Optimization in Private Stable Matching with Cost of Privacy Loss

Marius C. Silaghi[1], **Prashant Doshi**[2], **Toshihiro Matsui**[3], **Makoto Yokoo**[4], **Markus Zanker**[5]
[1]Florida Tech, [2]University of Georgia, [3]Nagoya Institute of Technology, [4]Kyushu University, [5]University of Klagenfurt

Abstract

We introduce, model and solve *private non-transitive-preference-based stable matching* using a new optimization framework that models privacy-loss as utility loss. Classic stable matching problems are a well known tractable application with many uses. However, some versions of stable matching problems are not tractable. One such version that we identified earlier is the stable matching with privacy of preferences. Privacy of preferences requirements preclude centralization of the data for running efficient algorithms. Distributed solving has to minimize privacy loss.

A further version of stable matching that is not tractable is when preferences are not transitive, namely when somebody prefers A over B, and prefers B over C, but prefers C over A. This situation occurs in nature under the name *Rock-Paper-Scissors mating pattern* (RPS). RPS stable matching (RPS-SMs) does not always have stable solutions and we define it as an optimization problem minimizing the number of instabilities. Private RPS stable matching problems can be represented as distributed constraint optimization problems (DCOPs), but DCOPs do not have a formal way to represent privacy requirements. We show how to represent RPS-SMs using a new framework, distributed private constraint optimization (DPCOP) which models privacy-loss as a utility loss, of the same nature as the utility loss specified by constraint weights.

Introduction

Classic stable matching is a well studied problem. It is known to be tractable, since the Gale-Shapley algorithm solves it in quadratic time. However, some of its variants are not solved by the Gale-Shapley, and remain a good research motivation for the constraint community. One such variant that we introduce here is characterized by:

- preferences are private to their agent, precluding centralization,
- preferences are not transitive, making it is possible that no stable solution exists.

We introduce these problems, proposing a random generator. Then we model and solve them with a new framework. The distributed private constraint satisfaction (DisPrivCSPs) framework (Silaghi & Faltings 2002) models problems with privacy requirements and enables qualitative and quantitative comparison of distributed CSP solvers. Here we will introduce a similarly powerful framework extending DisPrivCSPs to distributed constraint optimization (DCOP). Significant attention was previously given to the definition and analysis of privacy requirements in distributed constraint satisfaction problems (DisCSP) (Yokoo *et al.* 1998). For a given agent A_i, all the solutions of a DisCSP have the same reward (utility), U_i. At the basis of the DisPrivCSP theory is the observation that a rational agent A_i will drop out of search when it expects that the price of its future privacy loss is higher than U_i (intuitively leading to a negative total utility). Past privacy loss does not matter since its value was already lost. The quality of a solution to be searched for by a DisCSP algorithm is therefore defined only by the privacy criteria.

Distributed constraint optimization problems (DCOPs) are an extension of DisCSP where constraints have different costs. Some attention was already given to privacy in distributed constraint optimization (Silaghi & Mitra 2004; Maheswaran *et al.* 2006; Silaghi, Faltings, & Petcu 2006; Greenstadt, Grosz, & Smith 2007). In prior work, DCOPs with privacy requirements are treated as a multi-criteria optimization where the constraint weights are of a different nature from privacy (often perceived from an information theoretic perspective). The changes needed for generalizing the DisPrivCSP framework have not yet been analyzed in the original direction (a usage with DCOPs appears in (Maheswaran *et al.* 2006)). We start by noting that its original idea does not immediately apply to DCOPs, since in DCOPs the value of the optimal solution is not known in advance (otherwise the problem would become a constraint satisfaction problem). While DisPrivCSPs are optimization problems for the privacy criterion, DCOPs were so far seen as multi-criteria optimization problems along two incomparable metrics (privacy/entropy and cost/utility). We explicitly model the loss of privacy as a cost and we assume that this cost is provided as a part of the input specification. When the costs of revealing each secret can be obtained like this, they allow for much better targeted strategies, keeping the most valuable secrets and revealing less valuable secrets (rather than just maximizing the entropy by guarding many irrelevant secrets).

