Privacy for DisCSP-based Modeling in Multi-Agent Planning

Marius-Călin Silaghi*
Florida Institute of Technology (FIT)
Melbourne, Florida 32901-6988, USA
msilaghi@cs.fit.edu

Boi Faltings
Swiss Federal Institute of Technology (EPFL),
CH-1015 Lausanne, Switzerland
Boi.Faltings@epfl.ch

Abstract

Constraint Satisfaction and SAT can model planning problems (Kautz & Selman 1996) and this approach is quite successful. There is an increasing interest in distributed and asynchronous search algorithms for solving distributed constraint satisfaction problems (DisCSP). An important motivation for distributed problem solving is the agents' ability to keep their constraints private. Cryptographic techniques (Goldwasser & Bellare 1996) offer a certain protection from several types of attacks. However, when an attack succeeds, no agent can know how much privacy he has lost.

We assume that agents enforce their privacy by dropping out of the search process whenever the estimated value of the information that they need to reveal in the future exceeds that attached to a successful solution of the DisCSP. We compare several distributed search algorithms as to how likely they are to terminate prematurely for privacy reasons, and arrange the algorithms in a hierarchy that reflects this relation.

Introduction

Constraint Satisfaction and SAT can model planning problems (Kautz & Selman 1996) and this approach is quite successful. There is an increasing interest in distributed and asynchronous search algorithms for solving distributed constraint satisfaction problems (DisCSP). In a DisCSP, variables and constraints (predicates) belong to different agents. Agents exchange messages to find a single value assignment to all variables such that all their constraints are satisfied.

An important motivation for distributed problem solving is the agents' ability to keep their constraints private. One way of ensuring this is to anonymize constraints through cryptographic techniques. They are equivalent to replacing a trusted party by a set of agents that only together can act as trusted party, but alone are unable to cheat. Cryptographic techniques (Goldwasser & Bellare 1996) offer protection from several types of attacks. However, when an attack succeeds, no agent can know how much privacy he has lost. Another problem is that most existing cryptographic protocols add enormous complexity to asynchronous search, so they are often impractical (Hirt, Maurer, & Przydatek 2000).

*This work was performed while the first author was working at EPFL, supported by the Swiss National Science Foundation project number 21-52462.97.

We assume that each agent attaches a numerical value to information it is asked to reveal in a search process. For example, revealing whether the company is able to deliver a product by a certain date may have a very high value, while revealing whether the product can be blue may have a low value. A rational way for agents to ensure their privacy is then to abandon the search whenever the estimated value of the information that they must reveal in the future exceeds the value that they can gain from the successful solution.

Execution of asynchronous search algorithms is always nondeterministic because of the variable message delivery time. Therefore, often there is no strict comparison that guarantees that one algorithm is *always* better than another on a given problem instance. However, there are cases where for certain problem instances, one algorithm has the possibility of reaching a solution while the other will always terminate prematurely. We say that algorithm A is better[1] than algorithm B when there are cases when B always terminates prematurely but A may find a solution, and no cases where the opposite holds. In this paper, we show several such relations for known distributed constraint satisfaction algorithms and construct a hierarchy that reflects these relations.

Background

Cryptographic protocols are the most well known solutions for enforcing privacy in distributed computations (Goldwasser & Bellare 1996). Any given function can be *compiled* unto secure cryptographic protocols. When a certain function has to be computed on shared secrets, often, all its steps can be performed using corresponding secure protocols. This can be done in such a way that most attackers that cannot find the initial secrets, cannot find anything else than the *official output* of the protocol.

Secure multi-party protocols for DisCSPs

As said, any given function can be compiled unto secure cryptographic protocols. (Chaum, Crépeau, & Damgård 1988b; 1988a; Ben-Or, Goldwasser, & Widgerson 1988) show general compiling techniques for cases where less than n/3 players cheat and that are secure without intractability assumptions. (Goldreich, Micali, & Wigderson 1987) gives a technique for generating protocols robust to any minority

[1] Denoted $\prec$ or *looser* (not loser!) in the paper.

of colluders, and whose safety is based on intractability assumptions. That technique is complete in the sense that *"if a majority of players is not honest, some protocol problems have no efficient solution"* (Goldreich, Micali, & Wigderson 1987). Other limits of the privacy based on Shamir's schema (Shamir 1979) are explained in Section 3 of (Chor *et al.* 1985). At the end of a computation, no agent can know whether anything of its privacy is saved.

Recently, researchers have been trying to apply secure multi-party protocols to DisCSPs and related problems (e.g. (Yokoo & Suzuki 2002), and newer cases (Yokoo, Suzuki, & Hirayama 2002; Silaghi 2002)). (Silaghi 2002) describes some custom ways of applying secure multi-party protocols to discrete and numeric DisCSPs, based on intractability assumptions. One of its main ingredients is Merritt's election protocol (Merritt 1983) for shuffling, warping and reformulating the initial problem. Merritt's election protocol was initially meant for accounting the votes during elections and ensures the privacy of the relation vote-elector by shuffling the indexes of the votes. The shuffling is obtained by a chain of permutations (each permutation being the secret of an election center) on the encrypted votes.

In (Silaghi 2002), the Merritt's technique is slightly extended to allow both: shuffling variables and shuffling discrete values within discrete variables. The role of the election centers is taken by the participating agents. A chain of agents that compose their secret permutations is referred to as a *Merritt chain*.

