

Using additional information in DisCSPs search ^{*}

Amnon Meisels and Oz Lavee
{am, laveeo}@cs.bgu.ac.il

Department of Computer Science,
Ben-Gurion University of the Negev
Beer-Sheva, 84-105, Israel

Abstract. A method of volunteering information during asynchronous search on DisCSPs is presented. The meeting scheduling problem (MSP) is formulated as a distributed search problem. In order to implement asynchronous backtracking (ABT) for the MSP, a multi-variable version of ABT is described. Agents participate in multiple meetings, where each meeting is represented by a variable that needs to be assigned a time-slot. Assignments are constrained by arrival-time constraints, since meetings take place in different locations. All constraints are local to their agents.

Additional information is in the form of Nogoods. During search for a consistent schedule for all meetings, agents can generate and send additional Nogoods to those sent by the ABT algorithm. When additional Nogoods are sent, the efficiency of asynchronous backtracking is enhanced. This effect grows with the number of additional volunteered Nogoods.

1 Introduction

An important goal of search algorithms for the distributed constraint satisfaction problem is to support agents' privacy. During cooperative search for a globally consistent solution, agents exchange messages about their assignments and about conflicts with other agents' assignments. This creates a natural trade-off between information disclosure and the efficiency (and correctness) of the distributed search process. The first to investigate measures of privacy for DisCSPs were Meseguer et. al. [Brito and Meseguer2003]. In a series of two papers they presented algorithms for maintaining two types of privacy during the run of the asynchronous backtracking (ABT) algorithm [Meseguer and Jimenez2000, Brito and Meseguer2003].

A different approach for investigating the privacy of distributed search was presented first by [Wallace and Freuder2002]. This concrete family of problems was used to compare the amount of needed computations for finding a solution, when different quantities of information were exchanged among the searching agents [Wallace2003].

The present paper uses the family of meetings scheduling problems (MSPs) to achieve three goals. First, to define a general family of *Meetings scheduling search problems* that will serve as framework for the study of privacy. For the family of MSPs the agents need to solve a hard search problem. The second goal is to enhance the

^{*} Partially supported by the Lynn and William Frankel Center for Computer Science

asynchronous backtracking algorithm [Yokoo and Hirayama2000], for multiple variables per agent. This is needed in order for each agent to schedule its multiple meetings and then cooperate with other agents to search for compatible schedules for all meetings. The third and main goal of the present study is to define a consistent method for enhancing the information content of messages of the search algorithm (ABT) that will enable a more efficient computation.

This paper examines the effect of volunteering additional information, in the form of additional Nogood messages, on the efficiency of search. The trade-off between information and efficiency is investigated in the context of the meetings scheduling problem [Wallace and Freuder2002].

The meeting scheduling problem (*MSP*), is the problem of coordinating a meeting among several agents each one with its own calendar and has appeared first in [Garrido and Sycara1995, Sen and Durfee1995]. A very restricted form of the *MSP* was investigated with respect to privacy by [Wallace and Freuder2002]. The tradeoff between the privacy of agents' meetings and the efficiency of the search process is studied by the use of a simple instance of the meeting scheduling problem. The instances used in [Wallace and Freuder2002] have only one meeting to coordinate, which all agents have to attend. Agents must be able to get from their private meetings to the scheduled meeting according to the traveling time constraints. Each agent has its own private calendar that defines its constraints regarding the time and location of the meetings.

The *MSP* has two main characteristics that make it into a *DisCSP*. It is logically distributed among all agents and since calendars are privately owned, it must use a distributed search process in order to find a solution that is consistent with all agents. The meetings of every agent are constrained with each other and the solution is globally consistent if every agent is able to reach all meetings in which it participates.

The aspect of privacy is very natural to the *MSP*. Agents do not want to reveal information regarding their calendar. In the studies of [Wallace and Freuder2002, Wallace2003], privacy is in fact measured by the fraction of calendars of agents that becomes known to other agents during the search process.

Another simplified form of the *MSP* was used by Bessiere et. al. [Bessiere *et al.*2001] for testing the Asynchronous Backtracking algorithm. The problem used in [Bessiere *et al.*2001] has three groups. Each group has to schedule a meeting for all its members, with the constraint that two groups cannot meet at the same time and location. Perceived as a centralized constraints satisfaction problem (*CSP*), each meeting can be represented by a variable and the values to be assigned are the weekly time-slots. From this point of view, the *MSP* of [Wallace and Freuder2002] has one variable and the *MSP* of [Bessiere *et al.*2001] has three variables. The constraints of the [Wallace and Freuder2002] *MSP* are unary, consisting of all forbidden times and locations. The search space of *CSPs* is exponential in the number of variables [Dechter2003] and in this respect both of the above problems are not hard search problems.

