Constraints (2009) 14:199–234
DOI 10.1007/s10601-008-9048-x

Distributed constraint satisfaction with partially known constraints

Ismel Brito · Amnon Meisels · Pedro Meseguer · Roie Zivan

Published online: 15 May 2008

Abstract Distributed constraint satisfaction problems (DisCSPs) are composed of agents connected by constraints. The standard model for DisCSP search algorithms uses messages containing assignments of agents. It assumes that constraints are checked by one of the two agents involved in a binary constraint, hence the constraint is fully known to both agents. This paper presents a new DisCSP model in which constraints are kept private and are only partially known to agents. In addition, value assignments can also be kept private to agents and not be circulated in messages. Two versions of a new asynchronous backtracking algorithm that work with partially known constraints (PKC) are presented. One is a two-phase asynchronous backtracking algorithm and the other uses only a single phase. Another new algorithm preserves the privacy of assignments by performing distributed forward-

Partially supported by the Spanish project TIN2006-15387-C03-01. Partially supported by the Lynn and William Frankel center for Computer Sciences and the Paul Ivanier Center for Robotics and Production Management.

I. Brito · P. Meseguer
IIIA, Ins. Inv. Intel.ligència Artificial,
CSIC, Campus UAB, 08193 Bellaterra, Spain

I. Brito
e-mail: ismel@iiia.csic.es

P. Meseguer
e-mail: pedro@iiia.csic.es

A. Meisels
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel
e-mail: am@cs.bgu.ac.il

R. Zivan (✉)
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beer-Sheva, 84-105, Israel
e-mail: zivanr@bgu.ac.il

checking (DisFC). We propose to use entropy as quantitative measure for privacy. An extensive experimental evaluation demonstrates a trade-off between preserving privacy and the efficiency of search, among the different algorithms.

Keywords Distributed CSPs · Asynchronous search · Privacy · Entropy

1 Introduction

Distributed constraint satisfaction problems (DisCSPs) are composed of agents, each holding its local constraint network, which are connected by constraints among variables of different agents. Agents assign values to variables, attempting to generate a locally consistent assignment that is also consistent with all constraints between agents [21, 23]. To achieve this goal, agents check the value assignments of their variables for local consistency and exchange messages with other agents, to check consistency of their proposed assignments against constraints with variables owned by different agents [1].

DisCSP is an elegant model for many every day combinatorial problems that are distributed by nature. Take for example the meeting scheduling problem in which n agents attempt to schedule k meetings. In each meeting, a sub-group of the n agents participate [6, 12, 14, 22]. Arrival constraints define the time that must differentiate meetings with common participants.

Standard search algorithms for DisCSPs, like asynchronous backtracking (ABT) [24, 25], assume a static priority order among all agents. Higher priority agents perform assignments and send them via messages to lower priority agents. ABT assumes that every inter-agent constraint can be checked by the lower priority agent that is involved in the constraint, i.e. the lower priority agent must hold the entire constraint [23]. In dynamic ordering ABT [27], each assignment is checked for consistency according to the order at the time it is performed. Since both agents involved in a binary constraint can be with lower priority at the time the assignment is checked, both agents are required to hold the entire constraint.

In many real world problems the above assumptions are too strong. In the meeting scheduling example, people are usually not willing to reveal their private schedule which imposes constraints on the schedule of meetings they participate in. A more suitable model for a realistic DisCSP is the *partially known constraints* (PKC) model, where each inter-agent constraint is composed of two parts, each held by one of the two constraining agents [2]. When agents hold parts of the constraint privately, checking for consistency has to be performed by both of the constrained agents because some value combinations may be seen as permitted by one agent and forbidden by the other.

In [19] an algorithm which keeps the constraints of agents private was presented. *Asynchronous Aggregations Search* (AAS) enables the filtering of a global assignment by agents according to their private constraints. Unlike the common definition of DisCSP where variables are distributed among agents, AAS considers the dual case where constraints are distributed and controlled by a single agent. The use of AAS may cause that the problem to be solved has to be translated into a new one. This transformation could be inadequate in many naturally distributed problems where the initial problem structure must remain unchanged. Furthermore, in AAS there is no privacy of domains i.e. all agents hold the domains for all variables. This property makes AAS unsuitable for many real world problems that are concerned with privacy.

A number of secured protocols for DisCSPs which use cryptographic tools in order to preserve privacy were proposed in recent years [15, 18, 26]. All of these studies propose secured protocols which can solve asymmetric constraints. However, the overhead in communication and computation in all of these protocols is very large.

In this paper we differentiate between two types of privacy: privacy of constraints and privacy of assignments. Regarding privacy of constraints we assume the PKC model of DisCSPs, where each binary constraint is divided between the two constraining agents. A value tuple is allowed by the constraint if it is allowed by each part separately. To solve the resulting DisCSP, two asynchronous backtracking algorithms are proposed: ABT-2ph, a two-phase algorithm and ABT-1ph a single-phase one. Similarly to standard ABT, a static order of priorities is defined among all agents in both algorithms. In the first phase of ABT-2ph, an asynchronous backtracking algorithm is performed, in which only the constraints held by the lower priority agents are examined. In other words, only one of the two constraining agents in each binary constraint checks for consistency. When a solution is reached, a second phase is performed in which the consistency of the solution is checked again, according to the constraints held by the higher priority agents in each binary constraint. If no constraint is violated, a solution is reported. If there are violated constraints, the first phase is resumed after the necessary nogoods are recorded.

The first and immediate drawback of a two-phase algorithm is the effort of producing solutions in each first phase. Since constraints in the opposite direction are not examined, large parts of the search space, which could have been pruned if all constraints were considered, are being exhaustively scanned. The second drawback is the synchronized manner in which the algorithm switches between the two phases. For each such switch among phases, a termination detection mechanism must be performed which is a complicated task in asynchronous backtracking. Furthermore, all agents must be informed about every switch between phases.

In order to avoid these drawbacks, a single-phase distributed search algorithm (ABT-1ph) is proposed. ABT-1ph checks inter-agent constraints asynchronously at both of the constraining agents. Agents send their proposed assignments to all their neighbors in the constraint graph, with both higher and lower priority ones. Agents assign their local variables according to the priority order as in standard ABT, but check the constraints also against the assignment of lower priority agents. Nogoods are sent both from lower priority agents, as in standard ABT, and from higher to lower priority agents.

An algorithm for preserving privacy of assignments was proposed in [2], called *distributed forward checking* (DisFC). Unlike ABT, in DisFC constraints are checked by higher priority agents. Instead of sending its own assignment, every DisFC agent sends to each lower priority neighbor the subset of values of the neighboring agent which are consistent with its own assignment. To preserve privacy of constraints and assignments, this paper considers two versions of DisFC: DisFC-2ph [2] and DisFC-1ph, a double and a single-phase algorithm, respectively. Similar to the versions of ABT for PKC, all constraints are simultaneously considered in the single-phase of DisFC-1ph while some constraints are checked in the first phase of DisFC-2ph and the others in its second phase.

The evaluation of the proposed algorithms is done taking into account computation effort, communication cost and privacy loss. The evaluation of computation effort and communication cost in DisCSPs is performed according to the methods of [28]. To evaluate privacy loss, the natural measure is the entropy [5]. Entropy decrement, from the initial state of the search to its final state, is taken as a measure of the privacy loss during search. Regarding privacy of assignments, the loss during a complete search can be evaluated theoretically. Regarding privacy of constraints, we adjust the method of [11] to evaluate the constraint privacy alone by measuring the percentage of the conflicts in a constraint matrix held by an agent that are revealed to another agent involved in a conflict. This approach is motivated by the fact that both of our proposed algorithms will reveal some of the information that ABT would have revealed during its execution via standard *ok?* and *ngd* messages. However, in order to perform standard ABT, the entire part of every binary constraint held by the higher priority agent must be revealed to the lower priority agent. Our results focus on the part of the constraint that is revealed relatively to standard ABT.

The idea of entropy as privacy measure has already been considered in distributed constraint optimization problems, particularly in the context of the meeting scheduling problem [7, 11]. The entropy model that we present in this paper is an adjustment of these works to constraints and assignment privacy in DisCSPs, which is out of the scope of previous works.

The paper is organized as follows. A formal DisCSP definition appears in Section 2. Section 3 contains a summary of the standard ABT algorithm. Privacy of DisCSPs and the description of the PKC model appear in Section 4. In Section 5 we present two ABT versions for the PKC model, ABT-2ph and ABT-1ph. Following each algorithm description are its correctness and completeness proofs. In Section 6 we discuss an option for preserving assignment privacy by sending domain subsets instead of assignments (DisFC). In Section 7 we present a theoretical model for evaluating privacy loss by the entropy decrement during search, considering privacy of assignments and privacy of constraints. An extensive experimental evaluation, which demonstrates the difference between the algorithms with respect to performance, communication and loss of privacy appears in Section 8. Finally, Section 9 contains some conclusions of this work.

2 Preliminaries

A *constraint satisfaction problem* (CSP) involves a finite set of variables, each one taking a value in a finite domain. Variables are related by constraints that impose restrictions on the combinations of values that subsets of variables can take. A *solution* is an assignment of values to variables which satisfies every constraint. Formally, a finite CSP is defined by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables;
- $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is a collection of finite domains; $D(x_i)$ is the initial set of possible values for x_i;
- $\mathcal{C}$ is a set of constraints among variables. A constraint c_i on the ordered set of variables $var(c_i) = (x_{i_1}, \ldots, x_{i_{r(i)}})$ specifies the relation $prm(c_i)$ of the *permitted*

combinations of values for the variables in $var(c_i)$. An element of $prm(c_i)$ is a tuple $(v_{i_1}, \ldots, v_{i_{r(i)}}),\ v_i \in D(x_i)$.

A *distributed constraint satisfaction problem* (DisCSP) is a CSP where variables, domains and constraints are distributed among automated agents. Formally, a finite DisCSP is defined by a 5-tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{A}, \phi)$, where $\mathcal{X}$, $\mathcal{D}$ and $\mathcal{C}$ are as before, and

- $\mathcal{A} = \{1, \ldots, p\}$ is a set of p agents,
- $\phi : \mathcal{X} \rightarrow \mathcal{A}$ is a function that maps each variable to its agent.

Each variable belongs to one agent. The distribution of variables divides $\mathcal{C}$ into two disjoint subsets, $\mathcal{C}_{intra} = \{c_i | \forall x_j, x_k \in var(c_i), \phi(x_j) = \phi(x_k)\}$, and $\mathcal{C}_{inter} = \{c_i | \exists x_j, x_k \in var(c_i), \phi(x_j) \neq \phi(x_k)\}$, called intra-agent and inter-agent constraint sets, respectively. An intra-agent constraint c_i is known by the agent owner of $var(c_i)$, and it is unknown by the other agents. Usually, it is considered that an inter-agent constraint c_j is known by every agent that owns a variable of $var(c_j)$ [25].

As in the centralized case, a *solution* of a DisCSP is an assignment of values to variables satisfying every constraint. DisCSPs are solved by the collective and coordinated action of agents $\mathcal{A}$. Agents communicate by exchanging messages. It is assumed that the delay in delivering a message is finite but random. For a given pair of agents, messages are delivered in the order they were sent.

For simplicity purposes, and to emphasize the aspects of distribution, in the rest of this study we assume that each agent owns exactly one variable. We identify the agent number with its variable index ($\forall x_i \in \mathcal{X}, \phi(x_i) = i$). From this assumption, all constraints are inter-agent constraints, so $\mathcal{C} = \mathcal{C}_{inter}$ and $\mathcal{C}_{intra} = \emptyset$. Furthermore, we assume that all constraints are binary. We use the term c_{ij} to indicate a constraint that binds agents x_i and x_j.

We further assume that all information held by agents has an equal importance with respect to privacy (i.e. there is no part of the agents private information which is more important than other parts). This is a necessary assumption in order to be able to compute objectively a loss of privacy (see for example [11]). This is the basis for our experimental evaluation of privacy loss in Section 8.

3 Asynchronous backtracking

All the algorithms proposed in this paper are based on ABT. A summarized description of ABT is presented here. For details, the reader is addressed to the original papers [23–25].

Asynchronous backtracking (ABT) [23–25] was a pioneering algorithm to solve DisCSPs. ABT is an asynchronous algorithm executed autonomously by each agent, which makes its own decisions and informs other agents about them. The algorithm computes a solution (or detects that no solution exists) in finite time; the algorithm's correctness and completeness have been proven.

ABT requires constraints to be directed. A constraint causes a directed link between two constrained agents: the assignment-sending agent, from which the link leaves, and the constraint-evaluating agent, to which the link arrives. When the assignment-sending agent makes an assignment, it informs the constraint-evaluating agent, which then tries to find a consistent value assignment. To make the network cycle-free, there is a total order among agents that corresponds to the directed links. Agent i has higher priority than agent j if i appears before j in the total order.