Numeric constraints are broken and the pieces are shuffled separately through k distinct Merritt chains (Silaghi 2002).[2] Each element of a chain can therefore split, warp and transform the numeric constraints that they process.

The main feeble point of the technique described in (Silaghi 2002) for discrete DisCSPs is that agents that are able to get the shuffled DisCSP (e.g. by recording the trace of the algorithm), may be able to detect the homomorphism between some shuffled constraints and their own constraints. As a result, these agents can (partially) detect the obtained permutation and find the secrets of other agents. For numeric constraints, the permutation can be detected even easier by k colluding agents found on the same level on Merritt chains. Several techniques have been tried for increasing the intractability of detecting the permutations and the association constraint-agent.

Distributed search

While distributed resource allocation has been researched for several decades, the first asynchronous complete algorithm for DisCSPs appears in (Yokoo *et al.* 1992). The framework introduced in (Yokoo *et al.* 1992) approaches problems modeled with agents, variables and constraints, assuming that each variable has exactly one *owner*. The local problem of an agent A_i is denoted by CSP(A_i), and the involved variables by vars(A_i). The variables in vars(A_i) that are not owned by A_i are *foreign* to A_i. The domains can be represented as unary predicates. The local problem of an agent in a model with private domains is defined by:

- owned variables
- foreign variables
- predicates.

Such a local problem for an agent A_i can be denoted as:

$$\text{CSP}(A_i)=\{\text{owned variables}|\text{foreign variables}|\text{predicates}\}.$$

Issues on openness in incomplete search algorithms are discussed in (Modi *et al.* 2001; Denzinger 2001). Asynchronous backtracking (ABT), as presented in (Yokoo *et al.* 1992), is able to solve problems where proposals on variables can be made only by the owner of the variable. This peculiarity opens the door for the first approach to privacy: privacy of domains. Since only the owner, A, of a variable, x, can make proposals on the assignment of x, the domain of x can remain private to A. Parts of the domain of x may have to be revealed during search, but chances are that some values remain private:

- either by stopping when a solution is found, or
- by discovering that some proposals are already ruled out by constraints provided by other agents.

Slightly different frameworks have been mentioned in the literature (Zhang & Mackworth 1991; Khedro & Genesereth 1994; Solotorevsky, Gudes, & Meisels 1996), where an agent owns problems rather than variables. This removes the distinction between foreign and owned variables:

$$CSP(A_i) = \{variables|predicates\}.$$

In the following, we denote the k^{th} element of a tuple t by $t_{|k}$. It is interesting to mention the next proposition which actually already belongs to the folklore of DisCSPs:

Proposition 1 *Any distributed problem modeled with private variables can be modeled with private local problems. Any distributed problem modeled with private local problems can be modeled with private variables.*

Proof. This proposition is related to the equivalence between primal and dual CSP representations (Bacchus & Van Beek 1998).

Any DisCSP with private domains $P_1 = \{CSP(A_1)=\{ov_1|fv_1|p_1\},...\}$ is equivalent to the DisCSP with private problems $P_2 = \{CSP(A_1)=\{ov_1 \cup fv_1|p_1\},...\}$ where all public variables can take any value, but their owners know the unary constraints restricting them and that have the function of the corresponding private domains.

Any DisCSP with private problems $P_2 = \{CSP(A_1)=\{v_1|p_1\},...\}$ is equivalent to the DisCSP with private domains $P_1 = \{CSP(A_1)=\{x_1|v'_1|p'_1\},...\}$ defined as follows.

- x_k are new variables introduced in P_1 such that each agent A_i owns the variable x_i. The domain of x_i is the Cartesian product with domains of variables in v_i.
- Let $agents(v_i)$ be the set of agents enforcing in P_2 predicates involving variables in v_i. When $\{A\}$ is a set of agents, let *owned-vars*($\{A\}$) be the set of variables owned in P_1 by agents in $\{A\}$. Then:
  v'_i =*owned-vars*$(agents(v_i)) \setminus \{x_i\}$,

[2] This can be done after a first round of classic Merritt shuffling where pieces of numeric constraints are first dissociated from the initial variables. At the second shuffling (with plain constraints), the pieces of constraints can eventually be further split when these chains split themselves further forming trees.

- Let $idx(x, x_k)$ denote the index corresponding to valuations of the variable x in the tuples of x_k. p'_i consists of a set of predicates $\{x_{i|idx(x,x_i)}{=}x_{j|idx(x,x_j)}|x_j{\in}v'_i, x{\in}v_i\}$, and of a unary constraint restricting x_i to take values among tuples that are solutions for p_i.

□

Example 1 *Let us choose a problem modeled with private variables, consisting of two agents:* A_1 *and* A_2.
CSP(A_1*)=*$\{x, y||x, y{\in}\{0,1\}, x{+}y{=}1\}$*, and*
CSP(A_2*)=*$\{z, t|x, y|z, t{\in}\{0,1,2\}, z{+}t{=}2, x{+}z{-}y{-}t{=}2\}$.
The corresponding problem modeled with private local problems is:
CSP(A_1*)=*$\{x, y|x, y{\in}\{0,1\}, x{+}y{=}1\}$*, and*
CSP(A_2*)=*$\{z, tx, y|z, t{\in}\{0,1,2\}, z{+}t{=}2, x{+}z{-}y{-}t{=}2\}$.

