Improving Privacy in Distributed Constraint Satisfaction Problems

Salomeh Taherifard, Seyed Mahdi Jameii, Zahra Shojaeerad

1, 3 Sama Technical and Vocational Training College Tehran Branch (Andisheh), Islamic Azad University, Tehran, Iran
2 Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran

1 salomeh.taherifard@yahoo.com, 2 jamei@qodsiu.ac.ir, 3 zahra.shj@gmail.com

ABSTRACT

A distributed constraint satisfaction problem (DisCSP) is a constraint satisfaction problem in which variables and constraints are distributed among multiple agents. One of the main issues in DisCSPs is privacy: agents may not want to share their values or their constraint or their domain, and they may wish to keep constraints as private as possible. In order to improve constraint privacy in the DisCSPs, we use partially known constraints (PKC) in which constraints are kept private and are only partially known to agents. A new version of nogood-based Asynchronous Forward-Checking (AFC-ng) algorithm that works with PKC, 2phase AFC-ng algorithm (AFC-ng-2ph), a hybrid private algorithm to DisCSPs, is presented in this paper. The experimental results on randomly generated DisCSPs show that the AFC-ng-2ph is more efficient than some private algorithms like two-phase asynchronous backtracking algorithm (ABT-2ph) and two-phase distributed Forward-Checking (DisFC-2ph).

Keywords: Distributed artificial intelligence, distributed constraint satisfaction problems, asynchronous forward checking, privacy

1. INTRODUCTION

A Constraint satisfaction problem (CSP) is a problem to find a consistent assignment of values to variables that satisfy the constraints of the problem. CSP can formalize many applications such as picture processing [1], planning [2], job-shop scheduling [3], etc.

Many backtrack search algorithms have been designed for solving constraint satisfaction problems, like Forward Checking (FC) [4] and Maintaining Arc Consistency (MAC) [5]. A distributed CSP is a CSP in which variables and constraints are distributed among multiple automated agents [11, 12], each one holding its local constraint network, which are connected by constraints among variables of different agents. To solve a DisCSPs, Agents must assign values to variables, so that all constraints between agents are satisfied. To achieve this goal, agents exchange messages with other agents, to check consistency of their proposed assignments against constraints with variables owned by different agents [12, 13]. Examples of distributed problems are Sensor networks [6, 7] and distributed resource allocation problems [8, 9]. Several distributed algorithms on DisCSPs have been proposed in the last two decades.

Such as Asynchronous Backtracking (ABT) [15, 16, 18], Synchronous Backtrack (SBT) [19], Asynchronous Forward Checking (AFC) [21], Asynchronous Forward Checking (AFC-ng) [22].

Privacy is one of the main goals in distributed problems. Agents should not share their values, and constraints among agents should be kept as private as possible. So far, two main approaches developed to enforcing privacy. One uses the encryption techniques to conceal values and constraints [23, 24, 14]. The overhead of these protocols in communication and computation is very large. Another enforces privacy without using encryption methods. For example Distributed Forward Checking (DisFC) [25], DisFC-2ph and ABT-2ph are some private algorithms. DisFC is an algorithm for preserving privacy of assignments [25]. DisFC-2ph and ABT-2ph are two phase strategy that improve constraint privacy in solving the DisCSPs under the Partially Known Constraints (PKC) model [25, 11].

In this work, we propose a new solving approach that considers the constraints privacy. Regarding constraint Privacy, AFC-ng assumes that a constraint Cij (agent i→agent j in ordering) is totally known by agent I and j. Therefore, AFC-ng has no constraints or assignments privacy. We use the Partially Known Constraints (PKC) model [25], and present a 2phase nogood-based Asynchronous Forward-Checking algorithm (AFC-ng-2ph), based on AFC-ng. This paper is organized as follows. Section 2 gives the necessary background. Sections 2 describe the AFC-ng-2ph algorithm. Theoretical analysis and correctness proofs are given in Section 4. Section 5 presents an experimental evaluation of our proposed algorithms against other algorithms. Finally, we will conclude the paper in Section 6.