Each ABT agent keeps its own agent view and nogood list. Considering a generic agent $self$, the agent view of $self$ is the set of values that it believes to be assigned to agents connected to $self$ by incoming links. The nogood list keeps the nogoods received by $self$ as justifications for the removal of inconsistent values. Agents exchange assignments and nogoods.

The algorithm starts by each agent assigning its variable, and sending the assignment to its neighboring agents with lower priority. When an agent receives an assignment, it updates its agent view with the received assignment, removes inconsistent nogoods and checks the consistency of its current assignment with the updated agent view.

When receiving a nogood, it is accepted if it is consistent with the agent view of $self$. Otherwise, it is discarded since it is found to be obsolete. An accepted nogood is used to update the nogood list. It makes $self$ search for a new consistent value, since the received nogood is a justification that forbids its current value. When an agent cannot find any value consistent with its agent view, either because of the original constraints or because of the received nogoods, new nogoods are generated from its agent view and each one sent to the closest agent involved in it. This operation causes backtracking.

There are several versions of ABT, depending on the way that new nogoods are generated. In the simplest form of the ABT algorithm, the complete agent view is sent as a nogood [23]. The nogood is sent to the lowest priority agent whose assignment is included in the nogood. In contrast, in the version of ABT that appears in [1], when an agent cannot find a consistent value, it resolves its nogoods lists following a procedure based on dynamic backtracking methods. From this resolution, a new nogood is generated and sent to the agent with the lowest priority involved in the new nogood.

If $self$ receives a nogood including the assignment of an agent not connected with it, $self$ requires that a new link will be added from that agent to $self$. From this point on, a link from the other agent to $self$ will exist. ABT execution ends upon achieving quiescence in the agent network, meaning that a solution has been found, or when an empty nogood is generated, which indicates that the problem is unsolvable.

The ABT code appears in Fig. 1. This code uses the following data structures: Γ^-, Γ^+, *myAgentView* and *myNogoodStore*. Γ^- contains $self$'s higher priority constraining agents, whereas Γ^+ contains $self$'s lower priority constraining agents [1]. Agent $self$ stores assignments of higher priority agents in *myAgentView* and the received nogoods in *myNogoodStore*. Agents exchange four different kinds of messages: **ok?**, **ngd**, **adl** and **stp**. An **ok?** message comes from higher priority agents informing $self$ of the new assignment for the variable of the message's sending agent. A **ngd** message comes from lower priority agents, and includes a nogood which will serve as a justification for the removal of the value assigned to $self$'s variable. An **adl** message arrives from lower priority agents, requesting $self$ to add a new link between it and the sender. When an agent receives a **stp** message, it means that the problem is unsolvable since an empty nogood was found by at least one agent and the search terminates.

```
procedure ABT()
  $myValue \leftarrow$ empty; $end \leftarrow$ false; compute $\Gamma^+, \Gamma^-$;
  CheckAgentView();
  while ($\neg end$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?   : ProcessInfo($msg$);
      ngd   : ResolveConflict($msg$);
      adl   : SetLink($msg$);
      stp   : $end \leftarrow$ true;

procedure CheckAgentView(msg)
  if $\neg$consistent($myValue, myAgentView$) then
    $myValue \leftarrow$ ChooseValue();
    if ($myValue$) then for each $child \in \Gamma^+(self)$ do sendMsg:ok?($child, myValue$);
    else Backtrack();

procedure ProcessInfo($msg$)
  UpdateAgentView($msg.Assig$); CheckAgentView();

procedure ResolveConflict($msg$)
  if Coherent($msg.Nogood, \Gamma^-(self) \cup \{self\}$) then
    CheckAddLink($msg$);
    add($msg.Nogood, myNogoodStore$); $myValue \leftarrow$ empty;
    CheckAgentView();
  else if Coherent($msg.Nogood, self$) then SendMsg:ok?($msg.sender, myValue$);

procedure Backtrack()
  $newNogood \leftarrow$ solve($myNogoodStore$);
  if ($newNogood =$ empty) then
    $end \leftarrow$ true; sendMsg:stp($system$);
  else
    sendMsg:ngd($newNogood$);
    UpdateAgentView(rhs($newNogood$) $\leftarrow$ unknown);
    CheckAgentView();

function ChooseValue()
  for each $v \in D(self)$ not eliminated by $myNogoodStore$ do
    if consistent($v, myAgentView[\Gamma^-(self)]$) then return ($v$);
    else add($x_j = val_j \Rightarrow self \neq v, myNogoodStore$); /*$v$ is inconsistent with $x_j$'s value */
  return (empty);

procedure UpdateAgentView($newAssig$)
  add($newAssig, myAgentView$);
  for each $ng \in myNogoodStore$ do
    if $\neg$Coherent(lhs($ng$), $myAgentView$) then remove($ng, myNogoodStore$);

function Coherent($nogood, agents$)
  for each $var \in nogood \cup agents$ do
    if $nogood[var] \neq myAgentView[var]$ then return false;
  return true;

procedure SetLink($msg$)
  add($msg.sender, \Gamma^+(self)$); sendMsg:ok?($msg.sender, myValue$);

procedure CheckAddLink($msg$)
  for each ($var \in$ lhs($msg.Nogood$))
    if ($var \notin \Gamma^-(self)$) then
      sendMsg:adl($var, self$); add($var, \Gamma^-(self)$); UpdateAgentView($var \leftarrow varValue$);
```

Fig. 1 The ABT algorithm

4 Privacy in ABT-like algorithms

ABT agents exchange assignments and nogoods. Regarding privacy, there are two issues that reveal the private data of ABT agents:

- Constraints: An inter-agent constraint c_{ij} is totally known by the lower priority agent (j for $i < j$).
- Assignments: Agents notify other agents about their value assignments.

This approach may be inappropriate for applications in which privacy is the main reason for distributed solving. In this case, agents may desire to hide the actual values of their variables from other agents, considered to be potential competitors. For the same reasons, the information contained in the constraints may be considered as reserved and agents might not be willing to share it with other agents. In the following, we develop strategies to maintain privacy of variable values and of inter-agent constraints, allowing agents to share enough information to achieve a solution or detect that no solution exits.

4.1 The PKC model for constraint privacy

ABT assumes that an inter-agent constraint c_{ij} is *totally known* by the agents owning their related variables, that is, c_{ij} is totally known by agent i and agent j (see Section 2.2 of [25]). In fact, it is enough for ABT that the lower priority agent in each constraint knows the set of permitted tuples. To enforce constraint privacy, we introduce the *partially known constraints* (*PKC*) model of a DisCSP as follows. A constraint c_{ij} is partially known by its related agents. Agent i knows the constraint $c_{i(j)}$ where:

- $vars(c_{i(j)}) = \{x_i, x_j\}$;
- $c_{i(j)}$ is specified by three disjoint sets of value tuples for x_i and x_j:
  - $prm(c_{i(j)})$, the set of tuples that i knows to be permitted;
  - $fbd(c_{i(j)})$, the set of tuples that i knows to be forbidden;
  - $unk(c_{i(j)})$, the set of tuples whose consistency is unknown by i;
- Every possible tuple is included in one of the above sets, that is, $prm(c_{i(j)}) \cup fbd(c_{i(j)}) \cup unk(c_{i(j)}) = D_i \times D_j$.

Similarly, agent j knows $c_{(i)j}$, where $vars(c_{(i)j}) = \{x_i, x_j\}$. $c_{(i)j}$ is specified by the disjoint sets $prm(c_{(i)j})$, $fbd(c_{(i)j})$ and $unk(c_{(i)j})$. For the model to be truly partial, it is required that, there is at least one pair of constrained agents i and j that do not have the same information about the shared constraint (i.e. they differ in at least one of the three sets of tuples). The relation between a totally known constraint c_{ij} and its corresponding partially known constraints $c_{i(j)}$ and $c_{(i)j}$ is

$$c_{ij} = c_{i(j)} \otimes c_{(i)j}$$

where $\otimes$ depends on the semantic of the constraint. The above definitions satisfy the following conditions:

- If the combination of values k and l, for x_i and x_j is forbidden in at least one partial constraint, then it is forbidden in the corresponding total constraint: if $(k, l) \in fbd(c_{i(j)})$ or $(k, l) \in fbd(c_{i(j)})$ then $(k, l) \in fbd(c_{ij})$.

- If the combination of values k and l, for x_i and x_j is permitted in both partial constraints, then it is also permitted in the corresponding total constraint: if $(k, l) \in prm(c_{i(j)})$ and $(k, l) \in prm(c_{i(j)})$ then $(k, l) \in prm(c_{ij})$.

In this paper, we only consider constraints for which $unk(c_{(i)j}) = unk(c_{i(j)}) = \emptyset$.[1] In this case, a partially known constraint $c_{i(j)}$ is completely specified by its permitted tuples (tuples not in $prm(c_{i(j)})$ are in $fbd(c_{i(j)})$). Furthermore,

$$prm(c_{ij}) = prm\left(c_{i(j)}\right) \cap prm\left(c_{(i)j}\right) \tag{1}$$

For example, let us consider the *n-pieces m-chessboard* problem. Given a set of n chess pieces and a $m \times m$ chessboard, the goal is to put all pieces on the chessboard in such a way that no piece attacks any other. As DisCSP, the problem can be formulated as follows,

- Variables: one variable per piece.
- Domains: all variables share the domain $\{1, \dots, m^2\}$ of chessboard positions.
- Constraints: one constraint between every pair of pieces, following chess rules.
- Agents: one agent per variable.

For example, we can take $n = 5$ with the set of pieces {*queen, castle, bishop, bishop, knight*}, on a 4×4 chessboard, with the variables,

$$x_1 = queen,\ x_2 = castle,\ x_3 = bishop,\ x_4 = bishop,\ x_5 = knight.$$

If agent 1 knows that agent 5 holds a knight, and agent 5 knows that agent 1 holds a queen, the result is a completely known constraint $prm(c_{15})$ including the following tuples,

$$prm(c_{15}) = \left\{(1, 8), (1, 12), (1, 14), (1, 15), \dots\right\}$$

With the PKC model, agent 1 does not know which piece agent 5 holds. It only knows how a queen attacks, from which it can develop the constraint,

$$prm(c_{1(5)}) = \left\{(1, 7), (1, 8), (1, 10), (1, 12), \dots\right\}$$

Analogously, agent 5 does not know which piece agent 1 holds. Its only information is how a knight attacks, from which it can develop the constraint,

$$prm(c_{(1)5}) = \left\{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 8), \dots\right\}$$

The whole constraint c_{15} is equal to the intersection of these two constraints,

$$prm(c_{15}) = c_{1(5)} \cap c_{(1)5} = \left\{(1, 8), \dots\right\}$$

[1]In the PKC formulation for versions of stable marriage problem in which people desire to keep their marriage proposals (assignments) and preference lists (constraints) private during the search of a stable matching, $unk(c_{(i)j}) = unk(c_{i(j)}) \neq \emptyset$, see [3] for more details.

In fact, $prm(c_{1(5)})$ does not depend on agent 5. It codifies the way a queen attacks, which is independent of any other piece. In this problem, the PKC model allows each agent to represent its constraints, independently of other agents.

4.2 Assignment privacy

Agents may desire to keep the assigned values to their variables private. To achieve this, agents must avoid sending their assigned values to other agents. An ABT agent sends its value in two types of messages (between i and j, $i < j$):

1. **ok?**: when agent i informs low priority agents of its value assignment. This message is used by j to find a compatible value with i, so j has to know the constraint c_{ij}.
2. **ngd**: when agent j sends a backtrack message to i. This message contains the value assignments of the *AgentView* of j. It is used by i to check whether the nogood message is obsolete, by checking whether the assignments of common variables with agents of higher priority than i, in the agent views of i and j, are the same.

The first point could be solved if, instead of sending i's current value, the **ok?** message contains the subset of D_j values that are compatible with i's current value. From this subset, j may be consistently assigned with respect to the constraint with i.

The second point can be solved by the use of identifiers. We propose to use a sequential number over the assignments of the variable. Each variable keeps a sequence number that starts from 1 (or some random value), and increases monotonically each time the variable changes its assignment, acting as a unique identifier for each assignment. Messages include the sequential number of the assignment of the sending agent. The agent view of the receiver is composed of the sequence numbers it received in **ok?** messages from higher priority agents. Nogoods are composed of variables and their sequential numbers.

5 ABT algorithms for partially known constraints

We present two ABT algorithms for solving DisCSP that incorporate the PKC model of constraints. Initially, solving algorithms have two solving phases, one for each partial constraint. Later, these two phases are combined into one.

We emphasize that in the case of constraints privacy, both of our proposed algorithms reveal the same kind of information that standard ABT would have revealed in its messages when solving the same problem. However, standard ABT requires that for every binary constraint, the entire part of the constraint which is initially held by the higher priority agent will be revealed to the lower priority agent before the algorithm starts. Our proposed algorithms attempt to minimize this exposure of information to the minimum necessary.

