Secure Combinatorial Optimization using DFS-based Variable Elimination

[†]Silaghi, M. and [‡]Petcu, A. and [‡]Faltings, B.
[‡]Florida Tech
[‡]EPFL

July 19, 2005

Abstract

It is known that, in general, Constraint Optimization Problems (COP) are NP-hard. Existing arithmetic circuits for secure protocols solving such problems are exponential in the number of variables, n. Recently a combinatorial optimization algorithm was proposed whose cost is exponential only in a parameter of the Depth First Search tree (DFS) of the constraint graph, smaller than n. We show how to construct an arithmetic circuit with this property and solving any COP. For forest constraint graphs, this leads to a linear cost secure solver.

1 Introduction

Combinatorial Optimization is an important operation in many problems. 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 $D = \{D_1, ..., D_m\}$, and a set of weighted constraints $C = \{\phi_0, ..., \phi_m\}$, each such constraint ϕ_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 $v \in D_i$. A solution of the COP is a tuple of assignments ε with values for each variable in X such that the sum of the weights associated by the constraints in C to ε is maximized (minimized). 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 ε on a set of variables X_i by $\varepsilon_{|X_i}$, then the solution is:

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

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 ϕ_0 is a public constraint and that X_m, the set of variables in ϕ_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 combining the constraints such that there remains a single constraint per variable (with his ancestors in the tree).

Our work employs the following secure techniques:

- polynomial secret sharing: Each participant k out of n participants receives $\langle s \rangle_k^t = s + \sum_{i=1}^{t} a_i k^i$. The secret could be reconstructed with $s = \sum_{k=1}^{t+1} l_{k,t}$, where $l_{k,t}$ are the corresponding Lagrange coefficients.
- addition of shared secrets: $\langle s_1 + s_2 \rangle_k^t = \langle s_1 \rangle_k^t + \langle s_2 \rangle_k^t$
- resharing shared secrets: To reshare a secret $\langle s \rangle^t$ with another threshold t', each share $\langle s \rangle_k^t$ is shared with $(t'+1, n)$-polynomial sharing scheme.
- multiplication of shared secrets: $\langle s_1 * s_2 \rangle_k^{2t} = \langle s_1 \rangle_k^t * \langle s_2 \rangle_k^t$
- secure test: $\delta(x)$ returns 1 if $x = 0$ and 0 otherwise.
- secure Kronnecker's δ: $\delta_K(x, y) = \delta(x - y)$ returns 1 if $x = y$ and 0 otherwise.
- secure comparison: $cmp(x, y)$ returns 1 if $x < y$ and y 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. The former techniques seeking the strongest privacy guarantees are based on secure multiparty computation and scan several times the whole search space, i.e. Cartesian product of domains in D, once for each possible total weight. An optimization protocol specialized on generalized Vickrey auctions and based on dynamic programming is proposed in [12] and is significantly more efficient, but does not randomize the selection of the solution needed to increase privacy [10]. A dynamic programming algorithm for solving (D)COPs was proposed in [4] and consists of a Viterbi-like combination of a maximization and decoding [11]. The algorithm in [4] can also be seen as a clever heuristique for variable elimination [2], or as a parallelization of ADOPT [3], and is based on a different concept of privacy [8].

2.1 Variable Elimination

Variable Elimination is a principled technique for complexity reduction in COPs. It consists of replacing all the constraints linked to a variable chosen for elimination by the projection of their composition on the remaining variables. A heuristique for selecting the variables to be eliminated next is provided by the DFS tree [4].

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. A Depth First Search (DFS) tree associated to a COP is a spanning tree generated by the first arcs used for visiting each node during some depth first traversal of its primal graph. DFS trees were first successfuly used for Distributed Constraint problems in [1]. The property exploited there is that separate branches of the DFS-tree are completely independent once the assignments of common ancestors are decided.