We now introduce a new framework called Distributed

Private Constraint Optimization (DPCOP) where the two – previously incomparable – metrics of DCOPs are redefined and merged under the utility theory, yielding a unique and easy to analyze optimization criteria. Stable Matching is presented as a set of benchmarks for the new framework, and baseline algorithms are proposed and evaluated experimentally.

The DPCOP framework also defines a new hybrid between the DCOP and the multi-agent planning research areas, since a DPCOP solver becomes a planner (where the actions taken during search have an impact on the solution).

Classic stable matching have been modeled previously in the DCOP framework (Silaghi, Zanker, & Bartak 2004). Its modeling was also proposed in other distributed frameworks (Brito & Meseguer 2005), that also do not explicitly quantify privacy loss. Here we deal with the new type of stable matching and show results with the DPCOP framework.

Related Work

Privacy has been a fundamental motivation for distributed constraint optimization, since the early beginning of the field (Yokoo *et al.* 1998). However, due to wide disagreement on how to formalize privacy requirements, proposed techniques are often evaluated not from the privacy perspective but solely from the perspective of efficiency and cost. The first quantitative measurement of privacy loss was based on simply counting the number of disclosed tuple values (Freuder, Minca, & Wallace 2001). The distributed private constraint satisfaction problems introduced in (Silaghi & Faltings 2002) label each secret with a number corresponding to the cost induced by its privacy loss. The privacy loss incurred during a computation is given by the sum of the the privacy values of each leaked secret. Other approaches to quantifying privacy loss are based on information theory, maximizing entropy. The privacy loss is therefore expressed in bits of information, or some related units (Maheswaran *et al.* 2006). All these approaches lack a clear way to trade off privacy for solution quality, resulting in a difficult multi-criteria optimization.

Distributed Private CSPs

With DisPrivCSPs (Silaghi & Faltings 2002), each secret (cost/weight of a constraint or combination thereof) is associated with a *privacy value*. The privacy value of a secret specifies the incremental loss of utility due to the revelation of that secret. Note that DisPrivCSPs could only model additive privacy loss, restriction also removed in the definition proposed here. The *reward for solving the problem* is given as a constant. The weights (satisfying/unsatisfying) of a constraint have no direct relation to the utility. Agents in DisPrivCSPs abandon the search when the utility loss due to predictable privacy loss (next incremental privacy loss) is higher than the reward for finding a solution. A qualitative comparison of algorithms was possible based on the ability to solve problems without abandoning the search. Quantitative comparison is also possible and agents can minimize their privacy loss (Wallace & Silaghi 2004).

Baseline Algorithms

The simplest distributed algorithm for solving distributed CSPs is the one proposed in (Freuder, Minca, & Wallace 2001). In this algorithm, an agent proposes a value for the variables (a solution) at a time, and the other agents answer with messages specifying whether their constraints are satisfied by that assignment.

There are other algorithms for solving DCOPs such as ADOPT (Modi *et al.* 2005), DPOP (Petcu & Faltings 2005), and DisAO (Mailler & Lesser 2004). There also exist DCOP optimization techniques using cryptographic protocols (Silaghi, Zanker, & Bartak 2004; Greenstadt, Grosz, & Smith 2007), and which offer significantly high levels of privacy guarantees.

Stable matching.

We have earlier modeled classic stable matching in the DCOP framework (Silaghi, Zanker, & Bartak 2004). Its modeling was also proposed in other distributed frameworks (Brito & Meseguer 2005) that like DCOP do not model explicitly the privacy loss. Cryptographic solutions are given in (Silaghi, Zanker, & Bartak 2004; Golle 2006).

DPCOP Framework

Prior work treated distributed constraint optimization problems (DCOPs) with privacy requirements as a multi-criteria optimization, where the constraint weights are measured in utility, and privacy is measured in information bits or related metrics. Here we propose to measure privacy in the same type of utility as the constraint weights.

