya

Received: 29 November 2011 / Accepted: 22 September 2012 / Published online: 5 October 2012

Abstract Mangasarian (Optim. Lett., 6(3), 431–436, 2012) proposed a constraints transformation based approach to securely solving the horizontally partitioned linear programs among multiple entities—every entity holds its own private equality constraints. More recently, Li et al. (Optim. Lett., doi:10.1007/s11590-011-0403-2, 2012) extended the transformation approach to horizontally partitioned linear programs with inequality constraints. However, such transformation approach is *not sufficiently secure* – occasionally, the privately owned constraints are still under high risk of inference. In this paper, we present an inference–proof algorithm to enhance the security for privacy-preserving horizontally partitioned linear program with *arbitrary number of equality and inequality constraints*. Our approach reveals significantly less information than the prior work and resolves the potential inference attack.

Keywords Linear program · Privacy · Security · Horizontally partitioned data

1 Introduction

Securely solving distributed optimization problems has attracted considerable interest recently. Consider that an optimization problem is jointly formulated among multiple entities, the basic premise is to solve the problem without revealing any private information to each other. As a fundamental branch of optimization problems, numer-

Y. Hong (✉) · J. Vaidya
Management Science and Information Systems Department and CIMIC,
Rutgers University, 1 Washington Park, Newark, NJ 07102, USA
e-mail: yhong@cimic.rutgers.edu

J. Vaidya
e-mail: jsvaidya@business.rutgers.edu

ous privacy-preserving linear programming techniques have been proposed in literature [1–11] with respect to different data partition scenarios for linear programs.

Specifically, [4] and [7] proposed a secure simplex method and revised simplex method respectively for linear programs co-held by two entities. A matrix transformation approach has been proposed in [5,11] (some flaws with this approach are pointed out by Bednarz et al. [6]). In [1–3,8–10], the private information of two or more entities in the linear program are protected via mathematical transformation.

In particular, Mangasarian [1] proposed a constraints transformation based approach to securely solving the horizontally partitioned linear programs among multiple entities, where every entity holds its own set of private equality constraints. More recently, Li et al. [2] extended this approach to horizontally partitioned linear programs with inequality constraints. However, the approach is not sufficiently secure—occasionally, the privately owned constraints can still be inferred by adversaries (see Sect. 2.2). In this paper, we present an *inference–proof approach* to enhance the security for privacy-preserving horizontally partitioned linear program (PPHPLP) [1,2] with *arbitrary number of equality and inequality constraints*. Our approach extends the constraint matrix by including artificial constraints and extra slack variables (with random coefficients) to provide significantly more security than the existing work.

2 Inference attack against the existing work [1,2]

2.1 Privacy-preserving horizontally partitioned linear program

Consider a linear program with equality constraints: $\{\min : c^T x, \quad s.t.\ Ax = b, x \geq 0\}$. In horizontally partitioned linear program [1,2], the matrix $A \in R^{m \times n}$ and the right-hand side vector $b \in R^m$ in the constraints are partitioned into p horizontal blocks held by p entities respectively: $A_{I_1.}, A_{I_2.}, \ldots, A_{I_p.}$ and $b_{I_1}, b_{I_2}, \ldots, b_{I_p}$ where A_{I_i}, b_{I_i} and the index set I_i belong to entity i (note that $i = 1 \ldots p$, $\bigcup_{i=1}^{p} I_i = \{1, 2, \ldots, m\}$). With privacy concern, every entity $i = 1 \ldots p$ does not want to share its data block $A_{I_i.}, b_{I_i}$ with other entities while solving the problem. Therefore, Mangasarian [1] proposed a transformation approach to address this issue—given a random matrix $B \in R^{k \times m}$ with $k \geq m$ and rank m [12], a new linear program can be formulated as $\{\min : c^T x, \quad s.t.\ BAx = Bb, x \geq 0\}$.

In [1], the original and the transformed linear programs have been proven to have equivalent optimal solution and feasibility if the rank of $B \in R^{k \times m}$ (number of linear independent column vectors in B) is equal to m where $k \geq m$ (which naturally holds for random matrices [12]). Therefore, every entity $i = 1 \ldots p$ generates a privately held random matrix $B_{.I_i} \in R^{k \times m_i}$ such that:

$$BA = [B_{.I_1}, B_{.I_2}, \ldots, B_{.I_p}] \begin{bmatrix} A_{I_1.} \\ A_{I_2.} \\ \vdots \\ A_{I_p.} \end{bmatrix}$$
$$= B_{.I_1} A_{I_1.} + B_{.I_2} A_{I_2.} + \cdots + B_{.I_p} A_{I_p.} \in R^{k \times n} \quad (1)$$

and

$$Bb = [B._{I_1}, B._{I_2}, \ldots, B._{I_p}] \begin{bmatrix} b_{I_1} \\ b_{I_2} \\ \vdots \\ b_{I_p} \end{bmatrix}$$
$$= B._{I_1} b_{I_1} + B._{I_2} b_{I_2} + \cdots + B._{I_p} b_{I_p} \in R^k \quad (2)$$