The ancestors of x_i are all the nodes of the path between root of the tree and x_i, inclusively the root. The descendants of x_i are all the nodes for which x_i is an ancestor. We use the following notation. Let:

- F_x be the parent of x
- S_x be the children of x
- P_x be the neighbor ancestors of x
- G_x be the induced parents of x (ancestors that are neighbors for x or for some descendant of x)

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

2.3 DFS-based Variable Elimination

A heuristique for selecting the order to eliminate variables based on exploiting the DFS-tree is proposed in [4]. 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 heuristiq 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 heuristique is that the quality of an elimination order can be easily evaluated.

2.4 Secure Optimization

A secure optimization algorithm for DCOP is proposed in [9]. 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 [6] 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 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, un (upward) d ynamic programming step, and a (downward) decoding step.

3 Arithmetic Circuits

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:

$$W_i^{x_{F_i}}[S] = \max_{v \in D_i}(W_i^{x_i}[\langle x_i, v\rangle \cup S_{|P_{x_i}}] + \sum_{y \in S_{x_i}} (W_i^y[S \cup \{\langle x_i, v\rangle\}|_{G_y}])$$

The value $W_r^{\emptyset}$, where r is the index of the root node, is the weight of the optimal solution.

procedure *Upward(x_i)* **do**
  foreach *($y \in S_{x_i}$)* **do**
    Upward(y);
  foreach *tuple ε for $G_{x_i} \cup \{x_i\}$* **do**
    $W^{x_i}[\varepsilon] = W_i^{x_i}[\varepsilon_{|\{x_i\} \cup P_{x_i}}] + \sum_{y \in S_{x_i}} (W_i^y[\varepsilon_{|G_y}]$;
  foreach *tuple ε for G_{x_i}* **do**
    $W_i^{x_{F_i}}[\varepsilon] = \max_{v \in D_i}(W^{x_i}[\varepsilon \cup \langle x_i, v\rangle])$;

Algorithm 1: Arithmetic circuit for the upward (dynamic programming) step. At the first call, the parameter x_i is the root of the DFS tree.

3.1 Unsecure decoding of solution

If we also want to reaveal 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 is the shared secret $W_r^{\emptyset}$ computed at the upward step. At variable x_i with subtree value W, for each tuple ε with value W' in the vector W^{x_i} and with assignments equal to the ones selected at previous levels, compute and reveal $\delta_K(W, W')$. If the result is 1, the corresponding assignment of x_i is selected, and the corresponding values W_y in W_i^y 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, e.g. [5]. We fix this in the following algorithm.

