Privacy in Distributed Meeting Scheduling [1]

Ismel Brito and Pedro Meseguer [2]
IIIA, CSIC, Campus UAB, 08193 Bellaterra, Spain

Abstract. Meetings are an important vehicle for human interaction. The Meeting Scheduling problem (*MS*) considers several agents, each holding a personal calendar, and a number of meetings which have to be scheduled among subsets of agents. *MS* is a naturally distributed problem with a clear motivation to avoid centralization: agents desire to keep their personal calendars as private as possible during resolution. *MS* can be formulated as Distributed CSP, but due to the form of its constraints the PKC model does not bring any benefit here. We take entropy as a measure for privacy, and evaluate several distributed algorithms for *MS* according to efficiency and privacy loss. Experiments show interesting results with respect to the kind of tested algorithms.

Keywords. Distributed constraints, privacy, entropy.

1. Introduction

The Meeting Scheduling problem (*MS*) consists of a set of agents, each holding a personal calendar where previous appointments may appear, and a set of meetings among them. The goal is to determine *when* and *where* these meetings could occur [7]. This problem is naturally distributed because (i) each agent knows only his/her own personal calendar and (ii) agents usually desire to preserve their personal calendars during resolution. In a centralized approach, all agents must give their private information to a central server, which solves the problem and returns a solution. This causes a high privacy loss, because each agent has to give his/her personal calendar to the server. In a distributed approach, to find a solution some information of the personal calendars has to be revealed, but not all of them as in the centralized case.

A natural formulation of *MS* is as Distributed Constraint Satisfaction (*DisCSP*) with privacy requirements. To enforce privacy in *DisCSP*, two main approaches have been explored. One considers the use of cryptographic techniques [10], which causes significant overhead in solving. Alternatively, other authors try to enforce privacy by different search strategies [7,6]. In this paper, we follow this line. It is worth noting that the *partially-known constraints* (PKC) approach [2,4], developed to enforce privacy in *DisCSP*, it does not offer benefits for this problem because an equality constraint cannot

[1] This work is partially supported by the project TIN2006-15387-C03-01 and by the Generalitat de Catalunya grant 2005-SGR-00093.

[2] Corresponding Author: IIIA CSIC, Campus UAB, 08193 Bellaterra, Spain; email: pedro@iiia.csic.es

be divided in two private parts (one for each agent of a binary constraint), and this problem is modelled using equality constraints. We analyze three distributed approaches for *MS*, which can be seen as representative algorithms of the state-of-the-art in distributed constraint solving. One is the simple *RR* algorithm [7], developed for this problem. The other two are generic *DisCSP* algorithms: the synchronous *SCBJ* and the asynchronous *ABT*. We compare them experimentally, with especial emphasis on the privacy level they can reach. Privacy is measured using entropy from information theory, as done in [4].

This paper is structured as follows. In section 2, we give a formal definition of the *MS* problem, providing a *DisCSP* encoding. In section 3, we present entropy as a measure for privacy and we discuss its applicability for the considered solving approaches. In section 4, we discuss the idea of using lies to enforce privacy. In section 5, we compare empirically the considered algorithms, and we extract some conclusions in section 6.

2. The Meeting Scheduling Problem

The *MS* problem [7] involves a set of agents and a set of meetings among them. The goal is to decide *when* and *where* these meetings could be scheduled. Formally, a *MS* is defined as (A, M, S, P), where $A = \{a_1, a_2, ..., a_n\}$ is a set of n agents; $M = \{m_1, m_2, ..., m_q\}$ is a set of q meetings; $att(m_k)$ is the set of m_k attendees; $S = \{s_1, s_2, ..., s_r\}$ is the set of r slots in any agent's calendar; $P = \{p_1, p_2, ...p_o\}$ is the set of places where meetings can occur. We also define the set of meetings where agent a_i is involved as $M_i = \{m_j | a_i \in att(m_j)\}$, the common meetings between agents a_i and a_j as $M_{ij} = M_i \cap M_j$, and the set of agents connected with a_i as $A_i = \cup_{m_j \in M_i} att(m_j)$. Initially, agents may have several slots reserved for already filled planning in their calendars. A solution must assign a time and a place to each meeting, such that the following constraints are satisfied (i) a meeting attendees must agree *where* and *when* it will occur; (ii) m_i and m_j cannot be held at same time if they have one common attendee; (iii) each attendee a_i of meeting m_j must have enough time to travel from the place where he/she is before the meeting to the place where the meeting m_j will be; human agents need enough time to travel to the place where their next meeting will occur.