In Mangasarian's PPHPLP algorithm [1], every entity $i = 1 \ldots p$ only discloses publicly the transformed matrix product $B._{I_i} A_{I_i}.$ and the transformed vector $B._{I_i} b_{I_i}$ for solving the problem. Thus $B._{I_i}$, $A_{I_i}.$ and b_{I_i} can be kept private.

Furthermore, Li et al. [2] considers the horizontally partitioned linear program with inequality constraints as below:

$$\begin{aligned} min: \quad & c^T x \\ s.t. \quad & Ax \leq b, x \geq 0 \end{aligned} \quad (3)$$

Mangasarian [1] has pointed out that converting the inequality constraints in linear program (3) to equality constraints (by adding slack variables) and then transforming the problem per Mangasarian's approach [1] would reveal $B._{I_i}$. Therefore, Li et al. [2] resolved this issue by letting every entity $i = 1 \ldots p$ generate a diagonal random matrix $D_{I_i} \in R^{m_i \times m_i}$ besides $B._{I_i} \in R^{k \times m_i}$ for its slack variables (every inequality requires one slack variable). Consequently, a similar transformation is conducted $Ax + Dx_s = Bb \iff BAx + BDx_s = Bb$ to formulate:

$$\begin{aligned} min: \quad & c^T x \\ s.t. \quad & BAx + BDx_s = Bb, x \geq 0 \end{aligned} \quad (4)$$

where x_s denotes all the slack variables and $D = diag(D_{I_1}, D_{I_2}, \ldots, D_{I_p}) \in R^{m \times m}$. As a result, if (x^*, x_s^*) is the optimal solution of the transformed linear program (4), x^* is also the optimal solution for the linear program (3).

2.2 Inference attack

In Li et al.'s [2] extended transformation for linear program (3), every entity $i = 1 \ldots p$ generates a diagonal random matrix $D_{I_i} \in R^{m_i \times m_i}$ to prevent inferring its private random matrix $B._{I_i}$ from the slack variables. However, such transformation approach is still vulnerable to potential inference attack.

We now look at the illustrative example in [2]: entity 1 holds two constraints $x_1 + x_2 + 2x_3 \leq 2$ and $x_1 + 4x_2 - x_3 \leq 1$ while entity 2 holds one constraint $x_1 + 2x_2 - 4x_3 \leq 1$. In the transformed linear program, entity 1 and 2 generate their privately held random

matrices respectively:

$$B._{I_1} = \begin{bmatrix} 0.4562 & 0.1367 \\ 0.5469 & 0.8739 \\ 0.5633 & 0.7693 \end{bmatrix} \in R^{3\times 2}, \quad D_{I_1} = \begin{bmatrix} 12.4965 & 0 \\ 0 & 25.6433 \end{bmatrix} \in R^{2\times 2}$$

$$B._{I_2} = \begin{bmatrix} 0.3574 \\ 0.7763 \\ 0.6682 \end{bmatrix} \in R^{3\times 1}, \quad D_{I_2} = \begin{bmatrix} 15.7628 \end{bmatrix} \in R^{1\times 1}$$

Since the inequality constraints require three slack variables (this is revealed publicly while solving the linear program), entity 1 can learn that entity 2 has one inequality constraint ($m_2 = 1$) due to its known $m_1 = 2$. Then, the size of matrices $B._{I_2}$ and $A_{I_2.}$ can be learnt by entity 1 as $R^{3\times 1}$ and $R^{1\times 3}$. Thus, from entity 1's view, the elements in $B._{I_2} \in R^{3\times 1}$, $A_{I_2.} \in R^{1\times 3}$, and $b_{I_2} \in R^{1\times 1}$ can be regarded as real variables $\beta_1, \beta_2, \beta_3, \alpha_1, \alpha_2, \alpha_3$ and δ:

$$A_{I_2.} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix}, B._{I_2} = \begin{bmatrix} \beta_1, \beta_2, \beta_3 \end{bmatrix}^T, b_{I_2} = [\delta]$$