In order to extend the DisPrivCSP framework to DCOPs, we start from the observation that the DCOP constraint weights, normally used in the objective function of the optimization, can also be considered to be a positive (or negative) utility – or cost – of the same nature as the cost induced by privacy loss. As such, in DCOPs *minimizing* the sum of the constraint weights (i.e., where weights represent a cost, with negative utility), the total cost is given by the sum between the value of the total lost privacy and the cost of the selected solution. The reward for each agent of solving the problem will still be considered in this setting to be a previously known (possible infinite) value, like with DisPrivCSPs. A rational participating agent is expected to *abandon* the search if its next revelation would lead to a value for the *incremental privacy loss* which, together with a lowest bound on the cost of the solution, becomes larger than the reward for solving the problem.

For *maximization* DCOP problems, namely problems seeking a solution maximizing the sum of the constraint weights (i.e., where constraint weights represent rewards), privacy loss becomes the only cost. The utility is defined by the difference between the reward of the solution and the value of the privacy lost during the search. A rational agent will therefore abandon the search problem if its next (or expected) disclosures leads to an *incremental privacy loss* that is larger than the expected total reward of the solution.

In order to formally define the framework described so far, we first formalize the concept of privacy leaks in a way general enough to model non-additive functions. Given a set

of secrets, a leaked information about some of these secrets will be called *revelation*.

DEFINITION 1 (REVELATION). *Given a set of secrets S and a set of agents A, the set of possible revelations $R(S, A)$ is a function $R(S, A) : A \rightarrow (S \rightarrow [0, 1])$ which maps each peer agent to a functional relation specifying the probability learned by that agent for each secret.*

Note that this definition of revelation is more general than the version used by DisPrivCSPs, as here it can model statistical privacy losses. While the above definition has a rich modeling power, one can assume that sometimes users may find it difficult to provide the data related to all possible revelations defined in this way. We therefore also consider a simplified version that requires less data (but is somewhat less general):

DEFINITION 2 (SIMPLIFIED REVELATION). *Given a set of secrets S and a set of agents A, the set of possible revelations $R(S, A)$ is the function, $R(S, A) : A \rightarrow \mathcal{PS}(S)$, which maps each agent to an element in the power-set of the set of secrets, $\mathcal{PS}(S)$.*

The simplified revelation definition assumes that privacy is lost only when a secret is completely revealed. It does not account for secrets about which other probabilistic information is made available.

DEFINITION 3 (DPCOP). *A (minimization) Distributed Private Constraint Optimization Problem (DPCOP) is defined by a tuple (A, X, D, C, P, U). A is a set of agents $\{A_1, ..., A_K\}$. X is a set of variables $\{x_1, ..., x_n\}$, and D is a set of domains $\{D_1, ..., D_n\}$ such that each variable x_i may take values only from the domain D_i. The variables are subject to a set C of sets of weighted constraints $\{C_0, C_1, ..., C_K\}$, where $C_i = \{\phi_i^1, ..., \phi_i^{c_i}\}$ holds the secret weighted constraints of agent A_i, and C_0 holds the public constraints. Each weighted constraint is defined as a function $\phi_i : X_i \rightarrow \mathbb{R}_+$ where $X_i \subseteq X$. The value of such a function in an input point is called constraint entry. Each C_i can be viewed as the set of the secret constraint entries in its weighted constraints.*

P is a set of privacy loss cost functions $\{P_1, ..., P_K\}$, one for each agent. P_i defines the cost inflicted to A_i by each revelation r of its secrets, i.e., $P_i(r) : R(C_i, A) \rightarrow \mathbb{R}_+$.

A solution is an agreement between agents in A on a tuple ε^* of assignments of values to variables that minimizes the total cost:

$$\varepsilon^* = argmin_\varepsilon \sum_i (\sum_j \phi_i^j(\varepsilon)) + P_i(\Pi_i(\varepsilon))$$

where $\Pi_i(\varepsilon)$ is the revelation in $R(C_i, A)$ performed during the process leading to the agreement on the assignments ε.

U is a set of rewards $U_1, ..., U_K$, one for each agents, that the corresponding agent receives if a solution is found, and that agents use for deciding whether to abandon a search given their foreseen incremental privacy loss.

The set of rewards U can be used to qualitatively compare DCOP solvers, as to which solver can solve more problems than another solver without any agent abandoning the process. Such a hierarchy of solvers was built for DisPrivCSPs in (Silaghi & Faltings 2002). Formally, the agent A_i abandons the search if:

$$P_i(r*) - P_i(r) + W \geq U_i$$

where r is the revelation performed by A_i up to this moment, $r*$ is the revelation after the next planned sequence of actions, and W is a low bound on the quality of the expected solution.