MS is a truly distributed benchmark, in which each attendee may desire to keep the already planned meetings in his/her calendar private. This problem is very suitable to be treated by distributed techniques, trying to provide more autonomy to each agent while enforcing privacy. With this purpose, we formulate the Distributed Meeting Scheduling (*DisMS*) problem, which can be seen as a Distributed Constraint Satisfaction problem (*DisCSP*). One agent in *DisMS* corresponds exactly with an agent in *DisCSP*. Each agent includes one variable per meeting in which it participates. The variable domain enumerates all the possible alternatives of *where* and *when* the meeting may occur. Each domain has $o \times r$ values, where o is the number of places where meetings can be scheduled and r represents the number of slots in agents' calendars. There are two types of constraints: equality and difference constraints. There is a binary equality constraint between each pair of variables of different agents that corresponds to the same meeting. There is an all-different constraint between all variables that belong to the same agent.

3. Privacy in *DisMS* Algorithms

To solve a *DisMS* instance, agents exchange messages looking for a solution. During this process, agents leak some information about their personal calendars. Privacy loss is concerned with the amount of information that agents reveal to other agents. In the *DisCSP* formulation for *DisMS*, variable domains represent the availability of agents to hold a meeting at a given time and place, which is the information that agents desire to hide from other agents. In that sense, measuring the privacy loss of a *DisMS* modeled as *DisCSP* is the same as measuring the privacy loss of variable domains.

Following [4], we use entropy from information theory as a quantitative measure for privacy. The entropy of a random variable Z taking values in the discrete set S is,

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

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 [9,5]. If only one value k is possible for that variable, there is no uncertainty about the state of Z, and $H(Z) = 0$. Given a *DisCSP*, let us consider agent a_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 H_j associated with agent a_j and the entropy H associated with the whole problem as,

$$H_j = -\sum_{i \in S} p_i \; log_2 \; p_i \qquad H = \sum_{j \in A} H_j$$

Solving can be seen as an entropy-decrement process. Initially, agents know nothing about other agents. If a solution is reached after distributed search, agent a_i has no uncertainty about the values of its meetings (that also appear in other agents) so its entropy decrements. In addition, some information is leaked during search, which contributes to this decrement. We take the entropy decrement in solving as a measure of privacy loss, and this allows us to compare different algorithms with respect to privacy. We consider three particular states: *init*, it is the initial state, where agents know little about others; *sol*, it is when a solution has been found after solving; we are interested in the current states of the agents' calendars; *end*, it is the state after the solving process, no matter whether a solution has been found or not; here, we are interested in the initial state of agents' calendars. Assessing entropy of *sol* state, we evaluate how much of the reached solution is known by other agents. Assessing entropy of the *end* state, we evaluate how much information about the initial state has leaked in the solving process. In the following, we present *RR*, *SCBJ* and *ABT*, and their corresponding entropies for these states.

3.1. The RR Algorithm

RR was introduced in [7] to solve *DisMS*. This algorithm is based on a very simple communication protocol: agent a_i considers one of its meetings m_j, and proposes a time/place to the other agents in $att(m_j)$. These agents answer a_i with acceptance/rejection, depending whether the proposed time/place is acceptable or not accord-

ing to their calendars. In both cases, another agent takes control and proposes (i) a new time/place for m_j if it was not accepted in the previous round, or (ii) a time/place proposal for one of its meetings, if m_j was accepted. If an agent finds that no value exists for a particular meeting, backtracking occurs, and the latest decision taken is reconsidered. The process continues until finding a solution for every meeting (a whole solution for the *DisMS* problem), or when the first agent in the ordering performs backtracking (meaning that no global solution exists). Agent activation follows a Round Robin strategy.