2. BACKGROUND

2.1 Basic Definitions and Notations

A distributed constraint satisfaction problem (DisCSP) consists of a set of agents \( \{A_1, \ldots, A_n\} \), a set of \( n \) variables \( \{x_1, \ldots, x_n\} \), where each variable \( x_i \) belongs to one agent \( (a = n) \) and each agent taking a value from finite and discrete domains \( D(x_1), D(x_2), \ldots, D(x_n) \), respectively and a set of constraints that specify the combinations of values allowed for the variables they involve. In this paper, we all constraints are binary constraints and they involve two variables. A constraint \( C_{ij} \) between two variables \( x_i \) and \( x_j \) is a subset of the Cartesian product of their domains, i.e., \( C_{ij} \subseteq D(x_i) \times D(x_j) \).

In this work we assume that agents store a unique total order \( \prec \) on agents as it done in AFC-ng [22]. Thus, agents appearing before an agent \( A_i \) in the total order are...
the higher priority agents (predecessors) and conversely the lower priority agents (successors) are those appearing after \( A_i \). For sake of clarity, we assume that the total order on agents \( < \) is the lexicographic ordering \( \{A_1, A_2, \ldots, A_n\} \).

In the following we mention Definition 1 to Definition 5 of AFC-ng in [22].

- **Definition 1:** An assignment for an agent \( A_i \in A \) is a tuple \((x_i, v_i, t_i)\), where \( v_i \) is a value from the domain of \( x_i \) and \( t_i \) is the tag value. When comparing two assignments, the most up to date is the one with the greatest tag \( t_i \).

- **Definition 2:** A Current Partial Assignment CPA is an ordered set of assignments \( \{(x_1, v_1, t_1), \ldots, (x_i, v_i, t_i)\} \). Based on the constraints of the problem, agents can infer inconsistent sets of assignments called nogoods.

- **Definition 3:** A timestamp, associated with a CPA, is an ordered list of counters \([t_1, t_2, \ldots, t_i]\) where \( t_j \) is the tag of the variable \( x_j \). When comparing two CPAs, the strongest one is that associated with the lexicographically greater timestamp. That is, the CPA with greatest value on the first counter on which they differ, if any, otherwise the longest one.

- **Definition 4:** The AgentView of an agent \( A_i \in A \) stores the most up to date assignments received from higher priority agents in the agent ordering. It has a form similar to a CPA and is initialized to the set of empty assignments \( \{(x_i, \text{empty}, 0) \mid x_j < x_i\} \). Based on the constraints of the problem, agents can infer inconsistent sets of assignments called nogoods.

- **Definition 5:** A nogood ruling out value \( v_i \) from the initial domain of variable \( x_i \) is a clause of the form \( x_i = v_i \land \ldots \land x_j = v_j \rightarrow x_k \neq v_k \), meaning that the assignment \( x_i = v_i \) is inconsistent with the assignments \( x_i = v_i, \ldots, x_j = v_j \). The left hand side (lhs) and the right hand side (rhs) are defined from the position of \( \rightarrow \).

The current domain \( D(x_i) \) of a variable \( x_i \) contains all values from the initial domain \( D^0(x_i) \) that are not ruled out by a nogood. When all values of a variable \( x_i \) are ruled out by some nogoods (i.e., \( D(x_i) = \emptyset \)), these nogoods are resolved, producing a new nogood (ng). Let \( x_i \) be the lowest variable in the left-hand side of all these nogoods and \( x_j = v_j \). lhs(ng) is the conjunction of the left-hand sides of all nogoods except \( x_j = v_j \) and rhs(ng) is \( x_j \neq v_j \).

### 2.2 Nogood-based Asynchronous Forward-Checking

Nogood-based Asynchronous Forward-Checking (AFC-ng) is based on Asynchronous Forward-Checking (AFC) algorithm. Therefore, agents perform the forward checking phase asynchronously [26, 21]. Agents assign their variables only when they hold the current partial assignment (CPA) synchronously. The CPA is a unique message (token) that is passed from one agent to the next one in the ordering. The CPA message carries the partial assignment (CPA) that agents attempt to extend into a complete solution by assigning their variables on it. When an agent succeeds in assigning its variable on the CPA, it sends this CPA to its successor.