5.1 Two-phase strategy

In the PKC model, if agents i and j are constrained, i knows $c_{i(j)}$ and j knows $c_{(i)j}$, but none knows the total constraint c_{ij}. The first method for solving DisCSP under the PKC model appears in [2]. It consists of a cycle of two phases. In the first phase (phase I), the original problem is relaxed considering only one partial constraint for each pair of constrained agents. If no solution is found in phase I, the procedure ends returning failure, since no solution exists for the whole problem. If a solution is found, it is passed to the second phase (phase II) where it is checked against the partial constraints not considered in phase I. If it is also a solution of phase II, then it is a solution for the whole problem. Otherwise, one or several nogoods are generated and the search is resumed in phase I. Nogoods found in phase II are used in phase I to escape from incompatible assignments.

The two-phase strategy is a generic method which implementation details depend on the algorithm used to find a solution with respect to the considered partial constraints.

ABT-2ph. The ABT-2ph algorithm is a combination of the ABT algorithm with the two-phase strategy. It works as follows:

- Phase I. Constraints are directed, forming a DAG, and a compatible total order of agents is selected. The standard ABT algorithm finds a solution with respect to constraints $c_{(i)j}$, where agent i has higher priority than j (the constraint $c_{(i)j}$ is checked by the lower priority agent, j). A solution is identified by detecting quiescence in the network. If no solution is found, the process stops, reporting failure.
- Phase II. Constraints and the order of agents are reversed. Now $c_{i(j)}$ are considered, where j has higher priority than i (e.g. in the *reversed* order). j informs i of its value. If the value of i is consistent, i does nothing. Otherwise, i sends a **ngd** message to j, which receives that message and does nothing (this nogood will be used when Phase I is resumed). Quiescence is detected.

Figure 2 presents the code for the ABT-2ph algorithm. In the main procedure, the agents perform standard ABT (procedure `ABT-I()`) to find a solution compatible with all constraints held by lower priority agents. If such a solution is obtained, the agents reverse the total order by exchanging Γ^- and Γ^+. Then phase II is performed (procedure `ABT-II()`). If it is successful (no nogood generated during phase II), the algorithm terminates. Otherwise Γ^- and Γ^+ are exchanged again and phase I is resumed. Agents exchange ABT message types, plus the messages **qes**, **qnn** meaning quiescence in the network after **ngd** and after no **ngd** messages, respectively.

An extra agent called *system* is responsible for detecting network quiescence. Messages **qes** and **qnn** are sent by *system* to the rest of agents. Network quiescence state can be detected by *system* using specialized snapshot algorithms [4].

5.2 One-phase strategy

Instead of checking only part of the constraints in phase I and verifying the proposed solution in phase II, all constraints can be tested simultaneously in a single phase. An agent has to check all its partially known constraints with both higher and lower priority agents. To do this, an agent has to inform all its neighboring agents when it assigns a new value, and nogood messages can go in both directions (from lower priority to higher priority agents as in ABT but, also from higher to lower). In the

```
procedure ABT-2ph()
  compute $\Gamma^-, \Gamma^+$; $myValue \leftarrow$ empty; $end \leftarrow$ false; $nogoods \leftarrow$ false;
  repeat
    ABT-I();
    if ($\neg end$)
      exchange $\Gamma^-, \Gamma^+$;
      ABT-II();
      exchange $\Gamma^-, \Gamma^+$;
    until $end$ or $\neg nogoods$

procedure ABT-I()
  $quiescence \leftarrow$ false;
  CheckAgentView();
  while ($\neg end \wedge \neg quiescence$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?   : ProcessInfo($msg$);
      ngd   : ResolveConflict($msg$);
      adl   : SetLink($msg$);
      stp   : $end \leftarrow$ true;
      qes   : $quiescence \leftarrow$ true;

procedure ABT-II()
  $quiescence \leftarrow$ false;
  for each $child \in \Gamma^+(self)$ do sendMsg:ok?($child, myValue$));
  while ($\neg quiescence$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?  : if $\neg$ consistent($myValue, msg.Value$) then
                 sendMsg:ngd($self = myValue \Rightarrow msg.Sender \neq msg.Value$);
      ngd  : add(lhs($msg.Nogood, myNogoodStore$)); $myValue \leftarrow$ empty;
      qes  : $quiescence \leftarrow$ true; $nogoods \leftarrow$ true; /* quiescence with nogoods messages */
      qnn  : $quiescence \leftarrow$ true; $nogoods \leftarrow$ false; /* quiescence without nogoods messages */
```

Fig. 2 The ABT-2ph algorithm for the PKC model. Missing procedures appear in Fig. 1

following, we present the ABT-1ph algorithm obtained by combining ABT with the one-phase strategy.

ABT-1ph. In ABT, binary constraints are held completely by the lower priority agent involved in each constraint according to a total agent ordering. Lower priority agents check consistency of assignments received from higher priority agents via **ok?** messages. In the PKC model, constraints are only partially known to each of the participating agents. Consequently, both of the constrained agents need to check the consistency of their assignments against each other. This means that checking consistency of a pair of constrained assignments by the lower priority agent is no longer sufficient.

In single-phase asynchronous backtracking for the PKC model (ABT-1ph), each agent checks its constraints with all constraining agents (i.e. neighbors on the constraint graph). This includes higher priority, as well as lower priority constraining agents. Agents hold in their *myAgentView* the assignments of agents with higher and lower priorities. Values from the domain of agents are eliminated *only if they violate constraints with higher priority agents*. After a new assignment is found to

```
procedure ABT-1ph()
  $myValue \leftarrow$ empty; $end \leftarrow$ false; compute $\Gamma^+$, $\Gamma^-$;
  CheckAgentView();
  while ($\neg end$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?   : ProcessInfo($msg$);
      ngd   : ResolveConflict($msg$);
      adl   : SetLink($msg$);
      stp   : $end \leftarrow$ true;

procedure ProcessInfo($msg$)
  UpdateAgentView($msg.Assig$);
    if $\neg$consistent($myValue, msg.Assig$) then
      if ($msg.Sender \in \Gamma^+$) then
        SendMsg:ngd($msg.Sender, self = myValue \Rightarrow msg.Sender \neq msg.Assig$);
      else CheckAgentView();

procedure CheckAgentView(msg)
  if $\neg$consistent($myValue, myAgentView[\Gamma^-]$) then
    $myValue \leftarrow$ ChooseValue();
    if ($myValue$) then
      for each $child \in \Gamma^+(self) \cup \Gamma^-(self)$ do sendMsg:ok?($child, myValue$);
      for each $child \in \Gamma^+(self)$ such that $\neg$consistent($myValue, child.Assig$) do
                sendMsg:ngd($child, self = myValue \Rightarrow \neg child.Assig$);
    else Backtrack();
```

Fig. 3 The ABT-1ph algorithm for the PKC model. Missing procedures appear in Fig. 1

be consistent with all assignments of higher priority agents in *myAgentView* (agents in Γ^-), the selected assignment is checked against the assignments of lower priority agents (agents in Γ^+). If a conflict is detected, the agent keeps its assignment and sends a **ngd** message to the lower priority agent, including its own assignment and the conflicting assignment. An agent processes **ngd** messages in the same way, no matter if they come from higher or lower priority agents.

Figure 3 presents the changes in ABT to transform it into ABT-1ph. The only differences are `ProcessInfo()` and `CheckAgentView()`. In the first procedure, after receiving the assignment of a lower priority agent, if it is not compatible with *self* current value, a **ngd** message is sent to that agent. In the second procedure, after assigning *self* with a value consistent with all assignments of agents in Γ^-, it is sent to all agents in $\Gamma^- \cup \Gamma^+$. In addition, if there are agents in Γ^+ with inconsistent values, a **ngd** message is sent to them. Although a nogood is sent, the current assignment (*myValue*) is not replaced.

5.3 Formal properties

Here we prove that ABT-2ph and ABT-1ph inherit the good formal properties of ABT. The search space is defined by the variables and domains of the problem instance. The way this space is traversed depends on (1) the total order among agents and (2) the set of nogoods generated during asynchronous search. Assuming that all algorithms follow the same agent ordering, the proof will be based on the fact that all algorithms generate the same nogoods. This is proven in the following lemma.

Lemma 1 *A nogood can be generated by ABT-2ph/1ph iff it can be generated by ABT.*

Proof Let us differentiate between explicit and implicit nogoods. In ABT, an explicit nogood is generated as a consequence of an **ok?** message. An implicit nogood is generated by resolution of the set of nogoods that forbid all values of a variable.

Explicit nogoods Let i and j be two agents, $i < j$ in the total order, and let $x_i = v \Rightarrow x_j \neq w$ be a nogood generated by ABT-2ph/1ph. If the pair (v, w) is forbidden in $c_{(i)j}$, this nogood will be generated in j after receiving the **ok?** message containing $x_i = v$, and it will be stored in j. Otherwise, if the pair (v, w) is permitted in $c_{(i)j}$ but forbidden in $c_{i(j)}$, it will be generated in i, and it will be sent from i to j as a Nogood and stored in j. In any case, if it is forbidden by, at least, one partial constraint, the pair (v, w) is forbidden by the total constraint c_{ij}. Therefore, it will be generated by ABT.

Let us assume that (v, w) is forbidden by c_{ij}, so in ABT the nogood $x_i = v \Rightarrow x_j \neq w$ will be generated by j when it receives the *ok*? message from i containing $x_i = v$. We know that the pair (v, w) will be forbidden by, at least, one of the partial constraints in the PKC model. If it is forbidden by $c_{(i)j}$, the nogood will be generated in j after receiving the **ok?** message containing $x_i = v$, and stored in j. If it is forbidden by $c_{i(j)}$, the nogood will be generated in i after i sends $x_i = v$ to j and j sending $x_j = w$ to i (this requires two phases in ABT-2ph but a single one in ABT-1ph). This nogood will be sent to j, and stored there. So, in both cases the nogood is generated (and stored in j).

Implicit nogoods (Proof by induction on the number of implicit nogoods in a sequence of backtracking steps). The first implicit nogood in the sequence that appears in ABT-2ph/1ph is generated by resolving explicit nogoods. Since all explicit nogoods of ABT-2ph/1ph can be generated by ABT, and the nogood resolution mechanism is the same, this first implicit nogood can also be generated by ABT. Let us assume that this is true up to the n-th implicit nogood generated by ABT-2ph/1ph, and let us prove that this is the case for the $n + 1$-th implicit nogood. Let us consider the $n + 1$-th implicit nogood generated. It has been computed by resolving previous nogoods, either explicit or implicit. We have already proved that explicit nogoods can also be generated by ABT. All previous (n) implicit nogoods can be generated by ABT by the induction step. Therefore, since the nogoods involved in the resolution can all be generated by ABT and the resolution process is the same, the resolvent nogood could also be generated by ABT. A similar argument holds in the other direction of the lemma, starting from ABT implicit nogoods and implying nogoods of ABT-2ph/1ph. □

Proposition 1 *ABT-2ph/1ph are sound, complete and terminate.*

Proof

Soundness If ABT-2ph/1ph report a solution, it is because the network has reached quiescence. In that case, every partial constraint is satisfied (otherwise, quiescence cannot be reached). If every partial constraint is satisfied, every total constraint is satisfied as well (by definition of partial constraints, see Section 4.1). Therefore, the reported solution is a true solution.

Completeness ABT-2ph/1ph perform the same kind of search as ABT: total ordering of agents, asynchronous instantiation, resolving nogoods, adding links, etc. Their only difference is that (1) agents send their assignments to higher and lower priority agents through **ok?** messages, and (2) if a higher priority agent receives one of these **ok?** messages with a value that is inconsistent with its current value, it sends a nogood message to the sender. These points are crucial in the generation of nogoods. But we know, by Lemma 1, that these changes do not cause any modification in the set of nogoods generated by these algorithms with respect to ABT. Consequently the ABT-2ph/1ph algorithms will discard the same parts of the search space as ABT, but not other parts. Since ABT is complete, ABT-2ph/1ph are also complete.

Termination An argument similar to the one used in completeness applies here. Nogoods rule out parts of the search space. Because of the total ordering of agents, discarded parts accumulate, so ABT terminates in a finite search space (see [1]). Since ABT-2ph/1ph generate the same nogoods as ABT, and they perform the same kind of search, ABT-2ph/1ph also terminate. □

5.4 An example

Let us consider the problem of safely locating a queen Q and a knight k on a 4×4 chessboard (see Section 4.1 for the formal definition of the n-pieces m-chessboard problem). Each piece is handled by an independent agent, and none knows the identity of the other piece, so the PKC model applies here. There is a single constraint between Q and k. Initially we assume that Q has higher priority than k, so the constraint is directed $Q \rightarrow k$. Let us follow the execution of ABT-2ph/1ph on this problem.