In [Wallace2003] the family of *MSPs* as been extended to be the graph coloring problem for n meetings (variables). The problem is to assign time-slots to all n variables (meetings), such that each variable is owned by more than one agent. The constraints among the values assigned to meetings which include a specific agent are inequality constraints. This creates a graph coloring problem of a distributed nature. Each agent

owns the variables corresponding to meetings in which it participates and an inequality constraint holds among them [Wallace2003]. The family of MSPs of the present study are general DisCSPs and its arrival-time constraints are more general than inequalities.

Former studies of privacy efficiency trade-off in distributed search used either a simple iterative algorithm [Wallace and Freuder2002], or a synchronous distributed backtracking for solving the problem [Wallace2003]. The present study, investigates the privacy efficiency trade-off for a general MSP and for the enhanced asynchronous backtracking (ABT) algorithm. It measures the effect of asynchronous exchange of additional information on asynchronous search (see section 4).

In section 2 the Meeting Scheduling Problem (MSP) is defined, as well as its *CSP* representation and its distributed CSP form. Section 3 presents a version of the *ABT* algorithm [Bessiere *et al.*2001] for multi variable agents. The issue of additional information for search enhancement, in the context of the MSP, is at the center of section 4. It analyses ways of sending additional information during search and presents a form that sends additional Nogoods to standard ABT. An extensive experimental investigation of the behavior of the proposed method of voluntary information, with respect to search efficiency, is described in section 5.

2 The Meeting Scheduling Problem

The definition of the meeting scheduling problem is presented in three stages. First, the logical meeting scheduling problem. Second, its representation as a (centralized) CSP and third, the representation as a distributed CSP. The meeting scheduling problem (*MSP*) has been defined in many versions with different parameters, from duration of meetings [Wallace and Freuder2002] to preferences of agents [Sen and Durfee1995]. The family of MSPs that is at the focus of the present study is defined as follows:

- A group S of m agents
- A set T of n meetings
- Each meeting is associated with a set $s_i \subset S$ of agents that attend it
- Consequently, each agent has a set of meetings that it must attend
- Each meeting is associated with a location
- The scheduled time-slots for meetings in T must enable the participating agents to travel among their meetings

An example of a conflict of an agent's constraint is a meeting A, scheduled to 14:00 in Rome and a meeting B, that includes the same agent, that is scheduled for 16:00 in Paris. Each meeting is one hour long and the traveling time between Rome and Paris is two hours. It is assumed that there are no private meetings for any agent. This generates no loss of generality, since private meetings (or agents' private calendars, as in [Wallace and Freuder2002] for example) can be simply represented by unary constraints, removing values from domains of meetings. The duration of each meeting is one hour and the traveling time between any two locations is equal for all the agents (no agent is faster than another). The agents need to negotiate in order to search for a schedule of all meetings that meets all of the participants arrival-time constraints.

The meeting scheduling problem as described above can be represented as a constraints satisfaction problem in the following way:

- a set of variables Z - m_1, m_2, \dots, m_n the meetings to be scheduled
- domains of values D - all weekly time-slots
- a set of constraints C - for every pair of meetings m_i, m_j there is an arrival-time constraint, if there is an agent that participates in both meetings

As already mentioned, private meetings are equivalent to unary constraints removing values from domains of some meetings. Since all agents have the same arrival-times between any two locations, there is only one type of arrival-time constraint.

arrival-time constraint - Given two time-slots t_i, t_j there is a conflict if $|time(t_i) - time(t_j)| - duration \leq TravellingTime(location(m_i), location(m_j))$

The tightness of the arrival-time constraint can be measured for a given definition of distances between locations. Expressing distances in terms of time-slots, enlarging the arrival-times has the effect of tightening the constraints of the problem.

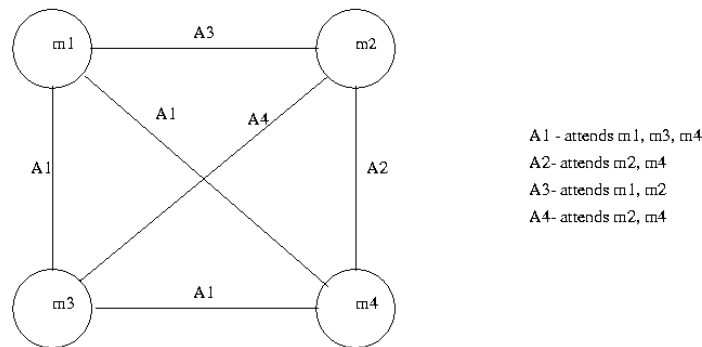


Fig. 1. the Meeting Scheduling Problem as a centralized CSP

Figure 1 presents the representation of a meeting scheduling problem as a CSP. The nodes are the meetings (the variables) and each edge represents a binary arrival-time constraint. Each edge is labeled by the agent, attending both meetings, that generates the arrival-time constraint.

