# Secure Combinatorial Optimization simulating DFS tree-based Variable Elimination

Marius-Călin Silaghi
Florida Institute of Technology at Melbourne
Boi Faltings and Adrian Petcu
Swiss Federal Institute of Technology at Lausanne

## Abstract

In general, Constraint Optimization Problems (COP) are NP-hard. Using variable elimination techniques [5, 13] COPs can be solved with computation that is exponential only in the induced-width of the constraint graph (given some order on the nodes), i.e. smaller than $n$. Orders on nodes allowing for some parallelism are offered by Depth First Search (DFS) trees of the constraint graph [3, 13].

Any arithmetic circuit can be compiled into a general secure multi-party computation where no participant learns anything except for the result [1, 8]. We show in [25] that a secure combinatorial problem solver must necessarily pick the result randomly among optimal solutions, to be really secure. We recently developed SMC [19], the first programming language that translates [1]'s theory into practice. SMC also supports constraint satisfaction problems (CSPs), but additional techniques were revealed needed to offer acceptably efficient support for COPs. In [24] we proposed arithmetic circuits for solving COPs but which are exponential in the number of variables, $n$, for any constraint graph.

Here we show how to construct an arithmetic circuit with the complexity properties of DFS-based variable elimination, and that finds a random optimal solution for any COP. For forest constraint graphs, this leads to a linear cost secure solver. Developing an arithmetic circuit performing the operations of the dynamic programming step in variable elimination proves to be quite straightforward and similar to previous work. We encountered a more interesting scientific challenge in choosing a secure scheme for the equivalent of the decoding step. The decoding step consists of traversing the dynamic programming data structures backward to detect the assignments that generate the winning alternative. What seems to be the straightforward arithmetic circuit translation reveals results before the end of the computation, compromising security. We show how to develop an arithmetic circuit comprising all processing until the end of the computation.

[1]Significant input was received from Benjamin Pflanz.

## 1 Introduction

Combinatorial optimization occurs in many situations. One important formalism for modeling combinatorial optimization is the constraint optimization problem (COP). A constraint optimization problem ($X$,$D$,$C$) is defined by a set of variables, $X = \{x_1, ..., x_m\}$, with domains from a corresponding set $D = \{D_1, ..., D_m\}$, and a set of weighted constraints $C = \{\phi_0, ..., \phi_m\}$, each such constraint $\phi_i$ specifying a distinct cost associated with each assignment of values to a subset $X_i$ of $X$.

An assignment is a pair $\langle x_i, v\rangle$ where $v \in D_i$. We will assume that $D_i = [1..k_i]$. A solution of the COP is a tuple of assignments $\varepsilon$ with values for each variable in $X$ such that the sum of the weights associated by the constraints in $C$ to $\varepsilon$ is maximized (minimized). The Cartesian product of the domains for a set of variables $X'$ is denoted $\Gamma_{X'}$. Without loss of generality we assume that by optimal solution we understand the solution with maximal weight. If we denote the projection of a tuple $\varepsilon$ on a set of variables $X_i$ by $\varepsilon_{|X_i}$, then the solution is:

$$\operatorname*{argmax}_{\varepsilon \in \Gamma_X} \sum_{\phi_i \in C} \phi_i(\varepsilon_{|X_i})$$

A distributed COP (DCOP) arises when some constraints are functions of secrets own by some agents from a set $A = \{A_1, ..., A_n\}$. Without loss of generality we assume that $\phi_0$ is the only public constraint and that $X_m$, the set of variables in $\phi_m$, contains besides $x_m$ only variables $x_i$ with $i < m$. Note that such a formulation can be obtained from any DCOP by building a Depth First Search (DFS) tree, introduced later and composing the constraints such that there remains a single constraint per variable (with her ancestors in the tree).

