A faster technique for distributed constraint satisfaction and optimization with privacy enforcement

Marius C. Silaghi
Florida Institute of Technology

January 16, 2004

Technical Report CS-2004-01

Abstract

A problem that received recent attention is the development of negotiation/cooperation techniques for solving naturally distributed problems with privacy requirements. An important amount of research focused on those problems that can be modeled with distributed constraint satisfaction, where the constraints are the secrets of the participants. Distributed AI develops techniques where the agents solve such problems without involving trusted servers. Some of the existing techniques aim for various tradeoffs between complexity and privacy guarantees [MTSY03], some aim only at high efficiency [ZM04], while others aim to offer maximal privacy [Sil03].

While the last mentioned work achieves an important level of privacy, it seems to be very slow. The technique we propose builds on that work, maintaining the same level of privacy, but being an order of magnitude faster, reaching the optimality in efficiency for this type of privacy. Unfortunately, all the versions of the new technique have an exponential space requirement, namely requiring the agents to store a value for each tuple in the search space. However, all existing techniques achieving some privacy were also applicable only to very small problems (typically 15 variables, 3 values), which for our technique means 14MB of required memory. Practically speaking, it improves the privacy with which this problems can be solved, and improves the efficiency with which $n/2$-privacy can be achieved, while remaining inapplicable for larger problems.

1 Introduction

Someone's private concerns can often be formulated in a general framework such as constraint satisfaction problems (i.e. where everything is modeled by either variables, values, or constraints on those variables) and then can be solved with any of the applicable CSP techniques. Often, one has to also find agreements with the other agents for a solution from the set of possible valuations of shared resources that satisfy her subproblem. The general framework modeling this kind of combinatorial problems is called Distributed Constraint Satisfaction.

In practice one also meets optimization problems. Distributed Weighted CSPs (DisWCSPs) is a general formalism that can model distributed problems with some optimization requirements. Now we introduce the distributed Weighted Constraint Satisfaction Problem and present shortly the *SecureRandomSolution* algorithm as well as the modifications proposed in this work. The new technique is called MPC-DisCSP2. Its exponential logic time complexity can be made logarithmic by parallelism. The technique is extended for solving DisWCSPs.

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. $C = \{f_1, ..., f_c\}$ is a set of constraints such that f_i is a predicate over an ordered subset X_i of the variables in X, $X_i \subseteq X$. An assignment is a pair $\langle x_i, v\rangle$ meaning that the variable x_i is assigned the value v. f_i specifies the legality of each combination of assignments to the variables in X_i.

A tuple is an ordered set. The projection of a tuple t of assignments over a tuple of variables X_i is denoted $t|_{X_i}$. A solution of a CSP $(X,\ D,\ C)$ is a tuple of assignments t with one assignment for each variable in X (i.e. $t \in D_1 \times ... \times D_m$) such that all the $f_i \in C$ are satisfied by $t|_{X_i}$.

Constraint Satisfaction Problems (CSPs) do not model optimization requirements. An extension allowing for modeling optimization concerns is given by Weighted CSPs.

Definition 1 ([Lar02]) *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=$\{f_1, ..., f_c\}$ is a set of functions, $f_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow [0..B]$ where k_i is the arity of k_i, and B is a maximal weight considered unacceptable for the solution.*

Its solution is $\underset{t \in D_1 \times ... \times D_m}{\text{argmin}} \sum_{i=1}^{c} f_i(t|_{X_i})$, *if its weight 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 predicate (aka *constraint*) f_i is known only by one agent.

Definition 2 (DisWCSP) *A Distributed Weighted CSP is defined by four sets (A, X, D, C). A, X, D are defined as for DisCSPs. In contrast to DisCSPs, the elements of C are functions $f_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow [0..B]$.*

Its solution is $\underset{t \in D_1 \times ... \times D_n}{\operatorname{argmin}} \sum_{i=1}^{c} f_i(t|_{X_i})$, *if its weight 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 when this is needed to hide the structure of the problem). Instead, agents want to avoid that others find details about the exact combinations allowed by the constraints that they enforce.