Note that Mangasarian [1] and Li et al. [2] make public every entity i's transformed matrix $B._{I_i} A_{I_i.}$ and vector $B._{I_i} b_{I_i}$. Therefore, in this two-entity case, entity 1 can express the real numbers in $B._{I_2} A_{I_2.} \in R^{3\times 3}$ and $B._{I_2} b_{I_2} \in R^{3\times 1}$ as:

$$B._{I_2} A_{I_2.} = \begin{bmatrix} \beta_1\alpha_1 & \beta_1\alpha_2 & \beta_1\alpha_3 \\ \beta_2\alpha_1 & \beta_2\alpha_2 & \beta_2\alpha_3 \\ \beta_3\alpha_1 & \beta_3\alpha_2 & \beta_3\alpha_3 \end{bmatrix} = \begin{bmatrix} 0.3574 & 0.7148 & -1.4296 \\ 0.7763 & 1.5526 & -3.1052 \\ 0.6682 & 1.3364 & -2.6728 \end{bmatrix}$$

$$B._{I_2} b_{I_2} = \begin{bmatrix} \beta_1\delta \\ \beta_2\delta \\ \beta_3\delta \end{bmatrix} = \begin{bmatrix} 0.3574 \\ 0.7763 \\ 0.6682 \end{bmatrix}$$

where all the elements in the above matrix and vector have been exactly revealed to entity 1. Since $\beta_1\alpha_1 = 0.3574$, $\beta_1\alpha_2 = 0.7148$, $\beta_1\alpha_3 = -1.4296$ and $\beta_1\delta = 0.3574$ are known to entity 1 as constants in $B._{I_1} A_{I_1.}$ and $B._{I_1} b_{I_1}$ (similarly, other rows in $B._{I_1} A_{I_1.}$ and $B._{I_1} b_{I_1}$ are also revealed), entity 1 can simply compute $\alpha_1{:}\alpha_2{:}\alpha_3{:}\delta = 1{:}2{:}(-4){:}1$. Although the exact value of β_1, α_1, α_2, α_3 and δ cannot be learnt by entity 1, the ratio $\alpha_1{:}\alpha_2{:}\alpha_3{:}\delta = 1{:}2{:}(-4){:}1$ is sufficient to reconstruct the constraint $x_1 + 2x_2 - 4x_3 \leq 1$ for adversaries.

Hence, revealing the number of constraints m in the linear program (and also every entity's transformed share per [1,2]) may result in serious privacy leakage: an adversarial entity might be able to infer other entities' private constraints by formulating equations with real variables. Indeed, two possible ways of learning m can be identified in the prior work [1,2] as follows:

1. *The number of slack variables* might be the number of constraints m (see above).
2. Since the transformed matrix BA and vector Bb are revealed to public in [1,2], every entity can compute the rank of matrix BA. Due to $rank(B \in R^{k\times m}) = m$ [12] and $rank(A \in R^{m\times n}) \leq m$, we have $rank(B) \geq rank(A)$.

Hence, $rank(BA) = rank(A)$, and $rank(A)$ can be inferred by all entities as $rank(BA)$. However, *the number of constraints is likely equal to* $rank(A)$, especially when the linear program contains small number of constraints (this applies to both equality [1] and inequality constraints [2]: in Li et al.'s illustrative example [2], $rank(BA) = rank(A) = 3$ can be computed, thus both entities can infer that $m = 3, m_1 = 2, m_2 = 1$). Therefore, the risk of inferring the number of constraints by $rank(BA)$ is still high in [1,2].

3 Inference–proof algorithm to PPHPLP

We now present an inference–proof approach for PPHPLP.

3.1 Transformation

We consider a general model of linear programs containing an arbitrary number of equality and inequality constraints. The slack form of the linear program is:

$$\begin{aligned} min: \quad & c^T x \\ s.t. \quad & Ax + Mx_s = b, \quad x \geq 0, \quad x_s \geq 0 \end{aligned} \tag{5}$$

where $A \in R^{m \times n}$, x_s includes ℓ slack variables [ℓ is the number of inequality constraints in linear program (5)] and $M = \begin{bmatrix} 0 \\ I \end{bmatrix} \in R^{m \times \ell}$ ($\ell \leq m$) is not always a diagonal matrix (different from D defined in Li et al. [2]) in this general linear program form: for each *equality constraint*, all the elements in the corresponding row of M are 0; the remaining rows (associated with *inequality constraints*) form a diagonal matrix. In horizontally partitioned linear program, every entity $i = 1 \ldots p$ holds constraints $A_{I_i} x + M_{I_i} x_s = b_{I_i} (A_{I_i} \in R^{m_i \times n}, b_{I_i} \in R^{m_i})$ where $M_{I_i} \in R^{m_i \times \ell}$ is the rows of M associated with entity i's all equality and inequality constraints, thus:

$$M = \begin{bmatrix} M_{I_1} \\ \vdots \\ M_{I_p} \end{bmatrix} \in R^{m \times \ell}, \quad \ell \leq m$$