Our work employs the following secure multi-party computation techniques:

- polynomial secret sharing [15]: Each participant $k$ out of $n$ participants receives $\langle s\rangle_k^t = s + \sum_{i=1}^{t} a_i k^i$, where $a_i$ is a secret random number. The secret can be reconstructed with the collaboration of $t+1$ participants using $s = \sum_{k=1}^{t+1} l_{k,t}\langle s\rangle_k^t$, where $l_{k,t}$ are the corresponding Lagrange coefficients.
- addition of shared secrets [1]: $\langle s_1 + s_2\rangle_k^t = \langle s_1\rangle_k^t + \langle s_2\rangle_k^t$
- resharing shared secrets [1]: To reshare a shared secret $\langle s\rangle^t$ with another threshold $t'$, each share $\langle s\rangle_k^t$ is also shared with $(t', n)$-polynomial sharing scheme, and $\langle s\rangle_k^{t'}$ is constructed with $\langle s\rangle_k^{t'} = \sum_{i=1}^{t+1} l_{i,t}\langle\langle s\rangle_i^t\rangle_k^{t'}$.
- multiplication of shared secrets [1]: $\langle s_1 * s_2\rangle_k^{2t} = \langle s_1\rangle_k^t * \langle s_2\rangle_k^t$.
- arithmetic circuit evaluation with additive secret sharing [8]: Each participant $k, k > 1$ out of $n$ participants receives $[s]_k = a_i$ where $a_i$ is a random number. Participant 1 gets $[s]_1 = s - \sum_{i=2}^{n} a_i$. The secret could be reconstructed with $s = \sum_{k=1}^{n}[s]_k$. Addition of additively shared secrets is done with $[s_1 + s_2]_k = [s_1]_k + [s_2]_k$. Multiplication is done using oblivious transfers [8].
- secure test [4]: $\delta(x)$ returns 1 if $x = 0$ and 0 otherwise.
- secure Kronecker's $\delta$ [9, 4]: $\delta_K(x, y) = \delta(x - y)$ returns 1 if $x = y$ and 0 otherwise.
- secure comparison [4]: $cmp(x, y)$ returns 0 if $x < y$ and 1 otherwise.
- secure max: $\max(x, y) = cmp(x, y) * (x - y) + y$.

## 2 Background

DCOPs have been addressed with various techniques that differ both in efficiency and in their privacy guarantees. Previous techniques seeking the strongest privacy guarantees are based on secure multi-party computation and scan several times the whole search space, i.e. Cartesian product of domains in $D$, once for each possible total weight [24]. An optimization protocol specialized on generalized Vickrey auctions and based on dynamic programming is proposed in [27] and is significantly more efficient, but does not randomize the selection of the solution, needed for reaching the highest level of privacy [25]. DPOP, a dynamic programming algorithm for solving (D)COPs was proposed in [13] and consists of a Viterbi-like combination of a maximization and decoding [26]. The algorithm in [13] can also be seen as a clever heuristic for variable elimination [5], or as a parallelization of ADOPT [11], and is based on a different concept of privacy [22].

### 2.1 Variable Elimination

Variable Elimination is a principled technique for complexity reduction in COPs. It consists of replacing all the constraints (objective functions) linked to a variable chosen for elimination by the projection of their composition on the remaining variables. A heuristic for selecting the variables to be eliminated next is provided by the DFS tree [13].

## 2.2 DFS tree

The primal graph of a COP is the graph having the variables as nodes and having an arc for each pair of variables linked by a constraint [6]. A Depth First Search (DFS) tree associated to a COP is a spanning tree generated by the arcs used for visiting once each node during some depth first traversal of its primal graph. DFS trees were first successfully used for Distributed Constraint problems in [3]. The property exploited there is that separate branches of the DFS-tree are completely independent once the assignments of common ancestors are decided. Two examples of DFS trees for a COP primal graph are shown in Figure 1.

**Definition 1 (neighbor nodes)** *The nodes directly connected to a node in a primal graph are said to be its* neighbors*.*

In Figure 1.a, the neighbors of $x_3$ are $\{x_1, x_5, x_4\}$.

**Definition 2 (ancestor nodes)** *The* ancestors *of a node are the nodes on the path between it and the root of the DFS tree, inclusively.*

In Figure 1.b, the ancestors of $x_2$ are $\{x_5, x_3\}$, while $x_3$ has no ancestors.

**Definition 3 (descendants nodes)** *The* descendants *of a node are its children in the DFS tree, as well as the children of any other of its descendants.*

In Figure 1.c, the descendants of $x_3$ are $\{x_1, x_4, x_2\}$, while $x_2$ has no descendants.

