Distributed Constraint Satisfaction and Optimization with Privacy Enforcement*

Marius C. Silaghi and Debasis Mitra
Florida Institute of Technology, Computer Sciences Department

Abstract

Several naturally distributed negotiation/cooperation problems with privacy requirements can be modeled within the distributed constraint satisfaction framework, where the constraints are secrets of the participants. Most of the existing techniques aim at various tradeoffs between complexity and privacy guarantees, while others aim to maximize privacy first [12, 7, 3, 4, 11]. In [7] we introduced a first technique allowing agents to solve distributed constraint problems (DisCSPs), without revealing anything and without trusting each other or some server. The technique we propose now is a dm times improvement for m variables of domain size d. On the negative side, the fastest versions of the new technique require storing of $O(d^m)$ big integers. From a practical point of view, we improve the privacy with which these problems can be solved, and improve the efficiency with which $\lfloor n-1/2 \rfloor$-privacy can be achieved, while it remains inapplicable for larger problems. The technique of [7] has a simple extension to optimization for distributed weighted CSPs. However, that obvious extension leaks to everybody sensitive information concerning the quality of the computed solution. We found a way to avoid this leak, which constitutes another contribution of this paper.

1. Introduction

CSPs can model problems like meeting scheduling, timetabling, stable matching, and resource allocation. One cannot always gather easily the parameters needed to formalize the problem. Some of the constraints may contain private information of different participants. We develop techniques that can be used by the agents in a distributed CSP to find a solution without losing any of their secrets, except for what is inherently revealed by the solution itself.

CSP A *constraint satisfaction problem* (CSP) is defined by three sets: (X, D, C). $X = \{x_1, ..., x_m\}$ is a set of variables and $D = \{D_1, ..., D_m\}$ is a set of domains such that x_i can take values only from $D_i = \{v_1^i, ..., v_{d_i}^i\}$. $C = \{\phi_1, ..., \phi_c\}$ is a set of constraints, ϕ_i involving an ordered subset $X_i = \{x_{i_1}, ..., x_{i_{k_i}}\}$ of the variables in X, $X_i \subseteq X$.

An assignment is a pair $\langle x_i, v_k^i \rangle$ meaning that the variable x_i is assigned the value v_k^i. ϕ_i constrains the legality of each combination of assignments to the variables in X_i. A tuple is an ordered set. The projection of a tuple ϵ of assignments over a tuple of variables X_i is denoted $\epsilon_{|_{X_i}}$. A solution of a CSP (X,D,C) is a tuple of assignments ϵ with one assignment for each variable in X such that each $\phi_i \in C$ is satisfied by $\epsilon_{|_{X_i}}$. CSPs do not model optimization requirements. An extension allowing for modeling some optimization concerns is given by Weighted CSPs.

Definition 1 *A Weighted CSP (WCSP) is defined by a triplet of sets (X, D, C) and a bound B. X and D are defined as in CSPs. In contrast to CSPs, C=$\{\phi_1, ..., \phi_c\}$ is a set of functions, $\phi_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow [0..b_i]$ where b_i is a maximal value (aka weight) for ϕ_i.*

Its solution is $\operatorname{argmin}_{\epsilon \in D_1 \times ... \times D_m} \sum_{i=1}^{c} \phi_i(\epsilon_{|_{X_i}})$, if the corresponding sum (aka. weight of solution) is smaller than B.

A Distributed CSP (DisCSP) is defined by four sets (A, X, D, C). A=$\{A_1, ..., A_n\}$ is a set of agents. X, D, C and the solution are defined like in CSPs. Each constraint ϕ_i is known only by one agent, being the secret of that agent.