RR agents exchange six types of messages: **pro**, **ok?**, **gd**, **ngd**, **sol**, **stp**. When a_i receives a **pro** message, this causes a_i to become the proposing agent. It considers one of its meetings m_j with no assigned value: a_i chooses a time/place for m_j and ask for agreement to other agents in $att(m_j)$ via **ok?** messages. When a_k receives an **ok?** message, it checks if the received proposal is valid with respect to previously scheduled appointments in its calendar. If so, a_k sends a **gd** message to a_i announcing that it accepts the proposal. Otherwise, a_k sends a **ngd** message to a_i meaning rejection. If a_i exhausts all its values for m_j without reaching agreement, it performs backtracking to the previous agent in the round-robin. Messages **sol** and **stp** announce to agents that a solution has been found or the problem is unsolvable, respectively.

Before search starts, the entropy of agent a_i is

$$H_i(init) = -\sum_{a_k \in A, k \neq i} \sum_{l=1}^{d} p_l \; log_2 \; p_l = \sum_{a_k \in A, k \neq i} log_2 \; d$$

where we assume that meetings have a common domain of size $d = r \times o$, whose values have the same probability $p_l = \frac{1}{d} \forall l$. In the sol state, when a solution is found, agent a_i knows the values of meetings in common with other agents, so its entropy is,

$$H_i(sol) = \sum_{a_k \in A_i, k \neq i} log_2(d - |M_{ik}|) + \sum_{a_k \in A - A_i} log_2 \; d$$

that is, the contribution of agents connected with a_i has decreased because a_i knows some entries their calendars, while the contribution of unconnected agents remains the same. Meetings in common with a_i is the minimum information that a_i will have in the *sol* state. Therefore, the previous expression is optimal for *DisMS*. The information leaked during the solving process has little influence in the entropy associated with the sol state, because an entry in a_i's calendar that is revealed as free in a round for a particular meeting m_j, it could be occupied by another meeting m_k involving a different set of attendees. However, this information is very relevant to dig into the initial state of other agents' calendars. During the solving process, each time a_i reveals that a slot is free (because it proposes that slot or because it accepts a proposal including that slot by another agent), it reveals that it was free at the beginning. This accumulates during search, and when search ends a_i knows some entries in the initial calendar of a_k. In the end statethe entropy is,

$$H_i(end) = \sum_{a_k \in A_i, k \neq i} log_2(d - free_k^i) + \sum_{a_k \in A - A_i} log_2 \; d$$

where $free_k^i$ is the number of different entries of a_k calendar that a_i knows were free at the beginning. It has been evaluated experimentally in section 5.

3.2. SCBJ

Synchronous Conflict-based Backjumping algorithm (*SCBJ*) is the synchronous distributed version of the well-known *CBJ* algorithm in the centralized case. *SCBJ* assigns variables sequentially, one by one. It sends to the next variable to assign the whole partial solution, that contains all assignments of previous variables. This variable tries a value and checks if this value is consistent with previous assignments. If so, that variable remains assigned and the new partial solution (that includes this new assignment) is sent to the next variable. If this value is inconsistent, a new value is tried. If the variable exhausts all its values, backjumping is performed to the last previous variable responsible for the conflict. Agents implement the described algorithm by exchanging assignments and nogoods through **ok?** and **ngd** messages, respectively. From the point of view of *DisMS*, agents accept or reject the proposals made by other agents. **ok?** messages are used for the agents to send proposals regarding time/place that are acceptable for a meeting. Contrary to what happens in *RR*, **ngd** messages only mean that someone has rejected the proposal, but the agent who has done such is not easily discovered. It is important to note that *SCBJ* loses some possible privacy in the sense that as the agents send **ok?** messages down the line, each agent knows that all the previous agents have accepted this proposal.

The entropy associated with the $init$ state is as in the *RR* case. In the sol state, a_i knows the value of every meeting in M_i, and the values of meetings scheduled prior its own meetings (there values were included in the last partial solution that reached a_i). So

$$H_i(sol) = \sum_{a_k \in A_i, k \neq i} log_2(d - |M_{ik}|) + \sum_{a_k \in A - A_i} log_2(d - known_k^i)$$

where $known_k^i$ is the number of meetings of a_k unconnected with a_i, whose time/place are known by a_i because they were scheduled before its own meetings. It is easy to see that *SCBJ* is less private than *RR* comparing their expressions of $H_i(sol)$. The first term is equal, but the second term is lower for *SCBJ* than for *RR* (it may be equal for some i, but for others has to be necessarily lower). This entropy decrement of *SCBJ* is due to the fact that a_i, besides knowing meetings in M_i (something shared with *RR*), it also knows the times/places of some other meetings where a_i does not participate. This is a serious drawback of this algorithm regarding privacy.