First, to prevent learning the number of constraints m from the rank of matrices BA and A, we can let every entity $i = 1 \ldots p$ locally generate some *artificial constraints* based on its original constraints, and add them to their constraints (we denote all entities' total number of original and artificial constraints as m'). Note that the feasible region of entity $i (i = 1 \ldots p)$'s all the artificial constraints should be a superset of the feasible region of entity i's original local constraints – *thus adding artificial constraints does not change the feasible region of the linear program*. For instance, if entity i owns inequality constraint $x_1 - 3x_2 \leq 9$ where $x \geq 0$, some artificial constraints can be generated as: $x_1 - 4x_2 \leq 9, 0.9x_1 - 3x_2 \leq 9, 0.5x_1 - 1.7x_2 \leq 4.5$. etc; if entity i owns equality constraint $x_1 + 3x_2 = 9$ where $x \geq 0$, we can generate: $x_1 + 2x_2 \leq 9$,

$0.5x_1 + 1.2x_2 \leq 4.5$. etc [creating artificial equality constraints are not recommended since adding them does not change $rank(A)$].

As a result, the constraint matrix and right-hand side vector are expanded to $A' \in R^{m' \times n}$ and $b' \in R^{m'}$, and all p entities can only infer $rank(A')$ rather than $rank(A)$ by computing $rank(BA')$ (*it is not difficult for every entity to generate artificial constraints with real number coefficients that increase the rank of the global constraint matrix* A). Therefore, the probability of inferring m can be significantly reduced by increasing the number of artificial constraints.

Second, we would also like to reduce the probability of inferring m from the number of slack variables ℓ. We denote the number of inequality constraints in the artificial constraints-added linear program as ℓ' (indeed, $\ell < \ell' \leq m'$, and the coefficient matrix of slack variables $M \in R^{m \times \ell}$ is expanded to $M' \in R^{m' \times \ell'}$ in the slack form of the artificial constraints-added linear program), thus m cannot be inferred by ℓ anymore. In addition, we can better reduce the inference probability by utilizing *more than one slack variable* to convert each of the inequality constraints to equality constraint in the slack form. Specifically, for the jth inequality constraint ($j = 1 \ldots \ell'$) in the artificial constraints-added linear program, we can let the constraint-owning entity determine t_j slack variables for converting it into an equality constraint. Thus, the joint linear program can have $t = \sum_{j=1}^{\ell'} t_j \gg \ell'$ slack variables.

We denote every entity i's ($i = 1 \ldots p$) total number of slack variables for its all inequality constraints as T_i. In general, T_i can be sufficiently large to better reduce the inference probability (e.g. let $T_i \gg m'_i$), thus we have $t = \sum_{i=1}^{p} T_i > m' \geq \ell' > \ell$ and $m' > m \geq \ell$ ($m = \ell$ [2] is the most vulnerable case). Hence, the total number of slack variables t can be completely different from m, and the linear program is:

$$\begin{aligned} min: \quad & c^T x \\ s.t. \quad & A'x + D'x'_s = b', \quad x \geq 0, \quad x'_s \geq 0 \end{aligned} \tag{6}$$

where $A' \in R^{m' \times n}$, slack variables $x'_s = \{s_1, s_2, \ldots, s_t\}$ and $D' \in R^{m' \times t}$ is a coefficient matrix for all slack variables in all constraints: for each *equality constraint*, all the elements in the corresponding row of D' are 0; for the *jth inequality constraint* ($j = 1 \ldots \ell'$), in the corresponding row of D', the coefficients of t_j slack variables are *randomly generated positive real numbers* while the remaining elements in such row are 0. In horizontally partitioned linear program, every entity $i = 1 \ldots p$ holds constraints $A'_{I_i.} x + D'_{I_i} x'_s = b'_{I_i}$ ($A'_{I_i.} \in R^{m'_i \times n}, b'_{I_i} \in R^{m'_i}$) where $D'_{I_i} \in R^{m'_i \times t}$ is the rows of D' associated with entity i's all inequality and equality constraints, thus:

$$D' = \begin{bmatrix} D'_{I_1} \\ \vdots \\ D'_{I_p} \end{bmatrix} \in R^{m' \times t}, x'_s = \begin{bmatrix} s_1 \\ \vdots \\ s_t \end{bmatrix}, \quad t > m' > m \geq \ell \text{ and } m' \geq \ell' > \ell \tag{7}$$