ABT-2ph works as follows. Phase I starts locating each piece in the first position of the chessboard. Afterwards, Q informs k of its current assignment. This causes k to take value 2, which is consistent according to its partial constraint with the value of Q. Then, quiescence is reached and phase II starts. The constraint direction is reversed, now $k \rightarrow Q$. k informs Q of its value. Q notes that its current value is not consistent with the value of k, so it sends a **ngd** message to k. Quiescence is reached and the constraint direction is reversed again to $Q \rightarrow k$. Phase I starts. k realizes that its value is eliminated by the *Nogood* contained in the **ngd** message received in phase II, so k changes its value to 3. Quiescence is reached and phase II starts. The constraint direction is reversed, now $k \rightarrow Q$. k informs Q of its value. Q finds that its current value is not consistent with the value 3 of k and then sends a **ngd** message to k. Quiescence is reached and the search is resumed in phase I. This process continues until k takes value 8. Then, k informs Q of the new assignment. Q checks that its assignment is consistent with k value. Quiescence is reached causing termination because no **ngd** message has been generated in phase II.

ABT-1ph works as follows. First, each piece takes value 1. Agents exchange two **ok?** messages. Q informs k that it has taken value 1, and k informs Q that it has taken value 1. When Q receives the **ok?** message, it sends a **ngd** message to k informing that its current value it is not consistent with the value of Q (both pieces are in the same chessboard cell). When k receives the **ok?** and **ngd** messages it takes value 2 and informs this change to Q via an **ok?** message. When Q receives this message it notes that its current value is not consistent with k value, so Q sends an **ngd** message to k informing it to change its value. Then, k takes the value 3 and informs Q. This process continues until k takes value 8. Then, it informs Q of the new assignment. Q finds that its current value is consistent with the value of k, so it does nothing. Quiescence is reached causing termination.

6 Distributed forward checking for assignment privacy

ABT-2ph/1ph reach a solution while keeping constraint privacy. However, they do not achieve assignment privacy because two constrained agents have to exchange their value assignments, in order to verify that their partial constraints are satisfied. Therefore, when a solution is found, an agent knows the value assigned to every agent constrained with it.

To enforce assignment privacy while keeping constraint privacy in the PKC model, we propose the *distributed forward checking* (DisFC) algorithm. It is inspired by the Forward Checking algorithm in the centralized case [8]. Considering two constrained agents i and j, $i < j$, the idea is, instead of i sending its assigned value to j, it sends the subset of D_j that is consistent with its assigned value. In addition, the current value assigned to each agent is replaced by the sequential number of assignments of each variable, as indicated in Section 4.2. As for ABT, we have two versions of this algorithm, with two phases or a single phase, which we term DisFC-2ph and DisFC-1ph respectively. Each of them works as its respective ABT counterpart, with the two differences mentioned above.

6.1 Two-phase/one-phase strategies

DisFC-2ph. It works like ABT-2ph, with the already mentioned differences: (1) instead of sending its current value assignment, agent i sends the subset of D_j consistent with it, and (2) the assigned value is replaced by a sequence number.

Let us consider DisFC-2ph and two constrained agents i, j, $i < j$. In phase I, the partial constraint $c_{i(j)}$ is tested by i, while in phase II $c_{(i)j}$ is tested by j. In ABT-2ph it happens exactly in the opposite order: in phase I $c_{(i)j}$ is tested by j and in phase II $c_{i(j)}$ is tested by i. This is due to the type of information sent (values in ABT-2ph, consistent sub-domains in DisFC-2ph) and to the partial constraint owned by each agent, but this is not a fundamental difference.

The DisFC-2ph algorithm appears in Fig. 4. Each agent stores assignments of higher priority agents in *myAgentView* and the received nogoods in *myNogood Store*. It uses the same types of messages as ABT-2ph.

DisFC-1ph. It works like ABT-1ph, with the already mentioned differences: (1) instead of sending its current value, agent i sends the subset of D_j consistent with

```
procedure DisFC-2ph()
  compute $\Gamma^-, \Gamma^+$; $myValue \leftarrow$ empty; $end \leftarrow$ false; $nogoods \leftarrow$ false;
  repeat
    DisFC-I();
    if ($\neg end$)
      exchange $\Gamma^-, \Gamma^+$;
      DisFC-II();
      exchange $\Gamma^-, \Gamma^+$;
  until $end$ or $\neg nogoods$

procedure DisFC-I()
  $quiescence \leftarrow$ false;
  CheckAgentView();
  while ($\neg end \wedge \neg quiescence$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?    : ProcessInfo($msg$);
      ngd    : ResolveConflict($msg$);
      adl    : SetLink($msg$);
      stp    : $end \leftarrow$ true;
      qes    : $quiescence \leftarrow$ true;

procedure ProcessInfo($msg$)
  UpdateAgentView($msg.Sender = msg.Seq$);
  UpdateDomain($msg$);
  CheckAgentView();

procedure ResolveConflict($msg$)
  if coherent($msg.Nogood, \Gamma^-(self) \cup \{self\}$) then
    CheckAddLink($msg$);
    add($msg.Nogood, myNogoodStore$); $myValue \leftarrow$ empty;
    CheckAgentView();
  else if coherent($msg.Nogood, self$) then
    SendMsg:ok?($msg.Sender, mySeq$, compatible($D(msg.Sender), myValue$));

procedure CheckAgentView()
  if ($myValue = empty \vee myValue$ eliminated by $myNogoodStore$) then
    $myValue \leftarrow$ ChooseValue();
    if ($myValue$) then
      $mySeq \leftarrow mySeq + 1$;
      for each $child \in \Gamma^+(self)$ do sendMsg:ok?($child, mySeq$, compatible($D(child), myValue$));
    else Backtrack();

procedure UpdateDomain($msg$)
  for each $v \in D(self) \wedge v \notin msg.Domain$ do
    add($msg.Sender = msg.Seq \Rightarrow self \neq v, myNogoodStore$);

procedure DisFC-II()
  $quiescence \leftarrow$ false;
  for each $child \in \Gamma^+(self)$ do sendMsg:ok?($child, mySeq$, compatible($D(child), myValue$));
  while ($\neg quiescence$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?    : if $myValue \notin msg.Domain$ then
                     sendMsg:ngd($self = mySeq \Rightarrow msg.Sender \neq msg.Seq$);
      ngd    : add($msg.Nogood, myNogoodStore$); $myValue \leftarrow$ empty;
      qes    : $quiescence \leftarrow$ true; $nogoods \leftarrow$ true; /* quiescence with nogoods messages */
      qnn    : $quiescence \leftarrow$ true; $nogoods \leftarrow$ false; /* quiescence without nogoods messages */
```

Fig. 4 The DisFC-2ph algorithm for the PKC model. Missing procedures appear in Fig. 1

it, and (2) the assigned value is replaced by a sequence number. The DisFC-1ph algorithm appears in Fig. 5. Beside the data structures of DisFC-2ph, each agent keeps in *myFilteredDomain*[*i*] the last filtered domain received from agent *i*, a lower priority constraining agent.

6.2 Formal properties

Regarding nogoods, the only difference between ABT-2ph/1ph and DisFC-2ph/DisFC-1ph is that actual values are replaced by sequence numbers. This is fine, as the only role of values/sequence numbers is to detect that an assignment is obsolete. However, it might occur that two different sequence numbers for one variable would represent the same value. If this happens after receiving a backtrack message, when comparing the agent view of the message with the agent view of the receiver, the message will be discarded as obsolete. But this will cause no problem. Since each time the sequence number changes, an **ok?** message is sent, if either the sender or the receiver of the backtrack message are not updated, it means that the message with

```
procedure DisFC-1ph()
  $myValue \leftarrow$ empty; $end \leftarrow$ false; compute $\Gamma^+, \Gamma^-$;
  CheckAgentView();
  while ($\neg end$) do
    $msg \leftarrow$ getMsg();
    switch($msg.type$)
      ok?    : ProcessInfo($msg$);
      ngd    : ResolveConflict($msg$);
      adl    : SetLink($msg$);
      stp    : $end \leftarrow$ true;

procedure ProcessInfo($msg$)
  UpdateAgentView($msg.Sender = msg.Seq$);
  if ($msg.Sender \in \Gamma^+(self)$) then $myFilteredDomain[msg.Sender] \leftarrow msg.Domain$;
  if ($msg.Sender \in \Gamma^-(self)$) then UpdateDomain($msg$);
  if $\neg(myValue \in msg.Domain)$ then
    if ($msg.Sender \in \Gamma^+(self)$) then
        SendMsg:ngd($msg.Sender, self = mySeq \Rightarrow msg.sender \neq msg.Seq$);
    else CheckAgentView();

procedure CheckAgentView()
  if ($myValue = empty \vee myValue$ eliminated by $myNogoodStore$) then
    $myValue \leftarrow$ ChooseValue();
    if ($myValue$) then
      $mySeq \leftarrow mySeq + 1$;
      for each $child \in \Gamma^+(self) \cup \Gamma^-(self)$ do
          sendMsg:ok?($child, mySeq$, compatible($D(child), myValue$));
      for each $child \in \Gamma^+(self)$ such that $\neg$ ($myValue \in myFilteredDomain[child]$) do
          sendMsg:ngd($child, self = mySeq \Rightarrow child \neq child.Seq$);
    else Backtrack();
```

Fig. 5 The DisFC-1ph algorithm for the PKC model. Missing procedures appear in Fig. 4

the most updated sequence number has not arrived yet, but it is on its way. After its arrival, the backtrack message will be accepted. After this clarification, we prove that the good theoretical properties of ABT-2ph/1ph also hold for DisFC-2ph/1ph.

Lemma 2 *A nogood can be generated by DisFC-2ph/1ph iff it can be generated by ABT.*

Proof Let us consider two constrained agents i, j, $i < j$. The only difference with ABT-2ph/1ph is that **ok?** messages may generate more than one nogood. In fact, an **ok?** message from i to j generates as many nogoods as values considered inconsistent in D_j. These nogoods would have been generated by ABT if these values would have been successively assigned to j. The rest of the proof is equal to the one of Lemma 1. □

Proposition 2 *DisFC-2ph/1ph are correct, complete and terminate.*

Proof Analogous to the proof of Proposition 1 □

6.3 An example

Let us consider the same problem of safely locating a queen Q and a knight k on a 4×4 chessboard we have discussed as an example for ABT-2ph/1ph. Each piece is handled by an independent agent, and none knows the identity of the other piece, so the PKC model applies here. There is a single constraint between Q and k. Initially, we assume that Q has higher priority than k. We show the execution of DisFC-2ph/1ph on this problem.

Considering DisFC-2ph, the algorithm execution is as follows (assuming that values are selected lexicographically). When phase I starts, both pieces take value 1. Q informs k of its filtered domain with respect to Q's assigned value, and k changes its assigned value to 7. Quiescence is reached and phase II starts. The constraint direction is reversed, now $k \rightarrow Q$. k informs Q of its filtered domain with respect to its assigned value. Q realizes that its current value assignment is forbidden, so it sends an **ngd** message to k. Quiescence is reached and the constraint direction is reversed again, now $Q \rightarrow k$. Phase I starts. k realizes that its assigned value is eliminated by the nogood contained in the **ngd** message received in phase II. Therefore, k changes its assignment to 8. Quiescence is reached and phase II starts. The constraint direction is reversed, now $k \rightarrow Q$. k informs Q of its filtered domain with respect to its assigned value. Q finds that its current assignment is consistent, so it does nothing. Quiescence is reached causing termination because no **ngd** message has been generated in phase II.

When DisFC-1ph starts, both pieces take value 1. Agents exchange two **ok?** messages. Q informs k of its filtered domain with respect to its assigned value, and k informs Q of its filtered domain with respect to its own assignment. When Q receives the message, it sends a **ngd** message to k informing that its current value assignment is not consistent with Q's assignment. When k receives the **ok?** and **ngd** messages it assigns value 7 and informs this change to Q via an **ok?** message. When Q receives this message it notes that its current assignment is not consistent, so Q sends an **ngd** message to k informing that it has to change its value assignment. Then, k assigns value 8 and informs Q. Q finds that its current assignment is consistent, so it does nothing. Quiescence is reached causing termination.

7 Entropy as a privacy measure

Given a discrete random variable Z which takes values in the discrete set S, its entropy is defined as,

$$H(Z) = -\sum_{i \in S} p_i \ log_2 \ p_i$$

where p_i is the probability that Z takes the value i. $H(Z)$ measures the amount of missing information about the possible values of the random variable Z [5, 17]. If only one value k is possible for that variable (that is, $p_i = 0, \forall i \neq k, p_k = 1$), there is no uncertainty about the state of Z, and $H(Z) = 0$. Entropy is a non-negative magnitude.