Definition 2 (DisWCSP) *A Distributed Weighted CSP is defined by four sets (A, X, D, C) and a bound B. A, X, D are defined as for DisCSPs. In contrast to DisCSPs, the elements of C are functions $\phi_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow [0..b_i]$, like for WCSPs. Its solution is $\operatorname{argmin}_{\epsilon \in D_1 \times ... \times D_n} \sum_{i=1}^{c} \phi_i(\epsilon_{|_{X_i}})$, if the corresponding weight of solution is smaller than B.*

We assume that agents know the variables involved in the constraints of each other (variables can be falsely declared as involved). Agents want to avoid that others find details about their secret constraints. The *secrets* of an agent are the satisfiability/weight specified by her constraints for each combination. The solving protocol should avoid revealing any secret except for what is inherently leaked by the solution itself, namely that all agents accept the selected solution. A *leak* is an involuntary revelation of a secret.

*We thank Richard Wallace and Markus Zanker whose comments helped to clarify the paper.

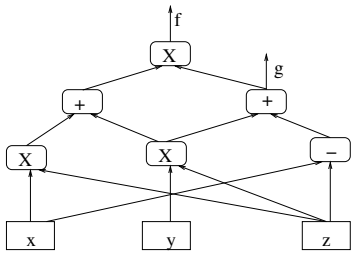

Figure 1. An arithmetic circuit: $g = yz+(x-z)$ **and** $f=(xz+yz)g$**.**

Example 1 *In a problem P, three persons Alice (A_1), Bob (A_2), and Carol (A_3) want to find a common place (x_1) and time (x_2) for meeting. x_1 is either Shanghai (S) or Halifax (H), i.e. $D_1 = \{S, H\}$. x_2 is either Monday (M) or Thursday (T), i.e. $D_2 = \{M, T\}$. Each of them has a secret constraint on the possible time and place of their meeting. The users know attached costs to the arrangements that they accept. The bound on the total cost of an accepted solution, B, is \$3000. We can consider the costs in units of \$500, such that we take B=6. Alice accepts only $\{\langle(H, M), \$500\rangle \langle(H, T), \$500\rangle\}$ which defines ϕ_1, such that $\phi_1(H, M)$=$\phi_1(H, T)$=1 and $\phi_1(S, M) = \phi_1(S, T)$=6, i.e. B. Bob accepts either of $\{\langle(S, M), \$1000\rangle, \langle(H, M), \$500\rangle, \langle(H, T), \$1000\rangle\}$, defining ϕ_2. $\phi_2(S, M)$=$\phi_2(H, T)$=2, $\phi_2(H, M)$=1, and $\phi_2(S, T)$=6. Carol's constraint is ϕ_3 defined as $\phi_3(S, M)$=6, $\phi_3(S, T)$=1, $\phi_3(H, M)$=$\phi_3(H, T)$=0.*

The problem is to find values for x_1 and x_2 minimizing the sum of ϕ_1, ϕ_2, and ϕ_3, such that it is lower than B, and without revealing anything else about ϕ_2 and ϕ_3 to Alice, about ϕ_1 and ϕ_3 to Bob, and about ϕ_1 and ϕ_2 to Carol.

Definition 3 (Arithmetic Circuit [1]) *An arithmetic circuit is a function f with one or several inputs and outputs, using solely the addition/subtraction and multiplication operations (see Figure 1). $\sum_{i=b}^{e} f(i)$ and $\prod_{i=b}^{e} f(i)$ can be part of an arithmetic circuit if b and e are public constants.*

Several multi-party techniques are known to compute general functions with secret inputs. These are mainly versions of oblivious evaluation of boolean circuits (boolean operations over $\{0, 1\}$), or arithmetic circuit evaluation [1]. But a DisWCSP is not a function. For a given input problem, a DisWCSP can have several solutions or no solution.

In [7] we proposed an algorithm for solving DisCSPs, called SecureRandomSolution. We will refer it from now on as MPC-DisCSP1 [9]. It uses evaluations of some arithmetic circuits as one of its building blocks. The algorithm described here, MPC-DisCSP2, is based on faster arithmetic circuits than the ones of MPC-DisCSP1. It also allows the n participating agents to securely find a solution by interacting directly without any external arbiters and without divulging any secrets. It is a *threshold scheme*, namely guaranteeing that no subset of t malicious agents that follow the protocol, $t<\lceil n/2 \rceil$, can find anything about others' problems except for what is revealed by the solution.