Each time an agent receives a CPA, performs a forward-check, it revises its initial domain, (including values already removed by a stored nogood) in order to store the best nogoods for removed values (one nogood per value). When comparing two nogoods eliminating the same value, the nogood with the highest possible lowest variable involved is selected (HPLV heuristic) [10]. As a result, when an empty domain is found, the resolved nogood contains variables as high as possible in the ordering, so that backtrack message is sent as high as possible, thus saving unnecessary search effort [18].

Each time an agent \( A_i \) generates an empty domain as a result of a forward-checking, it resolves the nogoods ruling out values from its domain, producing a new nogood. ng is the conjunction of lhs of all nogoods stored by \( A_i \). Then, \( A_i \) sends the resolved nogood ng in a ngd (backtrack) message to the lowest agent in ng.

Hence, multiple backtracks may be performed at the same time coming from different agents having an empty domain. These backtracks are sent concurrently by these different agents to different destinations. The reassignment of the destination agents then happen simultaneously and generate several CPAs. However, the strongest CPA coming from the highest level in the agent ordering will eventually dominate all others. Agents use the timestamp [22] to detect the strongest CPA.

Interestingly, the search process of higher levels with stronger CPAs can use nogoods reported by the (killed) lower level processes, so that it benefits from their computational effort.

When an agent receives a terminate message, it marks end flag true to stop the main loop. If the attached CPA is empty then there is no solution. Otherwise, the solution of the problem is retrieved from the CPA.

Therefore, AFC-ng uses 3 kind of message CPA, ngd and termination to solve a DisCSPs. The pseudo code of AFC-ng appears in Figure 1.

### 3. PROPOSED ALGORITHM

To enforce constraint privacy, Brito introduced partially known constraints (PKC) model of a DisCSPs. In the PKC model, all constraints are binary and each binary constraint \( C_{ij} \) is divided into two non-intersecting subsets \( C_{ij}^0 \) and \( C_{ij}^1 \). \( A_i \) knows \( C_{ij}^0 \) and \( A_j \) knows \( C_{ij}^1 \), but none knows the total constraint \( C_{ij} \). A two phase Strategy for
solving DisCSPs under the PKC model was suggested in. It consists of a cycle of two phases [11, 25].
Originally, we implemented the two-phase strategy using AFC-ng. This combination was conceived for enforcing constraint privacy. We called it two-phase Asynchronous Forward Checking nogood-based (AFC-ng-2ph). It works as follows:

The main loop starts performing Phase I.