Example 2 *Given a problem with three agents:*
CSP(A_1*)=*$\{x, y|x, y{\in}\{0,1,2\}, x{+}y{=}2\}$,
CSP(A_2*)=*$\{x, z|x, z{\in}\{0,1,2\}, x{+}z{=}2\}$*, and*
CSP(A_3*)=*$\{y, z|y{\in}\{0,1\}, z \in \mathbb{R}, y{+}z{=}2\}$.
An equivalent model with private variables is:
CSP(A_1*)=*$\{a||a{\in}\{(0,2),(1,1),(2,0)\}\}$,
CSP(A_2*)=*$\{b|a|b{\in}\{(0,2),(1,1),(2,0)\}, b_{|1}{=}a_{|1}\}$*, and*
CSP(A_3*)=*$\{c|a, b|c{\in}\{(0,2),(1,1)\}, c_{|1}{=}a_{|2}, c_{|2}{=}b_{|2}\}$.
While the initial version of asynchronous backtracking (ABT) requested that A_3 *owns at least one variable, we introduced the variable* c.

The difference between privacy on variables and privacy on problems is only algorithmic: With privacy on problems, several agents may cooperate for defining an assignment of a variable, allowing for abstractions which can have an impact on the amount of revelations (Silaghi, Sam-Haroud, & Faltings 2000a). With privacy on domains, the discussion about a variable can only be initiated by its owner. An estimation of privacy loss function of agent intelligence, described in (Freuder, Minca, & Wallace 2001), is based on the number of revealed tuples. The next sections introduce a framework for modeling agent behavior regarding privacy loss. It is then shown how features in protocols add new opportunities for privacy maintenance.

Distributed Private CSPs

Arguments referring to privacy have been often used in relation with algorithms for DisCSPs (Yokoo *et al.* 1998; Silaghi, Sam-Haroud, & Faltings 2000b), but the first quantitative experiments are detailed only in (Freuder, Minca, & Wallace 2001). It counts the number of revealed values during a synchronous protocol for meeting scheduling. That work was meant for assessing agent intelligence (by using or not using consistency in their local reasoning). Here we want to evaluate the capacity of distributed protocols to allow the search to proceed toward less private search space regions. In the following, when we speak about the feasibility of a tuple, we actually refer to the answer to the question: "Is the tuple feasible or not?"

Definition 1 *A Distributed Unitary Private CSP (DisUPrivCSP) is a DisCSP where each valuation of the local problem of an agent has associated a privacy value representing the loss suffered whenever the feasibility of that tuple is revealed:* $priv_{A_i} : D^{k_i} \rightarrow \mathbb{R}_+$.

Example 3 *A simple measure can be modeled in DisUPrivCSPs with* $priv_{A_i}(t) = f_1(feasible(t))$ *where* $f_1(false) = 0$ *and* $f_1(true) = 1$. *This is related to one of the measures used in* (Freuder, Minca, & Wallace 2001).

We need protocols that allow for selectively using less important tuples in solving the problems. In the following, we define a more complex version of DisUPrivCSP. Given a set S, usually one denotes with $\mathcal{P}(S)$ the set of subsets of S.

Definition 2 *A Distributed Private CSP (DisPrivCSP) is a DisCSP where:*

- *Each search subspace of the local problem of an agent has associated a privacy value representing the loss that appears whenever only the feasibility of that subspace is revealed:* $priv_{A_i}{:}\mathcal{P}(D^{k_i}){\rightarrow}\mathbb{R}_+$.
- *Each agent,* A_i*, prefers to minimize its privacy loss and abandons rather than accepting to incrementally lose a privacy with a value higher than* V_{A_i}*, the value that it can gain from a successful solution.*

The local problem of an agent in a model with private domains is defined by:

- owned variables,
- foreign variables,
- predicates, privacy function, and privacy threshold (expected gain).

For simplifying notations in descriptions of this paper, we use the following conventions. Constraint tuples for which no privacy value are specified are considered to have privacy value 0. When the privacy is specified for tuples formed only of owned variables, it will be considered that that privacy corresponds to the set of tuples whose projections on the owned variables is the given one. Lack of privacy threshold is considered to mean that its value is ∞. Such a local problem for an agent A_i can be denoted as:

CSP(A_i)={owned variables|foreign variables|predicates ...}.

Even if the privacy loss in general is not an additive function (as illustrated by the next example), for convenience, this is the default in the rest of the examples described in this work.

Example 4 *Let us analyze local problem CSP(*A_1*) with one variable* x *having a domain of three values* $\{a, b, c\}$ *and a publicly known unary constraint leaving exactly one value feasible. Revealing the feasibility of two values automatically reveals the feasibility of the third value. This implies that* $priv_{A_i}(\{a, b, c\}){=}priv_{A_i}(\{a, b\})$.
In this conditions, additivity,

$$priv_{A_i}(\{a, b, c\}){=}priv_{A_i}(\{a, b\}){+}priv_{A_i}(c)$$

can only be true when $priv_{A_i}(c) = 0$*, which cannot hold in general for each* c.
E.g. When:
$priv_{A_i}(a){=}priv_{A_i}(b){=}priv_{A_i}(c){=}1$*, and*
$priv_{A_i}(\{a, b, c\}){=}3$*, then*
$priv_{A_i}(\{a, b\}){=}priv_{A_i}(\{a, c\}){=}priv_{A_i}(\{b, c\}){=}3$.
Here $priv_{A_i}(\{a, b\}) \neq priv_{A_i}(a) + priv_{A_i}(b)$.

