Book Title
Book Editors
IOS Press, 2003

Distributed Meeting Scheduling

Ismel Brito [a] and Pedro Meseguer [a,1]
[a] *IIIA, Institut d'Investigació en Intel.ligència Artificial*
CSIC, Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Spain.

Abstract. Meetings are an important vehicle for human communication. The Meeting Scheduling problem (*MS*) is a decision-making process affecting several people, in which it is necessary to decide *when* and *where* several meetings could be scheduled. *MS* is a naturally distributed problem which has a clear motivation to be tried using distributed techniques: people may desire to preserve the already planned meetings in their personal calendars during resolution. In this paper, we evaluate three distributed algorithms for *MS* according to efficiency and privacy loss. Two of these algorithms view *MS* as a Distributed Constraint Satisfaction problem.

Keywords. meeting scheduling, distributed constraint satisfaction

1. Introduction

The Meeting Scheduling problem (*MS*) consists of a set of people which use their personal calendars to determine *when* and *where* one or more meeting(s) could take place [4]. This is problem is a naturally distributed problem because (1) each person knows only his/her own personal calendar before resolution and (2) people may desire to preserve the already planed meetings in their personal calendars during resolution. In the centralized approach, all people must give their private information to one person, who solves the problem and returns a solution. This approach results in a high privacy loss (each person must give his/her personal calendar to the solver). In a distributed approach, people work together, revealing some information of their personal calendars, in order to agree upon the time and the place that the meetings could be planned. In such context, it is natural to view *MS* as Distributed Constraint Satisfaction problem (*DisCSP*) with privacy requirements.

Regarding privacy, two main approaches have been explored in the context of *MS*. One considers the use of cryptographic techniques [6]. Alternatively, other authors try to enforce privacy by using different search strategies [4,3]. In this paper, we follow this line. Here, we provide an empirical comparison of three distributed approaches (two synchronous and one asynchronous) for *MS* in terms of efficiency and privacy loss. Among these approaches, two are *DisCSP* algorithms: Synchronous Conflict-backjumping (*SCBJ*) [2] and Asynchronous Backtracking (*ABT*) [7].

[1] Correspondence to: Pedro Meseguer, IIIA-CSIC, Campus UAB, 08193 Bellatera, Barcelona, Spain. Tel.: +34 93 580 9570; Fax: +34 93 580 9661; E-mail:pedro@iiia.csic.es.

2. The Meeting Scheduling Problem

The Meeting Scheduling [4] problem (in short *MS*) is a decision-making process affecting several people, in which it is necessary to decide *when* and *where* several meetings could be scheduled. Formally, a *MS* is defined by the following parameters:

- $P = \{p_1, p_2, ..., p_n\}$, the set of n people; each with his/her own calendar, which is divided into r slots, $S = \{s_1, s_2, ..., s_r\}$;
- $M = \{m_1, m_2, ..., m_k\}$, the set of k meetings;
- $At = \{at_1, at_2, ..., at_k\}$, the set of k collection of people that define which attendees must participate in each meeting, i.e. people in at_i must participate in the meeting m_i, $1 \leq i \leq k$ and $at_i \in P$;
- $c = \{pl_1, pl_2, ...pl_o\}$, the set of o places where meetings can be scheduled.

Initially, people may have several slots reserved for already filled planning in their calendars. A solution to this problem answers the *where* and *when* of each meeting. This solution must satisfy the next rules:

- attendees of a meeting must agree *where* and *when* the meeting is to take place,
- no two meetings m_i and m_j can be held at same time if they have at least one attendee in common,
- each attendee p_i of a meeting m_j must have enough time to travel from the place where he/she is before the meeting starts to the place where the meeting m_j will be. Similarly, people need sufficient time to travel to the place where their next meetings will take place.

2.1. Meeting Scheduling as Distributed CSP

The Meeting Scheduling problem is a truly distributed benchmark, in which each attendee may desire to keep the already planned meetings in his/her calendar private. So this problem is very suitable to be treated by distributed techniques, trying to provide more autonomy to each person, and to keep preferences private. For this purpose, we define the Distributed Meeting Scheduling problem (*DisMS*).