For example, entity 1 owns two constraints $3x_1 + 7x_2 + 2x_3 = 20$ and $x_1 + x_2 \leq 9$ while entity 2 owns one constraint $x_1 + 4x_2 + x_3 \leq 17$. Entity 1 creates an artificial constraint $2.9x_1 + 6.3x_2 + 2x_3 \leq 20$ while entity 2 creates an artificial constraint $0.5x_1 + 2x_2 + 0.3x_3 \leq 8.5$. Two slack variables are determined for converting each of

both entities' inequality constraints to equality. Thus, D' can be generated as ($m = 3$, $\ell = 2, m' = 5, \ell' = 4, t = 8$, only t and m' are possibly disclosed to all entities):

$$D' = \begin{bmatrix} D'_{I_1} \\ D'_{I_2} \end{bmatrix} = \left[\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2.5 & 3.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 1.4 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1.9 & 3.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8.5 & 5.7 \end{array}\right] \in R^{5\times 8}$$

where entity 1 and 2 own the first three rows and last two rows respectively.

Proposition 1 *If* $(x^*, (x'_s)^*)$ *is an optimal solution to linear program* (6)*, then* x^* *is optimal for linear program* (5).

Proof Suppose that x^* is not optimal to linear program (5) and $\exists \hat{x}$ such that $c^T\hat{x} < c^T x^*$ in linear program (5). Clearly, $x = \hat{x}$ satisfies $Ax + Mx_s = b$. Since A', M', b' are expanded from A, M, b by adding artificial constraints (without distorting the feasible region), $x = \hat{x}$ also satisfies $A'x + M'x_s = b'$ with the slack variables $x_s = \hat{x_s}$ (assuming that $\hat{x_s} = \{h_1 \ldots h_{\ell'}\}$ are the non-negative values to make the slack form $A'\hat{x} + M'\hat{x_s} = b'$ hold based on the satisfied standard form $A'\hat{x} \le b'$).

Hence, if $\exists M'\hat{x_s} = D'x'_s$, solution $x = \hat{x}$ (and t slack variables x'_s) is also feasible for $A'x + D'x'_s = b'$. Indeed, $M'\hat{x_s}$ and $D'x'_s$ are two length-m' vectors that convert all ℓ' inequalities in $A'x \le b'$ to equalities by two different sets of slack variables x_s and x'_s (note that $M'\hat{x_s}$ and $D'x'_s$ correspond to the same index of the constraints $A'x \le b'$). Taking D' and x'_s as below, we can have that $M'\hat{x_s} = D'x'_s$.

First, for all *equalities* in $A'x \le b'$, the corresponding numbers in $M'\hat{x_s}$ and $D'x'_s$ are 0 (which are equal for any D' and x'_s). Second, for the jth *inequality* ($j = 1 \ldots \ell'$) in $A'x \le b'$, since the corresponding number in $M'\hat{x_s}$ is a non-negative number h_j, taking the jth inequality's t_j slack variables (in x'_s) as any t_j non-negative values such that the linear combination of them (with random positive coefficients in D') equals h_j, can make the corresponding number in $M'\hat{x_s}$ and $D'x'_s$ equal. Similarly, given any D' per (7), we can always find x'_s (including all $t = \sum_{j=1}^{\ell'} t_j$ slack variables) to ensure $M'\hat{x_s} = D'x'_s$ for all constraints (since all the ℓ' inequalities do not share any slack variable). Hence, $x = \hat{x}$ is always feasible for $A'x + D'x'_s = b'$.

Thus, x^* is not optimal to linear program (6) since $c^T\hat{x} < c^T x^*$ and $x = \hat{x}$ satisfy all the constraints in linear program (6). This contradiction completes the proof. □

Similarly, we can transform linear program (6) to the inference–proof linear program by pre-multiplying a $k \times m'$ random matrix B to both sides of the constraints:

$$\begin{aligned} min: \quad & c^T x \\ s.t. \quad & BA'x + BD'x'_s = Bb', \quad x \ge 0, \quad x'_s \ge 0 \end{aligned} \tag{8}$$

where the random matrix $B \in R^{k\times m'}$, $k \ge m'$ and $rank(B) = m'$. Note that the constraint matrix A and D of [1,2] are replaced by A' and D' respectively (where the size and rank of the matrices are increased without changing the feasible region).