The *privacy-loss cost functions* P_i are a new concept. These functions are part of a problem model (just like utilities of auction outcomes, used to infer bids in Vickrey auctions). Just as utilities are an agent's input for auctions, a privacy-loss cost function is an agent's input for DPCOPs. An agent can infer a privacy-loss cost function by simulating how much utility it may lose when each revelation is performed.

The above DPCOP definition is for the general case where the constraint of an agent may involve all variables (Silaghi & Yokoo 2008). Many approaches consider a simplified version (equivalent in expressive power) where each agent *owns* some variables, and agents enforce only constraints with variables assigned by previous agents. We provide next the corresponding DPCOP simplification, allowing for most existing DCOP algorithms.

DEFINITION 4 (SIMPLIFIED DPCOP). *A (minimization) distributed private constraint optimization problem is defined by a tuple (A, X, D, C, P, U). A is a set of agents $\{A_1, ..., A_n\}$. X is a set of variables $\{x_1, ..., x_n\}$, and D is a set of domains $\{D_1, ..., D_n\}$ such that each variable x_i may take values only from the domain D_i. The variables are subject to a set C of sets of weighted constraints $\{C_0, C_1, ..., C_n\}$, where $C_i = \{\phi_i^1, ..., \phi_i^{c_i}\}$ holds the secret weighted constraints of agent A_i, and C_0 holds public constraints. Each weighted constraint is defined as a function $\phi_i^j : X_i \rightarrow \mathbb{R}_+$ where $X_i \subseteq \{x_1, ..., x_i\}$.*

P is a set of privacy loss cost functions $\{P_1, ..., P_n\}$, one for each agent. P_i defines the cost inflicted by the revelation of any subset of secret elements of $P_i : R(C_i, A) \rightarrow \mathbb{R}_+$.

A solution is an agreement between the agents in A on a tuple ε^* of assignments of values to variables that minimizes the total cost:

$$\varepsilon^* = argmin_\varepsilon \sum_i (\sum_j \phi_i^j(\varepsilon)) + P_i(\Pi_i(\varepsilon))$$

where $\Pi_i(\varepsilon)$ is the revelation in $R(C_i, A)$ performed during the process of agreeing on the assignments ε.

U is a set of rewards $U_1, ..., U_K$, one for each agents, that the corresponding agent receives if a solution is found.

Maximization.

Maximization DPCOPs are defined similarly, but without the element U, and redefining the solution as:

$$\varepsilon^* = argmax_\varepsilon \sum_i (\sum_j \phi_i^j(\varepsilon)) - P_i(\Pi_i(\varepsilon)).$$

An agent abandons the maximization search if:

$$W - (P_i(r*) - P_i(r)) \leq 0$$

where r is the revelation performed by A_i up to this moment, $r*$ is the revelation after the next planned sequence of actions, and W is an upper bound on the quality of the expected solution.

Simplified cost functions.

While (in general) privacy-loss cost functions are not additive, we expect that additive randomly generated benchmarks have the simplicity that can help in the theoretical understanding of the new framework. In a simplified version, the value of privacy leaks towards a peer agent can also be considered independent of privacy leaks towards other peers (assumption not applicable to all problems). For *additive* privacy cost functions, an array of privacy costs can simply be attached to each constraint tuple.

An important case of *non-additive* privacy-loss cost function is where the cost of a leak is independent of the agent (revelation to an agent being considered to be a revelation to all agents), while being additive along the dimension of the secrets. Such a privacy cost function can be represented by a single cost associated with each constraint tuple.

Comparison with previous frameworks

The closest previous framework is the Distributed Private CSPs (DisPrivCSPs) introduced in (Silaghi & Faltings 2002), which deals with distributed constraint satisfaction problems (DisCSPs). DisPrivCSPs also have costs for privacy loss, but that cost is not integrated in any way with the cost of the agreement tuples.