Regarding the initial state of calendars, a similar analysis to the one done for *RR* applies here. Each time a_i receives a partial solution with proposals, a free slot in the calendar of the proposing agent is revealed, and this information accumulates during search. At the end, the entropy associated with initial domains is,

$$H_i(end) = \sum_{a_k \in A, k \neq i} log_2(d - free_k^i)$$

where $free_k^i$ is the number of free slots that a_i detects in a_k. It has been evaluated experimentally in section 5.

3.3. ABT

Asynchronous Backtracking (*ABT*) [11] is an asynchronous algorithm that solves *DisCSP*s. Agents in *ABT* assign their variables asynchronously and concurrently. *ABT*

computes a solution (or detects that no solution exists) in finite time; it is correct and complete. ABT requires constraints to be directed. A constraint causes a directed link between two constrained agents. To make the network cycle-free, there is a total order among agents that corresponds to the directed links. Agent a_i has higher priority than agent a_j if a_i appears before a_j in the total order. Each *ABT* agent keeps its own agent view and nogood list. The agent view of a_i is the set of values that it believes to be assigned to agents connected to a_i by incoming links. The nogood list keeps the nogoods received by a_i as justifications of inconsistent values.

ABT agents exchange four types of messages: **ok?**, **ngd**, **addl**, **stp**. *ABT* starts by each agent assigning their variables, and sending these assignments to connected agents with lower priority via **ok?** messages. When an agent receives an assignment, it updates its agent view, removes inconsistent nogoods and checks the consistency of its current assignment with the updated agent view. If the current assignment of one of its variables is inconsistent, a new consistent value is searched. If no consistent value can be found, a **ngd** message is generated. When receiving a nogood, it is accepted if it is consistent with the agent view of a_i. Otherwise, it is discarded as obsolete. An accepted nogood is used to update the nogood list. It makes a_i search for a new consistent value of the considered variable, 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, causing backtracking. In addition, if a_i receives a **ngd** message mentioning an agent a_j not connected with a_i, a message **addl** is sent from a_i to a_j, asking for a new directed link, that will be permanent from this point on. The message **stp** means that no solution exists. When a solution is found, this is detected by quiescence in the network.

The entropy associated with the $init$ state is equal to the *RR* case. In the sol state, a_i knows the value of every meeting in M_i. It also knows the values of meetings corresponding to variables initially unconnected with a_i but later connected by added links,

$$H_i(sol) = \sum_{a_k \in A_i, k \neq i} log_2(d - |M_{ik}|) + \sum_{a_k \in A - A_i} log_2(d - link_k^i)$$

where $link_k^i$ is the number of meetings of a_k initially unconnected with a_i, whose time/place are known by a_i because they were connected by its new links during search. It has been evaluated experimentally in section 5.

It is easy to see that *ABT* is less private than *RR* comparing their expressions of $H_i(sol)$. The first term is equal, but the second term of *ABT* is lower than or equal to the second term of *RR*. This entropy decrement of *ABT* is due to the fact that a_i, in addition knowing meetings in M_i (something shared by *RR*), it also knows the times/places of some other meetings where a_i does not participate. It is worth noting here that there is a version of *ABT* that does not add new links during search [1]. For this version, called ABT_{not}, its privacy is equal to the obtained by *RR*, considering the sol state. With respect to *SCBJ*, no analytical result can be extracted: both share the first term, but their second terms are incomparable. Regarding initial domains, a similar analysis to the one done for *RR* and *SCBJ* applies here. Each time a_i receives a proposal, a free slot in the calendar of the proposing agent is revealed. This information accumulates during search. At the end, the entropy associated with initial domains is,

$$H_i(end) = \sum_{a_k \in A, k \neq i} log_2(d - free_k^i)$$

where $free_k^i$ is the number of free slots that a_i detects in a_k (a_k has to be higher than a_i in the total order). It has been evaluated experimentally in section 5.

4. Allowing Lies

A simple way to enforce privacy is to allow agents to lie: to declare they have assigned some value while this is not really true [3]. Formally, if variable i has m values $D_i = \{v_1, v_2, \ldots, v_m\}$, allowing lies means enlarging i's domain $D_i' = \{v_1, v_2, \ldots, v_m, v_{m+1}, \ldots, v_{m+t}\}$ with t extra values. The first m values are *true values*, while the rest are *false values*. If an agent assigns one of its variables with a false value and informs other agents, the agent is sending a lie. We assume that agents may lie when informing other agents, but they always say the truth when answering other agents.