Given a DisCSP, defined by $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{A}, \phi)$, let us consider agent j. The rest of the problem can be considered as a random variable, with a discrete set of possible states S. Applying the concept of information entropy, we define the entropy associated with agent j as,

$$H_j = -\sum_{i \in S} p_i log_2 \ p_i$$

and the entropy associated with the whole problem as,

$$H = \sum_{j \in A} H_j$$

Let us consider two imaginary states α and β of the whole problem. In state α, every agent knows nothing about the values of other agents, and knows the minimum about their constraints. In this state, privacy is at maximum, because each agent knows the minimum on the rest of the problem. It happens that entropy is also at maximum, because the missing information each agent has about the rest of the agents is maximal. Let us now consider state β, in which every agent knows everything about the rest of the problem. Obviously, in this case there is no privacy at all. It happens that the entropy of this state has its minimum value (zero) because there is no uncertainty on the state of any problem element. Given the relationship between privacy and entropy (coincidence at points of maxima and minima), and the good properties of entropy (continuity, symmetry, additivity), we propose to use entropy as a quantitative measure of privacy.

Assuming that the initial state of a DisCSP has maximum privacy (entropy), any solving process can be seen as an entropy-decreasing process. Entropy decrement is due to two factors: (1) if a global solution is finally reached, the solution condition decreases the uncertainty about the values assigned to agents, so the entropy of the whole problem should decrease, and (2) in the solving process some information about values or constraints of neighboring agents is leaked, increasing the knowledge of particular agents; therefore, their uncertainty (and entropy) decrease.

Following our proposal of entropy as quantitative measure for privacy, we suggest to take the entropy decrement in the solving process as a measure of privacy loss. This allows us to compare different algorithms with respect to privacy. We assume that every algorithm starts from the same initial state, where privacy (entropy) is at maximum. Entropy decrement depends on the solving algorithm. Comparing the entropy of the final states achieved by the different algorithms, we can compare about their relative performance with respect to privacy loss.

In this paper, we discuss privacy on two dimensions: assignment privacy and constraint privacy. This difference can be captured in our entropy approach by considering that the part of the problem considered is limited to the assignments of other agents (assignment privacy) or to their constraints (constraint privacy). Both approaches are further developed in the following.

7.1 Assignment privacy

To measure assignment privacy we use entropy limited to the assignments of other agents, for each agent *j*'s. Initially, an ABT-agent *j* knows the total number of agents *n*, the relative position of any agent with respect to *j* position in the ordering, the agents connected with it and above in the ordering Γ_j^-, the agents connected with it and below in the ordering Γ_j^+, its own value val_j and its own partial constraints $c_{j(l)}, l \in \Gamma_j^- \cup \Gamma_j^+$. We assume a common domain size *d*, and agent identifiers follow the total ordering. In the initial state, called *init*, the entropy associated with agent *j* is (using the additive property of entropy [5]),

$$H_j(init) = -\sum_{k=1,k\neq j}^{n} \sum_{i=1}^{d} p_i \log_2 p_i = -\sum_{k=1,k\neq j}^{n} d\,\frac{1}{d}\log_2\frac{1}{d} = \sum_{k=1,k\neq j}^{n} \log_2 d$$

where we assume that the *d* values for agent *k* have the same probability, $p_i = \frac{1}{d}\ \forall i$. For convenience, we rearrange this expression as follows,

$$H_j(init) = \sum_{k\in\Gamma_j^+} \log_2 d + \sum_{k>j,k\notin\Gamma_j^+} \log_2 d + \sum_{k\in\Gamma_j^-} \log_2 d + \sum_{k<j,k\notin\Gamma_j^-} \log_2 d \qquad (2)$$

where the first two terms correspond to agents below *j* (constrained and not constrained with *j*) and the last two terms correspond to agents above *j* in the ordering (again, constrained and not constrained with *j*). After finding a solution in the state called *sol*, the entropy of agent *j* depends on the algorithm used, as follows.

- ABT. Let us consider an agent *k* below *j* and directly constrained with it. Agent *j* does not know the value of *k*, val_k, but since it belongs to a solution, it must be consistent with its own value val_j. So val_k must be consistent with val_j in the partially known constraint $c_{j(k)}$. Assuming that there are $n_{j(k)}$ values consistent with val_j, the contribution of *k* to the entropy of *j* in *sol* is $\log_2 n_{j(k)}$. Therefore, the contribution of all agents constrained with *j* and below it in the ordering is

$$\sum_{k\in\Gamma_j^+} \log_2 n_{j(k)}$$

The second term of (2) does not change. Let us consider an agent i above j in the ordering and constrained with j. Agent j knows its value so its contribution to H_j is zero. Since this is true for any constrained agent above j, the third term of (2) becomes zero. The fourth term does not change. Therefore, the entropy associated with j restricted to assignments of other agents in the *sol* states is,

$$H_j^{ABT}(sol) = \sum_{k \in \Gamma_j^+} \log_2 n_{j(k)} + \sum_{k > j, k \notin \Gamma_j^+} \log_2 d + 0 + \sum_{k < j, k \notin \Gamma_j^-} \log_2 d$$

It is worth noting that new links may appear during the solving process, changing the sets Γ_j^- and Γ_j^+ with respect to their initial state. If a new link appears from i to j, $i < j$, the entropy associated with j decreases since the contribution of i in *init*, $\log_2 d$, goes to zero in *sol*.

- ABT-2ph/1ph. Let us consider agents i and k, constrained with j, $i < j < k$. Agent j knows the values of i and k, so it has no uncertainty about their values. Therefore, their contribution to entropy is zero. This argument applies for any agent constrained with j, so the first and third terms of (2) are zero. Therefore, the entropy associated with j restricted to assignments in the *sol* state is,

$$H_j^{ABT-2ph/1ph}(sol) = 0 + \sum_{k > j, k \notin \Gamma_j^+} \log_2 d + 0 + \sum_{k < j, k \notin \Gamma_j^-} \log_2 d$$

- DisFC-2ph/1ph. Let us consider agents i and k, constrained with j, $i < j < k$. Agent j does not know their values, but since they form a solution, their values must be consistent with the value of j in the partial constraints $c_{j(i)}$ and $c_{j(k)}$ (the argument used for ABT applies here). Assuming that there are $n_{j(k)}$ values consistent with val_j in $c_{j(k)}$, the entropy associated with j restricted to assignments in the *sol* state is,

$$H_j^{DisFC-2ph/1ph}(sol) = \sum_{k \in \Gamma_j^+} \log_2 n_{j(k)} + \sum_{k > j, k \notin \Gamma_j^+} \log_2 d + \sum_{k \in \Gamma_j^-} \log_2 n_{j(k)} + \sum_{k < j, k \notin \Gamma_j^-} \log_2 d$$

where the second and fourth terms of (2) remain unchanged.

These results allow us to rank these algorithms with respect to assignment privacy.

Proposition 3 *Regarding assignment privacy for ABT, ABT-2ph/1ph and DsFC-2ph/1ph, assuming that the sets Γ_j^- and Γ_j^+ are equal for the three algorithms in the sol state,*

- *The privacy loss of ABT-2ph/1ph is higher than or equal to the privacy loss of ABT,*
- *The privacy loss of ABT is higher than or equal to the privacy loss of DisFC-2ph/1ph.*

Proof Since these algorithms are complete, they will finish their execution finding a solution or proving that no solution exists. In case of no solution, there is no global assignment so it is meaningless to talk about assignment privacy. In case of solution, this proposition follows directly from the above results on entropy on these algorithms. Since entropy of the initial state is the same for the three algorithms

(2), we can compare the entropy of the *sol* state for the three algorithms [17]. Thus, we have,

$$H_j^{ABT}(sol) - H_j^{ABT-2ph/1p}(sol) = \sum_{k \in \Gamma_j^+} \log_2 \ n_{j(k)} \geq 0$$

$$H_j^{DisFC-2ph/1p}(sol) - H_j^{ABT}(sol) = \sum_{k \in \Gamma_j^-} \log_2 \ n_{j(k)} \geq 0$$

where subtraction is easily done because sets Γ_j^- and Γ_j^+ are the same for the three algorithms. From these expressions, we see that the entropy in the *sol* state of agent j is higher (or equal to in the limit case) when using ABT than when using ABT-2ph/1ph, and similar result occur between DisFC-2ph/1ph and ABT. Since this is computed for any agent j, we conclude that the *assignment* privacy loss of ABT-2ph/1ph is higher than or equal to the *assignment* privacy loss of ABT, and the *assignment* privacy loss of ABT is higher than or equal to the *assignment* privacy loss of DisFC-2ph/1ph. □

7.2 Constraint privacy

To measure constraint privacy we use entropy limited to the constraints involving j and other agents. In the initial state defined above (*init* state), the entropy associated with agent j is (using the additive property of entropy [5]),

$$H_j(init) = -\sum_{k=1, k \neq j}^{n} \sum_i p_i \ \log_2 \ p_i = -\sum_{k \in \Gamma_j^+} \sum_i p_i \ \log_2 \ p_i - \sum_{k \in \Gamma_j^-} \sum_i p_i \ \log_2 \ p_i$$

where the contribution of agents not constrained with j is zero (they are connected with j by the universal constraint that allows every tuple, so there is no uncertainty associated with these constraints). We are interested in knowing what agent j can infer about c_{jk}. Agent j knows that every zero entry (pair of incompatible values) in $c_{j(k)}$ will be a zero entry in c_{jk}. The number of matrices representing constraint c_{jk} compatible with $c_{j(k)}$ is $2^{d^2 - z_{j(k)}}$, where $z_{j(k)}$ is the number of zero entries in the matrix corresponding to $c_{j(k)}$. The probability for each of the possible states of agent k, with respect to its constraints with agent j, is constant and equals $\frac{1}{2^{d^2 - z_{j(k)}}}$. Since the number of possible states for agent k is $2^{d^2 - z_{j(k)}}$, the fixed probability can be taken out of the sum of the entropy for agent k and is summed up to 1. The contribution of each agent k is,

$$-\sum_i p_i \ \log_2 \ p_i = -2^{d^2 - z_{j(k)}} \frac{1}{2^{d^2 - z_{j(k)}}} \ \log_2 \ \frac{1}{2^{d^2 - z_{j(k)}}} = \log_2 \ 2^{d^2 - z_{j(k)}} = d^2 - z_{j(k)}$$

and the entropy of the *init* state associated with j is,

$$H_j(init) = \sum_{k \in \Gamma_j^+} d^2 - z_{j(k)} + \sum_{k \in \Gamma_j^-} d^2 - z_{j(k)} \tag{3}$$

When the solving process finishes, no matter whether a solution has been found or not, the system is in the *end* state. The entropy decrement depends on the solving algorithm, as follows.

- ABT. Let us consider an agent k above j and constrained with it. In order to run ABT agent j has to *know completely* c_{ij}, so there is no uncertainty about it. Since this argument applies to any agent above j and constrained with it, the second term of (3) becomes zero. Then, the entropy is,

$$H_j^{ABT}(end) = \sum_{k \in \Gamma_j^+} d^2 - z_{j(k)}$$

  How much is $z_{j(k)}$? This depends on the particular problem considered. We have evaluated the entropy of the *end* state empirically, in Section 8.
- ABT-2ph. Both terms of (3) depend on the execution of the particular algorithm. In ABT-2ph, an agent i above j may reveal to j some entries in $c_{i(j)}$. Combining this information with $c_{j(i)}$, j knows e'_{ij} entries of c_{ij}, $e'_{ij} \geq z_{j(i)}$. Analogously with agent k below j. After execution agent j may know e''_{jk} entries of c_{jk}, $e''_{jk} \geq z_{j(k)}$. Then, the entropy is,

$$H_j^{ABT-2ph}(end) = \sum_{k \in \Gamma_j^-} d^2 - e'_{jk} + \sum_{k \in \Gamma_j^+} d^2 - e''_{jk}$$

  e'_{jk} and e''_{jk} depend on particular executions, so we do not have their analytical expressions. We have evaluated the entropy of the *end* state empirically, in Section 8.
- ABT-1ph. The first term of (3) depends on the particular algorithm execution, in the same way of ABT-2ph. However, with agent k below j, after execution agent j cannot infer further entries in c_{jk}. Therefore, the entropy is,

$$H_j^{ABT-1ph}(end) = \sum_{k \in \Gamma_j^-} d^2 - e'_{jk} + \sum_{k \in \Gamma_j^+} d^2 - z_{j(k)}$$

  e'_{jk} depends on particular executions, so we do not have its analytical expression. We have evaluated the entropy of the *end* state empirically, in the Section 8.
- DisFC-2ph/1ph. Here it is a bit more complex to compute the entropy associated with the *end* state. Let us consider agent i above j and constrained with it. We have to compute $\#mc_{i(j)}$, the number of matrices which are compatible with the information exchanged between i and j during the solving process. One of these matrices is $c_{i(j)}$. Combining this information with $c_{j(i)}$, agent j can obtain the matrix c_{ij}. The entropy corresponding to the uncertainty of $c_{i(j)}$ is,

$$-\sum_i p_i \ \log_2 \ p_i = -\#mc_{i(j)} \frac{1}{\#mc_{i(j)}} \log_2 \frac{1}{\#mc_{i(j)}} = \log_2 \ \#mc_{i(j)}$$

  and the entropy of the *end* state is,

$$H_j^{DisFC-2ph/1ph}(end) = \sum_{i \in \Gamma^- \cup \Gamma^+} \log_2 \ \#mc_{i(j)}$$

Again, we do not have an analytical expression for $\#mc_{i(j)}$. The entropy expression is evaluated experimentally in Section 8.