Several multi-party computation techniques are known to solve general functions, mainly versions of oblivious evaluation of boolean circuits, or arithmetic circuit evaluation. However, a DisWCSP is not a function. For a given input problem a DisWCSP can have several solutions or no solution at all. The author of [Sil03] proposed an algorithm for compiling DisCSPs to a secure solution, called SecureRandomSolution, that we refer here as MPC-DisCSP1 and that uses evaluations of some arithmetic circuits as one of its building blocks. The algorithm described here is an extension 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. This is a *threshold scheme*, guaranteeing that no subset of t malicious agents that follow the protocol (aka honest but curious agents [CGH00]), $t<(n+1)/2$, can find anything about others' problems except what is revealed by the solution.

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. But, this reveals to everybody the quality of the found solution! We then propose a new extension called MPC-DisWCSP2 which reveals the quality of the solution only to a set of agents chosen by the participants, or to nobody.

2 Overview of MPC-DisCSP

MPC-DisCSP1 uses general multi-party computation techniques. General multi-party computation techniques can solve securely only certain functions, one of the most general classes of solved problems being the arithmetic circuits. A Distributed CSP is not a function. A DisCSP can have several solutions for an input problem, or can even have no solution. Three reformulations of DisCSPs as a function 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 function DisCSP() which picks randomly a solution if it exists, respectively returns τ when there is no solution.

For privacy purposes only the 2nd definition is satisfactory. DisCSP() only reveals what we expect to get from a DisCSP, namely one solution. MPC-DisCSP1 proposes to implement DisCSP() in three phases:

1. The input DisCSP problem is jointly shuffled by reordering variables and values randomly and with permutations hidden from each subset of participant agents.
2. A version of DisCSP^1() where operations performed by agents are independent of the input secrets, is computed (e.g. by evaluating arithmetic circuits).
3. The solution returned by the DisCSP^1() at step 2 is translated into the initial problem definition using a transformation that is inverse of the shuffling at step 1.

At step 2 MPC-DisCSP1 requires a version of the DisCSP^1() function whose cost is independent of the input since otherwise the users can learn things like: *The returned solution is the only one as it was found after unsuccessfully checking all other valuations, all other valuations being infeasible.* However, the DisCSP^1() used by MPC-DisCSP1 is very complex and we propose a much simpler and faster solution.

3 Background

Secure evaluation of functions with secret inputs (where sometimes secret functions can also be treated as secret inputs and vice-versa) have been often addressed in literature. Several recent versions are based on (oblivious) boolean circuit evaluation [CDG88, AF88, Kil88, Hab88, GHY88]. Others, like MPC-DisCSP1, are based on arithmetic circuit evaluation.

MPC-DisCSP uses $(+, \times)$-homomorphic encryption functions $E_{K_E} : D_P \rightarrow D_C$ (D_P is the domain of the plaintext and D_C the domain of the ciphertext), 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. Sometimes 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 [Pai99].

To destroy the visibility of the relations between the initial problem formulation and the formulation actually used in computations one can exploit random joint permulations that are not known to any participant. Here we reformulate the initial problem by reordering the values and the variables. Such permutations appeared in Chaum's work and in Merritt's election protocol [Mer83, Cha81]. The shuffling is obtained by a chain of permutations (each being the secret of a participant) on the encrypted votes. n agents, $A_1, ..., A_n$, are ordered in a chain (called in the following, Merritt chain). The agents encrypt their votes with the public keys of all agents in the order of the chain, twice. The votes are then decrypted by the corresponding agents by passing the messages along the chain of agents, while each agent secretly shuffles all messages at each such step. The shuffling of each agent remains its secret.

Shamir's secret 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. This can be done 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. Therefore, in order to share a secret s to n participants $A_1, A_2, ..., A_n$, 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 participant A_i. Each participant A_i is then communicated over a secure channel (e.g. encrypted with E_i) the value of the pair $(k_i, f(k_i))$. This is called a (t, n)-scheme.