Representing the MSP as a distributed CSP needs to associate variables with the different agents. Our distributed CSP representation can be described as follows:

- Agents - the Group S of agents
- For each Agent $s_i \in S$ there is a variable x_j^i , for Every meeting m_j that s_i attends.
- Each agent s_i includes arrival-time constraint between every pair of its local variables x_j^i, x_k^i .
- for each two agents s_i, s_j that attend meeting m_k there is an equality inter-constraint between the variables x_k^i, x_k^j , corresponding to the meeting m_k .

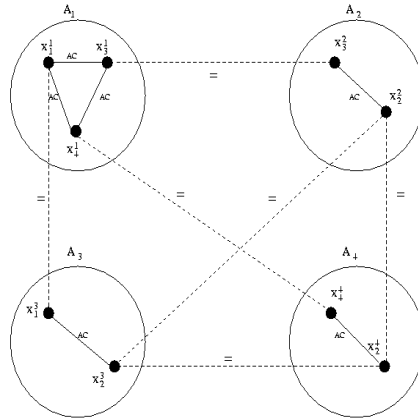


Fig. 2. the Meeting Scheduling Problem as a DisCSP

The representation of the *MSP* of figure 1 as a *DisCSP* can be seen in figure 2, where agents include multiple local variables connected by arrival-time constraints. Edges between variables of different agents represent the equality inter-constraint.

2.1 Random Meeting Scheduling problems (RMSPs)

Random Meeting Scheduling Problems (*RMSPs*) can be parametrized in numerous ways. Parameters can be the number of meetings, meetings' locations, number of agents, etc. To simplify the experimental design, one can use the relevant features of the CSP representation. Let us first denote a set of parameters:

- number of meetings - m
- number of agents - n
- number of meetings per agent- k
- distances between locations of meetings
- domain size - number of time-slots

The meetings are the set of m variables of the constraints network, each representing a meeting at a specific location. The domains of values are the time-slots. An edge between any pair of variables represents an agent that participates in both meetings. The density of the constraints network depends on the number of agents and the distribution of meetings that each agent attends. If each agent participates in k meetings, one can generate the resulting CSP as follows. For each of the n agents a clique of k variables is selected randomly, such that not all of the edges of the clique are already in the network. Each clique is added to the CSP, representing the arrival-time constraint between the meetings of each agent.

Similarly to randomly generated CSPs, one can calculate the resulting density p_1 of the network and the tightness p_2 . p_1 is the ratio of the total number of edges to the

maximal number - $m \times (m - 1)/2$. The tightness of the generated CSPs, p_2 , can be calculated by using the average distance between locations. For two meetings m_i, m_j connected by an arrival-time constraint, each value in the domain of m_i is inconsistent with $2 \times \text{distance}(m_i, m_j)$ values (time-slots) in the domain of m_j .

The representation of the above CSP as a distributed CSP is straightforward. Each meeting variable m_j in the CSP corresponds to the variable x_j^i within each agent A_i that participates in m_j . These variables are connected by the equality constraint on x_j^i , meaning that for all agents A_i , the meeting m_j is at the same time. When ordering the DisCSP, the first of the variables that represent the same meeting is connected to all other variables of the same meeting by an equality constraint. Each meeting has one participating agent that is first in the global order. This agent proposes time-slots for the meeting, during the run of asynchronous search. No other pair of participating agents need to be connected by an equality constraint.

3 Multi-variable Asynchronous Backtracking

Every agent of the meeting scheduling problem includes multiple variables, one for each meeting it attends. As a result, the distributed search algorithm must be able to deal with multiple local variables. For asynchronous backtracking (ABT) this is a special version of the algorithm that has not been described in the fundamental publications [Yokoo and Hirayama2000, Bessiere *et al.*2001]. The multi-variables version of ABT that is presented below is an adaptation from [Bessiere *et al.*2001], but, uses conflict-based backjumping (CBJ) [Prosser1993] for the local CSP of each agent.

As in standard ABT, all agents are assumed to be ordered [Bessiere *et al.*2001]. All variables of each agent are ordered successively, so that the variables of agent A_{i+1} follow successively the variables of agent A_i . The pseudo-code of the *ABT - CBJ* algorithm for multi-variable agents is presented in Figure 3. Agents running the algorithm wait for messages and upon receiving a message call the suitable procedure for the type of the received message. Elimination explanations are kept for each value of every variable (cf. [Ginsberg1993, Bessiere *et al.*2001]). Explanations may contain either local variables with higher priority or variables of other agents, with higher priority.

The *processInfo* procedure is called when an **ok?** message is received. It updates the *AgentView* with the received assignment and removes all eliminating explanations in all the local variables that contain the obsolete assignment of the received variable.