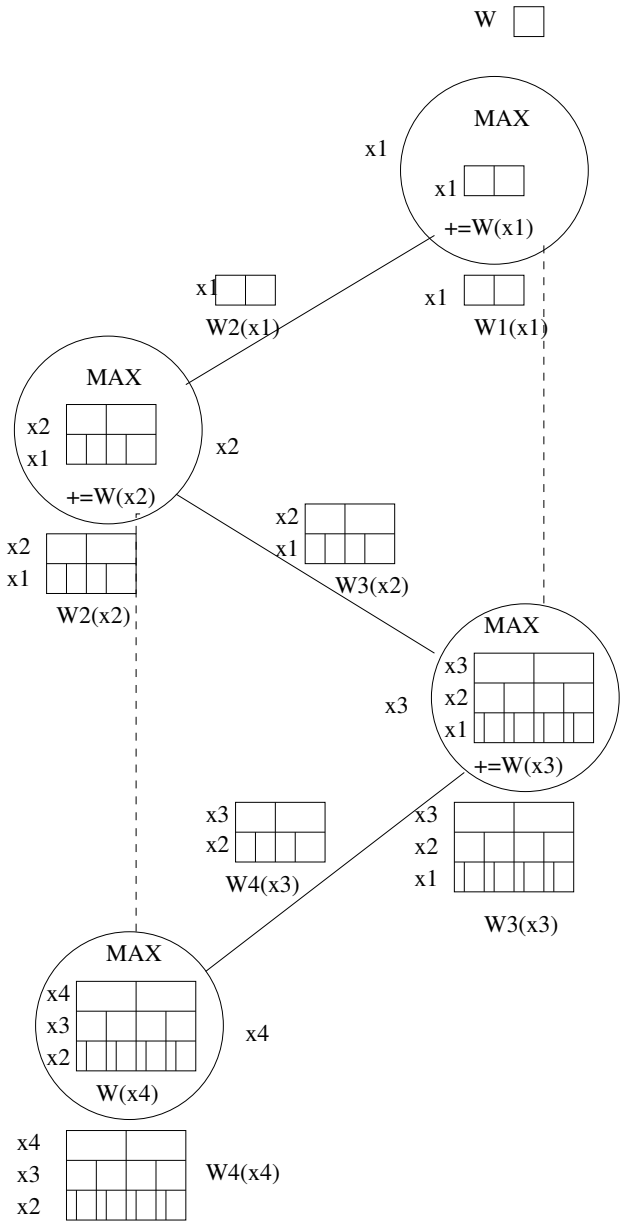

Figure 1: Data structures in the Upward computation.

3.2 Secure decoding of solution

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_{x_k}, and the shared selected weight of this variable, $W^{\emptyset}_{x_i}$. The procedure also computes the inputs for the next recursive procedure calls, at the descendents of x_i, namely the selected weight $W^{\emptyset}_{y}$ of each child $y \in S_{x_i}$.

$$W^{\emptyset}_{y} = \sum_{\varepsilon} (\prod_{p \in G_{x_i}} d_p[\varepsilon_{|p}]) \delta_K(W^{\emptyset}_{x_i}, W^{x_i}[\varepsilon]) W^{x_i}_{y}[\varepsilon]$$

$$d_{x_i}[v] = \sum_{\varepsilon, \varepsilon_{|\{x_i\}}=v} (\prod_{p \in G_{x_i}} d_p[\varepsilon_{|p}]) \delta_K(W^{\emptyset}_{x_i}, W^{x_i}(\varepsilon))$$

procedure *Downward(CSP)* **do**
  while $x_i \leftarrow$ *get_InOrder_Next(DFS(CSP))* **do**
    foreach $v \in D_i$ **do**
      $d_{x_i}[v] = \sum_{\varepsilon, \varepsilon_{|\{x_i\}}=v} (\prod_{p \in G_{x_i}} d_p[\varepsilon_{|p}]) \delta_K(W^{\emptyset}_{x_i}, W^{x_i}(\varepsilon))$;
    foreach $y \in S_{x_i}$ **do**
      $W^{\emptyset}_{y} = \sum_{\varepsilon} (\prod_{p \in G_{x_i}} d_p[\varepsilon_{|p}]) \delta_K(W^{\emptyset}_{x_i}, W^{x_i}[\varepsilon]) W^{x_i}_{y}[\varepsilon]$;

Algorithm 2: Arithmetic circuit for the downward step.

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.

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 $|G_{x_i}|$.

In the Downward step there exist a *while* loop for each variable x_i and each such cycle has two summations for each element in W^{x_i}, each term having $|G_{x_i}| + 1$ multiplications. The total cost for the downward step is $O(ngd^{g+1})$.

Therefore, the total complexity of the secure version is $O(ngd^{g+1})$. If the downward version with immediate revelation of assignments in solutions is used, then the complexity is only $O(nd^{g+1})$. 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 [7].

5 Conclusion

We have shown how to speed up the secure computation for constraint optimization, by exploiting fix-cost DFS-based Variable Elimination. The previ-

ous 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.

References

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

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

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

[4] A. Petcu and B. Faltings. Distributed variable elimination. In *CP'04 DCR Workshop*, 2004.

[5] M. Silaghi. Secure generalized vickrey auctions. In *IJCAI05 DCR Workshop*, 2005.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.

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