If lies are allowed, an agent receiving a message cannot be sure whether it contains true or false information. The accumulation of true information during search is more difficult than when lies were not allowed. Lies are a way to conceal true information by adding some kind of *noise* that makes more difficult any inference on the data revealed during search. But this has a price in efficiency. Since the search space is artificially enlarged with false values, its traversal requires more effort. So using lies increases the privacy level achieved by solving algorithms but decreases their efficiency.

A solution reported by the algorithm must be a true solution, and it should not be based on a lie. If an agent has sent a lie, it has to change and say the truth in finite time afterwards. This is a sufficient condition to guarantee correctness in asynchronous algorithms (assuming that the time to say the truth is shorter than the quiescence time of the network) [3]. The inclusion of the lie strategy in synchronous algorithms presents some difficulties, as we see next. *RR* cannot use lies. If the first proposing agent sends a lie that is accepted by all other agents, this agent has no way to retract its lie. The control is passed to another agent, which if it does not see any conflict it passes control to another agent, etc. until all agents have been activated following the round-robin strategy. If lies are allowed, the RR algorithm might generate incorrect solutions. *SCBJ* cannot use lies, for similar reasons. An agent proposes when the partial solution reaches it. After assigning its variables, the new partial solution departs from it, and the agent has no longer control of that solution. If the agent introduces a lie in the partial solution, it may be the case that it could not retract the lie in the future, causing incorrect solutions. So using lies causes *SCBJ* to lose its correctness. *ABT* can use lies. An agent may change its value asynchronously. If an agent has sent a lie, it can send another true value in finite time afterwards Sending lies is not a problem for *ABT*-based algorithms [3].

The use of lies in *ABT* causes some changes in the information revealed during search. Each time a_i receives a proposal, it is unsure whether it contains a true value (a free slot in the proposing agent's calendar) or a false value (an occupied slot taken as free). Even if the proposing agent changes afterwards, it is unclear whether this is caused because the previous proposal was false or because it has been forced to change by search. Since there is uncertainty about this, no true information can be accumulated during search. At the end and only when there is a solution, the entropy associated with initial domains is equal to the entropy of the sol state.