As it was suggested for MPC-DisCSP1, MPC-DisCSP2 can be extended to perform optimization in Distributed Weighted CSPs. The extension consists in first redesigning one of the basic arithmetic circuits involved in MPC-DisCSP2 such that the algorithm is enabled to find a solution with a predefined weight. Then one can simply find the optimal solution of the DisWCSP by scanning for a solution with weight 0, then weight 1, etc. until the first solution is found. However, this reveals to everybody the weight of the found solution! We propose a new technique called MPC-DisWCSP2 which reveals the weight of the solution only to a set of agents chosen by the participants, or to nobody.

Overview of MPC-DisCSP1. MPC-DisCSP1 is a multi-party computation technique. Former multi-party computations can solve securely certain functions, one of the most general classes of solved problems being the arithmetic circuits over fields. A Distributed CSP is not a function. A DisCSP can have several solutions for a given problem instance, or can even have no solution. Two of the three reformulations of DisCSPs as a function (see [8]) are relevant:

- i A function DisCSP^1() returning the first solution in lexicographic order, respectively an invalid valuation τ when there is no solution.
- ii A probabilistic function DisCSP() which picks randomly a solution if it exists, respectively returns τ when there is no solution.

For privacy purposes only the 2^{nd} alternative is satisfactory. DisCSP() only reveals what we usually expect to get from a DisCSP, namely *some* solution. DisCSP^1() intrinsically reveals more [10]. MPC-DisCSP1 implements DisCSP() in three phases:

1. The input DisCSP problem is shuffled in a cooperative way by reordering values (and eventually variables) randomly, by composing secret permutations from each participant agent.
2. A version of DisCSP^1() where operations performed by agents are independent of the input secrets, is computed by simulating arithmetic circuit evaluation with the technique in [1].
3. The solution returned by the DisCSP^1() at Step 2 is translated into the initial problem formulation using a transformation that is inverse of the shuffling at Step 1.

function value-to-unary-constraint2(v, M)
  $\{x_i\}_{0\leq i\leq M}, x_0{=}1, x_{i+1}{=}x_i * (v{-}i)$
  $\{y_i\}_{0\leq i\leq M}, y_M{=}1, y_{i-1}{=}y_i * (i{-}v)$
  $\{u_k\}_{0\leq k\leq M}, u_k = \frac{1}{k!(M-k)!}x_k y_k$, where $0! \stackrel{\text{def}}{=} 1$.
  Return u.

Algorithm 1: Transforming secret value $v{\in}\{0, ..., M\}$ to a shared secret unary constraint, i.e. $u[x]{=}1$ iff $u{=}v$.

Overview of Multi-party computations. In shuffling, we use $(+, \times)$-homomorphic encryption functions E_{K_E} : $D_P{\rightarrow}D_C$ i.e. respecting:

$$\forall m_1, m_2 \in D_P : E_{K_E}(m_1)E_{K_E}(m_2) = E_{K_E}(m_1 + m_2).$$

Some encryption functions take a randomizing parameter r. We write $E_i(m)$ instead of $E_i(m, r)$, to simplify the notation. A good example of a $(+, \times)$-homomorphic scheme with randomizing parameter is the Paillier encryption [5].

To destroy the visibility of the relations between the initial problem formulation and the formulation actually used in computations one can exploit random joint permutations that are not known to any participant. Here we reformulate the initial problem by reordering its parameters. Related permutations appeared in Chaum's mix-nets [2]. The shuffling is obtained by a chain of permutations (each being the secret of a participant) on the encrypted secrets.