Parameters e'_{jk}, e''_{jk} and $\#mc_{i(j)}$, appearing in the previous expressions, are computed as follows:

- e'_{jk}, e''_{jk}(ABT-2ph/1ph), (agents $i < j < k$):
  - *Explicit nogood in ABT-2ph/1ph.* If j receives an explicit **ngd** message from an agent i, then j may deduce that the entry corresponding to the combination of current values of i and j in $c_{i(j)}$ is forbidden, and therefore, j may deduce that this entry in c_{ij} is also forbidden.
  - *Explicit good in ABT-2ph.* If during the second phase of ABT-2ph, j does not receive an explicit **ngd** message from an agent i, then j may deduce that the entry corresponding to the combination of current values of i and j in $c_{i(j)}$ is permitted, and therefore, j may deduce that this entry in c_{ij} is equal to this entry in $c_{(i)j}$.
  - *Explicit good in ABT-2ph.* If during the second phase of ABT-2ph, j receives an **ok?** message from an agent k, then j may deduce the entry corresponding to the combination of current values of j and k in $c_{(j)k}$ is permitted, and therefore, j may deduce that this entry in c_{jk} is equal to this entry in $c_{j(k)}$.
- $\#mc_{i(j)}$ (DisFC-2ph/1ph). Let us consider a constraint as a $d \times d$, 0/1 matrix. In DisFC-2ph/1ph, agent i sends rows of $c_{i(j)}$ to j; at the end j has a subset of rows without knowing their position in $c_{i(j)}$. In addition, some search episodes (information exchanged by agents in phase II in DisFC-2ph, nogood messages from high to low priority agents in DisFC-1ph) may reduce the number of acceptable positions for a particular row. To assess the amount of information revealed, we compute the number of matrices that are compatible with the exchanged rows. With this aim, we construct a CSP instance where the variables are the rows, their domains are the acceptable positions, under the constraints that two different rows cannot go to the same position and every row must get a position. Computing all solutions of this instance (an NP-hard task) we obtain all compatible matrices. One of these matrices represents $c_{i(j)}$.
  Figure 6 presents an example of DisFC-1ph execution that illustrates when and what kind of information about constraints that agents can infer after message

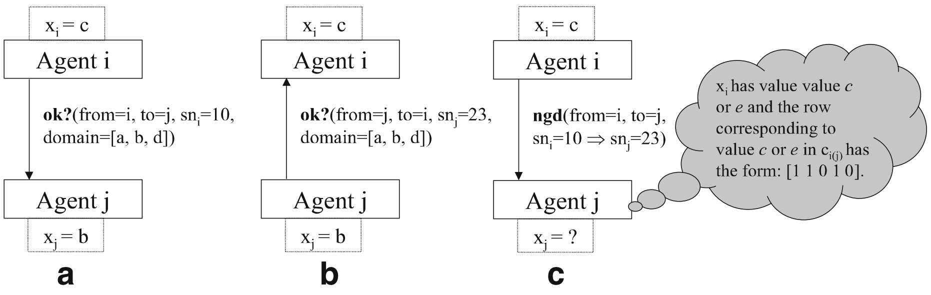

x_i = c
Agent i
ok?(from=i, to=j, sn_i=10, domain=[a, b, d])
Agent j
x_j = b
a
x_i = c
Agent i
ok?(from=j, to=i, sn_j=23, domain=[a, b, d])
Agent j
x_j = b
b
x_i = c
Agent i
ngd(from=i, to=j, sn_i=10 ⇒ sn_j=23)
Agent j
x_j = ?
x_i has value value c or e and the row corresponding to value c or e in c_i(j) has the form: [1 1 0 1 0].
c

Fig. 6 Information deduced by a DisFC-1ph agent after receiving a **ngd** message from a higher priority agent

reception. The example consists of two agents i and j, each having one variable with domain: $\{a, b, c, d, e\}$. We look at each constraint as a matrix in which every entry represents the compatibility of two values, assuming that values are lexicographically ordered. An entry with 0 or 1 indicates that the corresponding value pair is forbidden or permitted, respectively.

The example starts when agent i sends an **ok?** message to agent j saying that agent i's current value is compatible only with the values $\{a, b, d\}$ of agent j's domain (Fig. 6a). From this message, agent j may deduce that there is a row in $c_{i(j)}$ with the form: [11010]. However, with this information agent j cannot infer the position of this row in $c_{i(j)}$. After receiving the **ok?** message, agent j takes a new value and sends an **ok?** message to agent i saying that the only permitted values for agent i are in the domain $\{a, b, d\}$ (Fig. 6b). Similar to agent j, agent i may deduce that $c_{(i)j}$ has a row with the form [11010]. When agent i receives the **ok?** message from j, it discovers that the received message is incompatible with its own value and sends a **ngd** to agent j (Fig. 6c). From this message, agent j can deduce that agent i's current value is not included in the compatible domain that j just sent to i. Therefore, the current value of agent i is either c or e. Thus, agent j may deduce that the row corresponding to c or e in $c_{i(j)}$ has the form [11010] (i.e. the value c or e for agent i are compatible only with a, b and d for agent j). Next, agent j changes its value and sends a new **ok?** message, with the compatible values $\{a, b, e\}$. Agent i answers with a **ngd** message, and agent j discovers that $c_{i(j)}$ has a row with the form [11010] that corresponds to value c or d.

In the above example, the CSP that agent j constructs includes two variables x_1 and x_2, one for each time the row [11010] has been discovered. The domains of these variables are: $\{c, e\}$ and $\{c, d\}$, respectively. There exists a constraint between variables to avoid that both rows be associated to value c. The CSP has 4 solutions: $s_1 = \{x_1 = x_2 = c\}$; $s_2 = \{x_1 = c, x_2 = d\}$; $s_3 = \{x_1 = e, x_2 = c\}$; $s_4 = \{x_1 = e, x_2 = d\}$. The number of consistent matrices with $c_{(i)j}$ is: $2^{(25-5)} + 2^{(25-10)} + 2^{(25-10)} + 2^{(25-10)} - rep = 2^{(25-5)} + 3 \times 2^{(25-10)} - rep$. Each term expresses the number of consistent matrices related to each solution of the CSP. The first represents s_1; the second, s_2 and so on. The interpretation of s_1 is that both identical rows correspond to the same value c. That is to say, only one row of $c_{(i)j}$ has been revealed (5 entries out of 25) and the rest have not (25 - 5 entries out of 25). Then, there exist 2^{25-5} matrices consistent with the information derived from s_1. Following the same analysis, one can obtain the terms of the formula related to the solutions s_2, s_3 and s_4, where 10 entries out of 25 are revealed. By substracting the term rep from the first four terms, one avoids counting one matrix more than once. This, $rep = 2^{(25-5)} + 2 \times 2^{(25-10)}$, because the matrices for s_2 and s_3 are included in the matrices for s_1, and as well as matrices for s_1 are included in the matrices for s_4. Hence, according to the information that j has at the end of the search, the original $c_{i(j)}$ could be one of those $2^{15} = 32768$ matrices.

In DisFC-2ph, the process that agent j follows to reconstruct $c_{i(j)}$ is the almost same as for DisFC-1ph with some minor differences. Similar to ABT-2ph, all inferences are done in **phase II**. When agent j receives a **ngd** message from a higher priority agent, the process is the same as for DisFC-1ph. In addition, if after agent j has sent an **ok?** message to a higher priority agent i, agent j does not receive a **ngd** message from i, this means that i's value appears in the domains sent by j to i in the **ok?** message. Thus, the rows previously sent from agent i to agent j correspond to one of the values that appear in the domain that was included in the **ok?** message sent from agent j to agent i.

In any case, after agent j resolves the associated CSP and it identifies the matrices that are consistent with $c_{i(j)}$, it must combine each of them with the matrix that represents $c_{(i)j}$ following (1) (Section 4.1), to compute the matrices that are consistent with the total constraint c_{ij}.

The above results allow us to rank ABT and ABT-1ph with respect to constraint privacy [the other algorithms are not ranked because they depend on parameters e'_{jk}, e''_{jk}, $\#mc_{i(j)}$ which are not comparable with $z_j(k)$].

Proposition 4 *The privacy loss of ABT-1ph is lower than or equal to the privacy loss of ABT.*

Proof Analogous to the proof of Proposition 3 but replacing the *sol* state by the *end* state and performing substraction between $H_j^{ABT-1ph}$ and H_j^{ABT}. □

8 Experimental evaluation

The common approach in evaluating the efficiency performance of distributed algorithms is to compute two independent measures: search effort, in terms of computation steps [10, 23], and communication load, in terms of the total number of exchanged messages (*msg*) [10].

Nonconcurrent steps of computation are counted by a method similar to the logical clocks of Lamport [9]. Every agent holds a counter of computation steps. Each message carries the value of the sending agent's counter. When an agent receives a message it stores the data received together with the corresponding counter. When the agent first uses the received message, it updates its counter to the largest value between its own counter and the stored counter value which was carried by the message [28]. By reporting the cost of the search as the largest counter held by some agent at the end of the search, a measure of non-concurrent search effort that is close to Lamport's logical time is achieved [9]. If instead of steps of computation, the number of constraint checks is counted, then the local computational effort of agents is measured as the number of non-concurrent constraint checks (*NCCC*s) [13, 28].

In this study, besides efficiency, we are interested in empirically evaluating the assignment and constraint privacy that ABT and the proposed PKC algorithms achieve. As we have discussed in Section 7, we use the entropy of agents to measure each privacy type. Since each agent's entropy in the *init* state is the same for every algorithm, we restrict the privacy analysis to compare the algorithms according to the entropies that agents have in the *sol* and *end* states as measures of assignment and constraint privacy, respectively. In the experiments, we report for each algorithm and privacy type a *global entropy* value, which simply is the sum of entropy values of the agents when the algorithm ends. Larger values of global entropy correspond to higher privacy and lower privacy loss.

Experiments are performed on random constraint networks of n variables, k values in each domain, a constraint density p_1 and tightness of each constraint p_2

(which are commonly used in experimental evaluations of CSP algorithms [16, 20]). The generated constraint networks included 15 agents ($n = 15$) each holding one variable, 10 values for each variable ($k = 10$) and two values of constraints density $p_1 = 0.4$ and $p_1 = 0.7$. The tightness p_2 varies between 0.1 and 0.9, to cover all ranges of problem difficulty. For each pair of density and tightness (p_1, p_2), 100 different instances are solved by each algorithm and results are averaged over these 100 runs.

8.1 Efficiency evaluation

Figure 7 (top) presents the number of nonconcurrent constraints checks to find a solution for ABT, ABT-1ph, ABT-2ph, DisFC-1ph and DisFC-2ph on random instances with $p_1 = 0.4$. The less efficient algorithms are DisFC-1ph and DisFC-2ph, algorithms that achieve some kind of assignment privacy. Considering ABT-2ph and ABT-1ph, they both run more than twice slower than standard ABT. An interesting difference is between the two-phase and single-phase version. For problems in the phase transition region, the single-phase version outperforms the double phase version. On instances with high tightness, the single-phase version behaves like the

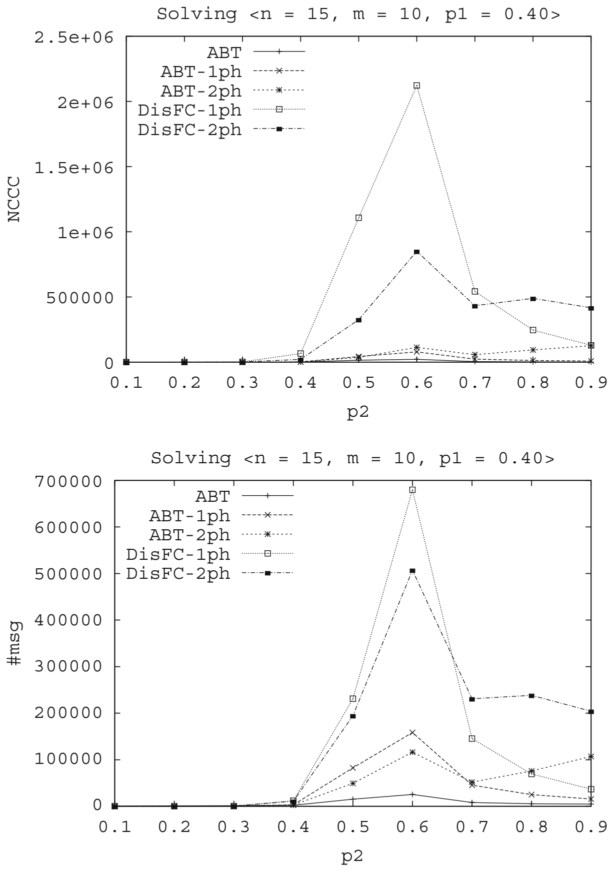


Fig. 7 Number of nonconcurrent constraints checks (*top*) and exchanged messages (*bottom*) performed by ABT, ABT-1ph, ABT-2ph, DisFC-1ph and DisFC-2ph when solving PKC DisCSPs with $p_1 = 0.4$

standard algorithm (i.e. the difference between the algorithms is constant) while the performance of the two-phase version deteriorates. To understand this behavior one must keep in mind that the problem solved by the first phase in the two-phase algorithm is actually less tight than the problem solved by the single-phase algorithm. Therefore when the single-phase algorithm detects that the problem is too tight to be solved, the two-phase algorithm works hard to solve a problem with lower tightness.