Proposition 2 [1] *Let* $k \ge m'$ *for the random matrix* $B \in R^{k\times m'}$*. The secure linear program* (8) *is solvable if and only if the linear program* (6) *is solvable in which case the solution sets of the two linear programs are identical.*

Therefore, linear programs (5) and (8) have the identical optimal solution.

3.2 Inference–proof algorithm

Algorithm 1 Secure Sum Algorithm

Input: Entity i holds R_i (matrix, vector or real number) where $i = 1 \ldots p$, and $\forall i$, R_i has the same size
Output: $R = \sum_{i=1}^{p} R_i$

1: $R \leftarrow 0$.
2: **for** each entity $i = 1 \ldots p$ **do**
3: generates a privately held random (matrix, vector or real number) R'_i with the same size as R_i.
4: $R \leftarrow R + R_i + R'_i$.
5: **end for**
6: **for** each entity $i = 1 \ldots p$ **do**
7: $R \leftarrow R - R'_i$.
8: **end for**
{Note that R_1 (or R_2) might be computed by entity 2 (or 1) if $p = 2$, but this does not affect the security of Algorithm 2 (see detailed analysis later on)}

We let every entity $i = 1 \ldots p$ locally create $(m'_i - m_i)$ artificial constraints, convert each of its inequality constraints to equality with one or more slack variables and compute $B._{I_i} A'_{I_i.}$, $B._{I_i} b'_{I_i}$, $B._{I_i} D'_{I_i}$ with its privately held matrices $B._{I_i}$ and D'_{I_i}. To obtain linear program (8), we can securely sum $BA' = \sum_{i=1}^{p} B._{I_i} A'_{I_i.}$, $BD' = \sum_{i=1}^{p} B._{I_i} D'_{I_i}$ and $Bb' = \sum_{i=1}^{p} B._{I_i} b'_{I_i}$ over p entities with Algorithm 1.

Algorithm 1 ensures that only the sum of the distributed matrix products shares are disclosed. Since $B._{I_i} A'_{I_i.}$, $B._{I_i} D'_{I_i}$ and $B._{I_i} b'_{I_i}$ are unknown to other entities (the special case $p = 2$ is discussed later on), the equations with real variables cannot be established. Thus, besides hiding m, the secure sum algorithm (Algorithm 1) can also reduce the inference probability, compared with revealing every share of the matrix products to public in the prior work [1,2].

Algorithm 2 Inference-proof Algorithm to PPHPLP

1: Each entity $i = 1 \ldots p$ locally creates some artificial constraints, thus $A_{I_i.} \in R^{m_i \times n}$ and $b_{I_i} \in R^{m_i}$ are expanded to $A'_{I_i.} \in R^{m'_i \times n}$ and $b'_{I_i} \in R^{m'_i}$ respectively where $m'_i > m_i$ and $rank(A'_{I_i.}) > rank(A_{I_i.})$.
2: Each entity $i = 1 \ldots p$ locally converts each of its inequality constraints into equality with one or more slack variables where the total number of slack variables $T_i \geq m'_i$.
3: All p entities securely sum $m' = \sum_{i=1}^{p} m'_i$ and $t = \sum_{i=1}^{p} T_i$ where $t > m' > m$ (Algorithm 1).
4: Each entity $i = 1 \ldots p$ generates its privately held random matrix $D'_{I_i} \in R^{m'_i \times t}$ for the slack variables generated in Step 2 (as discussed in Section 3.1), where m'_i is the number of rows held by entity i in D' as shown in (7).
5: All p entities agree on an integer value for $k \geq m'$, where k is the number of rows of the random matrix $B \in R^{k \times m'}$ as defined in (1) and (2).
6: Each entity $i = 1 \ldots p$ generates its privately held random matrix $B._{I_i} \in R^{k \times m'_i}$, where m'_i is the number of columns held by entity i in $B = [B._{I_1}, B._{I_2}, \ldots, B._{I_p}] \in R^{k \times m'}$.
7: Each entity $i = 1 \ldots p$ locally computes the matrix products $B._{I_i} A'_{I_i.}$, $B._{I_i} D'_{I_i}$ and $B._{I_i} b'_{I_i}$.
8: All p entities securely sum the matrix products (Algorithm 1) to get the constraint matrix of the inference-proof linear program (8):

$$BA' = [B._{I_1} A'_{I_1.} + B._{I_2} A'_{I_2.} + \cdots + B._{I_p} A'_{I_p.}] \in R^{k \times n}$$

$$BD' = [B._{I_1} D'_{I_1} + B._{I_2} D'_{I_2} + \cdots + B._{I_p} D'_{I_p}] \in R^{k \times t}$$

and the right hand side vector for linear program (8):

$$Bb' = [B._{I_1} b'_{I_1} + B._{I_2} b'_{I_2} + \cdots + B._{I_p} b'_{I_p}] \in R^{k}$$

9: Solve LP (8) to get the optimal solution $(x^*, (x'_s)^*)$, thus x^* is optimal for LP (5).