Every *DisMS* instance can be encoded as a Distributed Constraint Satisfaction problem (*DisCSP*). A *DisCSP* is a constraint satisfaction problem whose variables and constraints are distributed among autonomous agents [7]. In the *DisCSP* formulation for *DisMS*, there exists one agent per person. Every agent includes one variable for each meeting in which the corresponding person wishes to participate. The domains of the variables enumerate the possible alternatives of *where* and *when* meetings may occur. That is, each domain includes $k \times o$ values, where k means the number of places where meetings can be scheduling and o represents the number of slots in agents' calendars. There are two types of binary constraints between variables: equality and difference constraints. There exists a binary equality constraint between each pair of variables that belongs to different agents and corresponds to the same meeting. There exists a binary difference constraint between each pair of variable which belongs to the same agent.

3. Privacy on *DisMS* Algorithms

To solve a *DisMS* instance, agents must cooperate and communicate among them in order to determine *when* and *where* meetings will take place. 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 *DisSM*, variable domains represent the availability of people to hold a meeting at a given time and place, which actually is the information that agents desire hide from other agents. In that sense, measuring the privacy loss of a *DisMS* modeled as *DisCSP* is actually the same as measuring the privacy loss of variable domains.

Later on this section we will analyze privacy loss on three distributed algorithms for *DisMS*. From *DisMS* perspective, agents in these algorithms make proposals to other agents about *when* and *where* meetings could take place. A proposal can be accepted or rejected by recipient agents. Depending on the answers of recipient agents, the proposing agent can infer some information about the other agents. Similarly, when an agent receives an assignment proposal, some information is leaked about the proposing agent. In following we describe which knowledge can be inferred by agents in each case [3]:

1. When a proposal is rejected, the proposing agent can infer that it may be because the rejecting agent either has a meeting in that slot already or has a meeting that could not be reached if the proposed meeting was accepted.
2. When a proposal is accepted, the proposing agent can infer that the accepting agent does not have a meeting in that slot, that possible meetings that are incompatible with the proposal do not occur in the possible another agent's calendar.
3. When an agent receives a proposal from another agent, the recipient agent can infer that the proposing agent has not a meeting in that slot, nor in any slot that would be incompatible because of the distance constraints.

The aforementioned points constitute what we call *the process of knowledge inference*. In this work, we actually consider only part of the information that agents can infer by using the first point. The inferred knowledge in this case is very vague because the agent that receives a rejection cannot deduce anything for certain regarding the personal calendar of the rejecting agent. From the other two cases (points 2 and 3), we identify three kinds of information that can be deduced from agents:

Positive Information This is the information that denotes that can have a meeting in certain locations at certain times.

Negative Information This is the information that denotes that an agent cannot have a meeting in certain locations at certain times.

Open Slots This is the information related to slots in which an agent surely does not have any meeting already in any of the places.

The concepts of **Positive Information** and **Negative Information** are similar to the definitions of "present-meeting information" and "future-meeting information" given in [3]. Regarding **Open Slots**, this information can be deduced by an agent if its proposal is accepted by another agent. In this case, the accepting agent does not any meeting already in a time-and-city that is incompatible with the proposal because the distance constraints.

In the following subsections we analyze the details of the process of knowledge inference within each considered algorithm presuming that the number of meetings to be scheduled is simply one ($k = 1$).

3.1. The RR Algorithm

RR was presented and used in [4] to solve *DisMS*. This algorithm is based on a very simple communication protocol: one agent at a time proposes to the others agents the time and the location that meeting may occur. The ordering in which proposals are made follows the Round Robin strategy. When an agent receives a proposal, it responds only to the proposing agent if this proposal is possible according to its calendar.

RR agents exchange six types of messages: **pro**, **ok?**, **gd**, **ngd**, **sol**, **stp**. **pro** messages are used by agents to select the proposing agent. When an agent receives a **pro** message, this cause the agent to become the proposing agent. After the proposing agent chooses the time/place that the meeting can be scheduled, it informs about its decision to rest of agents via **ok?** messages. When an agent receives an **ok?** message, it checks if the received proposal is valid with respect to the previously scheduled appointments in its calendar. If this proposal is consistent, the agent sends a **gd** message to the proposing agent announcing it accepts the proposal. Otherwise, the agent sends a **ngd** message to the proposing agent saying that it rejects the proposal. Messages **sol** and **stp** are responsible for announcing to agents that a solution has been found or the problem is unsolvable, respectively.