Considering DisFC-2ph and DisFC-1ph, we observe that DisFC-1ph is more than twice slower than DisFC-2ph for problems close to the complexity peak. However, DisFC-2ph is worse than DisFC-1ph for tighter problems ($p_2 \geq 0.7$). These results differ from those obtained for ABT versions where ABT-1ph is faster than ABT-2ph for instances close to the complexity peak. This is explained by the following fact. Before sending an **ok?** message, a DisFC-1ph agent has to check consistency with each value in the domain of every agent constrained with it. Conversely, a DisFC-2ph agent has to check consistency with every lower priority agent constrained with it, which generates a lower number of constraint checks.

Figure 7 (bottom) presents the results for the total number of exchanged messages among agents during algorithm execution. Again, DisFC algorithms require a number of messages substantially larger than the other algorithms. In contrast to the run-time measure, for low tightness instances, the number of messages sent by ABT-2ph is smaller than for ABT-1ph. This is because the agents in the single-phase algorithm send **ok?** messages to all their neighbors while in the two-phase algorithm in each phase the agents send **ok?** messages only in one direction. However, for tighter problems, the single-phase algorithm sends less messages than the two-phase algorithm when solving instances to the right of the complexity peak. Considering DisFC versions, the relative ordering of algorithms is, mainly, the same as that shown in the ABT versions. DisFC-1ph agents exchange more messages than DisFC-2ph for problems with constraint tightness lower than 0.6 ($p_2 \leq 0.6$). Although, DisFC-2ph is more costly than DisFC-1ph for the rest of the problems ($p_2 \geq 0.7$).

Comparing these results with the ABT-2ph/1ph, we see that DisFC algorithms are much slower. Similarly, agents in DisFC algorithms send more messages. This inefficiency of DisFC algorithms can be explained. Regarding *NCCC*, in ABT-2ph/1ph as in standard ABT, assignments are sent to neighboring agents which concurrently check their consistency with the local assignments. In DisFC, in order to keep the assignments private, agents must perform the consistency checks of their proposed assignments sequentially, checking the entire domains of their neighboring agents. This increases the non-concurrent effort of DisFC algorithms. Regarding *msg*, we observe that DisFC algorithms are more costly than ABT versions basically because two facts. First, DisFC agents exchange sequence numbers instead of their assignments. Sequence numbers are used to detect the obsolescence of nogoods messages. Since two different sequence numbers used by an agent may represent the same variable valuation, DisFC algorithms may temporally discard nogood messages as obsolete that are actually valid if considering the agents' assignments (like it happens in ABT versions). Second, an **ok?** message may contain several conflicts in DisFC while at most one in ABT algorithms. Thus, some updated conflicts may be discarded in DisFC due to storage limitation of nogoods (one conflict as justification of each forbidden value in all the considered algorithms) before they have been resolved. In such case, DisFC algorithms will rediscover and resent these conflicts, which results in an increase in the number of exchanged messages. The combination of the above two facts causes DisFC algorithms to exchange more messages than ABT versions.

In Fig. 8 we report the number of non-concurrent constraint checks (top) and exchanged messages (bottom) for ABT, ABT-1ph, ABT-2ph DisFC-1ph and DisFC-2ph when solving high density problems. Similar results to those of low density instances appear here. The PKC algorithms are much more costly than ABT. Regarding nonconcurrent constraint checks, ABT-2ph is much worse than ABT-1ph on the right of the peak (the same phenomenon already observed for low density instances occurs here). The relative order of algorithms remains unchanged for the total number of exchanged messages: ABT-1ph sends more messages than ABT-2ph for low tightness instances, while it sends less messages for high tightness instances. Considering DisFC versions, the relative order in the algorithmic performance remains unchanged for both parameters. DisFC-1ph is almost four times slower and sends more messages than DisFC-2ph for harder problems (i.e. close to the complexity peak). However, DisFC-1ph outperforms DisFC-2ph, mainly in communication cost, for problems on the right of the complexity peak (i.e. unsolvable problems). Comparing these algorithms with ABT-2ph/1ph, the DisFC algorithms have to perform a much larger number of constraint checks (because

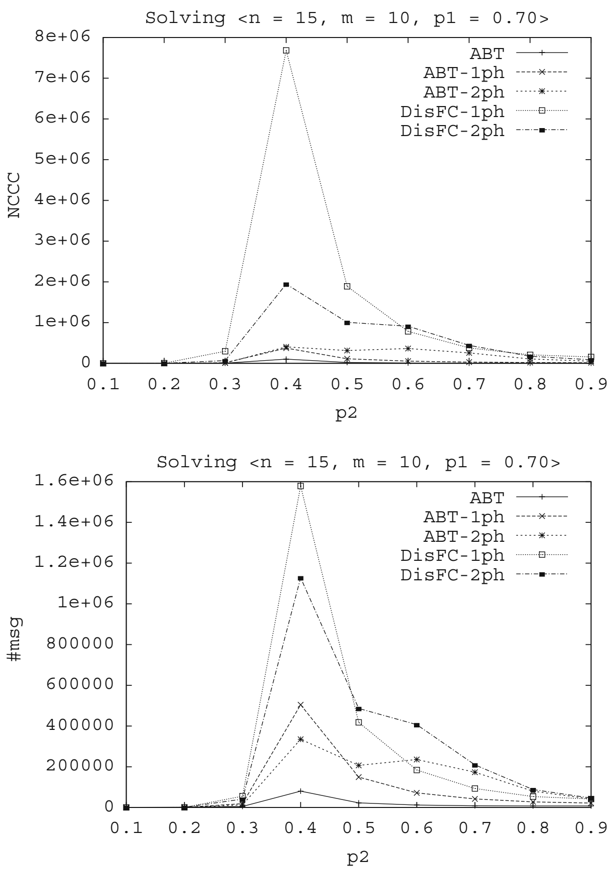


Fig. 8 Number of nonconcurrent constraints checks (*top*) and exchanged messages (*bottom*) performed by ABT, ABT-1ph, ABT-2ph, DisFC-1ph and DisFC-2ph when solving PKC DisCSPs with $p_1 = 0.7$

they send filtered domains, not just single values) and exchange more messages (as consequence of exchanging of sequence numbers instead of assignments) than the latter, which justify these results.

8.2 Privacy evaluation

In Fig. 9 we present the results of assignment privacy in terms of global entropy for ABT, ABT-2ph/1ph and DisFC2ph/1ph when solving low and high density random instances. For $p_1 = 0.4$ (top), DisFC-2ph/1ph and ABT-2ph/1ph show the largest and smallest values of global entropy, respectively. In terms of privacy this means that, on solvable instances, DisFC algorithms keep agents' final assignments more private than ABT and ABT-2ph/1ph. For $p_1 = 0.7$ (bottom), the results bring out the same conclusion as for low density instances: DisFC-2ph/1ph offer higher final assignment privacy than ABT, while ABT-2ph/1ph reveal more.

Somehow, these results were advanced in Proposition 3 and they are due to the following issues. During the search, agents in DisFC algorithms exchange sequence numbers instead of assignments in order to hide agents' valuations. Thus, at the end of the search, an agent can only make inferences about other agents' assignments

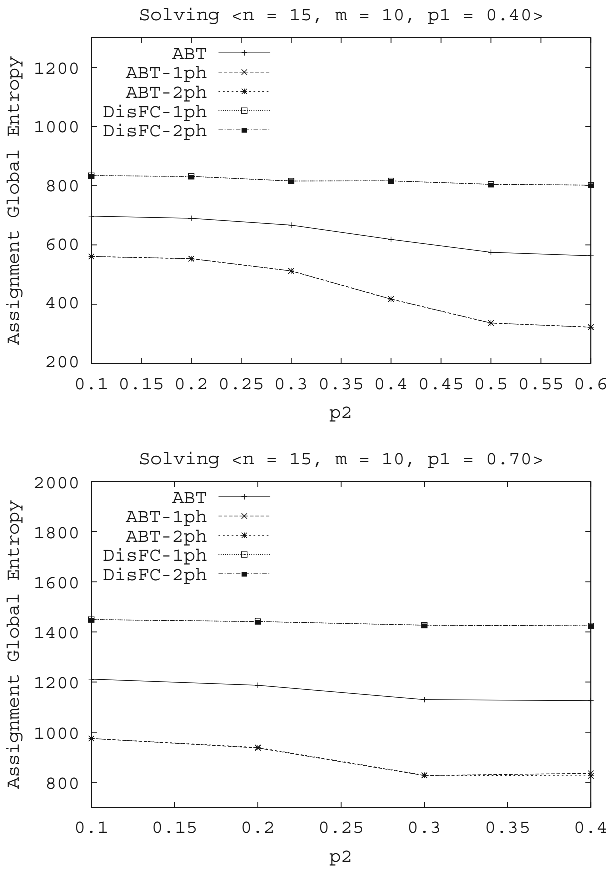
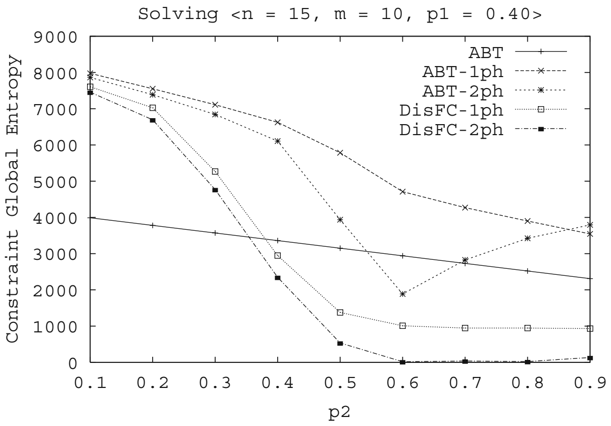


Fig. 9 Assignment privacy in terms of entropy for ABT, ABT-2ph/1ph, DisFC-2ph/1ph for $p_1 = 0.4$ (*top*) and $p_1 = 0.7$ (*bottom*)

based on the information contained in the partial constraints that the agent knows. On the other hand, and similar to ABT, agents in ABT-2ph/1ph explicitly reveal their assignments in **ok?** messages. Nevertheless, the number of agents that know other agents' final assignments in ABT-2ph/1ph is larger than in ABT because **ok?** messages goes from higher to lower priority agents in the standard algorithm while they travel in both directions, from higher to lower priority agents and vice versa, in the new versions of the algorithm.

In Fig. 10, we report the results of constraint privacy in terms of global entropy for ABT, ABT-2ph/1ph and DisFC-2ph/1ph when solving low and high density random instances. For $p_1 = 0.4$ (top), we observe that ABT-1ph always has larger values of global entropy than ABT. This result is supported by Proposition 4, where we have proven that the privacy loss of ABT is higher than or equal to the privacy loss of ABT-1ph.

In contrast, ABT-2ph may reveal more information about total constraints than ABT for some random instances. In the plot we observe this phenomenon at the complexity peak ($p_2 = 0.6$), where ABT has higher values of global entropy than the two-phase algorithm. To understand this behavior we must remind that agents in



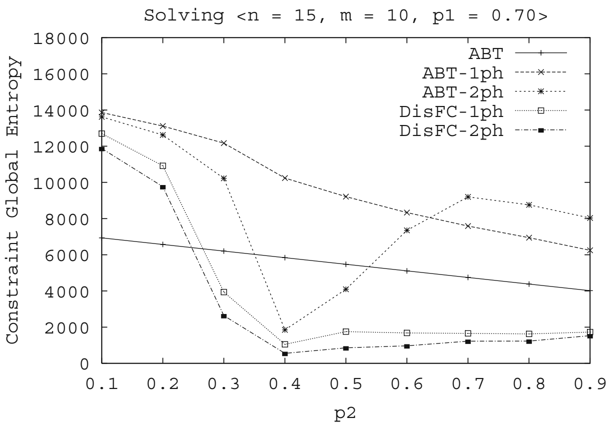


Fig. 10 Constraint privacy in terms of entropy for ABT, ABT-2ph/1ph and DisFC-2ph/1ph for $p_1 = 0.4$ (*top*) and $p_1 = 0.7$ (*bottom*)

the two-phase algorithm reveal only explicit information in phase II. On the left of the peak ($p_2 \leq 0.5$), most of the instances are solvable and the number of times that ABT-2ph passes to phase II is small. On the right of the peak ($p_2 \geq 0.7$), most of the problems have no solution, and, as soon as p_2 increases, the first phase of ABT-2ph becomes enough to detect inconsistency. At the complexity peak, instances are very difficult which makes that ABT-2ph passes to phase II (where explicit information is exchanged) in many times, causing to reveal more information than ABT on constraints.