A *neighboring ancestor* of a node is any node that is both a neighbor in the primal graph and an ancestor in the used DFS tree. Considering the DFS tree as a reverse order on the constraint graph, we adapt the typical definition of the induced width [6] as follows. The *induced width of a node $x$ in the DFS tree* is the number of ancestors that are neighboring $x$ or some of its descendants. The *induced width of a DFS tree* is given by the maximum induced width of all its nodes. Several heuristics had been used in the past for building DFS trees with reduced width, a preferred optimal technique consisting of a branch & bound procedure.

We use the following notation. Let:

- $F_x$ be the parent of $x$ (in Figure 1.b $F_{x_5} = x_3$). If $x$ is the root node of the tree then $F_x = \emptyset$.
- $S_x$ be the children of $x$ (in Figure 1.c $S_{x_3} = \{x_1, x_4\}$).
- $P_x$ be the neighbor ancestors of $x$ (in Figure 1.c $P_{x_2} = \{x_1, x_5\}$).
- $G_x$ be the induced parents of $x$, i.e., ancestors that are neighbors for $x$ or for some descendant of $x$ (in Figure 1.c $G_{x_1} = \{x_3, x_5\}$, since $x_5$ is the neighboring ancestor of the descendant $x_2$).

In this work we do not address heuristics for building DFS trees, but consider that such a tree is provided.

## 2.3 DFS-based Variable Elimination

A heuristic for selecting the order to eliminate variables based on exploiting the DFS-tree is proposed in [13]. The idea is that before eliminating a node in the DFS tree one should first eliminate its children. In a centralized approach, such an order could be generated by either a postorder traversal or a reversed level-order traversal. This heuristic guarantees that the arity of the largest constraint that will be added to the problem (and therefore the complexity of the algorithm) is bounded by the distance between two neighbors in the DFS tree. This is bounded by the depth of the tree and potentially much smaller than $n$. The advantage of this heuristic is that the quality of an elimination order can be easily evaluated.

## 2.4 Random selection and security

We prove in [25] that a constraint satisfaction problem solver is not secure if it does not pick the answer randomly among the existing solutions. Those proofs apply straightforwardly to optimization, and a constraint optimization problem solver is not secure if it does not pick the answer randomly among the existing solutions.

The solutions we proposed were based on shuffling the search space prior to solving, such that the solving is done with an order unknown to any participant. We propose several uniform and non-uniform

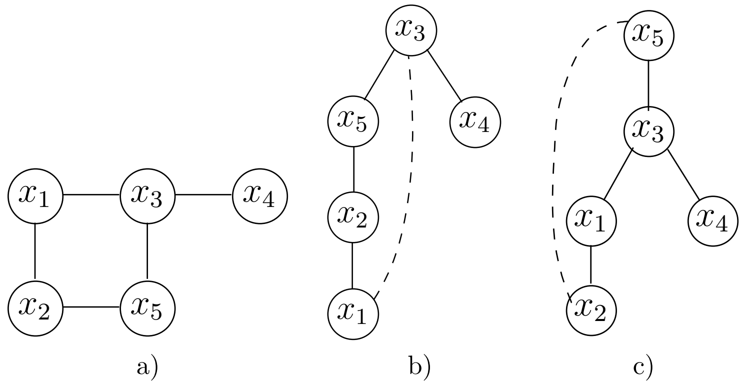

Figure 1: For a COP with primal graph depicted in (a), two possible DFS trees (pseudotrees) are (b) and (c) with induced width 2. The interrupted lines show constraint graph neighboring relations that do not belong to the DFS tree.

shuffling techniques in [21]. The result of the computation will be unshuffled prior to revelation to the participants. Shuffling can be performed using a mix-net, build from the participants [2, 10, 21], i.e., where each participant applies a secret permutation on the secrets in turn.

**Remark 1** *The fact that a mix-net leads to a random solution holds for an agent (or a set of agents) only if the 'other' agents shuffle all variables.*

*If at least one agent does not shuffle a variable, this does not hold for the sub-group that shuffle, and the benefit of the shuffling is completely lost*

**Theorem 1** *All agents must be involved in shuffling each variable's domain of a problem.*

**Proof.** If some agent $A_i$ does not shuffle the domain of a variable $x_j$, then all the other agents can make a coalition, find the order on $x_j$ with which the problem was solved, and learn a set of values of $x_j$ for which there is no optimal solution. Eliminating $x_j$ for those values from their constraints leads to a projection of their constraint on the remaining variables, $X'$, from which they can infer bounds on $A_i$'s costs on those remaining variables $X'$ (lower bounds at minimization respectively upper bounds at maximization). □

## 2.5 Secure Optimization

A secure optimization algorithm for DCOP is proposed in [24]. It chooses randomly one of the values with the optimal value and reveals the total weight of the solution and the corresponding assignments only if desired and only to agreed participants. To ensure random selection of the solution, shuffling of values is done prior to solving. The result of the computation will be unshuffled. In [18] it is shown how to make the selection with a uniform random distribution. However, the complete versions of these techniques are always exponential in the size of the search space (as left after applying any pruning defined by the public constraints).