The behaviors in DisPrivCSPs are related to behaviors in auctions where all bidders pay their last bid. Namely, even if the privacy loss undergone so far by A_i is higher than V_{A_i}, it is rational for A_i to continue to cooperate in search whenever it believes it can find a solution without increasing its privacy loss with V_{A_i}.

The simplifying assumptions made here are:

- environment invariant privacy: the value of the privacy loss is invariant with completely external events.
- peer invariant privacy: the value of the privacy loss is invariant with the set of agents learning the divulged information (everybody discloses everything known about third parties).[3]

Definition 3 (looser[4]) *A problem solving algorithm P_1 is strictly looser than another algorithm P_2, denoted $P_1 \prec P_2$, if there exists a DisPrivCSP that can be solved by some instance of P_1 but cannot be solved by any instance of P_2 for the same initial order on values and agents. It is also required that any problem solvable with P_2 can be solved with some instance of P_1.*

Proposition 2 *The relation $\prec$ is transitive.*

Proof. If $P_1 \prec P_2 \prec P_3$, it means that $\exists$ a problem solvable with P_1 but not solvable with P_2, therefore not solvable with P_3. Also, all problems solvable by P_3 are solvable with P_2 and therefore solvable with P_1. □

Note that only the initial order is considered for comparing algorithms with reordering. For DisCSPs, (Yokoo 1995) and (Meisels & Razgon 2001) present algorithms with value reordering. Several protocols for solving DisCSPs are compared for DisPrivCSPs in next section.

Comparison of Protocols

The simplest complete distributed algorithm that we can imagine now is the Synchronous Backtracking (SyncBT) (Yokoo *et al.* 1992). SyncBT is the distributed counterpart of centralized backtracking.

(Collin, Dechter, & Katz 2000) presents an algorithm we refer to as Distributed Depth-First-Search Branches (DisDFSB). DisDFSB improves on SyncBT by allowing statically detected independent subproblems on branches of the search to be scanned in parallel. When one of these subproblems generates a failure, the whole search branch can be abandoned.

Proposition 3 *DisDFSB $\prec$ SyncBT.*

Proof. Let P be a DisPrivCSP with three agents and privacy functions on unary constraints:
CSP(A_1)=$\{a||a\in\{0..3\}, a\not\in\{2\}, a{=}b{=}c, priv_{A_1}(2){=}4, V_{A_1}{=}3\}$
CSP(A_2)=$\{b|a|b\in\{0..3\}, b{>}1, a{=}b, priv_{A_2}(1){=}1,$
$priv_{A_2}(0){=}4, V_{A_2}{=}3\}$
CSP(A_3)=$\{c|a|c\in\{0..3\}, c{>}1, a{=}c, priv_{A_3}(1){=}4,$
$priv_{A_3}(0){=}1, V_{A_3}{=}3\}$.

[3]This assumption is the inverse of the one behind most cryptographic protocols.

[4]For the non-English readers, please note that looser is not the same as loser.

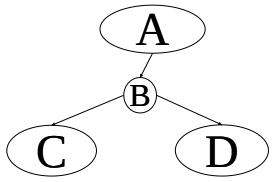

Figure 1: DisDFSB DFS branches: A,C,D are sets of agents. B is an agent.

For simplification, here as overall in the paper, $priv_{A_2}(1)$ is a short notation for $priv_{A_2}(b = 1)$, where the missing variable in the unary constraint is the owned one. As previously mentioned, $priv_{A_2}(b = 1)$ means $priv_{A_2}(b = 1, \forall a)$, where the added universal quantifier precedes all the foreign variables.

With DisDFSB, A_1 starts proposing $a = 0$ then A_2 waits for a while, hoping that A_3 can detect infeasibility, since A_2 prefers to abandon the search rather than admitting that it does not accept $b = 0$. Infeasibility is fortunately found by A_3 and A_1 proposes $a = 1$. For $a = 1$ it is A_2 that accepts to reveal infeasibility. A_1 avoids mentioning about $a = 2$ and a solution is found for $a = 3$.

Given this order on values, SyncBT cannot find any solution for this order on agents, and neither SyncBT nor DisDFSB can find any solution with other DFS orders on agents.

Obviously, whenever SyncBT finds a solution for some order on agents, DisDFSB will also find a solution for a compatible DFS order on agents. □

The Synchronous (arc-) consistency maintenance protocol (SyncMC) mentioned in (Collin, Dechter, & Katz 2000), and detailed in (Tel 1999) consists of a sequence of proposals/backtracking interleaved with consistency maintenance. In addition, for DisPrivCSPs, the strength of the achieved consistency can be optional and agents can be allowed to divulge only the labels/value eliminations that they want. With this amendment:

Proposition 4 *SyncMC $\prec$ SyncBT.*

Proof. For the problem discussed in the proof of Proposition 3, SyncMC finds a solution in the same conditions as DisDFSB. Whenever SyncBT finds a solution, an instance of SyncMC where agents prefer not to reveal any label behaves like SyncBT and finds the same solution. □

Proposition 5 *SyncMC and DisDFSB are incomparable with the relation $\prec$.*

Proof. Any DisCSP can be extended with constraints between all variables, such that DisDFSB behaves like SyncBT. According to Proposition 4, SyncMC can avoid inconsistency and finds solutions when DisDFSB must abort.