Next, we present the detailed steps of the inference–proof algorithm in Algorithm 2. While solving linear program (8), entity $i = 1 \ldots p$ can only learn:

- the value m' (securely summed in Step 3), which can be far greater than m.
- the value of $rank(BA') = rank(A')$, which can be far greater than m and $rank(A)$ ($m = rank(A)$ if all the rows in A are linearly independent).
- all p entities' total number of slack variables t (securely summed in Step 3), which can be far greater than m and ℓ ($m = \ell$ if all the constraints are inequalities).

Note that in two-entity case ($p = 2$), even though $B._{I_1} A'_{I_1.}$, $B._{I_1} D'_{I_1}$ and $B._{I_1} b'_{I_1}$ can be computed from Algorithm 1 by entity 2 (or vice-versa), entity 1's number of constraints m_1 can be only inferred as any number no greater than $m' - m'_2$ or $t - m'_2$ for entity 2 in Algorithm 2 since entity 2 knows at most m', t and m'_2 (m' and t can be sufficiently large). With unknown m_1, entity 2 cannot further infer entity 1's privately held constraints (or vice-versa). Thus, Algorithm 2 is still secure when $p = 2$.

Overall, Algorithm 2 reveals significantly less information than [1,2]: $m, \ell, \{i = 1 \ldots p, m_i, B._{I_i} A_{I_i.}, B._{I_i} D_{I_i}, B._{I_i} b_{I_i}\}$, BA, BD, Bb and $rank(A)$. etc are not disclosed to public. Moreover, every entity's private constraints cannot be inferred by formulating equations with real variables since the number of constraints are unknown anymore. To further reduce the inference risk, every entity $i = 1 \ldots p$ can locally increase m'_i and T_i by adding more artificial constraints and slack variables.

3.3 Efficiency and security tradeoff discussion

Recall that the transformation approach in [1,2] tends to suffer high risk of inference attack in some vulnerable cases (e.g. $p = 2$). Indeed, Algorithm 2 can resolve the inference attack for any horizontally partitioned linear program held by p entities, but expands the problem size and thus trades off some efficiency for enhanced security. We now discuss this efficiency and security tradeoff for different PPHPLPs.

First, if $p = 2$ (two-entity), clearly, we should utilize Algorithm 2 to tackle the potential inference attack since every entity's number of constraints m_1 or m_2 are explicitly disclosed to each other in [1,2].

Second, if $p > 2$ (multi-entity), Algorithm 2 is also effective to enhance security. In this case, since the probability of inferring any entity i's number of constraints m_i by another entity $j \neq i$ with its known m_j is quite low: m_i can be any integer in $[1, m - m_j - (p - 2)]$ (assuming that each of the remaining $p - 2$ entities should include at least 1 constraint), the probability of inferring entity i's constraints is even lower. Note that the prior approach [1,2] remains applicable with better efficiency.

However, for any $p > 2$, there exist some special cases which are still vulnerable if directly applying the approach in [1,2]: e.g. $p = 4, m = 9, m_j = 6$, entity j can infer that each of the remaining three entities has exactly only 1 constraint, and then their constraints can be reconstructed by the inference attack described in Section 2.2 if their transformed share are explicitly revealed [1,2]. Therefore, we should adapt our secure sum into the prior PPHPLP algorithm (simply replacing step (III) in Algorithm 3.1 in [1] with Algorithm 1) for keeping every entity's transformed share private (note that this has not resolved the inference on every entity's number of constraints yet).

In summary, if stronger security is desired and the slightly declined efficiency is tolerable, it would be better to apply our inference–proof algorithm (Algorithm 2) for PPHPLP co-held by any number of entities p. If more efficiency is required and every entity does not care the potential inference on its number of constraints when $p > 2$, we can integrate Algorithm 1 and the prior PPHPLP algorithm [1,2].