The secure arithmetic circuit evaluation technique in [1] exploits Shamir's secret sharing [6]. This sharing is based on the fact that a polynomial $f(x)$ of degree $t{-}1$ with unknown parameters can be reconstructed given the evaluation of f in at least t distinct values of x, using Lagrange interpolation. Instead, absolutely no information is given about the value of $f(0)$ by revealing the valuation of f in any at most $t{-}1$ non-zero values of x. In order to share a secret number s to n participants $A_1, ..., A_n$, one first selects $t{-}1$ random numbers $a_1, ..., a_{t-1}$ that will define the polynomial $f(x) = s{+}\sum_{i=1}^{t-1}(a_i x^i)$. A distinct non-zero number k_i is assigned to each A_i. The value of the pair $(k_i, f(k_i))$ is sent securely to A_i. This is called a (t, n)-threshold scheme.

Once secret numbers are split and shared with a (t, n)-scheme, computations of an arbitrary agreed function of a certain class can be performed over the shared secrets, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $t{-}1$) [1]. For [6]'s technique, one knows to perform additions and multiplications when $t \leq (n + 1)/2$.

2. New Arithmetic Circuits for DisCSP^1()

The main building block of DisCSP^1() consist of evaluating some arithmetic circuits. It is for this step that we are proposing a simpler and faster version. An implementation of DisCSP^1() can be easily obtained by checking all tuples until one satisfies all the constraints. Such a solution has a number of operations dependent on the secret constraints of the problem. This is why it cannot be used in DisCSP().

Consider the CSP $P{=}(X, D, C)$. One can interpret the constraints of C as functions with results in the set $\{0, 1\}$ (0 is infeasible and 1 is feasible). The solutions of P are the tuples of assignments ϵ^* (of type $\langle(x_1, v^1_{\epsilon^1}), ..., (x_m, v^m_{\epsilon^m})\rangle$) with $\prod_{\phi_k\in C}\phi_k(\epsilon^*_{|_{X_k}}){=}1$. The size of the search space (i.e. total number of tuples) is $\Theta = \prod_{k=1}^{m} d_k$. Let us detail now MPC-DisCSP2. If $p(\epsilon) = \prod_{\phi_k\in C}\phi_k(\epsilon_{|_{X_k}})$, and ϵ_k denotes the k^{th} tuple in the lexicographic order, then define:

$$\begin{aligned} h_1(P) &= 1 \\ h_i(P) &= h_{i-1}(P) * (1 - p(\epsilon_{i-1})) \end{aligned}$$

The index of the lexicographically first solution can be computed by accumulating the weighted terms of the h series:

$$id(P) = \sum_{i=1}^{\Theta} i * p(\epsilon_i) * h_i(P) \qquad (1)$$

A result of 0 means that there is no solution. The cost of this computation is $(c + 1)\Theta$ multiplications of secrets, $O(md)$ times less than the technique in MPC-DisCSP1, which is $O((cm + m^2)d^{m+1})$, where $d{=}\max_i(d_i)$.

One can then compute the values of the different variables in the found solution. We first transform the index id of the solution computed with Equation 1 into a shared vector S, of size Θ where only the id^{th} element is 1 and all other elements are 0. This is achieved using Equation 2. The technique for transforming the solution to a vector, shown in Algorithm 1, has $3M$ multiplications, M less than Algorithm **value-to-unary-constraint1**, proposed in [7].

The value of the u^{th} variable in the t^{th} tuple of the search space is $\eta_u(t)$, computed with Equation 3. An arithmetic circuit, $f_i(P)$, (see Equation 4), can now be used to compute the value of each variable x_i in the solution.

$$S{=}\text{value-to-unary-constraint2}(id{-}1, \Theta{-}1) \qquad (2)$$

$$\eta_u(t) = \lfloor (t-1)/\prod_{k=1}^{u-1} d_k \rfloor \mod d_u \qquad (3)$$

$$f_i(P) = \sum_{t=1}^{\Theta}(\eta_i(t) + 1) * S[t{-}1] \qquad (4)$$

The space required for computing S is $O(d^m)$. This can be reduced by not reusing intermediary results in Algorithm 1 and computing S on demand during the evaluation of f functions, but with efficiency losses of $O(d^{2m})$ rounds of messages. We call this circuit, DisCSP2^1().