One can also construct problems where DisDFSB finds solutions and SyncMC aborts. Here we do not present a complete example since it would be lengthy and redundant, but explain how it can be built:

In Figure 1, both sets of agents: C and D, defining DFS branches under the agent B are infeasible for two different search tree branches b_1 and b_2, but the privacy loss leads to abortion once for C on b_1 and then for D on b_2.

When the full search on D can reach failure on b_1 but the maintained degree of consistency of SyncMC fails to find inconsistency (such situations are known to exist), then SyncMC forces C to find

the inconsistency. If the order on agents is lucky and the failing subset of D comes first, then SyncMC aborts when it expands the search tree on b_2. □

Asynchronous Protocols

During distributed search, agents exchange information on current proposals via messages, and the set of proposals known by an agent is referred to as its *view*. Protocols are asynchronous when at no moment an agent can make assumptions about the current view of another agent (except A_1 with static order).

Most distributed algorithms consist of several asynchronous *epochs*, meaning that there exist some uncertainty when messages are on their ways, but at some moments one agent can make safe assumptions about the view of the other agents. These moments delimit epochs. In SyncMC, each start and termination of a distributed consistency round delimits consistency epochs from backtracking epochs. In (Meisels & Razgon 2001), the search consists of a series of: consistency, ordering, backtracking epochs. (Armstrong & Durfee 1997) describes an algorithm consisting of asynchronous backtracking epochs interleaved with reordering epochs. In SyncBT, each epoch consists of exactly one message.

Definition 4 *We say that a protocol is asynchronous when it consists of a single asynchronous epoch.*

Asynchronous Backtracking (ABT) (Yokoo *et al.* 1992) is an asynchronous protocol for solving DisCSPs. In ABT, agents can make concurrently and in parallel proposals on distinct variables.

Proposition 6 *ABT $\prec$ SyncDFSB.*

Proof. We take a DisPrivCSP with agents A_1, A_2, A_3, A_4, A_5 enforcing a CSP with a fully connected graph. Given this order on agents we consider that the agents A_3, A_4, A_5 compose an infeasible problem for the first proposal of the first agent. Let A_2 also want to refuse this proposal but the privacy loss for this refusal would oblige him to abandon the search. ABT is able to detect the infeasibility of this value and solutions with subsequent values can be found. SyncDFSB instead obliges A_2 to divulge infeasibility of this same proposal and the search is aborted.

Whenever SyncDFSB solves a problem, the same problem can be solved by ABT, since a certain timing in ABT yields SyncDFSB. □

(Silaghi, Sam-Haroud, & Faltings 2000a) proposes an asynchronous algorithm, Asynchronous Aggregation Search (AAS), for a model with privacy on constraints. We discuss here the version of AAS described in (Silaghi 2002). AAS allows agents to aggregate several proposals in one message. While the order on values is difficult to define when aggregation is used (AAS), we assume in this report that each new proposal contains the first available tuple in lexicographic order.

Proposition 7 *AAS $\prec$ ABT.*

Proof. AAS can emulate ABT by not performing aggregations, therefore when ABT finds a solution, AAS can also find it. Instead, by the fact that AAS can aggregate two proposals a and b, lower priority agents may be able to find a solution with b, avoiding to abort on a. □

Proposition 8 *ABT is incomparable versus SyncMC using the relation $\prec$.*

Proof. Arc consistency achievement can remove a branch of the search defined by a proposal of the first agent, by propagating the removal of the last proposal of the second agent. ABT cannot do this, especially since nogoods involving that last value can only be computed after that value is proposed. Since the propagation of such a value can rule out proposals leading to abortion, SyncMC may find solutions when ABT does not.

SyncMC may not itself find solutions when search with ABT in lower priority agents can detect infeasibility of challenging proposals.

An example where the previous argument can be analyzed can be easily built. Consider the next DisPrivCSP:
CSP(A_1)=$\{x_1 || x_1 \in \{1,2\}, |x_2 - x_1| \leq 1\}$
CSP(A_2)=$\{x_2 | x_1 | x_2 \in \{1,2,3\}, |x_2 - x_1| \leq 1, x_2 > 1,$
$priv_{A_2}(1)=4, V_{A_2}=3\}$

When CSP(A_3), CSP(A_4), CSP(A_5),... define a problem such that $x_1=1$ leads to failure by search but $x_1=1, x_2 \in \{1,2\}$ does not fail with consistency, then a solution for $x_1=2$ can be found with ABT but not with SyncMC.

Instead, when CSP(A_3), CSP(A_4), CSP(A_5),... define a problem such that $x_1=1$ does not lead to failure by search (due to the existence of a solution for $x_1=1, x_2=3$) but consistency with $x_1=1, x_2 \in \{1,2\}$ can prove infeasibility, then a solution for $x_1=2$ can be found with SyncMC but not with ABT.

To avoid lengthy redundant description, we do not give fully described cases. However, these two types of problems obviously exist, therefore the claim is proven. □

A method called Distributed Maintaining Asynchronously Consistency (DMAC-ABT), for maintaining consistency in ABT, is introduced in (Silaghi, Sam-Haroud, & Faltings 2001b). DMAC-ABT allows to exchange a type of nogoods called consistency-nogoods, corresponding to labels in SyncMC.