In the following we show how to formulate arithmetic circuits for securely computing an optimal solution of a DCOP using DFS-based Variable Elimination. As in the non-secure version, the algorithm has two parts, an (upward) dynamic programming step, and a (downward) decoding step.

# 3 Arithmetic Circuits

The data structure we employ as well as their usage is depicted graphically in Figure 2. Continuous lines show arcs in the used DFS tree. Interrupted

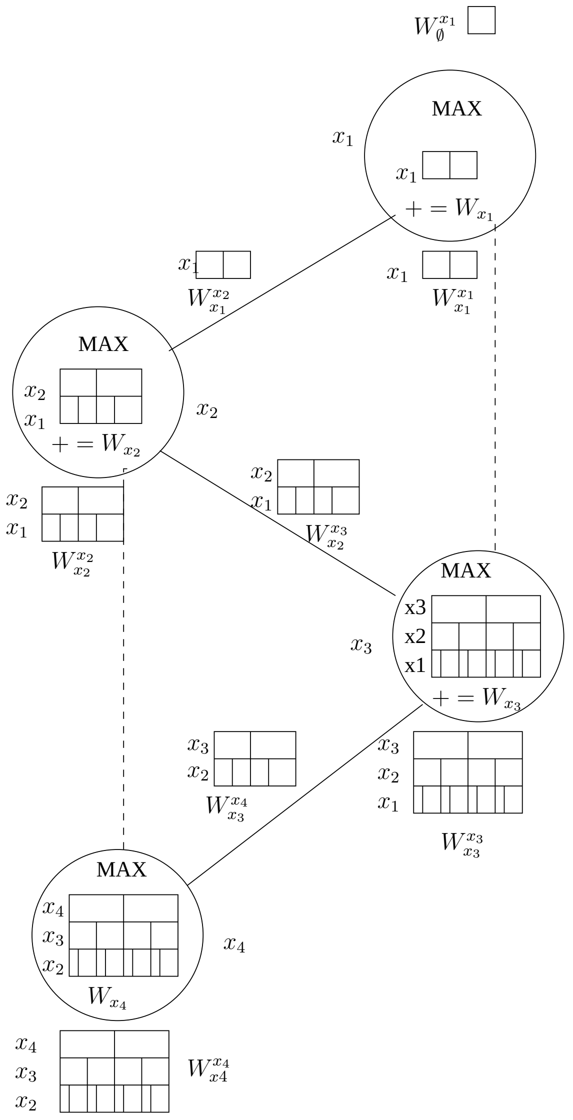

Figure 2: Data structures in the Upward computation for a problem with 4 variables.

lines show arcs not belonging to the DFS tree. Constraints are depicted as vectors whose columns are selected by the possible values of the combination of involved variables. Constraints depicted below nodes are the ones between the variable and its neighboring ancestors, associated to the variable in the initial problem. Constraints shown on arcs are the result of projections of the child's aggregated constraint on the variables of its induced parents. Constraints inside a node are obtained by composing the original constraint of the node, situated below, to the projected constraints coming along arcs from children.

For each node $x_i$ there is a separate set of data structures $W_{F_{x_i}}^{x_i}, W_{x_i}^{x_i}, W_{x_i}$, representing projected, original and composed constraints. These data structures are accessed by indexing with partial assignment of variables. They are implementable as multi-dimensional matrices, where $W_{x_i}^{x_i}$ has $|X_i|$ dimensions (one for each variable in $X_i$), $W_{F_{x_i}}^{x_i}$ has $|Gx_i|$ dimensions (one for each variable in $G_{x_i}$), and $W_{x_i}$ has $|Gx_i| + 1$ dimensions (one for $x_i$ and one for each variable in $G_{x_i}$).