Example 2 *Consider a DisCSP induced by Example 1, i.e.* $p(S, M){=}0$, $p(H, M){=}1$, $p(S, T){=}0$, $p(H, T){=}1$. $h_1(P){=}1$, $h_2(P){=}1$, $h_3(P){=}0$, $h_4(P){=}0$.

The index of the solution is computed with Equation 1, yielding id(P)=2. This is used according to Equation 2 to generate the vector S=$\{0,1,0,0\}$.

The vector S is used to compute the values of the variables in the solution, using Equations 3 and 4:
$\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=0, $\eta_1(4)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1, $\eta_2(4)$=1. $f_1(P)$=2, $f_2(P)$=1.

This signifies that the solution chosen by this arithmetic circuit is x_1=Halifax and x_2=Monday.

3. Computation of DisCSP()

Revealing the first solution, ϵ_0, in a lexicographic order leaks two distinct things: ϵ_0 is a solution (or at least that the elements communicated to each participant are part of a solution ϵ_0), and there exists no solution lexicographically ordered before ϵ_0. To avoid the second leak, MPC-DisCSP2 (similar to MPC-DisCSP1) returns a solution picked randomly from the existing solutions by rephrasing the input DisCSP with a hidden permutation after sharing its secrets.

MPC-DisCSP2's mix-net for reordering shared secret DisCSPs. MPC-DisCSP2 shuffles the DisCSP's domains (and eventually variables). Each agent chooses a random secret permutation π_i for each domain D_i:

$$\pi_i : \quad [1..d_i] \rightarrow [1..d_i], \qquad i \in [1..m]$$

The secret shares, of the {0,1} value associated by the extensional representation of each constraint ϕ to a tuple, are encrypted with the owner's publick key and then are serialized according to the current lexicographic order on domains. The serialized encrypted constraints are passed to each agent in a predefined order, each agent A_i shuffling them according to her secret permutations π_i (mix-net).

To avoid that agents get a chance to learn the final permutation by matching final shares with the ones they generated, a randomization step is applied at each shuffling. Each agent applies a randomization step on the set of shares for each secret constraint value, by adding corresponding Shamir shares of a 0 (a distinct sharing for each randomization). Because of the encryption, this randomization step is based on $(+, \times)$-homomorphic encryption, multiplying encryptions of shares to obtain encryptions of their sums [8].

Decoding the solution After DisCSP[1] is run on the shared problem shuffled as shown by the previous technique, the shares of the results of functions f (processed with Equation 5) have to be revealed without revealing the permutation. Randomization steps are performed as for encoding. Therefore, each agent A_k generates md random sets of shares of zero, $z_k^j[t]$ being A_j's share of the t^{th} zero.

The vectors $\{\langle\{E_j(f_i'^j[t])\}_{t\in[1..d_i]}, j\rangle\}_{i\in[1..m]}$, for each j, are sent backward through the mix-net, where $f_i'^j[t]$ is A_j's share for $f_i'[t]$. When A_k receives $\{\langle\{E_j(f_i'^j[t])\}_{t\in[1..d_i]}, j\rangle\}_{i\in[1..m]}$, it generates and sends to A_{k-1} the vector $\{\langle\pi_i^{-1}(\{E_j(f_i'^j[t])E_j(z_k^j[(i-1)d+t])\}_{t\in[1..d]}), j\rangle\}_{i\in[1..m]}$. A_1 broadcasts them.

$$f_i' = \text{value-to-unary-constraint2}(f_i - 1, d_i) \qquad (5)$$

Complexity The total number of messages that have to be sent with MPC-DisCSP2 is $3n$ (for shuffling), $n^2(c+1)d^m$ for Equation 1, $n^2 3d^m$ for Equation 2, $n^2 m 3d$ for m Equations 5, and n^2+2n for decoding the solution. The total number of messages is $n^2(d^m(c+4)+3md+1)+5n$.