When a backtrack message is received, the *resolveConflict* procedure is called. This procedure is similar to the *resolveConflict* of *ABT* in [Bessiere *et al.*2001]. It checks the consistency of the received Nogood with the *AgentView*. If it is consistent, then *resolveConflict* updates the relevant assignments in the *AgentView* (the non Γ^- variables in the *AgentView*, in terms of [Bessiere *et al.*2001]). It also removes the eliminated value from the relevant local variable.

The *chooseValues()* procedure assigns values to all the local variables, checking that all the eliminators in all the variables are consistent with the *AgentView*. Lines 1,2 deal with the case that the current assignment of all the local variables is consistent with the *AgentView* and no changes needed. If this is not the case, then lines 4-18 find a consistent assignment for all the local variables. The order of all variables is

```

– ABT-CBJ:
1. SelfVars  $\leftarrow$  empty, end  $\leftarrow$  false
2. chooseValues()
3. while ( $\neg$ end)
4. msgs  $\leftarrow$  recvieve_all()
5. foreach msg  $\in$  msgs do
6.   switch (msg.type)
7.   info : processInfo(msg)
8.   Back : resolveConflict(msg)
9.   Stop : end  $\leftarrow$  true

– processInfo(msg):
1. update(AgentView, msg.variable, msg.value)
2. remove eliminators inconsistent with AgentView
3. chooseValues()

– chooseValues:
1. if consistent(SelfVars, AgentView)
2. then return
3. else
4. for p = 0 to SelfVars.size
5.   found  $\leftarrow$  find_assignment(selfVars[p])
6.   if found = false
7.     Nogood  $\leftarrow$  resolve(SelfVars[p].NogoodStore)
8.     if rhs(Nogood)  $\in$  SelfVars
9.       q  $\leftarrow$  index of rhs(Nogood)
10.      for i = q + 1 to p
11.        remove from selfVars Nogoods containing SelfVars[i]
12.        removeValue(SelfVars[q], Nogood)
13.        p = q
14.      else
15.        remove rhs(Nogood) from AgentView
16.        backtrack(Nogood)
17.        remove all Nogoods containing variables  $\in$  SelfVars
18.        p=0
19. for each agent  $\in$   $\Gamma^+$ (updatedVariable) sendMsg:Info(agent, updatedVariable)

– removeValue(variable, Nogood):
1. set eliminator Nogood at variable
2. for each var  $\in$  SelfVars
3.   remove eliminators contianing variable

– resolveConflict(msg):
1. if consistent(msg.Nogood,  $\Gamma^- \cup \{SelfVars\}$ )
2. for each assign  $\in$  lhs(msg.Nogood)  $\setminus$   $\Gamma^-$  do
3.   update(AgentView, ngVar)
4.   remove eleiminators inconsistent with AgentView
5.   rhsVariable  $\leftarrow$  rhs(Nogood)
6.   removeValue(rhsVariable, Nogood)
7.   chooseValues()
8. else if msg.sender  $\in$   $\Gamma^+ \wedge$  Consistent(msg.Nogood, SelfVars[rhs(msg.Nogood)])
then sendMsg:Info(msg.sender, SelfVars[rhs(msg.Nogood)])

```

Fig. 3. The multi-variable ABT algorithm

static. In line 5 a search for a value for the current variable x_j^i is performed, such that it is consistent with all assigned local variables and with the *AgentView*. If a value is not consistent an eliminator is added for this value. If no consistent value is found, the eliminators of the current variable are resolved to form a Nogood, in line 7.

When the Nogood points to a local variable x_k^i then a backjump to x_k^i will be performed, with the Nogood as eliminator for the assignment of x_k^i (lines 9-13). The backjump requires the removal of all eliminators from all the local variables $x_{k+1..j}^i$, that were jumped over (lines 10-11). This procedure implements the backjumping algorithm for multi local variables. The backjumping algorithm that is implemented for local variables is similar to [Ginsberg1993]. If the right hand side of the Nogood is a variable of a different agent, then it contains no local variables (since all the local variables are ordered successively). Therefore, the Nogood is sent in a backtrack message and the right hand side assignment of the Nogood is removed from the *AgentView* (lines 15-16). Next, the local process for consistent assignments to all local variables starts from the beginning (line 17-18). When consistent assignments for all local variables have been found, all new assignments are sent by an **ok?** message to all the agents that are later in the order of the problem (agents in I^+ of the updated variable, similarly to standard *ABT* [Bessiere *et al.*2001]).