Based on the previously discussed message system, it follows that the process of knowledge inference is clear-cut. The message system is simple: proposals are sent via **ok?** messages; which are accepted via **gd** messages or rejected via **ngd** messages.

3.2. SCBJ

Synchronous Conflict-backjumping algorithm (*SCBJ*) is a very simple distributed algorithm for solving *DisCSP* [2]. In *SCBJ* algorithm assign variables sequentially []. They exchange assignments and nogoods through **ok?** and **ngd** messages, respectively. From the point of view of *DisMS*, agents propose or reject the proposals made by other agents. **ok?** messages are used for the agents to send proposals regarding the time and the 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 subsequent agent knows that all the previous agents have accepted this proposal.

For the purpose of clarification, take for example, a problem consisting of five agents each one representing a person with its own personal calendar. Suppose that the first agent sends a proposal to the second agent about meeting Monday at 9:00 am in Barcelona. The second agent accepts the proposal and sends it to the third agent. Then, the third agent finds this proposal to unacceptable and therefore sends a **ngd** message to the second agent, effectively eliminating the possibility of meeting Monday at 9:00 in Barcelona. In this case, it is impossible for agent 1 or 2 to to know where the rejection originated, because any of the agents situated ahead of them, could be responsible, an the **ngd** message came via the third agent. Furthermore, it is impossible for both the first and second agents to discover the agent that rejected the proposal, as it could have been any of the agents situated ahead of them, as was simply relayed back to them via the third agent. However, in such systems, there is one specific case in which it is possible to determine which agent has rejected a proposal. In this example, such a case would occur

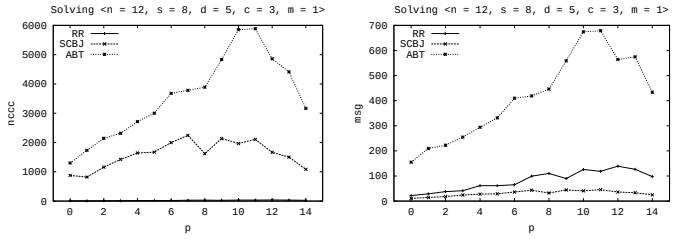

Figure 1. Constraint checks and number of messages for RR, SCBJ and ABT on Distributed Meeting Scheduling instances.

if all of the agents 1-4 have already received the proposal and then the fifth agent rejects. When this happens, the fourth agent knows that it was the fifth agent that rejected the proposal as the latter is the only agent capable of sending such a message, assuming that the fourth agent knows that the system contains only five agents in total.

3.3. ABT

The Asynchronous Backtracking algorithm (*ABT*) is a reference algorithm for *DisCSP* [7]. Agents in *ABT* assign the variables asynchronously and concurrently. Mainly, agents exchange assignments and nogoods through **ok?** and **ngd** messages, respectively. Similar to *SCBJ*, **ok?** messages represent proposals, while **ngd** messages represent rejections. The pervious discussion regarding **ngd** message in *SCBJ* is still valid for this algorithm, however, there is a difference with respect to **ok?** messages. This difference is that since a sending agent may send an **ok?** to all of the lower priority agents, the information contained in this message is only valid for the sending agent and is revealed only to the receiving agents.

4. Experimental Results

In this section, we evaluate two synchronous algorithms (*RR* and *SCBJ*) as well as one asynchronous algorithm (*ABT*) for solving *DisMS* instances. In order to compare the algorithms, we make use of three measure: computation effort, communication cost, and privacy loss. We measure computation effort using the number of non concurrent constraint checks (*nccc*) [5], communication cost in terms of the number of messages exchanged (*msg*) and privacy loss using the three types of information that agents may deduce regarding other agents' calendars: **Positive Information**, **Negative Information**, **Free Slots**.

Lower priority agents in *SCBJ* and *ABT* tend to work more than higher priority ones, which causes them to reveal more information than higher priority agents. In order to analyze the difference in the amount of privacy loss, we give the minimum, maximum and average amount data for each type of information that agents can find out from other agents' plans.

In our experiments, we deal with *DisMS* instances in which there has to be only one meeting scheduled and which admit at least one solution. Each problem is composed of 12 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 agent'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. The value of p varies from 0 to 14.