Many of these messages can be sent in parallel. Namely, the number of needed rounds for arithmetic circuit evaluation is given by the depth of the circuit [1]. For our circuits, the depth is $\log_2(c)$ for p, $\log_2(\Theta)$ for id(P), $\log_2(\Theta)$ for value-to-unary-constraint2, and 1 for f. The total number of rounds is therefore O($m\log(d)+\log(c)$). Nevertheless, this parallelism requires $c\Theta(\Theta+1)/4$, i.e. $O(cd^{2m})$, concurrent messages in the first round.

4. MPC-DisWCSP2

The weak extension to distributed weighted CSPs. Let $q(\epsilon)=\sum_{\phi\in C}\phi(\epsilon)$. A solution of a CSP (X,D,C) is a valuation ϵ with $q(\epsilon)$=c. For addressing Distributed WCSPs, the function p has to be further adapted as follows. Now the maximum value of $q(\epsilon)$ is no longer c, but $b=\sum_{i=1}^{c} b_i$. We still want to isolate solutions ϵ whose $q(\epsilon)$ is some value, x_0 (actually we now need the minimal such x_0 allowing for a solution). This is achieved by a $p(\epsilon)$ defined as:

$$p(\epsilon) = \frac{\prod_{i=0}^{x_0-1}(q(\epsilon)-i)\prod_{i=x_0+1}^{b}(i-q(\epsilon))}{x_0!(b-x_0)!}. \qquad (6)$$

A solution with the lowest weight for a DisWCSP can be found by iterating MPC-DisCSP2 with the Definition 6 for p, for x_0 increasing from 0 to $B-1$. Note that this technique reveals to everybody the weight of the solution, i.e. the sum of constraint weights in the solution.

MPC-DisWCSP2: Solving a DisWCSP while hiding the weight of the solution. To hide the weight of the solution to a DisWCSP we must hide the number of rounds needed to find it with the weak extension. Therefore we will perform all the B possible rounds, keeping secret the round where a solution is found. A new set of vectors of secrets $\{w_i^j\}_i$ are defined, w_i^j holds the value of x_i in the best solution found in rounds 0 to j.

$$w_i^j \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } j=-1 \\ f_i(P) & \text{if } w_1^{j-1}=0 \\ w_i^{j-1} & \text{if } w_1^{j-1}\neq 0 \end{cases}$$

This can be computed with (for $i \in [1..d]$ and $j \in [1..(B-1)]$):

$$\begin{aligned} w_i^{-1} &= 0 \\ w_i^j &= w_i^{j-1}(1 - \frac{\prod_{k=1}^{d}(k - w_1^{j-1})}{d!}) \\ &\quad + f_i(P)\frac{\prod_{k\in[1..d]}(k - w_1^{j-1})}{d!} \end{aligned}$$

MPC-DisWCSP2 also consists in three phases:

1. First the DisWCSP is shared and then shuffled through the mix-net in the same way as it was done with the DisCSP (except that the values assigned by the constraint ϕ_k to tuples are in $[0..b_k]$ rather than $\{0,1\}$).

2. The vector $\{w_i^B\}_{i\in 1..m}$ is computed by iteratively building the vectors $\{w_i^j\}_{i\in 1..m}$ for j increasing from 0 to $B-1$. The computation in each iteration j is performed according to the arithmetic circuit DisCSP2[1] but with the new definition of p and with $x_0{=}j$. It is followed by a secure evaluation of $\{w_i^j\}_{i\in 1..m}$.

3. The solution is decoded and distributed as in MPC-DisCSP2, except that the solution vectors are the ones containing the results of the functions w_i^{B-1}.

The complexity of MPC-DisWCSP2 is B times higher than the complexity of MPC-DisCSP2. For the most parallel version of MPC-DisCSP2, the number of rounds increases with $B \log d$, compared to the weak extension.