Corollary 8.1 *DMAC-ABT $\prec$ ABT.*

Proof. Any of the problems that can be solved with ABT can be solved with DMAC-ABT for DisPrivCSPs, since DMAC-ABT emulates ABT when no agent sends labels due to privacy enforcement policies. According to Proposition 8, there exist problems that can be solved with SyncMC but cannot be solved with ABT. Those problems can be solved with DMAC-ABT, which can also emulate SyncMC. □

Corollary 8.2 *DMAC-ABT $\prec$ SyncMC.*

Proof. Any of the problems that can be solved with SyncMC can be solved with DMAC-ABT for DisPrivCSPs, since DMAC-ABT emulates SyncMC for certain timing policies. The problems that can be solved with ABT but cannot be solved with SyncMC (see Proposition 8), can be solved with DMAC-ABT, which can also emulate ABT when no agent sends labels due to privacy enforcement policies. □

The following two statements may be proved similarly to the previous propositions.

Proposition 9 *AAS is incomparable versus SyncMC using the relation $\prec$.*

Proof outline. Note that in contrast to the proof for Proposition 8, for proving the previous statement one has to design examples that work for all possible aggregations containing the first values that can be proposed by A_1. For simplicity, the designed examples can let A_1 manage at least two variables where the set of feasible tuples for A_1 cannot be represented by a single Cartesian product (e.g. an equality constraint for two variables with identical domains with two values each.

One can argue that such an example is not convincing since it can be reformulated by clustering the variables of the first agent and aggregating the resulting feasible values. To stand against this criticism, the designed example has to deal with branches defined by the proposals of A_2 which cannot all be aggregated with the proposal of A_1. A_3 is therefore the agent risking to abort and the agents $A_4, ...$ respectively $A_3, ...$ are the agents that can/cannot refuse the proposal of A_2 with AAS respectively SyncMC. □

(Silaghi, Sam-Haroud, & Faltings 2001b) mentions a protocol called DMAC, combining AAS and DMAC-ABT. Similarly with the previous propositions it can be shown that based on the last proposition:

Corollary 9.1 *DMAC is looser than both AAS and SyncMC.*

A recent general protocol is the replica-based DMAC (R-DMAC)[5], a.k.a. Asynchronous Dichotomous Maintaining Bound-consistency (ADMB) (Silaghi *et al.* 2001). While none of the previously mentioned protocols can be $\prec$ than R-DMAC that is a generalization, the superiority of R-DMAC remains to be proven.

Comparison for reordering

Now we discuss several algorithms allowing for reordering. ABT with reordering (ABTR) is a protocol described in (Silaghi, Sam-Haroud, & Faltings 2001a) and allowing agents to propose new orders based on heuristic information obtained from other agents.

Proposition 10 *ABTR $\prec$ ABT.*

Proof. In ABTR, an agent owning private tuples that must be revealed for proving infeasibility of a branch may be allowed to generate a heuristic message and can be placed on the first position. This way it can launch another proposal and escapes the need to abort. ABTR can always emulate ABT when agents do not use heuristics. □

Asynchronous Weak Commitment (AWC) (Yokoo 1995) is an algorithm for reordering values and agents during asynchronous search. Its ability to reorder values allows it to solve problems that algorithms with static order cannot solve. In AWC, agents can propose new reordering whenever a new nogood is multicast. The standard reordering policy in AWC is to position each agent discovering a new nogood before all the agents that have generated proposals found in that nogood, but in practice, other positions can be selected.

Proposition 11 *AWC $\prec$ ABT.*

Proof. Since, as mentioned above, agents can choose not to modify their priority on the discovery of new nogoods, at extreme AWC emulates ABT, and any problem solved with ABT can be solved with some variant of AWC. However, due to its reordering ability, AWC can solve the next DisPrivCSP that is abandoned in ABT.
CSP(A_1)=$\{a|b|a\in\{1,2,3\}, a=b\}$
CSP(A_2)=$\{b|a|b\in\{1,2,3\}, a=b, a\notin\{1,2\}, priv_{A_2}(2)=4, V=3\}$

When A_2 receives the first proposal for $a = 1$, it generates a nogood and gets higher priority. It can then propose $b = 3$ and a solution is found. With ABT, A_1 proposes first $a = 1$, then on refusal it proposes $a = 2$ and A_2 must abort. □

However, in case of a series of unlucky reordering events, ABTR and AWC may have to abandon a DisPrivCSP that can be solved with ABT.

Example 5 *For the DisPrivCSP:*
CSP(A_1)=$\{a|b|a\in\{1,2,3\}, a=b, a\neq 2, priv_{A_1}(2)=4, V=3\}$
CSP(A_2)=$\{b|a|b\in\{1,2,3\}, a=b, a\neq 1\}$

AWC with the reordering strategy advised for efficiency on DisCSPs in (Yokoo 1993), A_1 aborts when after the first nogood detected by A_2, A_2 gets the highest priority and proposes $b = 2$.

Proposition 12 *AWC and ABTR cannot be compared with the relation $\prec$.*

Proof. AWC cannot change an order before nogoods are discovered. Given the DisPrivCSP:
CSP(A_1)=$\{a|b|a\in\{1,2\}, a=b\}$
CSP(A_2)=$\{b|a|b\in\{1,2\}, b\neq 1, a=b, priv_{A_2}(1)=4, V=3\}$,
AWC leads A_2 to abandon the search immediately after receiving the proposal $a = 1$, since A_2 don't accept to divulge the infeasibility of $b = 1$. ABTR instead allows A_2 to generate a heuristic message and a reordering can be issued where A_2 gets the highest priority. A_2 can propose $b = 2$ and the solution is found immediately.