DisCSPs can be modeled as a special case of DCOPs, namely when the constraints are functions with results only in $\{0, \infty\}$, rather than in $\mathbb{R}_+$. This is because:

$$\sum_i (\sum_j \phi_i^j(\varepsilon))$$

has the same value for all the satisfying tuples of the DisPrivCSP.

Previous research related to privacy in DCOPs has already found inspiration in DisPrivCSPs (Maheswaran *et al.* 2006), and can be seen as straightforward applications of DisPrivCSPs to DCOPs. DPCOPs are a less straightforward extension of DisPrivCSPs. We think that the main innovation in DPCOPs versus a straightforward DisPrivCSPs usage with DCOPs is:

- DPCOPs unify the metric for cost of privacy loss with the metric used for specifying weights of constraints (in DisPrivCSPs they were incomparable metrics).
- The revelation is more general in DPCOPs, allowing for statistical and non-additive privacy loss functions.

Among smaller differences, while with DisPrivCSPs an agent A_i will abandon the search when incremental costs are higher than U_i, with maximization DPCOPs there may be no known finite limit on the reward of the agent. Also, for DisPrivCSPs we provided only theoretical and qualitative comparison of techniques, while with DPCOPs we provide benchmarks, random problem generators, and experimental analysis of techniques (`http://www.cs.fit.edu/~msilaghi/DPCOP`).

Baseline DPCOP Solvers

Any of the existing DCOP techniques can be used to solve DPCOPs. Techniques using cryptographic methods, such as the ones in (Silaghi, Zanker, & Bartak 2004; Greenstadt, Grosz, & Smith 2007), can guarantee optimality with minimal privacy leak. Other techniques may offer more efficiency at the expense of optimality. We evaluate simple algorithms for solving DPCOPs. Probably the simplest technique consists of an agent consecutively asking each publicly possible tuple one after another, while the other agents answer with their costs. This is an adaptation to optimization of the technique proposed in (Freuder, Minca, & Wallace 2001). The agent asking the questions in this **1-leader** version is called *the leader*. In the **N-leaders** variant, the search space is distributed between agents (related to (Hamadi 2001)), and each agent asks costs for his part. The baseline version we evaluate in the N-leaders version is even simpler, with agents acting in turn rather than simultaneously, each question also delegates the leader for the next question. At the end, the agents publish the best tuples for their sub-parts, and the best overall tuple is selected.

procedure *leader* **do**
  foreach *next tuple ε with better local weight than currently best tuple* **do**
    decide next_leader // only N-leaders version;
    send ask(ε,next_leader);
    set next leader // only N-leaders version;
    wait answers;
    update identity of best tuple;
  end
end do.
procedure *slaves* **do**
  when *ask (ε, next_leader)* **do**
    compute local cost for ε;
    send answer(ε, cost) to leader;
    recompute privacy_loss;
    leader := next_leader // only N-leaders version;
    if *(leader = myself)* **then**
      change to leader mode // N-leaders version;
    end
  end do.
end do.

Algorithm 1: Baseline (1-leader and N-leaders versions)

Leaders may propose tuples that are suboptimal (with worse local cost than their currently best tuple), lying to increase privacy (lying occurs also in (Brito & Meseguer 2007)).

RPS Stable Matchings Benchmarks

It is easy to learn a secret weight of a constraint entry for an agent when a message sent by this agent is based solely on the weight of that secret constraint entry. If an agent controls a single secret constraint, each message that the agent sends in response to a leader's challenge reveals a secret weight. If an agent holds several secret constraints, a message is an aggregation of secrets from those constraints, and learning the component secrets is sometimes possible, but

more computationally involved (solving the corresponding systems of equations, when they are determined). First we perform an experimental study for the simpler case where *each agent enforces a single private constraint*.

DPCOP files.

Each randomly generated stable matching DPCOP is stored in a file in the following format:

```
<nb variables>
<var_name1> <dom_size> <val_1> ... <val_d>
<var_name2> <dom_size> <val_1> ... <val_d>
...
<nb constraints>
<arity1>
<owner>
<var1_name>
...
<size privacy-vector/tuple>
<weight1> [<privacy vector>] ...
```

Stable matching.