4 Volunteering additional information

In studies of privacy issues of DisCSP search, the option of hiding information about assignments was considered by Meseguer and Jimenez [Meseguer and Jimenez2000]. In order to keep privacy of assignments, Meseguer *et. al.* propose to send forward a list of allowed values instead of the assignment itself. This idea cannot work for the MSP, because the equality constraint identifies the assignment of a single time-slot with the list of legal assignments for the agent receiving the **ok?** message. By the same token, constraints of the MSP cannot be kept private by the method of [Brito and Meseguer2003]. All the constraints of the meeting scheduling problem are arrival-time constraints which are *internal* to each agent. The inter-agent constraints are just equality constraints, for which hiding is meaningless.

Recently, several studies have investigated the trade-off between privacy and the efficiency of search. Wallace and Freuder [Wallace and Freuder2002] have looked at a simple MSP to show that loss of privacy enhances search efficiency. In a later study, Wallace have shown a similar trade-off to hold for a distributed graph coloring problem [Wallace2003]. The version of the MSP that was presented in section 2, is more general than the graph coloring problem of [Wallace2003]. However, by the above analysis of the privacy of MSPs, both types of privacy are not simply connected to the run of the search algorithm. For the family of meeting scheduling problems, the privacy question can be replaced by the possibility of volunteering additional information, to help agents arrive faster at a consistent solution.

Backtracking messages of the ABT algorithm contain Nogoods, which represent unsolvable sub-search spaces. Backtracking messages are based on violation of constraints. Since Nogoods contain the only information about constraints, one can *add Nogoods in order to volunteer relevant information*. Adding Nogoods to the asynchronous back-

tracking algorithm is presented below. It forms a method of adding viable information to agents, to enhance their efficiency in arriving at a solution.

Let us start with an example in which agent A_i has received an **ok?** message from agent A_l , proposing an assignment for its variable $\langle x_r^i = 15 : 00 \rangle$. Agent A_i has another meeting, with an assignment $\langle x_s^i = 14 : 00 \rangle$, proposed by agent A_j . As a result of arrival-time conflict, A_i has to reject the new assignment by sending the Nogood $\{(x_s^j = 14 : 00 \rightarrow x_r^l \neq 15 : 00)\}$. This Nogood informs agent A_l that meeting m_r , for which it is responsible, is in conflict with meeting m_s . It can also deduce that the agent responsible for meeting m_s is A_j . If the arrival time constraint between meetings m_r and m_s is three hours, than the following Nogood holds - $\{(x_s^j = 14 : 00 \rightarrow x_r^l \neq 16 : 00)\}$. In other words, agent A_i can generate additional Nogoods when the conflict occurs between its local variables. It is important to note here that Nogoods retain their meaning. In other words, Nogoods are valid during all stages of search. In that sense, the additional information retains its validity through all of the search process.

Different versions of asynchronous backtracking retain Nogoods in different ways. From retaining all of them in the first versions of ABT [Yokoo *et al.*1998], to erasing all Nogoods that are not currently consistent [Bessiere *et al.*2001]. It is important to note that the variety of strategies for retaining received Nogoods during asynchronous backtracking relates to space efficiency and not to completeness. This is why different correct algorithms choose differently [Bessiere *et al.*2001]. The present method does not interfere with the correctness of the multiple-variable ABT and it is independent of the question whether additional (volunteered) Nogoods are retained or not.

An absolute measure of the information content of a Nogood is the size of the eliminated subtree from the search tree. This measure depends on the size of the Nogood, the shorter the Nogood, the larger the eliminated subtree. It is easy to compute the fraction of the search space that is eliminated. If the LHS of the Nogood is $\langle X_k^1, T_k^1 \rangle \dots \langle X_m^i, T_m^i \rangle$ and there are n agents, then the eliminated subtree is of size $D^{i+1} \times \dots \times D^n$. The fraction of the search space that is eliminated by the Nogood is simply $\frac{D^{i+1} \times \dots \times D^n}{D^1 \times \dots \times D^n}$

For the meeting scheduling problem (*MSP*) a Nogood is a partial schedule, that conflicts with a proposed assignment of a time-slot to a given meeting. If a Nogood is generated by a conflict within the sending agent, it reflects a conflict of two or more of the meetings of the sending agent. Alternatively, the Nogood sent has been received (in longer form) by the sending agent and could not be resolved by it (see function **resolveConflicts()** in Figure 3).