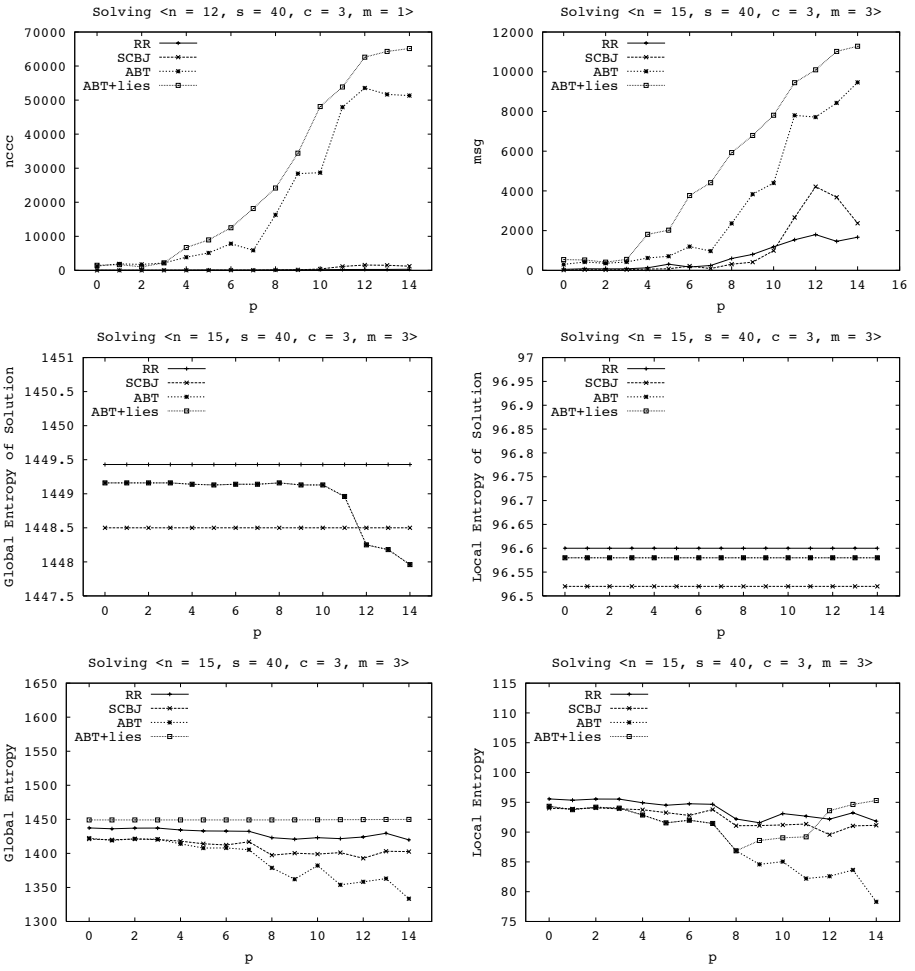

Figure 1. Experimental results of *RR*, *SCBJ*, *ABT* and ABT_{lies} on random *DisMS* instances. Top row: non–concurrent constraint checks and number of exchanged messages. Middle row: global and local entropy of sol state. Bottom row: global and local entropy of end state.

5. Experimental Results

We evaluate *RR*, *SCBJ*, *ABT* and ABT_{lies} (the *ABT* version where lies are allowed), on random meeting scheduling instances. We compare these algorithms using three measures: computation effort, communication cost and privacy loss. We measure computation effort using the number of non concurrent constraint checks (*nccc*) [8], communication cost as the number of messages exchanged (*msg*) and privacy loss in terms of entropy in *sol* and *end* states (entropy of the *init* state depends on the instance only).

In *SCBJ* and *ABT*, lower priority agents tend to work more than higher priority ones, which causes them to reveal more information. This generates an uneven distribution of privacy loss among agents as search progresses. Because of this, we provide values of global and local entropy for each algorithm. The global entropy, H, is the sum of agents' individual entropies. The local entropy is the minimum of agents' entropies, and it aims

at assessing the amount of information that the best informed agent infers from others' calendars. In both cases, higher values of entropy mean higher privacy.

The experimental setting considers *DisMS* instances with three meetings to schedule. Each instance is composed of 15 people, 5 days with 8 time slots per day and 3 meeting places. This gives $5 \cdot 8 \cdot 3 = 120$ possible values in each variable's domain. Meetings and time slots are both one hour long. The time required for travel among the three cities is 1 hour, 1 hour and 2 hours. *DisMS* instances are generated by randomly establishing p predefined meetings in each agent's calendar. Parameter p varies from 0 to 14.

In *RR*, we count one constraint check each time that an agent checks if a meeting can occur at a certain time/place. In all algorithms, each time an agent has to make a proposal, it chooses a time/place at random. Agents in *ABT* process messages by packets instead of processing one by one and implement the strategy of selecting the best nogood [1]. In ABT_{lies}, when an agent searches for a new consistent value, it randomly decides if it chooses a true or false value, so sending a truth value or a lie is equally probable.

Figure 1 shows the average results of 20 *DisMS* instances for each value of p. Regarding computation effort (*nccc*), we observe that the synchronous algorithms need fewer *nccc* than asynchronous ones, with *RR* showing the best performance. This phenomenon can be explained by analyzing how agents reject proposals. In *RR* the proposing agent broadcasts its meeting proposal to all agents sharing a constraint with it. When those agents receive the proposal, they determine if it is valid or not according to their own calendars. The proposal checking process can be concurrently executed by informed agents. In terms of *nccc*, this means that the total cost of all checking processes executed by informed agents is equal to the cost of one checking process (1 *nccc*). In *SBCJ* the checking process for a proposal involves more agents because a proposal made by agent a_i will reach a_j passing through intermediate agents (the number of intermediate agents depends on the ordering among agents) and further includes other proposals made by these intermediate agents. Finally, if the proposal made by a_i is found inconsistent by a_j, this will represent a substantial number of *nccc*. Contrary to the previous algorithms, *ABT* agents send proposals asynchronously, while knowing the proposals of higher priority agents. Since consistency among constraining proposals occurs when they have the same value, and values for the proposals are randomly selected, consistency occurs only after several trials. When considering lies, ABT_{lies} exhibits worse performance than pure *ABT* as expected - with lies performance deteriorates. Regarding communication cost (*msg*), the same ordering among algorithms occurs, for similar reasons.