AWC can reorder values and this can allow for solving DisPrivCSPs that are abandoned with ABTR, such as:
CSP(A_1)=$\{a|b|a \in \{0,1,2,3\}, a=b, a\neq 2, priv_{A_1}(2)=4, V=3\}$
CSP(A_2)=$\{b|a|b\in\{1,2,3\}, a=b, a\neq 1, |a-b|\neq 2, priv_{A_2}(1)=4, V=3\}$

□

In the previous proof we did consider that agents in ABTR stubbornly stick to the initial order on their values. Clearly, the ABTR protocol can easily be extended to allow the agents to use strategies where they can choose values in a wiser way. It remains an open question whether such strategies can solve all the problems that can be solved with AWC.

The protocol combining DMAC with the reordering of ABTR is called Multiply Asynchronous Search (MAS). It can be noted that no algorithm has been yet described that would combine DMAC-ABT with the reordering of ABTR (DMAC-ABTR).

Proposition 13 *MAS $\prec$ DMAC.*

Proof. This relation is similar to the one at Proposition 10. □

It is obvious that ABTR cannot be looser than MAS since MAS can emulate ABTR. However, we have not yet succeeded in designing DisPrivCSP examples that can be solved by MAS and cannot be solved by ABTR, such that the relationship between these two algorithms remains an open question.

[5] R-MAS without reordering.

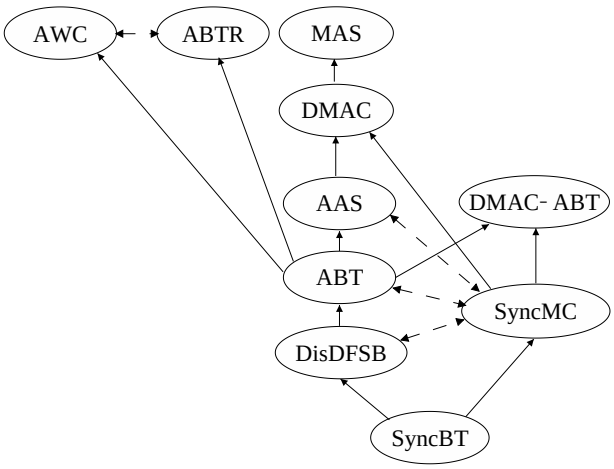

Figure 2: Dashed lines with arrows at both sides show protocols proven to be incomparable with the relation looser. For lines with one arrow, the arrows point towards looser protocols. The relation $\prec$ is transitive.

Summary of Distributed Search Protocols

In Figure 2 we give a global image of the aforementioned results in the comparison of protocols for DisPrivCSPs. Several very recent protocols (e.g. (Bessière, Maestre, & Meseguer 2001; Meisels & Razgon 2001)) are not yet analyzed. Some protocols (e.g. (Solotorevsky, Gudes, & Meisels 1996)) initially designed for problems without privacy could also be studied in the future from the point of view of DisPrivCSPs.

The relation $\prec$ is transitive (Proposition 2) and therefore, several additional relations can be inferred from the proven ones. Some questions remain open, e.g. the relation between ABTR, AWC, and AAS or DMAC-ABT.

Secure protocols vs. distributed search

In the previous sections we have seen several search protocols for solving distributed CSPs. One of the main motivations behind DisCSPs is the privacy that they offer. In the Background section we have first introduced secure multiparty computations, the main competitor for distributed search.

Before closing this report, we will discuss some of the trade-offs between the two techniques. Clearly, each of them is appropriate for a different type of problems, while for some problem the choice can be difficult.

Secure protocols have the main advantage that *it is probable* that they lead the computation without any privacy loss, other than the official result. In contrast, distributed protocols by their nature divulge additional constraints and values (except when the first checked tuple is a solution).

Secure protocols lead to deterministic results that depend only on the initial description of the problem and strategies (Yao 1982), and of the way the distribution/shuffling is done. The result in distributed search is influenced by agent strategies, network load, divulged information.

Distributed search has the advantage that in any moment, an agent knows and enforces an upper bound of how much information it may have divulged. In typical secure protocols, there is no way for an agent to know that any privacy was actually saved.

Distributed search fits well human-machine interaction, while secure protocols are cryptic and are only appropriate for purely machine-based problem solving.

Distributed search seems able to accommodated easily to very dynamic problems like the one in (Jung *et al.* 2000).

Overall, we think that distributed search can be preferred when the participants are not trusted enough for the available cryptographic protocols to be secure (e.g. for numeric constraints or techniques from (Chaum, Crépeau, & Damgård 1988a; Ben-Or, Goldwasser, & Widgerson 1988)), or when the agents strongly need to make sure that the upper bound of the divulged information is lower than a threshold. They are also preferred when the size of the problems is sufficiently small such that 'secure protocols' based on intractability assumptions are too insecure. Secure protocols are preferred in most other cases, especially when it is very important to find a solution and the global problem is suspected to be very difficult such that almost everything would be revealed with distributed search.