Distributed stable matching consist in matching m participants of a type to m participants of another type. We developed a generator for DPCOP models of stable matching between m agents of a type and m agents of another type. The first m agents are of one type, and the last m agents are of the second type. To model secret preferences we introduce a variable $P^i_{j,k}$ for each pair j, k, $0 \leq j < k \leq m$ (Silaghi 2004b; Silaghi, Zanker, & Bartak 2004; Silaghi 2004a). In a version with 2 preferences, variable $P^i_{j,k}$ has one of two values (0 meaning that A_i prefers A_{2m-j} to A_{2m-k}, and 1 meaning that it does not prefer A_{2m-j} to A_{2m-k}. Each agent A_i receives $m*(m-1)/2$ private unary constraints on the variables $P^i_{j,k}$. In a version with 3 preferences, a private variable has one of three values: 0 meaning that A_i prefers A_{2m-j} to A_{2m-k}, 1 meaning that it equally prefers A_{2m-j} and A_{2m-k}, and 2 stands for the remaining situation.

In general someones preferences may not be transitive (as in rock/paper/scissors mating patterns (Sinervo & Livley 1996)). For such problems it is possible that $P^i_{j,k}$ and $P^i_{k,t}$ are both 0, but $P^i_{j,t}$ is 1. We call this version RPS Stable Matching Problems, and remark that instances of such problems may not have any stable solution.

The definition of the problem is based on m additional variables, x_i, each of them with m values. The value of x_i give the index of the participant that is matched with A_i in a stable solution. The conditions of stability and the fact that each agent can be matched with exactly one agent of the other type, are specified using a set of public quaternary constraints. A quaternary constraint is created between each quadruplet x_i, x_j, $P^i_{u,v}$, and $P^u_{j,i}$. This constraint specifies that:

"If A_i is matched with A_{2m-u} and A_j is matched with A_{2m-v}, then u must be different from v; and if A_i prefers A_{2m-u} to A_{2m-v}, then A_{2m-v} prefers A_j to A_i."

In the generated DPCOP models, the private constraints are hard constraints (with weights in $\{0, \infty\}$), to anchor in reality the evaluation of a solution. The public constraints are soft constraints, each unstable matching having cost 1. The fact that each agent is matched with exactly one other agent of the opposite type remains a hard constraint. The rewards generated for reaching a solution with minimization DPCOPs are infinite.

A pseudocode of the RPS-SM problem generator is given in Algorithm 2.

print the number of variables;
print the variables with their domains;
print total number of constraints;
foreach *participant* A_i **do**
  foreach *participant pair* (A_k, A_j) *of opposite type* **do**
    print i's unary secret constraint on $P^i_{k,j}$;
  end
end
foreach *guadruple:* (x_i, x_j, $P^i_{u,v}$, $P^u_{j,i}$) **do**
  print the public constraint x_i, x_j, $P^i_{u,v}$, and $P^u_{j,i}$. This constraint specifies that, if A_i is matched with A_{2m-u} and A_j is matched with A_{2m-v}, then u must be different from v; and if A_i prefers A_{2m-u} to A_{2m-v}, then A_{2m-v} prefers A_j to A_i.
end