- $W_{x_i}^{x_i}$ holds $\phi_i$, namely the local cost associated to each combination of assignments of $x_i$ and neighboring ancestors of $x_i$.

- $W_{x_i}$ holds the cumulated cost associated by $\phi_i$ and the projection of the constraints of all its children to each assignment of $x_i$ and induced parents of $x_i$, $G_{x_i} \cup \{x_i\}$.

- $W_{F_{x_i}}^{x_i}$ holds the cumulated cost associated by $\phi_i$ and the projection of the constraints of all its children to each assignment of induced parents of $x_i$, $G_{x_i}$.

On the upward path in the DFS tree, for each node $x_i$ one computes for each assignment $S$ of the induced parents $G_x$:

$$\begin{aligned} W_{F_{x_i}}^{x_i}[S] \quad = \quad & \max_{v \in D_i}(W_{x_i}^{x_i}[\langle x_i, v\rangle \cup S_{|P_{x_i}}] + \\ & \sum_{y \in S_{x_i}} (W_{x_i}^{y}[(S \cup \{\langle x_i, v\rangle\})_{|G_y}])) \end{aligned}$$

The value $W_{\emptyset}^{r}$, where $r$ is the index of the root node, is the weight of the optimal solution. The steps of computation are detailed in the Algorithm 1.

```
procedure $Upward(x_i)$ do
    foreach $(y \in S_{x_i})$ do
        // recursive call for each child;
        $Upward(y)$;
    foreach tuple $\varepsilon \in \Gamma_{G_{x_i} \cup \{x_i\}}$ do
        // compose costs for the same tuple;
        $W_{x_i}[\varepsilon] = W^{x_i}_{x_i}[\varepsilon_{|\{x_i\} \cup P_{x_i}}] + \sum_{y \in S_{x_i}} (W^y_{x_i}[\varepsilon_{|G_y}]$;
    foreach tuple $\varepsilon \in \Gamma_{G_{x_i}}$ do
        // project composed constraint onto induced parents;
        $W^{x_i}_{F_{x_i}}[\varepsilon] = \max_{v \in D_i}(W_{x_i}[\varepsilon \cup \langle x_i, v\rangle])$;
```

Algorithm 1: Arithmetic circuit for the upward (dynamic programming) step. At the first call, the parameter $x_i$ is the root, $r$, of the DFS tree. It would have to be called several times for disconnected graphs.

## 3.1 Unsecure decoding of solution

If we also want to reveal the assignment with the optimal value, it can be done with the following arithmetic circuit, for the downward path (e.g., level-order traversal from root).

*The value of the whole tree, $V_r$, is the shared secret $W^r_\emptyset$ computed at the upward step. At variable $x_i$ with subtree value $V_{x_i}$, for each tuple $\varepsilon$ with value W' in the structure $W_{x_i}$ at assignments equal to the ones selected at previous levels, compute and reveal $\delta_K(V_{x_i}, W')$. If the result is 1, the corresponding assignment of $x_i$ in $\varepsilon$ is selected, and the corresponding values $V_y$ in $W^y_{x_i}$ for each child variable $y \in S_{x_i}$ is selected as value of the subtree with root $y$.*

The problems with this approach, of revealing the solution, is that the algorithm cannot be used in cases where the solution of the COP is only an intermediary computation [16]. In that kind of application the revelation of the intermediary results on each involved optimization is a leak of a secret that should not have been disclosed. Besides, this revelation of some results before others could be computed offers opportunities to some participants to stop cooperating after they learn secrets and before reaching a conclusion [7]. We fix this in the following algorithm.

## 3.2 Secure decoding of solution

The data structures proposed for the secure downward computation (decoding) are again similar for each node. At node $x_i$ we store:

- $d_{x_i}$ is a shared secret vector of $|D_i|$ boolean values, $d_{x_i}[v] = 1$ indicating the selection of the corresponding value $v$ of $D_i$. Only one value can be set to 1, the others being 0.
- $V_{x_i}$ is the shared secret weight associated to the selected optimal solution by the subtree with root $x_i$.