4 Computational results

We conduct experiments similar to those in [1] to demonstrate the computation cost with MATLAB on a 6G-RAM PC. Specifically, we generate $m = 400$ constraints for $n = 1{,}000$ variables ($\ell = 200$ inequality constraints) for linear program (5): $A \in R^{400\times 1{,}000}$ is randomly generated where every element in A is uniformly distributed in the interval $[-50, 50]$; $b \in R^{400\times 1}$ is randomly generated in the interval $[300, 500]$. The linear program is horizontally partitioned among 3 entities: $m_1 = 100, m_2 = 100, m_3 = 200$ (half of each entity's constraints are inequalities).

To formulate linear program (8), we generate 100, 100 and 200 artificial constraints for three entities respectively: given an equality or inequality constraint, we reduce a positive coefficient or increase a negative coefficient of the constraint while retaining the same righthand side value (this is not the only way to create artificial constraints). Thus, we have $m'_1 = 200$, $m'_2 = 200$, $m'_3 = 400$ and $\ell' = 600$. We let every entity utilize two slack variables to convert each of its local inequality constraints to equality, then $t = 1200$. Every entity generates $D'_{I_1} \in R^{200\times 1200}$, $D'_{I_2} \in R^{200\times 1200}$ and $D'_{I_3} \in R^{400\times 1200}$ where the positive random numbers in them are uniformly distributed in the interval (0, 1]. We let $k = 1{,}000 > m' = 800$ and generate three random matrices $B_{\cdot I_1} \in R^{1{,}000\times 200}$, $B_{\cdot I_2} \in R^{1{,}000\times 200}$ and $B_{\cdot I_3} \in R^{1{,}000\times 400}$ with elements uniformly distributed in the interval [0, 1]. Thus, we have $A' \in R^{800\times 1{,}000}$, $D' \in R^{800\times 1200}$, $B \in R^{1{,}000\times 800}$ and $b' \in R^{800\times 1}$.

We solve linear programs (5) and (8), and compare the optimal solutions of them – which are identical. The computation time was 16.583s for the inference–proof linear program (8) which is comparable to 8.34s for linear program (5).

5 Conclusion and future work

We have detected a potential inference attack to the recent work on privacy-preserving horizontally partitioned linear programs [1,2]—the complete constraint can be inferred by adversaries in some vulnerable cases. We presented an approach to tackle the above issue for PPHPLP with arbitrary number of equality and inequality constraints, and also discussed how to select an appropriate approach for different PPHPLPs according to the security and efficiency tradeoff. In the future, we plan to look at how to securely solve some other interesting collaborative optimization problems (e.g. traveler salesman problem [13], graph coloring [14]).

References

1. Mangasarian, O.L.: Privacy-preserving horizontally partitioned linear programs. Optim. Lett. **6**(3), 431–436 (2012)

2. Li, W., Li, H., Deng, C.:. Privacy-preserving horizontally partitioned linear programs with inequality constraints. Optim. Lett. doi:10.1007/s11590-011-0403-2
3. Mangasarian, O.L.: Privacy-preserving linear programming. Optim. Lett. **5**(1), 165–172 (2011)
4. Li, J., Atallah, M.J.: Secure and private collaborative linear programming. In: CollaborateCom, pp. 1–8 (2006)
5. Vaidya, J.: Privacy-preserving linear programming. In: SAC, pp. 2002–2007 (2009)
6. Bednarz, A., Bean, A.N., Roughan, M.: Hiccups on the road to privacy-preserving linear programming. In: WPES '09, pp. 117–120 (2009)
7. Vaidya, J.: A secure revised simplex algorithm for privacy-preserving linear programming. In: AINA '09, pp. 347–354 (2009)
8. Hong, Y., Vaidya, J., Lu, H.: Efficient distributed linear programming with limited disclosure. In: DBSec, pp. 172–187 (2011)
9. Hong, Y., Vaidya, J., Lu, H.: Secure and efficient distributed linear programming. J. Comput. Secur. (2012, to appear)
10. Dreier, J., Kerschbaum, F.: Practical privacy-preserving multiparty linear programming based on problem transformation. In: PASSAT (2011)
11. Du, W.: A study of several specific secure two-party computation problems. Ph.D. thesis, Purdue University, West Lafayette (2001)
12. Feng, X., Zhang, Z.: The rank of a random matrix. Appl. Math. Comput. **185**(1), 689–694 (2007)
13. Sakuma, J., Kobayashi, S.: A genetic algorithm for privacy preserving combinatorial optimization. In: GECCO, pp. 1372–1379 (2007)
14. Hong, Y., Vaidya, J., Lu, H., Shafiq, B.: Privacy-preserving tabu search for distributed graph coloring. In: PASSAT, pp. 951–958 (2011)