The present investigation uses locally generated *additional* Nogoods as a form of volunteering information. The proposed method is to generate additional Nogoods and add them to every backtrack message. In order to add Nogoods to backtrack messages, the refined backtrack procedure is presented in Figure 4. The improved procedure checks for additional time slots of x_k^i , that create an immediate conflict between the local variable x_k^i of the meeting m_k and local variable x_w^i of meeting m_w . In the code of Figure 4, x_k^i has an equality constraint with the right hand side of the Nogood that forbids $\langle x_k^j, value \rangle$ and x_w^i has an equality constraint with one of the variables on the left hand side of the Nogood that is being sent back in the backtrack message

(lines 3-5). When the enhanced backtrack procedure finds such a conflict, it adds it to the *additionalNogoods* list (line 6). When the *additionalNogoods* list reaches the size of the predefined parameter - *informationFactor*, it is sent with the original nogood in a backtrack message (lines 7-8).

```

- backtrack(nogood):
  1. additionalNogoods  $\leftarrow \emptyset$ 
  2.  $\langle x_k^j, value \rangle \leftarrow rhs(nogood)$ 
  3. for each  $\langle x_w^l, val' \rangle \in lhs(nogood)$  do
  4. for each  $val \in domain(x_k^i)$  do
  5. if not consistent( $\langle x_w^l, val' \rangle, x_k^i, val$ ) then
  6.   additionalNogoods.add( $\langle x_w^l, val' \rangle \rightarrow \langle x_k^j, val \rangle$ )
  7. if additionalNogoods.size  $\geq informationFactor$  then
  8.   send:BT(Nogood, additionalNogoods)
  9. return
  10. send:BT(Nogood, additionalNogoods)

end procedure

```

Fig. 4. Backtrack procedure - sending additional Nogoods for agent A^i

5 Experimental Results

To simulate asynchronous agents, a Distributed CSP simulator is used, that implements agents as *Java Threads*. Threads (agents) run asynchronously, exchanging messages by using a common mailer. After the algorithm is initiated, agents block on incoming message queues and become active when messages are received.

Search algorithms on DisCSPs are run concurrently by all agents and their performance must be measured in terms of distributed computation. Two measures are commonly used to evaluate distributed algorithms - time, which is measured in terms of computational effort and network load [Lynch1997]. The time performance of search algorithms on DisCSPs has traditionally been measured by the number of computation cycles or steps (cf. [Yokoo and Hirayama2000]). In order to take into account the effort an agent makes during its local assignment the computational effort can be measured by the number of concurrent constraints checks that agents perform ([Meisels *et al.*2002]). Measuring the network load poses a much simpler problem. Network load is generally measured by counting the total number of messages sent during search [Lynch1997].

In the asynchronous simulator, concurrent steps of computation are counted by a method similar to that of [Lamport1978, Meisels *et al.*2002]. Every agent holds a counter of computation steps. Every message carries the value of the sending agent's counter. When an agent receives a message it updates its counter to the largest value between its own counter and the counter value carried by the message. By reporting the cost of the search as the largest counter held by some agent at the end of the search,

we achieve a measure of concurrent search effort that is similar to Lamport’s logical time [Lamport1978].

Meeting scheduling problems were generated randomly, as described in section 2.1. Locations of meetings were selected randomly from a set of 4 $\{P_1, P_2, P_3, P_4\}$. The distances among the 4 locations, in terms of time of travel, generate the arrival-time constraints among meetings. Two sets of experiments were performed. The first set of experiments has 9 meetings and 16 agents. Each agent participates in 3 meetings. The meetings of each agent were selected randomly, as described in section 2.1, by selecting cliques of 3 meetings. The random selection was performed so that no two agents attend exactly the same 3 meetings. The second set of experiments used 9 meetings and 24 agents with 2 meetings per agent. Each agent in this experiment adds an edge and not a clique, during the generation of random problems.

Two sets of distances among the meetings’ locations are used in the first set of experiments. The distances are described in Figures 5. The domains of all meetings contain 24 time-slots. Each experiment was performed 10 times and average results are reported.

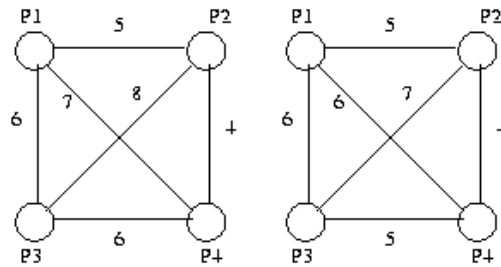


Fig. 5. (a) Distances between locations of meetings, (b) Smaller distances between locations of meetings.

All experiments consist of performing search for a consistent solution of the randomly generated problems, with different amounts of volunteered information. The runs of the problems had 6 different numbers of added Nogoods, or in terms of the parameter of algorithm `backtrack()` of Figure 4 $informationFactor = \{0,1,2,4,6,8,10\}$. The *informationFactor* is the maximum number of additional Nogoods per backtrack. The actual number in the experiments was very close and the results are parametrized by the average number of actual Nogoods sent.

Figures 6, 7 present the behavior of the three measures of performance, for growing number of additional Nogoods, in the first set of experiments (16 meetings and 9 agents). Two different distance graphs are presented in these figures. The grey columns use the distance map of Figure 5(a) and the dark columns use a distance map of smaller distances (Figure 5(b)). The smaller distances rule out fewer values per arrival-time constraint, thus generating a CSP with lower tightness.