After performing the upward computation of dynamic programming, we can decode the solution securely during a preorder or level-order traversal of the tree. On visiting each node $x_i$, a procedure is run to compute securely the shared secret assignment of $x_i$ using the shared secret assignments of the induced parents, $d_p$, for $p \in G_{x_i}$, and the shared secret selected weight of this variable, $V_{x_i}$. The procedure also computes the inputs for the next recursive procedure calls, at the descendents of $x_i$, namely the shared secret selected weight $V_y$ of each child $y \in S_{x_i}$.

$$h_{x_i}[v] = 1 - \sum_{k=1}^{v-1} d_{x_i}[k]$$

$h_{x_i}[v]$ is 1 if there is no optimal value for $x_i$ before $v$, and 0 otherwise.

```
procedure Downward(COP,$W_*^*$,$V_r$) do
    foreach variable $x_i$ in the COP do
        $\zeta_{x_i}[\emptyset] = 1$;
    while $x_i \leftarrow$ get_In_PreOrder_Next(DFS(COP)) do
        foreach $v \in D_i$ do
            $h_{x_i}[v] = 1 - \sum_{k=1}^{v-1} d_{x_i}[k]$;
            $d_{x_i}[v] = h_{x_i}[v] \sum_{\varepsilon \in \Gamma_{G_{x_i} \cup \{x_i\}}, \varepsilon_{|\{x_i\}}=v} \zeta_{x_i}[\varepsilon_{|G_{x_i}}]\delta_K(V_{x_i}, W_{x_i}[\varepsilon])$;
        foreach $y \in S_{x_i}$ do
            // Compute $\zeta_y[\varepsilon]$ with Eq. 2 (or Eq. 1);
            if cost($|G_y| - 2$ multiplications) > cost($d^{|G_{x_i} \setminus G_y|}$ additions) then
                foreach $\varepsilon \in \Gamma_{G_y}$ do
1.1                 $\zeta_y[\varepsilon] = d_{x_i}[\varepsilon_{|\{x_i\}}]((G_{x_i} == \emptyset)?1 : \sum_{\varepsilon' \in \Gamma_{G_{x_i}}, \varepsilon'_{|G_y}=\varepsilon} \zeta_{x_i}[\varepsilon'])$;
            else
                foreach $\varepsilon \in \Gamma_{G_y}$ do
                    $\zeta_y[\varepsilon] = \prod_{p \in G_y} d_p[\varepsilon_{|p}]$;
            $V_y = \sum_{\varepsilon \in \Gamma_{G_y}} \zeta_y[\varepsilon] W_{x_i}^y[\varepsilon]$;
```

Algorithm 2: Arithmetic circuit for a secure downward phase. Standard "C" operator ?: is used at Line 1.1.

One can check that the ancestors of a node $x_i$ have selected a tuple $\varepsilon$ of assignments for $G_{x_i}$ by computing

$$\zeta_{x_i}[\varepsilon] = \prod_{p \in G_{x_i}} d_p[\varepsilon_{|p}] \quad (1)$$

which will have value 1 if that holds and 0 otherwise. The following sums are over all tuples $\varepsilon$ of assignments for $G_{x_i} \cup \{x_i\}$.

$$d_{x_i}[v] = h_{x_i}[v] \sum_{\varepsilon_{|\{x_i\}}=v} \zeta_{x_i}[\varepsilon_{|G_{x_i}}]\delta_K(V_{x_i}, W_{x_i}[\varepsilon])$$

Here the factor $\zeta_{x_i}[\varepsilon_{|G_{x_i}}] = (\prod_{p \in G_{x_i}} d_p[\varepsilon_{|p}])$ is meant to return 1 if $\varepsilon_{|G_{x_i}}$ is included in the partial assignment selected by the ancestor nodes and 0 otherwise. The factor $\delta_K(V_{x_i}, W_{x_i}[\varepsilon])$ is meant to return 1 if the extension of the ancestor's assignment with $x_i = v$ leads to an optimal cost for the current subtree. The product of these two factors is summed over all tuples where $x_i = v$. At most one of the terms of this summation is 1 because the first factor is 1 only for a single $\varepsilon_{|G_{x_i}}$ (the one selected by the ancestors). This computation inverses the projection with *max* done on the Upward path.

The following computation, of $V_y$ where $y$ is a child of $x_i$, is done over all tuples $\varepsilon$ of assignments for $G_y$ and selects the optimal value of the subtree rooted in $y$.

$$V_y = \sum_{\varepsilon \in \Gamma_{G_y}} \zeta_y[\varepsilon] W_{x_i}^y[\varepsilon]$$

The factor $\zeta_y[\varepsilon] = (\prod_{p \in G_y} d_p[\varepsilon_{|p}])$ is 1 only for the actual tuple selected by ancestors of $y$, and the factor $W_{x_i}^y[\varepsilon]$ weights the optimal tuple with its cost in the subtree. Note that $\zeta_y$ can be reused at the computation of $d_y$.