As discussed in Section 3 , we have used entropy at *sol* and *end* states as quantitative metrics of privacy of the four algorithms. The middle row of Figure 1 corresponds to the global and local entropies at the *sol* state. In *RR*, we see that both metrics of entropy are constant because they are independent as far as search effects go. They also show a constant behavior for *SCBJ* because a static variable ordering has been used. The global entropy of *ABT* decreases when the number of predefined meetings increases. This is because finding a solution is harder when *p* increases, making agents add new links and thus decreasing privacy. The entropy of the best informed *ABT* agent (given by the local entropy) remains practically constant with respect to p. These points corroborate what was theoretically proven in Section 3: *RR* has higher entropy in *sol* than *SCBJ* and *ABT*. The bottom row of Figure 1 shows the global and local entropies at the *end* state of the four algorithms. Interestingly, the algorithm offering the highest global privacy is ABT_{lies}, while *ABT* is the algorithm offering the lowest global privacy. Allowing lies is a simple

way to achieve high privacy (at the extra cost of more computation and communication costs). The entropy of the best informed agent tends to decrease as p increases, keeping the relative order *RR*, *SCBJ*, *ABT*. At some point, ABT_{lies} separates from *ABT* and becomes the most private among them. This is due to the presence of unsolvable instances in the right part of the plot, which have the minimum contribution to the entropy of any agent of the system.

6. Conclusions

We have analyzed privacy loss of several distributed algorithms for *DisMS*, a naturally distributed problem of practical interest. We have used entropy as a quantitative measure for privacy, considering how much information leaks form the final solution and from initial calendars during search. We have also discussed the use of lies during resolution, to enforce privacy. Our experimental results show that the two synchronous approaches outperform the asynchronous ones in computation effort and communication cost. Regarding privacy, the picture is a bit more complex. Considering privacy of the final solution, the synchronous *RR* algorithm shows the best privacy. Considering privacy of initial calendars (due to information leaked during search), ABT_{lies} reaches the highest global privacy. All in all, the simple *RR* offers a very high privacy with unbeaten performance.

References

[1] C. Bessière, A. Maestre, I. Brito, P. Meseguer. Asynchronous Backtracking without Adding Links: a New Member to ABT Family. *Artificial Intelligence*, **161(1–2)**, 7–24, 2005.

[2] I. Brito and P. Meseguer. Distributed Forward Checking. *Proc. of 8th CP*, 801-806, 2003.

[3] I. Brito and P. Meseguer. Distributed Forward Checking May lie for Privacy. *Recent Advances in Constraints*, LNAI 4651, Ed. F. Azevedo, 93-107, 2007.

[4] I. Brito and A. Meisels and P. Meseguer and R. Zivan. Distributed Constraint satisfaction with Partially Known Constraints. *Constraints*, accepted for publication. 2008.

[5] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, Wiley-Interscience, 2nd edition, 2006.

[6] M. S. Franzin and F. Rossi and E. C. Freuder and R. Wallace. Multi-Agent Constraint Systems with Preferences: Efficiency, Solution Quality, and Privacy Loss. *Computational Intelligence*, **20**, 264–286, 2004.

[7] Freuder E.C., Minca M., Wallace R.J. Privacy/efficiency trade-offs in distributed meeting scheduling by constraint-based agents. *Proc. of DCR Workshop at IJCAI-01*, 63–71, USA, 2001.

[8] Meisels A., Kaplansky E., Razgon I., Zivan R. Comparing performance of distributed constraint processing algorithms.*Proc. of DCR Workshop at AAMAS-02*, 86–93, Italy, 2002.

[9] C. E. Shanon. *The Mathematical Theory of Communication*, University of Illinois Press, 1963.

[10] M. C. Silaghi. Meeting Scheduling Guaranteeing n/2-Privacy and Resistant to Statistical Analysis (Applicable to any DisCSP). *Proc. of the 3th Conference on Web Intelligence*, 711–715, 2004.

[11] M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Trans. Knowledge and Data Engineering*, **10**, 673–685, 1998.