Once secret numbers are split and distributed with a (t, n)-scheme, computations of an arbitrary agreed function of a certain class can be performed over the distributed secrets, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $t{-}1$) [Yao82]. For [Sha79]'s technique, one knows to perform addition and multiplications when $t \leq (n + 1)/2$.

4 Arithmetic circuits for DisCSP^1()

The main building blocks 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 secrets of the problem and this is why [Sil03] claims that it cannot be used in DisCSP(). DisCSP^1() is a building block of DisCSP(). Consider the CSP $P = (X, D, C)$. One can interpret the predicates of C as functions with results in the set $\{0, 1\}$ (0 is infeasible and 1 is feasible). All domains are extended to d values. The solutions of P are the valuations ϵ (of type

```
function value-to-unary-constraint2($v$, $M$)
    1. Jointly, all agents build a vector $u$,
       $u = \langle u_0, u_1, ..., u_M \rangle$ with $3M{-}1$
       multiplications of secrets, by computing:
       1. the shared secret vector:
            $\{x_i\}_{0 \le i \le M}$, $x_0{=}1$, $x_{i+1}{=}x_i * (v{-}i)$
       2. the shared secret vector:
            $\{y_i\}_{0 \le i \le M}$, $y_M{=}1$, $y_{i-1}{=}y_i * (i{-}v)$
       then, $u_k = \frac{1}{k!(M-k)!} x_k y_k$, where $0! \overset{\text{def}}{=} 1$.
    2. Return $u$.
```

Algorithm 1: Transforming secret value $v \in \{0, 1, 2, ..., M\}$ to a shared secret unary constraint. This is a multi-party computation using the shares of secret v.

$\langle (x_1, v^1_{\epsilon_1}), ..., (x_m, v^m_{\epsilon_m}) \rangle \rangle$ with $\prod_{f \in C} f(\epsilon|_f){=}1$. Here $\epsilon_{|_f}$ denotes the projection of ϵ to the variables in f.

If $p(\epsilon) = \prod_{f \in C} f(\epsilon_{|_f})$, 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 terms of the h series, weighted as follows:

$$id(P) = \sum_{i=1}^{d^m} i * p(\epsilon_i) * h_i \tag{1}$$

A result of 0 means that there is no solution. The cost of this computation is $(c{+}1)d^m$ multiplications of secrets, md times less than the technique in MPC-DisCSP1, which is $O((fm + m^2)d^{m+1})$. We call this DisCSP2[1].

The method proposed now in MPC-DisCSP2 is to first transform the index id of the solution computed with Equation 1 into a shared vector S, of size d^m 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 **value-to-unary-constraint1** used in MPC-DisCSP1.

$$S = \text{value-to-unary-constraint2}(id, d^m{+}1) \tag{2}$$

It is possible to end here this stage and to start unshuffling S (which would take rather large messages, of size d^m). In this last case the obtained technique for this stage is called DisCSP3[1].

Alternatively, we can translate already now the vector S into values for each variable, and we believe that this approach is slightly better. The obtained technique for this step is referred to as DisCSP2[1].

The value of the $(u+1)^{th}$ 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.

$$\eta_u(t) = [(t-1)/d^u] \mod d \tag{3}$$

$$f_i(P) = \sum_{t=1}^{d^m} \eta_i(t) * S[t] \tag{4}$$

$$\begin{aligned} \theta_k &= \theta_{k-1} * (id - 1) \\ S &= \frac{\theta_k \prod_{i=k}^{d^m} (id - i)}{k!(M-k)!} \end{aligned} \tag{5}$$

It can be noticed that the space required for computing and storing 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, using Equation 5, but with efficiency losses that are inacceptable, $O(d^{2m})$.

5 Computation of DisCSP()

Revealing the first solution ϵ_0 in a lexicographic order reveals 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. MPC-DisCSP1 returns a solution picked randomly from the existing solutions by rephrasing the DisCSP with a hidden permutation after its secrets were shared.