In *RR*, we look at one constraint check each time that an agent checks to see if a meeting can occur at a certain time/place. In all algorithms, each time an agent has to propose, it chooses a time/place at random. Agents in *ABT* process messages by packets instead of processing one by one [2] and implement the strategy of selecting the best nogood [1].

Figure 1 gives the results in terms of $nccc$ (on the left) and msg (on the right) for each algorithm averaged over 100 instances. For every value of p, we observe that *RR* requires less $nccc$ than *SCBJ* and *ABT* has the worst results. This can be explained by looking at how agents reject invalid proposals in each algorithm. In *RR*, the proposing agent broadcasts its proposal to all the other agents. Then, the receiving agents check if this proposal is valid or not. This process can be performed concurrently by all receiving agents, and therefore, its computation effort is just one non concurrent constraint check. (Actually, the $nccc$ value for *RR* is equal to number of proposals made before finding a solution.) In *SCBJ*, the active agent sends the proposal (received from prior agents) to the next agent when this is valid for him/her. It could be happen that a proposal make by the proposing agent in *RR* and by the first agent in the ordering in *SCBJ* and *ABT* decide to meet at certain time and certain place which is inconsistent for an agent lower in the ordering for *SCBJ* and *ABT*. In *RR*, this inconsistency will be found as soon as this agent responds to the proposing agent. In *SCBJ*, otherwise, this will be found when this agent receives the proposal, after that all the prior agents have accepted it and have performed several non concurrent constraint checks. Regarding *ABT*, this results can be explained because (1) agents choose their proposals randomly and (2) these proposals are made possibly without knowing the proposals of higher priority agents. The combination of these two facts make *ABT* agents more likely to fail when trying to reach an agreement regarding *where* and *when* the meeting could take place. Considering msg, the relative ordering among agents changes only in the sense that *RR* is worse than *SCBJ*. This difference between both algorithms occurs probably because *SCBJ* omits the accept messages used by *RR*.

Figure 2 report the privacy loss with respect to each information type. Regarding **Positive Information** (plots on the left in Figure 2), we observe that according to minimum values of **Positive Information**, *ABT* and *SCBJ* have similar behavior, while *RR* is a little worse, especially for more difficult problems ($p > 6$). This plot indicates that the less informed agent in terms for each algorithm infers only the **Positive Information** derived from the problem solution. That is, when a solution is reached, this agent can deduce that all the other agents can meet at the time and the location given by the found solution.

In terms of maximum values it is apparent that the difference between algorithms is greater. *ABT* is always less private than *SCBJ* and *RR* is the algorithm with the best results. From these values we may conclude that the better informed agent in *RR* has less **Positive Information** than the better informed agent in the other two algorithms. In terms of average values of **Positive Information**, the plot shows that *ABT* agents discover on average approximately two time slots in which each agent is available while for agents in the other two algorithms this value is approximately one. *SCBJ* shows better results than *RR* on instances with larger numbers of already planned appointments.