In certain cases, namely for small $|G_{x_i} \setminus G_y|$, it will be faster to compute $\zeta_y$ from $\zeta_{x_i}$ rather than with its definition in Eq. 1. This can be done with a single multiplication per item:

$$\zeta_y[\varepsilon] = d_{x_i}[\varepsilon_{|\{x_i\}}] \sum_{\varepsilon' \in \Gamma_{G_{x_i}}, \varepsilon'_{|G_y}=\varepsilon} \zeta_{x_i}[\varepsilon']. \quad (2)$$

The computation steps required by the downward phase are detailed in Algorithm 2. At the end of this computation, the vectors $d_{x_i}$ hold a shared unary constraint allowing a single value for $x_i$, namely the one in the optimal solution. These unary constraints can then be unshuffled [21]. Note that security also requires the use of dedicated techniques for ensuring fairness, e.g., by quasi-simultaneous revelation of solutions [7].

**Security Analysis** Since the whole computation is an arithmetic circuit, we are guaranteed that nothing else is learned except for what can be inferred from the initial knowledge, the result and the used algorithm (order on values and variables). To reduce to statistical leaks what can be inferred from the algorithm, we had shown earlier [25] that the algorithm needs to pick a solution randomly over the optimal solutions. This is achieved here by shuffling the domains of the variables [25]. Optionally the variables can also be shuffled [17]. It was proven in [25] that shuffling values (and variables) does not lead to a uniform distribution in the probability of selecting a given solution among other alternatives (uniform distributions minimizing statistical leaks at repeated involvement of a secret in different computations [18]). Therefore the level of privacy offered by the MPC-DisWCSP3 and MPC-DisWCSP4 optimization algorithms proposed in [18, 24, 20] is still higher for secrets involved in different computations.

## 4 Complexity Analysis

The *Upward* step is called once for each variable $x_i$ and the number of operations for each variable is linear in the number of elements of $W_{x_i}$, i.e., exponential in $|G_{x_i}|+1$. The total cost for the upward step is $O(nd^{g+1})$, where $d$ is the maximum size of a domain of a variable, and $g$ is the maximum value for $|G_{x_i}|$, i.e. the induced width of the used DFS tree.

In the *Downward* step there exists a *while* loop for each variable $x_i$ and for each variable $x_i$ there is a summations for each element in $W_{x_i}$ and two summations for each element of $W^{x_i}_{F_{x_i}}$, each term having one multiplication (assuming the first branch of the if). The total cost for the downward step is therefore $O(nd^{g+1})$ multiplications.

Therefore, the total complexity of the secure version is $O(nd^{g+1})$. If the unsecure downward version with immediate revelation of assignments in solutions is used, then the complexity is also $O(nd^{g+1})$. For forest constraint graphs ($g = 1$) this implies a linear complexity, tractable and much better than the algorithms in [24]. If shuffling of constraints and unshuffling of solution vectors $d_x$ are used to randomize the selection of the solution, then the cost of the shuffling is also added [21].

## 5 Extensions and Applications

A version of Secure Stochastic Optimization can be obtained by trimming the intermediary data structures at each node on the *Upward* phase. Something similar was proposed for non-cryptographic techniques in [12].

**Remark 2** *Note that sorting the set of weights before trimming (in a beam-search like approach) is possible, but only for some applications. E.g., in auctions one has to ensure that the same tuples survive the pruning at each different optimization subproblem appearing in the Clarke tax computations [16].*

An immediate application for secure DCOPs is in performing the intermediary optimizations steps for Clarke tax in generalized Vickrey auctions, and related auction clearance mechanisms. If a secure stochastic version is used [23], then one has to use the same pruning (with the same surviving tuples) for each different optimization task on a given problem, as explained in [16].

**Extension to omnidirectional propagations** It has been shown in [14] that the utility propagation can be made *omnidirectional* by circulating messages in all directions along the DFS tree (also top-down, from each node to its children). In this version, a message from a parent to its child summarizes the utility information from all the problem except the subtree of that child. Joining the message from the parent with the ones received from the children gives each node a global view of the system, logically making each node in the system equivalent to the root in the simple DPOP scheme. Projecting out all dimensions from this joined message gives each node the optimal value in the overall optimal solution.