In MPC-DisWCSP2 nobody can learn the total weight of the solution. In some problems one may nevertheless want to let some particular agents learn the total weight of the solution, while the rest of the agents should not learn it. This can be achieved by computing at the end of the second phase the first element of the solution vector according to:

$$\begin{aligned} w_0 &= \sum_{k\in[1..(B-1)]} k(1 - \frac{\prod_{k_1\in[1..d]}(k_1 - w_1^k)}{d!}) \\ &\quad \frac{\prod_{k_2\in[1..d]}(k_2 - w_1^{k-1})}{d!} \end{aligned}$$

The single non-zero term in the summation defining w_0 is for the round k where w_1^k is for the first time non-zero. w_0 specifies the weight of the solution and after decoding, is revealed only to the agents that should learn it.

Example 3 *Let us see a full example of how this arithmetic circuit is applied to Example 1, (assuming the secret shuffling does not change any order).*
w_1^{-1}=0, w_2^{-1}=0, $\eta_1(1)$=0, $\eta_1(2)$=1, $\eta_1(3)$=0, $\eta_1(4)$=1. $\eta_2(1)$=0, $\eta_2(2)$=0, $\eta_2(3)$=1, $\eta_2(4)$=1.

$x_0 = 0$*: p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=0.* $h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=1, $h_4(P)$=1.
S={*0,0,0,0*}*, id(P)=0.* $f_1(P)$=0, $f_2(P)$=0. w_1^0=0, w_2^0=0.

$x_0 = 1$*: p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=0. ...* $f_1(P)$=0, $f_2(P)$=0. $w_1^1 = 0$, $w_2^1 = 0$.

$x_0 = 2$*: p(S,M)=0, p(H,M)=1, p(S,T)=0, p(H,T)=0.* $h_1(P)$=1, $h_2(P)$=1, $h_3(P)$=0, $h_4(P)$=0.
S={*0,1,0,0*}*, id(P)=2.* $f_1(P)$=2, $f_2(P)$=1. w_1^2=2, w_2^2=1.

$x_0 = 3$*: p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=1.*
S={*0,0,0,1*}*, id(P)=4.* $f_1(P)$=2, $f_2(P)$=2. w_1^3=2, w_2^3=1.

$x_0 = 4$*: p(S,M)=0, p(H,M)=0, p(S,T)=0, p(H,T)=0. ...* $w_1^4 = 2$, $w_2^4 = 1$. ...
The iteration ends computing for x_0 equal to the maximum admissible cost for P (which according to Definition 2 is B-1). $w_1^5 = 2$, $w_3^5 = 1$. $w_0 = 2$.

This signifies that the solution chosen by this arithmetic circuit is x_1=Halifax and x_2=Monday.

Conclusions. We presented a technique where agents that need to cooperate and whose problems can be modeled as CSPs can find a random solution without leaks of additional information about their constraints. We also show how the technique can be extended to perform optimization in Distributed Weighted CSPs. In particular we find a way to avoid the privacy leaks concerning the solution weight, as exhibited by the obvious approach.

References

[1] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *STOC*, pages 1–10, 1988.

[2] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Com. of ACM*, 24(2):84–88, 1981.

[3] B. Faltings. Incentive compatible open constraint optimization. In *Electronic Commerce*, 2003.

[4] J. Liu, H. Jing, and Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136:101–144, 2002.

[5] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[6] A. Shamir. How to share a secret. *Comm. of the ACM*, 22:612–613, 1979.

[7] M. Silaghi. Arithmetic circuit for the first solution of distributed CSPs with cryptographic multi-party computations. In *IAT*, Halifax, 2003.

[8] M. Silaghi. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *IJCAI-DCR*, 2003.

[9] M. Silaghi. A suite of secure multi-party algorithms for solving DisCSPs. Technical Report CS-2004-04, FIT, 2004.

[10] M. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a DisCSP on privacy loss. In *AAMAS*, 2004.

[11] R. Wallace and M. Silaghi. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*, 2004.

[12] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.