5.0.1 MPC-DisCSP's mix net for reordering shared secret DisCSPs

MPC-DisCSP2 shares and shuffles the DisCSP in the same way as MPC-DisCSP1. To recall, each agent A_i has to share a set of secrets $\{s^i_k\}$ where indexes k, are taken from disjoint sets for distinct agents $(\forall s^i_{k_1}, s^j_{k_2}, (i{\neq}j){\Rightarrow}(k_1{\neq}k_2))$. A pair (ε_k, v_k), where $v_k \in \{0,1\}$ is the evaluation of a constraint of A_i for the partial valuation tuple ε_k, is called an *atomic predicate*. For each (partial) valuation $\varepsilon_k = ((x_{k_1}, v^{k_1}_{k^1}), ..., (x_{k_t}, v^{k_t}_{k^t}))$, having A_j's share of the secret feasibility value $v_k \in \{0,1\}$ equal to s^j_k, the submission takes the form: $\langle\langle(k_l, k^l)\rangle_{l\in[1..t]}, k, E_j(s^j_k), j\rangle$. All the submissions are made to A_1, the first in the chain of permutation agents. Assuming the total number of submitted atomic predicates is K, each agent A_i generates $Z{=}K{+}mn$ sets of Shamir secret shares of 0, $\{z^k_j | j \in [1..n], z^k_j{=}\sum_{u=1}^{t-1} a_{k,u}(k_j)^u\}\}$, for some $a_{k,u}$, $k{\in}[1..Z]$, and the secret permutations:

$$\begin{aligned} \pi :&\quad [1..m] \rightarrow [1..m], &&\text{(variables)}\\ \pi_1, ..., \pi_m :&\quad [1..d] \rightarrow [1..d], &&\text{(domains)}\\ \pi_0 :&\quad [1..K] \rightarrow [1..K]. &&\text{(atomic predicates)} \end{aligned}$$

For performing the mix net, when A_1 or a subsequent A_i receives (all the elements of) a vector $\{\langle\{(k_l, k^l)\}_{l\in[1..t]}, k, E_j(s^j_k), j\rangle\}_{k\in[1..K]}$, it generates a permutation w^k and the vector $\pi_0(\{\langle\{(\pi(k_{w^k(l)}), \pi_{k_{w^k(l)}}(k^{w^k(l)}))\}_{l\in[1..t]}, \pi_0(k),$
$E_j(s^j_k)E_j(z^k_j), j\rangle\}_{k\in[1..K]})$, which is sent to A_2, respectively to A_{i+1}. Namely, the position of pairs inside each permuted valuation are randomly shuffled according to w^k. A_n distributes the vectors to corresponding A_j.

Hiding shares As in MPC-DisCSP1, to avoid that everybody learns all the secret shares, these are sent encrypted with the public key of their destination participant. To also avoid that agents recognize shares that they have sent and retrieve part of the overall permutation, each agent A_i in the Merritt chain generates random sets of n shares for 0 with the technique of Shamir. The j^{th} share in the k^{th} set is denoted by z^k_j. Whenever A_i performs a shuffling/unshuffling of a set of encrypted secret shares, A_i uses a new set of shares for 0 and multiplies the corresponding encrypted shares with the point product. Therefore, since we use a $(+,\times)$-homomorphic encryption, the obtained shares represent the sum of the secret with 0 and is a re-sharing of the initial secret.

$$f'_i = \text{value-to-unary-constraint2}(f_i{-}1, d) \tag{6}$$

Decoding solutions after DisCSP3[1] After DisCSP2[1] is run on the shared problem shuffled as shown by the previous technique, the solution has to be revealed without revealing the permutation. Let S^j be j's share of S. The vectors $\{\langle\{E_j(S^j[t])\}_{t\in[1..d^m]}, j\rangle\}$, for each j, are sent backward through the Merritt chain of agents. When A_k receives $\{\langle\{E_j(S^j[t])\}_{t\in[1..d^m]}, j\rangle\}$, it generates and sends to A_{k-1} the vector $\{\langle\{E_j(S^j[\sum_{i=0}^{m-1} d^i * \pi_{\pi^{-1}(i)}(\eta_{\pi^{-1}(i)}(t))]\}_{t\in[1..d^m]},$
$j\rangle\}$. A_1 broadcasts them.