Conclusions

While many distributed protocols have been developed recently for coping with privacy, no existing formal framework allows to go beyond questionable claims. In this article we propose a framework called Distributed Private Constraint Satisfaction Problems (DisPrivCSPs), related with the work in (Freuder, Minca, & Wallace 2001). This framework models the privacy loss for individual revelations. It also models the effect of the privacy loss by assuming that agents are determined to abandon when the incremental privacy loss overcomes the expected gains from cooperation. Applications of such protocols have been recently described in (Freuder, Minca, & Wallace 2001). The newly introduced framework is supported by showing how several existing protocols behave and compare against each other within the new definitions. Several relations remain to be discovered. Observations made during this analysis can direct users in choosing good strategies for their agents (Silaghi 2002).

References

Armstrong, A., and Durfee, E. F. 1997. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of 15th IJCAI*.

Bacchus, F., and Van Beek, P. 1998. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence*.

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

Bessière, C.; Maestre, A.; and Meseguer, P. 2001. Distributed dynamic backtracking. In *Proc. IJCAI DCR Workshop*, 9–16.

Chaum, D.; Crépeau, C.; and Damgård, I. 1988a. Multiparty unconditionally secure protocols. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, 11–19.

Chaum, D.; Crépeau, C.; and Damgård, I. 1988b. Multiparty unconditionally secure protocols. In Springer-Verlag., ed., *Proc. CRYPTO 87, LNCS 293*, 462.

Chor, B.; Goldwasser, S.; Micali, S.; and Awerbuch, B. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, 383–395.

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

Denzinger, J. 2001. Tutorial on distributed knowledge based search. IJCAI-01.

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

Goldreich, O.; Micali, S.; and Wigderson, A. 1987. How to play any mental game - a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, 218–229.

Goldwasser, S., and Bellare, M. 1996. Lecture notes on cryptography. MIT.

Hirt, M.; Maurer, U.; and Przydatek, B. 2000. Efficient secure multi-party computation. In *Advances in Cryptology - ASIACRYPT'00*, volume 1976 of *LNCS*, 143–161.

Jung, H.; Tambe, M.; Zhang, W.; and Shen, W.-M. 2000. On modeling argumentation as distributed constraint satisfaction: Initial results. In *Proceedings of the International Workshop on Distributed Constraint Satisfaction*, 47–56. CP'00.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI*, 1194–1201.

Khedro, T., and Genesereth, M. R. 1994. Modeling multiagent cooperation as distributed constraint satisfaction problem solving. In *Proceedings of ECAI'94*, 249–253.

Meisels, A., and Razgon, I. 2001. Distributed forward checking with dynamic ordering. In *CP01 COSOLV Workshop*, 21–27.

Merritt, M. 1983. *Cryptographic Protocols*. Ph.D. Dissertation, Georgia Institute of Technology.

Modi, P. J.; Jung, H.; Tambe, M.; Shen, W.-M.; and Kulkarni, S. 2001. Dynamic distributed resource allocation: A distributed constraint satisfaction approach. In *Distributed Constraint Reasoning, Proc. of the IJCAI'01 Workshop*, 73–79. Seattle: IJCAI.

Shamir, A. 1979. How to share a secret. *Communications of the ACM* 22:612–613.

Silaghi, M.-C.; Sabău, Ş.; Sam-Haroud, D.; and Faltings, B. 2001. Asynchronous search for numeric DisCSPs. In *Proc. of CP'2001*, 786.

Silaghi, M.-C.; Sam-Haroud, D.; and Faltings, B. 2000a. Asynchronous search with aggregations. In *Proc. of AAAI2000*, 917–922.

Silaghi, M.-C.; Sam-Haroud, D.; and Faltings, B. 2000b. Asynchronous search with private constraints. In *Proc. of AA2000*, 177–178.

Silaghi, M.-C.; Sam-Haroud, D.; and Faltings, B. 2001a. ABT with asynchronous reordering. In 2^{nd} *A-P Conf. on Intelligent Agent Technology*, 54–63.

Silaghi, M.-C.; Sam-Haroud, D.; and Faltings, B. 2001b. Consistency maintenance for ABT. In *Proc. of CP'2001*, 271–285.

Silaghi, M.-C. 2002. *Polynomial-Space Asynchronous Search*. Ph.D. Dissertation, Swiss Federal Institute of Technology (EPFL), CH-1015 Ecublens. submitted.

Solotorevsky, G.; Gudes, E.; and Meisels, A. 1996. Algorithms for solving distributed constraint satisfaction problems (DCSPs). In *Proceedings of AIPS96*.

Tel, G. 1999. *Multiagent Systems, A Modern Approach to Distributed AI*. MIT Press. chapter Distributed Control Algorithms for AI, 539–580.

Yao, A. 1982. Protocols for secure computations. In *Proceedings of 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, 160–164.

Yokoo, M., and Suzuki, K. 2002. Secure multi-agent dynamic programming based on homomorphic encryption and its application to combinatorial auctions. In *Proc. of AAMAS-02*. to appear.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *ICDCS*, 614–621.

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

Yokoo, M.; Suzuki, K.; and Hirayama, K. 2002. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *Proc. of the AAMAS-02 DCR Workshop*. submitted.

Yokoo, M. 1993. Dynamic value and variable ordering heuristics for distributed constraint satisfaction. In *Workshop on Intelligent Agents'93*.

Yokoo, M. 1995. Asynchronous weak-commitment search for solving large-scale distributed constraint satisfaction problems. In *1st ICMAS*, 467–318.

Zhang, Y., and Mackworth, A. K. 1991. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proc. of Third IEEE Symposium on Parallel and Distributed Processing*, 394–397.