## 6 Conclusion

We have shown how to speed up the secure computation for constraint optimization, by a secure equivalent of the fix-cost DFS-based Variable Elimination using general arithmetic circuits. The previous secure DCOP techniques performed secure verifications separately for each possible tuple weight, and were significantly more expensive, specially for sparse graphs and for problems with a large range of possible weights for tuples. We show in [25] that a secure combinatorial problem solver must necessarily pick the result randomly among optimal solutions, to be really secure. Other related optimization techniques previously tailored for auctions [27] did not support shuffling of the problem for randomizing the selection of the solution (as needed for privacy guarantees in DCOPs), and did not exploit parallelism as enabled by different branches of DFS trees. The presented technique is linear for DCOPs with constraint graphs consisting of forests.

The translation into arithmetic circuits of the dynamic programming part of the variable elimination has proven to be straightforward (even if somewhat lengthy) and similar to previous work. However, for a secure integration with shuffling, and in general to guarantee that nothing is learned before the end of the computations [7], the last (decoding) part of the algorithm where a winning path has to be traced back through the dynamic programming structures has been less straightforward and raised more interesting design challenges. The result is fully based on arithmetic circuit and mix-nets, guaranteeing the correctness and security of the compiled protocol, with the high privacy insured by random but non-uniform shuffling as described in [25].

The new optimization technique seems to be the suitable technique needed to integrate constraint optimization into SMC [19], the first programming language that translates [1]'s theory of secure compilations of arithmetic circuits into practice.

## 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] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, 2000.

[4] I. Damgård, M. Fitzi, J. B. Nielsen, and T. Toft. How to split a shared number into bits in constant round and unconditionally secure. Cryptology ePrint Archive, Report 2005/140, 2005. `http://eprint.iacr.org`.

[5] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *AI'90*, 1990.

[6] Rina Dechter. *Constraint Programming.* Morgan Kaufman, 2003.

[7] Juan Garay. Fair multi-party computation. In *Workshop on Secure Multiparty Protocols*, Amsterdam, October 2004.

[8] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge, 2004.

[9] E. Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. Cryptology ePrint Archive, Report 2005/066, 2005. `http://eprint.iacr.org`.

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

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

[12] Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *Principles and Practice of Constraint Programming CP 2005*, 2005.

[13] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.

[14] Adrian Petcu and Boi Faltings. Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, Pittsburgh, Pennsylvania, USA, July 2005. AAAI.

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

[16] M. Silaghi. Using secure discsp solvers for generalized vickrey auctions, complete and stochastic secure techniques. In *IJCAI05 DCR Workshop*, 2005.

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

[18] M.-C. Silaghi. Meeting scheduling system guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any DisCSP). In *3rd IC on Web Intelligence*, pages 711–715, 2004.

[19] M.-C. Silaghi. Secure multi-party computation (SMC) programming language. `http://www.cs.fit.edu/~msilaghi/SMC/`, 2004.

[20] M.-C. Silaghi. Hiding absence of solution for a discsp. In *FLAIRS'05*, 2005.

[21] M.-C. Silaghi. Zero-knowledge proofs for mixnets of secret shares and a version of ElGamal with modular homomorphism. Cryptology ePrint Archive, Report 2005/079, 2005. `http://eprint.iacr.org`.

[22] M.-C. Silaghi and B. Faltings. A comparison of DisCSP algorithms with respect to privacy. In *AAMAS-DCR*, 2002.

[23] M.-C. Silaghi and G. Friedrich. Secure stochastic multi-party computation for combinatorial problems. Technical Report CS-2005-14, FIT, 2005.

[24] M.-C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *3rd IC on Intelligent Agent Technology*, pages 531–535, 2004.

[25] M.-C. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a DisCSP on privacy loss. In *AAMAS*, pages 1396–1397, 2004.

[26] A.J. Viterbi. Error bounds for convolutional codes and an asymtotically opti mum decoding algorithm. *IEEE Trans. on Information Theory*, 13(2):260–267, 1967.

[27] M. Yokoo and K. Suzuki. Generalized Vickrey Auctions without Third-Party Servers. In *FC04*, 2004.