- Phase I: In this phase, the problem considers only one partial constraint for each pair of constrained agents. Agent \( A_i \) starts the search by calling procedure AFC-ng-I(). AFC-ng-I() is exactly similar to the standard AFC-ng algorithm. In AFC-ng-I(), agents exchange AFC-ng message types (where \( A_i \) is the sender) plus the messages qes, qnn. It try to find a solution with regard to partially known constraints. Assume agent \( A_i \) tries to find an assignment, which is consistent with its Agent View. If \( A_i \) could found a consistent assignment then \( A_i \) send the CPA to next agent \( A_j \), the receiver will try to extend the CPA (when it is the next agent on the ordering) now agent \( A_j \) try to find a consistent assignment on \( C_{ij} \) (\( i < j \)) which is held by \( A_j \) (limited \( C_{ij} \) will be considered in phase II). If no solution is found in phase I, the process stops with the failure reporting, since the whole problem has no solution. If a solution is found (a CPA consist of all agents generated according to one partial constraints), the generated CPA is passed to the second phase (phase II) where it is checked.

```plaintext
procedure AFC-ng()
InitAgentView();
end ← false; AgentView.Consistent ← true;
if (Ai = IA) then Assign();
while (~end) do
msg ← getMsg();
switch (msg.type)
CPA: ProcessCPA(msg);
ng: ProcessNogood(msg);
terminate: ProcessTerminate(msg);

procedure InitAgentView()
for each (\( x_i < x_k \)) do AgentView[i] ← \{(x_i, empty, 0)\};

procedure Assign()
if (\( D(x_i) \neq \emptyset \)) then
\( v_i \leftarrow \text{Choose Value}() \); \( t_i \leftarrow t_i + 1 \);
CPA ← \{AgentView \( \cup (x_i, v_i, t_i)\)\};
Send CPA(CPA);
else Backtrack();

procedure Send CPA(CPA)
if (size(CPA) = n) /* A_i is the last agent in the total ordering*/
broadcast Msg: terminate(CPA);
end ← true
else for each (\( x_k \rightarrow x_i \)) do send Msg: CPA(CPA) to \( x_k \);

procedure ProcessCPA(msg)
if (AgentView.Consistent ∧ AgentView ⊆ msg.CPA) then return;
if (msg.CPA is stronger than AgentView) then
Update Agent View (msg.CPA);
AgentView.Consistent ← true;
Revise();
if (\( D(x_i) = \emptyset \)) then Backtrack();
else Check Assign (msg, Sender);

procedure Check Assign (sender)
if (\( A_{i+1} \) = sender) then Assign(); /* the sender is the predecessor of \( A_i \)*/

procedure Backtrack()
ng ← solve (my No good Store);
if (ng = empty) then
broadcast Msg: terminate(∅);
```

- #2019-2015 CIS Journal. All rights reserved.

http://www.cisjournal.org
end ← true;
else
  for each (x_i > x_j) do /* x_j denotes the variable on rhs(ng) */
    AgentView[k].value ← empty;
  for each (nogood ∈ Nogood Store) do
    if (~Compatible(nogood, AgentView) \ V x_j ∈ nogood) then
      remove(nogood, Nogood Store);
      AgentView.Consistent ← false; v_i ← empty;
      Send Msg : ngd(ng) to A_j;
    end
  procedure ProcessNogood(msg)
    if (consistent(msg, Nogood, Agent View)) then
      add(msg. nogood, Nogood Store); /* according to the HPLV */
      if (rhs(msg. nogood).Value = v_i) then v_i ← empty; Assign();
  end
  procedure ProcessTerminate(msg)
    end ← true; v_i ← empty;
    if (msg.CPA ≠ ∅) then solution ← msg.CP A;
  end
  procedure Revise()
    for each (v ∈ D(x_i)) do
      if (v is ruled out by AgentView) then
        store the best nogood for v; /* according to the HPLV */
    end
  procedure Update Agent View(CPA)
    AgentView ← CPA; /* update values and tags */
    for each (ng ∈ myNogood Store) do
      if (~Consistent(ng, Agent View)) then remove(ng, Nogood Store);
    end
end

Figure 1: The AFC-ng algorithm [22]

against the partial constraints not considered in phase I. A solution is identified by detecting quiescence in the network.

- **Phase II.** In this phase, the partially known constraints that are not checked in phase I like C_{u,j}, are checked. The generated CPA is passed to all agents sequentially according to agent ordering. If CPA is also a solution of phase II, then it is a solution for the whole problem. Otherwise, A_i, the first agent that found inconsistency generates a nogood and store it in the nogood store. A_i cause the next agent A_j to change its value and does nothing. So, quiescence is detected. Now A_i causes the search is resumed in phase I from the A_j. Therefore, nogood found in phase II is used in phase I to escape from incompatible assignments.

The main loop ends when the new resolved nogood ng is empty (the problem is unsolvable) in phase I, or when Phase II does not generate nogood message (the solution of Phase I satisfies the constraints of Phase II).

In AFC-ng-2ph Agents exchange AFC-ng message types (CPA, ngd, terminate), plus the messages qes, qnn meaning quiescence in the network after ngd and after no ngd messages, respectively. AFC-ng-2ph inherits the good properties of AFC-ng: completeness, correctness and termination.