Algorithm 2: Random RPS Stable Matching generator

An example file (with additive privacy costs) is:

```
# generated with ./stableGenMax 2 2 4 2
4 # nb agents
6 # nb variables
# variable_name
#      domain_size list_of_values
x0      2 0 1
x1      2 0 1
PA_0_1_0        2 0 1
PA_1_1_0        2 0 1
PB_0_1_0        2 0 1
PB_1_1_0        2 0 1

8 # nb constraints

1 # arity
0 # owner agent
PA_0_1_0 # variable
4 # size of privacy-cost list
  0 [ 0.012 2.1 3.3 0.32 ]
INF [ 2.2 1.1 2.1 3.7 ]

1 # arity
1 # owner agent
PA_1_1_0 # variable
4 # size of privacy-cost list
INF [ 1.4 4 2.9 3 ]
  0 [ 2.5 2.7 2.3 0.72 ]

1
2
```

```
PB_0_1_0
4
INF [ 3.5 2 2.3 2.4 ]
  0 [ 2.1 0.65 3.5 0.42 ]

1
3
PB_1_1_0
4
INF [ 0.058 0.46 3.3 3.3 ]
  0 [ 1.8 1.4 1.8 3.9 ]

4 # arity
-1 # owner: public - no owner
x0      # variable
x1      # variable
PA_0_1_0        # variable
PB_0_1_0        # variable
0 # privacy loss: no private element
 INF INF
 INF INF
 0 0
 0 0
 1 0
 0 0
 INF INF
 INF INF

4 # arity
-1 # owner: public - no owner
x0      # variable
x1      # variable
PA_0_1_0        # variable
PB_1_1_0        # variable
0 # privacy loss: no private element
 INF INF
 INF INF
 0 0
 1 0
 0 0
 0 0
 INF INF
 INF INF

4 # arity
-1 # owner: public - no owner
x1      # variable
x0      # variable
PA_1_1_0        # variable
PB_0_1_0        # variable
0 # privacy loss: no private element
 INF INF
 INF INF
 0 0
 0 0
 0 1
 0 0
 INF INF
 INF INF

4 # arity
-1 # owner: public - no owner
x1      # variable
x0      # variable
PA_1_1_0        # variable
PB_1_1_0        # variable
0 # privacy loss: no private element
 INF INF
 INF INF
 0 0
 0 1
 0 0
 0 0
 INF INF
 INF INF
```

These files are meant as inputs to simulators of DPCOP solvers. In real solvers, each agent could take as inputs a similar file, but where the constraints owned by other agents are empty (detection of an owner field naming another agent signifies that the token after the last variable belongs to the next constraint).

With the baseline algorithms, the public constraints are evaluated only by the leader, while the other agents evaluate only the private constraints.

Privacy Leaks.

For problems with $m = 2$ the number of private constraints per agent is 1, and therefore secrets are lost each time that they are used for answering to a leader with a cost (weight).

For problems with more participants, one positive (0 weight) answer in any of the two baseline techniques will reveal all three secrets involved, but a infinite weight answer contains an aggregated information about a set of secrets. It reveals one secret only if the remaining secrets aggregated with it are revealed by finite weight answers, and requires additional data storage for reconstructing shadow COPs (Wallace & Silaghi 2004).

The inference technique is shown in Algorithm 3.

This algorithm creates in each agent a shadow of each secret unary constraint of the other agents. This shadow is filled with each new information received from the corresponding agent. Each time a finite weight is received from an agent for a given assignments tuple, the projection of the tuple on the variables of the secret unary constraints of that agent are marked as having cost 0. Whenever an infinite cost is received for a tuple, the set of projections of the tuple on the variables in the secret unary constraints of the sender is enqueued in the list of ∞ cost answers (if the number of unknown secrets involved is smaller than a bound k, used to bound the space complexity). Each time that a set from this list contains a known infinite cost element, it is removed. Any 0-cost element is removed from its set. When a set from this list contains a single unknown element (is unary), we infer that this element has infinite weight.

For non-RPS Stable Matching problems, where preferences present transitivity properties, one can also apply

shadow constraint ← unknown;
list of ∞ cost answers ← $\emptyset$;
when *finite cost answer* **do**
  set values in shadow COP to 0;
  remove assignments of newly learned unary constraints from ∞ cost answers;
  revisit ∞ cost answers becoming unary;
end do.
when ∞ *cost answer* **do**
  remove variables of known constraints from answer;
  if *answer less than* k*-ary* **then**
    add answer to list of ∞ cost answers;
  end
  revisit ∞ cost answers becoming unary;
end do.
procedure *revisit* ∞ *cost answers becoming unary* **do**
  set shadow tuples in unary ∞ cost answers to infinity;
  remove the cost answer from the list of answers;
end do.

Algorithm 3: SMI: Stable Matching Inference of secrets

Algo	DPCOP	Size	Pref	Cost	Cycles	Time
HP	STM	4	2	3	14	1.47
BL	STM	4	2	257	9.6	0.85
BL	RPSM	4	2	249	8.8	0.56
BL	RPSX	4	2	310	27.6	0.82
BL	RPSX	4	3	443	116.7	0.7
BL	RPSX	6	2	1502	$9.6*10^5$	5822

Table 1: Stable matching versions.