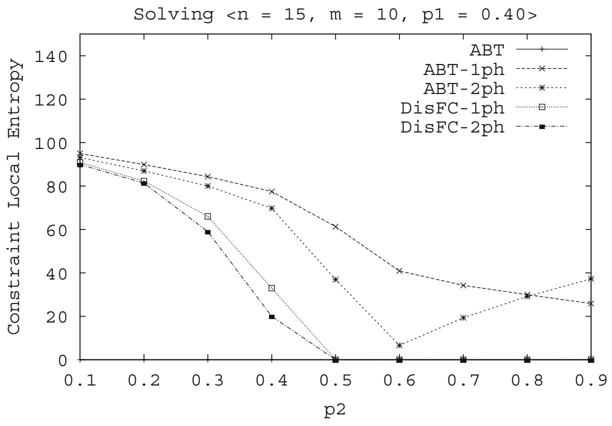
Regarding DisFC algorithms, DisFC-2ph/1ph report higher global entropy than ABT for very low tightness instances ($p_2 \leq 0.3$). This occurs because the number of forbidden tuples in total constraints of these instances is too small and therefore the filtered domains received by each DisFC-2ph/1ph agent include large number of values. Thus, an agent is more likely to success each time it tries to find a consistent value, which causes DisFC-2ph/1ph agents to reveal very few rows of their constraints. For instances with higher tightness ($p_2 \geq 0.4$), the number of forbidden tuples in total constraints becomes larger when p_2 increases. This makes that every agent tends to fail more when it tries to find a consistent value. In DisFC-2ph/1ph the above implies that every agent reveals a larger number of rows from its constraints, which makes that the uncertainties of other agents about the agent's constraints decreases. It worth noting that the global entropy of DisFC-2ph is almost zero for values of $p_2 \geq 0.4$. This represents that most of total constraints may be completely inferred by agents.

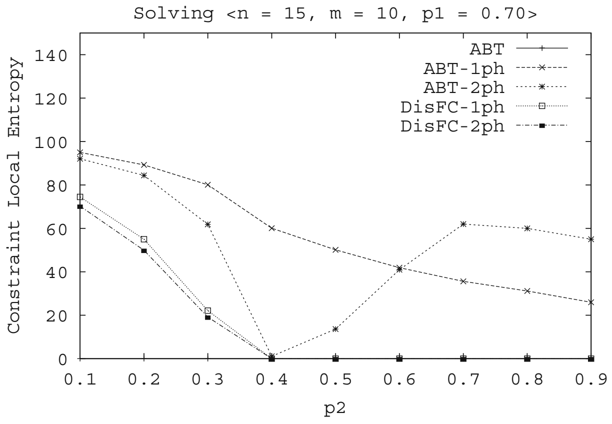
Comparing DisFC-2ph/1ph versus ABT-2ph/1ph, the latter algorithms have larger values of global entropy than the former ones for all the values of p_2. From the privacy point of view, this means that ABT-2ph/1ph always maintain total constraints more private than DisFC versions. The explanation of this is based on the explicit information type that agents reveal in each algorithm type. In DisFC algorithms, when an agent j receives an **ok?** message from an agent i, it actually receives a complete row of partial constraint $c_{i(j)}$, which may allow j to deduce one of the rows of c_{ij}. As discussed in Section 7, agent j must find all the solutions of a CSP in order to know the possible positions of the received rows in $c_{i(j)}$. After the resolution of this CSP, j may identify several rows in c_{ij}. Conversely, when j receives explicit information from i in ABT-2ph/1ph, it actually receives one entry of $c_{i(j)}$, and consequently, only one entry in c_{ij}.

Similar results for dense problems ($p_1 = 0.7$) appear in Fig. 10 (bottom). The relative ordering of algorithms remains unchanged. ABT-1ph reports the largest values of global entropy except for $p_2 \geq 0.7$, where ABT-2ph has the best results. DisFC algorithms show lower values of global entropy than ABT for instances with $p_2 > 0.2$. DisFC-2ph/1ph show worse results than ABT-2ph/1ph.

In addition to global entropy, we have evaluated the algorithms according to the entropy contribution of the constraint that is best known by an agent. This is computed in the following way. For each agent j, we first find min_j, which is the minimum contribution of any other agent i to $H_j(end)$. Then, we find the median value of all the values of min_j for all agents. The entropy value returned by the above procedure captures how close agents have been to completely discover one of their total constraints. In Fig. 11 we report the entropy contribution of the constraint that is best known by an agent. Median values are reported for ABT, ABT-2ph/1ph and DisFC-2p/1ph for low and high density problems. Regarding $p_1 = 0.4$ (top), results

Fig. 11 Constraint privacy in terms of the entropy contribution of the constraint that is best known by an agent. Median values are reported for ABT, ABT-2ph/1ph and DisFC-2ph/1ph for $p_1 = 0.4$ (*top*) and $p_1 = 0.7$ (*bottom*)





demonstrate the following. For $p_2 \geq 0.5$, the local entropies of DisFC-2ph/1ph are zero, which expresses that at least the half of the agents in DisFC algorithms may reconstruct completely one of the total constraints in which they are involved. As expected, the entropy of ABT is always zero because when the algorithm ends, the half of agents know completely at least one of the total constraints. The information revealed by ABT-2ph/1ph is not enough to allow the half of the agents to be able to reconstruct total constraints. Very similar results appear for dense problems (Fig. 11, bottom).

8.3 Discussion

The results in Section 8.2 reveal a clear trade-off between efficiency and privacy. Standard *ABT* is both faster and makes a more economic use of the network than the proposed PKC algorithms. However, ABT is not suitable for the PKC model and all the information about constraints must be revealed in advance.

ABT-2ph performs ABT multiple times on a partial problem. Therefore, its efficiency is dependent on the properties of the partial problem. The overhead in efficiency of multiple termination detection was not measured in our experiments but should be considered. In terms of constraint privacy, agents are aware of each phase change, so they are aware of the consistency of their assignment with respect to their neighbors at each change. ABT-1ph sends explicit information (**ngd** messages) when needed. Therefore, its privacy loss is related to the problem's difficulty. Like ABT it considers all constraints in a single phase but requires more time and network load than ABT. In terms of assignment privacy, both ABT-2ph and ABT-1ph keep agents' final assignments less private than ABT.

In addition to enforcing constraint privacy, DisFC-2ph and DisFC-1ph allow agents to conceal their assignments. Both algorithms are versions of DisFC, an ABT-like algorithm in which, instead of exchanging assignments with neighboring agents, agents send the list of compatible values in the neighbors domain. Similarly to the ABT versions for the PKC model, DisFC-2ph performs DisFC multiple times on a partial problem and DisFC-1ph considers all constraints in a single phase. Empirically, we observe that the cost of preserving assignments in DisFC-2ph/1ph is high in terms of efficiency and in terms of constraint privacy.

9 Conclusions

Privacy is one of the main motivations for solving constraint satisfaction problems in a distributed form. The model for DisCSP solving by ABT (the reference algorithm) does not consider privacy as a major goal. This paper addresses the inclusion of privacy in ABT. First, we differentiate between privacy of constraints and privacy of assignments. Privacy of constraints is concerned with constraints that are initially private (the PKC model) between agents, and they remain as private as possible during the solving process. Privacy of assignments considers that actual assigned values are not made public in the solving process. Second, we propose two families of algorithms, ABT-2ph/1ph and DisFC-2ph/1ph, to perform the actual solving while trying to keep the above mentioned privacy levels. These algorithms are clear descendants of ABT, they use the same kind of messages (plus some extra ones) and keep its good properties. They were initially conceived as two-phase algorithms, although later both phases were joined into a single one. Regarding privacy, these algorithms are 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 ABT-1ph always leaks less information than standard ABT. In the case of privacy of assignments, the ABT-2ph/1ph reveal more than standard ABT, while DisFC-2ph/1ph reveal less. The proposed algorithms have been implemented and evaluated on random DisCSP instances. Empirically we observe that to achieve privacy, algorithms degrade their performance (because they have to conceal some values, exchange more messages, etc.). To quantify privacy (and privacy loss) we have used entropy as defined in information theory.

References

1. Bessiere, C., Brito, I., Maestre, A., Meseguer, P. (2005). Asynchronous backtracking without adding links: A new member in the abt family. *Artificial Intelligence*, *161*(1–2), 7–24.
2. Brito, I., & Meseguer, P. (2003). Distributed forward checking. *Proc. of 8th CP*, *2833*, 801–806.

3. Brito, I., & Meseguer, P. (2005). Distributed stable matching problems. *Proc. of the 10th CP, Lecture Notes in Computer Science*, *3709*, 152–166.
4. Chandy, K., & Lamport, L. (1985). Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Computer Systems*, *3*(2), 63–75.
5. Cover, T. M., & Thomas, J. A. (2006). *Elements of information theory*. Wiley (2nd ed).
6. Gent, I. P., & Walsh, T. (1999). CSPlib: A benchmark library for constraints. *Technical report APES-09-1999.* Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proc. of the 5th CP (CP-1999) (pp. 480–481).
7. Greenstadt, R. (2007). Improving privacy in distributed constraint optimization. *Harvard Ph.D. thesis.* Available from http://www.eecs.harvard.edu/~greenie/hthesis.pdf.
8. Haralick, R., & Elliot, G. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, *14*, 263–313.
9. Lamport, L. (1978). Time, clocks, and the ordering of events in distributed system. *Communication of the ACM*, *2*, 95–114.
10. Lynch, N. A. (1997). *Distributed algorithms.* Morgan Kaufmann.
11. Maheswaran, R. T., Pearce, J. P., Bowring, E., Varakantham, P., Tambe, M. (2006). Privacy loss in distributed constraint reasoning: A quantative framework for analysis and its applications. *Autonomous Agents and Multi Agent Systems*, *13*, 27–60.
12. Meisels, A., & Lavee, O. (2004). Using additional information in DisCSP search. *In Proc. 5th workshop on distributed constraints reasoning, DCR-04.*
13. Meisels, A., Razgon, I., Kaplansky, E., Zivan, R. (2002). Comparing performance of distributed constraints processing algorithms. *In Proc. 3th workshop on distributed constraints reasoning, DCR-02 (AAMAS-2002)* (pp. 86–93).
14. Modi, P. J., & Veloso, M (2004). Multiagent meeting scheduling with rescheduling. *In Proc. of the fifth workshop on distributed constraint reasoning, DCR, CP 2004.*
15. Nissim, K., & Zivan, R. (2005). Secure DisCSP protocols—From centralized towards distributed solutions. *In Proc. 6th workshop on distributed constraints reasoning, DCR-05.*
16. Prosser, P. (1996). An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, *81*, 81–109.
17. Shanon, C. E. (1963). *The mathematical theory of communication.* University of Illinois Press.
18. Silaghi, M. C., & Mitra, D. (2004). Distributed constraint satisfaction and optimization with privacy enforcement. *In Proc. 3rd IC on intelligence agent technology* (pp. 531–535).
19. Silaghi, M. C., Sam-Haroud, D., Faltings, B. (2000). Asynchronous search with aggregations. *In Proc. of the 17th. AAAI* (pp. 917–922).
20. Smith, B. M. (1996). Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, *81*, 155–181.
21. Solotoresvsky, G., Gudes, E., Meisels, A. (1996). Modeling and solving distributed constraint satisfaction problems (DCSPs). *In Constraint Processing, 96*, 561–562.
22. Wallace, R. J., & Freuder, E. (2002). Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. *In Proc. 3rd workshop on distributed constrait reasoning, DCR-02* (pp. 176–182).
23. Yokoo, M., & Hirayama, K. (2000). Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, *3*(2), 185–207.
24. Yokoo, M., Durfee, E., Ishida, T., Kuwabara, K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. *In Proc. of the 12th. DCS.*
25. Yokoo, M., Durfee, E., Ishida, T., Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowledge and Data Engineering*, *10*, 673–685.
26. Yokoo, M., Suzuki, K., Hirayama, K. (2005). Secure distributed constraint satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, *161*(1–2), 229–246.
27. Zivan, R., & Meisels, A. (2006). Dynamic ordering for asynchronous backtracking on discsps. *Constraints*, *11*(2, 3), 179–197.
28. Zivan, R., & Meisels, A. (2006). Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI)*, *46*, 415–439, October.