Decoding solutions after DisCSP3[1] 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 6) have to be revealed without revealing the permutation. The vectors $\{\langle\{E_j(f'^j_i[t])\}_{t\in[1..d]}, j\rangle\}_{i\in[1..m]}$,

for each j, are sent backward through the chain of agents, where $f_i'^j[t]$ is A_j's share for $f_i'[t]$. When A_k receives $\{\langle\{E_j(f_i'^j[t])\}_{t\in[1..d]},j\rangle\}_{i\in[1..m]}$, it generates and sends to A_{k-1} the vector $\pi^{-1}(\{\langle\pi^{-1}_{\pi^{-1}(i)}(\{E_j(f_i'^j[t])E_j(z_j^{K-i-t+2})\}_{t\in[1..d]}),$
$j\rangle\}_{i\in[1..m]})$. A_1 broadcasts them.

Complexity and parallelism The total number of messages that have to be sent with MPC-DisCSP2 is $2n$ (for shuffling), $n^2(c+1)d^m$ for Equation 1, $n^2 3d^m$ for Equation 2, n^2m3d for m Equations 6, and $O(n^2)$ for decoding the solution. The total number of messages is $n^2(4+d^m(c+4)+3md)$. n^2 of them are always sent concurrently, for multiplications, obtaining $O(d^mc+m)$ rounds.

One can enable additional parallelism, performing some of the multiplications of Equations 1, 2, and 6 in a parallel divide and conquer manner. This is dependent of the amount of parallelim supported by the network: Ideally, one can reduce the total number of rounds to $O(m\log_2(d)+\log_2(c))$, but unfortunately that requires $O(n^2d^m)$ concurrent messages.

6 MPC-DisWCSP2: extension to distributed weighted CSPs

Let $q(\epsilon)=\sum_{f\in C}f(\epsilon)$. 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 $|C|B$. We still want to isolate solutions whose $q(\epsilon)$ is given by some value x_0 (actually we now need the minimal x_0 allowing for a solution).

$$p(\epsilon)=\frac{\prod_{i=0}^{x_0-1}(q(\epsilon)-i)\prod_{i=x_0+1}^{|C|B}(i-q(\epsilon))}{x_0!(|C|B-x_0)!}. \tag{7}$$

A solution with lowest weight of a DisWCSP can be found by iterating MPC-DisCSP with the definition 7 for p, for x_0 increasing from 0 to $|C|B$. Note that the last two techniques reveal to everybody the quality of the solution, i.e. the number of satisfied constraints respectively the sum of constraint weights in the solution.

Theorem 1 *The described technique offers t-privacy (No collusion of less than t attackers can learn anything, but the final solution with its quality, and what can be inferred from it).*

Proof. Each step of the previous technique is based on the evaluation of a set of functions consisting solely of additions and multiplication. It has been proven in [Yao82, GMW86] that the compilation to multi-party computations of such a technique is t-private. The mix net is hiding the permutation (and the order under which the solution was found). It is always possible that the found solution was the first in lexicographic order, as it is possible that is was the last one. No information can be extracted about the acceptance of the other possible allocations.

The information revealed to everybody is the quality of the solution and the allocation of their resources in that solution. The quality of the solution is revealed by the number of computation rounds. □

MPC-DisWCSP2: Solving a DisWCSP while hiding the weight of the solution DisCSP[1] can hide the number of rounds needed to find the first solution to a DisCSP. In a similar way we hide the number of rounds needed to find the first solution to a DisWCSP. A new set of functions w_i is defined to hold the value of x_i in the best solution found so far.