It is easy to see that the computational effort is decreasing with increasing number of additional Nogoods. The overall factor of improvement in communication load is larger than 3 for the experiment with larger distances. This is the harder problem to solve. For the easier problem, with smaller distances, the overall scale is much smaller (i.e. easier problem) and the improvement is less dramatic. The improvement in communication load is easy to understand. Backtracking messages rule out a larger number of assignments for the receiving agent. As a result, less **info** messages are sent forward.

The improvement in the number of steps of computation can be explained by the following example. When an additional Nogood eliminating $x_k^i = 4$ is received by agent A_i , it eliminates a cycle of steps: assigning $x_k^i = 4$, assigning the rest of the local variables of A_i , sending **ok?** messages to all agents in I^+ and finally receiving a backtrack message that eliminates $x_k^i = 4$.

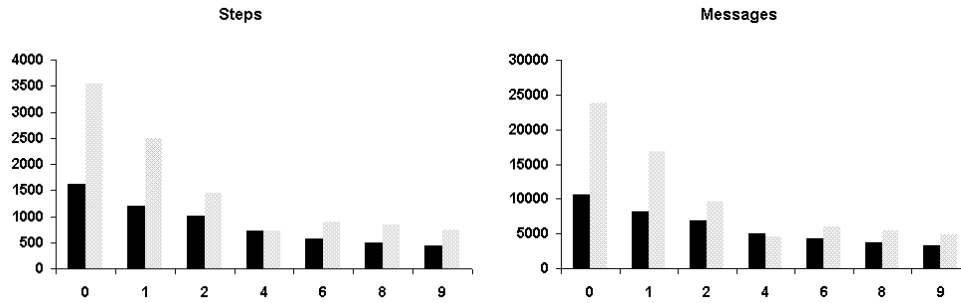


Fig. 6. (a) steps of computation vs. actual additional Nogoods, (b) total number of messages vs. additional Nogoods.

It is important to note that the production of additional Nogoods has a computational cost. For the MSP, the computational effort required for producing additional Nogoods is relatively small. This is a result of the structure of the problem, because all the intra-constraints are equality constraints. In other words, a Nogood ($x_k^i = 16 : 00 \rightarrow x_r^j \neq 17 : 00$) can be generated easily in agent A_i that attends both meetings m_k, m_r .

The computational effort of generating the additional Nogoods should affect the CCCs measure most, since the generation of Nogoods requires constraint checks. It is therefore interesting that the CCC performance measure in Figure 7 shows an improvement with increasing number of additional Nogoods. It is important to note that additional Nogoods are not always relevant for the receiving agent. The removed value of the additional Nogood may have already been erased, thus wasting the effort of generating the additional Nogood.

In the second set of experiments we used 9 meetings and 24 agents, with 2 meetings per agent. Each agent in this experiment adds an edge and not a clique, during the generation of random problems. This set of problems require less computational effort

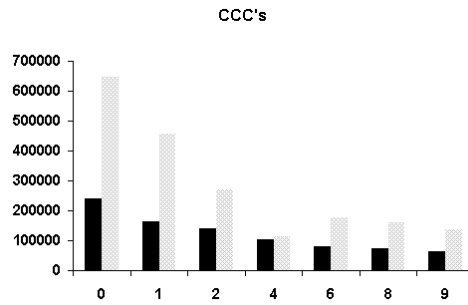


Fig. 7. Concurrent constraints checks (CCC's) vs. actual additional Nogoods

than the first two experiments, but the decrease of computational effort with increasing number of additional Nogoods is clear (Figures 8).

6 Discussion

The first investigation of the trade-off between privacy and efficiency of search was done by [Wallace and Freuder2002]. In their paper the agents tried to find a time-slot for a *single meeting of all agents*. The additional information in [Wallace and Freuder2002] was sets of time-slots that are already taken in individual calendars of agents. The addition of such information is immediately related to the privacy of agents. Sending lists of taken time-slots (i.e. with former meetings) reveals parts of the calendar of the sending agent.

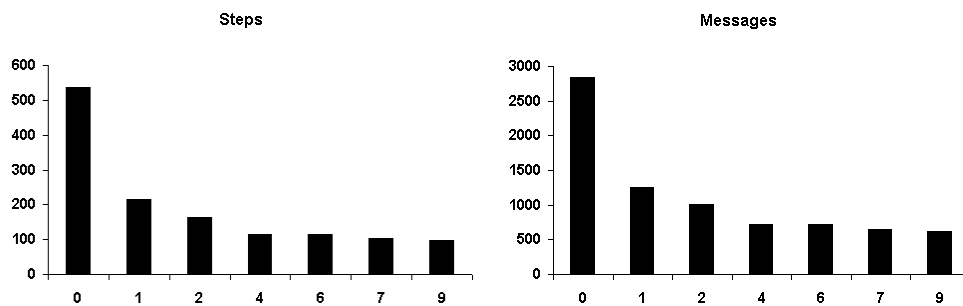


Fig. 8. Two meetings per agent: (a) steps vs. actual additional Nogoods, (b) total number of messages vs. average additional Nogoods.