Floyd-Warshall to compute the transitive closure of these preferences, recovering additional secrets.

Privacy loss avoidance.

Inferred shadows for the secret constraints of other participants are used by the leader to predict the answers of other agents and to even skip asking the question if the prediction proves that the current tuple is suboptimal (e.g., one of the other agents have preferences that make a matching unstable).

Experimental Results.

For stable matching problems, results averaged over 25 instances are given in Table 1. The cases are: RPS stable matching with soft constraints (RPSX), RPS stable matching with hard constraints (RPSM), classic stable matching with transitive preferences (STM). Results are given for the baseline algorithm with one leader (BL), and for Hamilton and Premkumar's (HP) cryptographic implementation of the (Silaghi 2004a; Hamilton, Premkumar, & Silaghi 2005) technique. The cryptographic algorithm leaks only the secrets implied by the fact that the solution is stable (if an agent A prefers another than its match in the solution, that other did not prefer A to his match).

Some cryptographic solvers are guaranteed to find optimal solutions for DPCOPs, at the expense of efficiency (Silaghi, Zanker, & Bartak 2004). Assuming that no two agents exchange information about peers, there exist partially cryptographic solvers that are quite efficient but may, rarely, leak information due to solution vulnerabilities (Greenstadt, Grosz, & Smith 2007).

Conclusion

We introduce, model and solve private stable matching with non-transitive preferences. We define the framework of Distributed Private Constraint Optimization to model problems with complex privacy requirements. We show how it models the application of stable matching with secret non-transitive preferences. A generator for random stable matching problems is implemented.

Baseline algorithms are evaluated for problems in the new framework. All existing DCOP solvers apply to DPCOPs. Some cryptographic solvers provide the optimal solution, at the expense of efficiency. This set of benchmarks together with a random DPCOP generator are made available at `http://www.cs.fit.edu/~msilaghi/DPCOP`.

References

Brito, I., and Meseguer, P. 2005. Distributed stable matching problems. In *CP*.

Brito, I., and Meseguer, P. 2007. Distributed forward checking may lie for privacy. In *CP DCR Workshop*.

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

Golle, P. 2006. A private stable matching algorithm. In *FC*.

Greenstadt, R.; Grosz, B.; and Smith, M. D. 2007. SSDPOP: Improving the privacy of PDCOP with secret sharing.

Hamadi, Y. 2001. Interleaved backtracking in distributed constraint networks. In *ICTAI*, 33–41.

Hamilton, J.; Premkumar, M.; and Silaghi, M. 2005. Private stable marriages implementation. `http://www.cs.fit.edu/~msilaghi/SMC/examples/stable-marriages`.

Maheswaran, R. T.; Pearce, J. P.; Bowring, E.; Varakantham, P.; and Tambe, M. 2006. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*.

Mailler, R., and Lesser, V. 2004. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, 438–445.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AIJ* 161.

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In *IJCAI*.

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

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

Silaghi, M., and Yokoo, M. 2008. Distributed constraint reasoning. Encyclopedia of AI, Information Science Reference.

Silaghi, M.-C.; Faltings, B.; and Petcu, A. 2006. Secure combinatorial optimization using DFS-based variable elimination. In *Symposium on AI and Maths*.

Silaghi, M.-C.; Zanker, M.; and Bartak, R. 2004. Deskmates (stable matching) with privacy of preferences, and a new distributed CSP framework. In *Proc. of CP'2004 Immediate Applications of Constraint Programming Workshop*.

Silaghi, M.-C. 2004a. Incentive auctions and stable marriages problems solved with privacy of human preferences. Technical Report TR-FIT-11/2004, Florida Institute of Technology, Melbourne, FL.

Silaghi, M.-C. 2004b. Secure multi-party computation for selecting a solution according to a uniform distribution over all solutions of a general combinatorial problem. Cryptology ePrint Archive, Report 2004/333. `http://eprint.iacr.org/`.

Sinervo, B., and Livley, C. M. 1996. The rock-paper-scissors game and the evolution of alternative male strategies. Nature 340:240-243.

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

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE* 10(5):673–685.