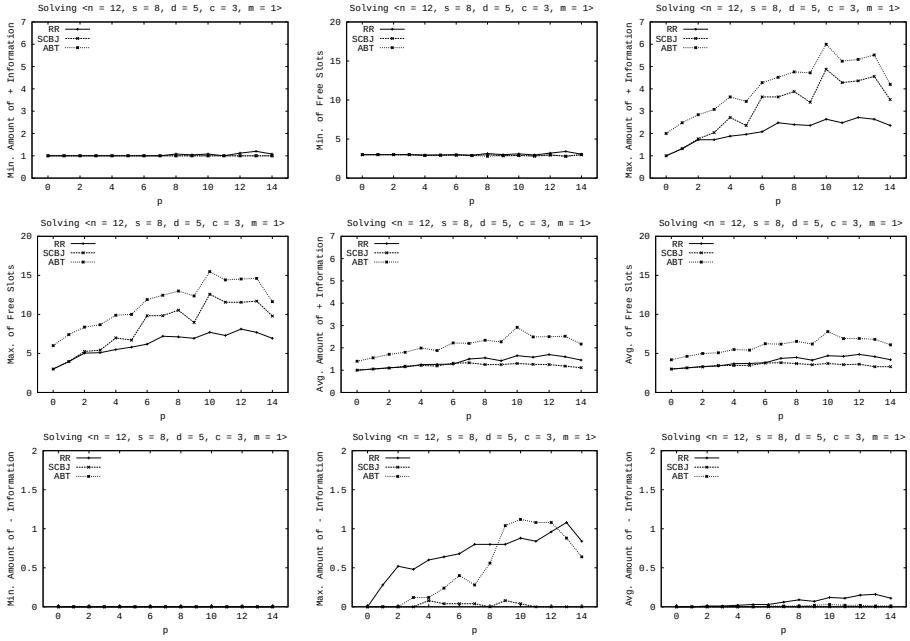

Figure 2. Privacy loss for RR, SCBJ and ABT on Distributed Meeting Scheduling instances.

Considering the number of **Free Slots** that agents leak from other agents, in terms of minimum values, we observe that the three algorithms have results similar to **Positive Information** ones. The less informed agent for each algorithms identifies almost 10 free slots from the other agents' calendars. In terms of maximum values, the better informed agent in *ABT* infers almost twice more **Free Slots** than the better informed agent in *RR*, while the better informed agent in *SCBJ* discovers more than this agent in *RR*. In terms of average values, *ABT* agents also find more **Free Slots** than the other two algorithms. *SCBJ* lightly outperforms *RR* on instances with $p > 6$.

The amount of **Negative Information** deduced in each algorithm is practically null. Only for instances with higher number of already planned appointments, the better informed agent in *ABT*/*RR* can identify at most one rejection from the other agents.

From the above results, we observe the following. Regarding computation effort and communication cost, *ABT*, the asynchronous algorithm, is less economic than the other two algorithms. This is because *ABT* agents work concurrently and select their proposals randomly, which makes more difficult *ABT* agents to reach an agreement regarding *when* and *where* they can meet together. Consistently for all tested instances, *SCBJ* requires less messages than *RR*, however, it performs more constraint checks. Regarding privacy, for the three algorithm the greater amount of information revealed identifies time slots in which agents surely does not have any meeting in any of the places. In terms of this parameter, *ABT* is always worse than the synchronous algorithms. On average, *SCBJ* agents reveal less information than *RR*. However, the better informed agent in *SCBJ* deduce more information than in *RR*.

5. Scheduling Multiple Meetings

This work is focused on *DisMS* instances in which only a meeting have to be planned. Previous works [4,3], that also use *RR* to solve *DisMS*, make the same assumption. This is motivated by the fact that agents in *RR* have to plan only one meeting at a time. Nevertheless, the extension of *RR* to deal with *DisMS* instances in which several meetings must be scheduled is direct. For these instances, *RR* agents schedule one meeting after the other. If agents reach an agreement about the time and the place where the current meeting may occur, another not yet scheduled meeting is considered as the current meeting. Otherwise, a different plan for the previous scheduled meeting is searched. This process continues until all the meetings have been planned or every alternative of place and time for the first meeting has been considered without reaching an agreement. Conversely to *RR*, agents in *SCBJ* and *ABT* may schedule several meetings simultaneously. For instances with multiple meetings to be scheduled, the only difference with respect to the details described above is that agents in these algorithms may hold multiple variables.

6. Conclusions

In this work we have evaluated privacy loss of three distributed algorithms for the Meeting Scheduling problem, a naturally distributed benchmark. Our experimental results show that the two synchronous approaches outperform the asynchronous one regarding computation effort, communication cost as well privacy loss. These results do not implies that synchronous algorithms should be considered the chosen algorithms for solving this problem. Synchronous and asynchronous algorithms have different functionalities (i.e. synchronous algorithms are less robust to network failures). Regarding privacy, neither of the distributed algorithms that we have considered is worse than the centralized approach, which needs to gather the whole problem into a single agent to solve it.

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)**, pp. 7–24, 2005.

[2] I. Brito and P. Meseguer. Synchronous, Asynchronous and Hybrid Algorithms for DisCSP. *Proc. of DCR Workshop at CP-2004*, 2004.

[3] 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**, pp. 264–286, 2004.

[4] 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*, pp. 63–71, USA, 2001.

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

[6] 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*, pp. 711–715, 2004.

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