Regarding unsolvable instances, Phase I will eventually detect that there is no solution.

Figure 2 presents the pseudo code for AFC-ng-2ph algorithm. AFC-ng-2ph agents have the same data structures as AFC-ng agents. In the main procedure, the agents perform standard AFC-ng (procedure AFC-ng-II()) to find a solution compatible with half of partial constraints. If such a solution is obtained, Then phase II is performed (procedure AFC-ng-II()). If it is successful, the solution of the problem is retrieved from the CPA (no nogood generated during phase II) and the algorithm ends, otherwise, phase I is resumed.

4. THEORETICAL ANALYSIS (CORRECTNESS PROOFS)

Here, we prove that AFC-ng-2ph inherit the good properties of AFC-ng: completeness, correctness and termination. We assume that all algorithms follow the same agent ordering. The proof will be based on the fact that all algorithms can generate the same nogoods.
Theorem 1: The spatial complexity of AFC-ng-2ph is polynomially bounded by $O(n^2d)$.

Proof 1: In [22], it is proved that the spatial complexity of AFC-ng is polynomially bounded by $O(nd)$ per agent where $d$ is the size of the initial domain and $n$ the number of variables. Now about AFC-ng-2ph, since in phase1 the standard AFC-ng runs, spatial complexity of phase I is polynomially bounded by $O(nd)$ and phase2 take a little time. Also in worse case phase1 runs $n$ times. So the spatial complexity of AFC-ng-2ph is polynomially bounded by $O(n^2d)$.

Lemma 1: AFC-ng2-ph is guaranteed to terminate.

Proof 2: [22] proved by induction on the agent ordering that there will be a finite number of new generated CPAs (at most $d^n$, where $d$ is the size of the initial domain and $n$ the number of variables.), and that agents can never fall into an infinite loop for a given CPA. Therefore, AFC-ng-2ph inherit termination of AFC-ng.

Lemma 2: AFC-ng-2ph cannot infer inconsistency if a solution exists.

Proof 3: In [22] proved that AFC-ng is able to produce all solutions. So, AFC-ng cannot infer inconsistency if a solution exists. Therefore, AFC-ng-2ph cannot infer inconsistency if a solution exists.

Theorem 2: AFC-ng-2ph is correct.

Proof 4: The argument for soundness is close to the one given in [21, 28]. In [22] proved that the agents in AFC-ng receive only consistent assignments. A solution is reported by the last agent only. At this point, all agents have assigned their variables, and their assignments are consistent. Thus the AFC-ng-2ph algorithm is sound. Completeness comes from the fact that AFC-ng-2ph is able to terminate and does not report inconsistency if a solution exists (Lemmas 1 and 2).

```
Procedure AFC-ng-2ph()
InitAgentView();
end ← false; AgentView.Consistent ← true;
if ( A_i = IA ) then Assign();
repeat
    AFC-ng-I();
    if (~end)
        AFC-ng -II();
    until end or ~nogoods

procedure AFC-ng-I() quiescence ← false;
    CheckAgentView();
    while (~end ∧ ~quiescence) do
        msg ← getMsg();
        switch[msg.type]
            CPA: ProcessCPA(msg);
            ngd: ProcessNogood(msg);
            terminate: ProcessTerminate(msg);
            qes: quiescence ← true;

procedure AFC-ng-II()
    quiescence ← false;
    for each ( x_k > x_i) do sendMsg : CPA(CPA) to A_k;
    while (~quiescence) do
        msg ← getMsg();
        switch[msg.type]
            CPA: if ~consistent(v_i ,msg.CPA) then
                sendMsg:ngd(v_i = myValue ⇒ v_j ≠ msg.CPA);
                quiescence ← true; nogoods ← true;
                ngd: add(msg.nogood, NogoodStore) ; /* according to the HPLV [10] */
                if rhs(msg.nogood).value = v_i then v_i ← empty;
            qes: quiescence ← true; nogoods ← false; /* quiescence with nogoods messages */
            qnn: quiescence ← true; nogoods ← false; /* quiescence without nogoods messages */
```