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

This can be computing with the following arithmetic circuits (for $i{\in}[1..d]$ and $j{\in}[1..|C|B]$):

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

MPC-DisWCSP 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 DisWCSP (except that the values assigned by constraints to tuples are in [0..B] rather than {0,1}).
2. The vector $\{w_i^{|C|B}\}_{i\in 1..m}$ is computed by iteratively building the vectors $\{w_i^j\}_{i\in 1..m}$ for j increasing from 0 to $|C|B$. The computation in each iteration j is performed according to DisCSP2[1] but with the new definition of p and with $x_0{=}j$. It is followed by evaluating the arithmetic circuit $\{w_i^j\}_{i\in 1..m}$ with a multiparty computation.
3. The solution is unshuffled and distributed as in MPC-DisCSP, except that the solution vector is the one containing the results of the functions $w_i^{|C|B}$ rather than f_i.

The complexity of MPC-DisWCSP is $|C|B$ times higher than the complexity of MPC-DisCSP. For the most parallel version of MPC-DisCSP, the longest chain of messages is $|C|B$ times longer.

In MPC-DisWCSP 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:

$$\begin{aligned} w_0^{|C|B} &= \sum_{k\in[0..|C|B]} k(1- \\ &\quad \frac{\prod_{k\in[1..d]} k - w_1^k(P)}{d!}) \\ &\quad \frac{\prod_{k\in[1..d]} k - w_1^{k-1}(P)}{d!} \end{aligned}$$

The single non-zero term in the summation defining $w_0^{|C|B}$ is for the round k where w_1^k is for the first time non-zero. $w_0^{|C|B}$ specifies the weight of the solution and after unshuffling is revealed only to the agents that should learn it.

7 Conclusions

DisCSPs are a very active research area and secrecy within DisCSPs has been recently stressed in [MJ00, FMW01, WF02, YSH02, Sil03] as an important issue. The described technique is a $\lceil n/2\rceil$-privacy threshold scheme. While we work on techniques that may provide such robustness as in other multiparty computations, they were not addressed in this discussion (notably see ($\lceil t/3\rceil$,n) threshold schemes) [BOGW88].

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. The technique is exponential in space such that only problems up to 15 variables can be addressed. To be noted that this is the typical problem size that can be addressed by other techniques that aim at some lesser privacy. We also show how the technique can be extended to perform optimization in distributed Weighted CSPs.

References

[AF88] M. Abadi and J. Feigenbaum. A simple protocol for secure circuit evaluation. In *STACS'88 Proceedings*, pages 264–272, 1988.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[CDG88] D. Chaum, I. Damgard, and J. Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO'87 Proceedings*, pages 87–119, 1988.

[CGH00] D. Catalano, R. Gennaro, and S. Halevi. Computing inverses over a shared secret modulus. In *EUROCRYPT*, pages 190–206, 2000.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[FMW01] E.C. Freuder, M. Minca, and R.J. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *Proc. IJCAI DCR*, pages 63–72, 2001.

[GHY88] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *CRYPTO'87*, pages 135–155, 1988.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proc. of 27th IEEE FOCS*, pages 174–187, Toronto, 1986.

[Hab88] Stuart Haber. *Multi-party Cryptographic Computation: Techniques and Applications*. PhD thesis, Columbia University, 1988.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[Lar02] J. Larrosa. Node and arc consistency in weighted csp. In *AAAI-2002*, Edmonton, 2002.

[Mer83] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Inst. of Tech., Feb 1983.

[MJ00] P. Meseguer and M. Jiménez. Distributed forward checking. In *DCS*, 2000.

[MTSY03] P.J. Modi, M. Tambe, W.-M. Shen, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS*, Melbourne, 2003.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[Sha79] A. Shamir. How to share a secret. *Comm. of the ACM*, 22:612–613, 1979.

[Sil03] M.C. Silaghi. Arithmetic circuit for the first solution of distributed CSPs with cryptographic multi-party computations. In *IAT*, Halifax, 2003.

[WF02] R.J. Wallace and E.C. Freuder. Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. In *DCR*, pages 176–182, 2002.

[Yao82] A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[YSH02] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.

[ZM04] R. Zivan and A. Meisels. Concurrent backtrack search on discsps. In *to appear in FLAIRS2004*, 2004.