Figure 2: The AFC-ng-2ph algorithm for the PKC model. Missing procedures appear in Figure1.
5. EXPERIMENTAL RESULTS

In this section we experimentally compare AFC-ng-2ph algorithms to AFC-ng, ABT-2ph [11] and DisFC-2ph [11]. These algorithms are evaluated on uniform random binary DisCSPs. All experiments were performed on the DisChoco2.0 platform [27], in which agents are simulated by Java threads that communicate only through message passing. All algorithms are tested on the same static agents ordering (lexicographic ordering) and the same nogood selection heuristic (HPLV) [10].

We evaluate the performance of the algorithms by communication load [20] and computation effort. Communication load is measured by the total number of exchanged messages among agents during algorithm execution (#msg), including those of termination detection (system messages). Computation effort is measured by the number of non-concurrent constraint checks (#ncccs) [17]. #ncccs is the metric used in distributed constraint solving to simulate the computation time.

The algorithms are tested on uniform random binary DisCSPs which are characterized by \(<n, d, P_1, P_2>\), where \(n\) is the number of agents/variables, \(d\) is the number of values in each of the domains, \(P_1\) is the network connectivity defined as the ratio of existing binary constraints, and \(P_2\) is the constraint tightness defined as the ratio of forbidden value pairs. We solved instances of two classes of constraint networks: sparse networks \(<20, 10, 0.25, P_2>\) and dense ones \(<20, 10, 0.7, P_2>\). We vary the constraint tightness (i.e., \(P_2\)) from 0.1 to 0.9 by steps of 0.1. For each pair of fixed density and tightness \((P_1, P_2)\) we generated 100 instances. The average over the 100 instances is reported.

In Figure 3, we present the performance of the algorithms on the sparse instances \((P_1= 0.25)\). Concerning the computational effort (Figure 3(a)), AFC-ng-2ph is better than DisFC-2ph because it has no domain privacy unlike DisFC-2ph also, it is worse than ABT-2ph because in phase II it should check all agents’ value in the CPA. Among these algorithms AFC-ng is the fastest one.

Concerning the communication load (Figure 3(b)), AFC-ng-2ph is better than DisFC-2ph and ABT-2ph. Because in phase II, AFC-ng-2ph sends only one CPA message synchronously, But DisFC-2ph and ABT-2ph require numbers of exchanged messages. AFC-ng-2ph exchanges slightly fewer messages than ABT-2ph and DisFC-2ph at the complexity peak. It thus seems that on sparse problems, synchronous search algorithm provides more benefit than asynchronous ones.

In Figure 4, we present the performance of the algorithms on the dense instances \((P_1= 0.7)\). Concerning the computational of effort also it is better than DisFC-2ph and ABT-2ph in terms of communication load.
Figure 3(b): Number of messages sent on sparse random DisCSPs

Figure 4(a): #ncccs performed on dense random DisCSPs.

Figure 4(b): Number of messages sent on dense random DisCSPs
6. CONCLUSION

Privacy is one of the main motivations for solving distributed constraint satisfaction problems.

Privacy of constraints is concerned with constraints that are initially private between agents, and they remain as private as possible during the solving process. We have proposed a new synchronous search algorithm for solving DisCSPs. Our new algorithm, AFC-ng-2ph, improve Privacy of constraints of AFC-ng by using the PKC model. AFC-ng-2ph algorithm is clear descendant of AFC-ng, it uses the same kind of messages plus some extra ones and keeps its good properties. AFC-ng-2ph is not perfect and leak some information in the solving process. When considering constraints privacy, we have observed that the amount of revealed information depends on constraint tightness, although AFC-ng-2ph always leaks less information than standard AFC-ng. Experiments also show that AFC-ng-2ph is better than DisFC-2ph in terms of computational effort also it is better than DisFC-2ph and ABT-2ph in terms of communication load.

REFERENCES