The present paper differs completely from [Wallace and Freuder2002], in that it solves a search problem. A set of meetings, each with different subsets of the agents, has to be assigned non conflicting time-slots. The assignment problem is exponential in the number of meetings, thus different than a problem with one global meeting to schedule. In the context of a CSP, prior meetings of agents can be represented by different domains for agents. The constraints of the problem are arrival-time constraints, arising from the different locations of the meetings that need assignments. This generates a standard CSP with a clear set of variables, domains of values and constraints among variables (see section 2). When a distributed search process is performed on the MSP, partial assignments are temporary (i.e. the AgentView). This makes the privacy issue less clear. Additional information about the state of assignments carries little information about calendars of agents. Moreover, assignments are dynamic and change during search. In the view of the present paper, the only private information that is revealed during the distributed search process is the meetings in which each agent participates.

Because of the less clear nature of privacy during asynchronous search, the present investigation focuses on volunteered (additional) information. For asynchronous backtracking Nogoods are a clear form of information [Yokoo and Hirayama2000]. In all versions of ABT, differing amounts of Nogoods are kept as constraints discovered during search [Bessiere *et al.*2001]. The choice of the present paper is to volunteer information in the form of additional Nogoods that are sent during search (section 4). Nogoods are units of information about the search space that are of permanent validity. However, their *relevance* to asynchronous backtracking at any given moment can change dynamically (see [Bessiere *et al.*2001]). In other words, the usefulness of additional Nogoods is not guaranteed.

The main experimental result of the present paper is that additional information improves search efficiency. Sending additional Nogoods improves the performance of asynchronous backtracking on the distributed meetings scheduling problem, in 3 different measures. The total number of messages decreases and so does the number of computation cycles and the number of concurrent constraint checks (Figures 6, 7). The largest improvement occurs for adding the first and second additional Nogoods. The marginal gain, as more and more Nogoods are being added, becomes smaller. This is evident for all sets of experiments (see also Figures 8). Problems with a larger number of participating agents (i.e. 24), and a smaller number of meetings per agent are in general easier. It will be interesting to perform further experiments with larger number of meetings per agent. This will need much care during the process of problem generation, as the resulting network is very dense and can become insoluble.

It is interesting to try and clear the impact of the additional information to the present investigation, on the privacy of agents. To this end, one can think of internal constraints of arrival as private information. These constraints represent the set of meetings in which a given agent participates. In the model of the present investigation an arrival-time constraint of agent A_i can be resolved by another agent A_l after receiving a group of immediate Nogoods from A_i . Take for example the following group of immediate Nogoods, sent by agent A_i to agent A_l : $\{(x_k^j = 14 : 00 \rightarrow x_w^l \neq 14 : 00), (x_k^j = 14 : 00 \rightarrow x_w^l \neq 15 : 00), (x_k^j = 14 : 00 \rightarrow x_w^l \neq 16 : 00)\}$ This set of Nogoods can be interpreted by agent A_l as a lower bound on the traveling time of agent A_i from meet-

ing m_k to meeting m_w , in this case two hours. In this way, each immediate Nogood generates knowledge on the meetings and locations of the sending agent.

The effect of additional Nogoods on the efficiency of asynchronous search on general random DisCSPs can also be studied. The generation of additional Nogoods for general DisCSPs can be described as follows. Before sending back a Nogood, for each value in the domain of the destination agent, check whether this value (combined with the left hand side of the Nogood) is in conflict with all remaining values in the domain of the current (sending) agent. It is clear that this computation can be heavy for a general DisCSP and a general set of constraints. For the MSP family of problems the number of allowed domain values is extremely small, due to the equality constraint. Therefore, the computational effort required for additional Nogoods generation in MSP can still reduce the overall concurrent effort.

References

- [Bessiere *et al.*2001] C. Bessiere, A. Maestre, and P. Messeguer. Distributed dynamic backtracking. In *Workshop on Distributed Constraints in IJCAI-01*, Seattle, 2001.
- [Brito and Messeguer2003] I. Brito and P. Messeguer. Distributed forward checking. In *CP 2003: 9th International Conference*, pages 801–806, Kinsale, Ireland, 2003.
- [Dechter2003] R. Dechter. *Constraints Processing*. Morgan Kaufmann, 2003.
- [Garrido and Sycara1995] L. Garrido and K. Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In Victor Lesser, editor, *Proc. 1st Intern. Conf. on Multi-Agent Systems (ICMAS'95)*. MIT Press, 1995.
- [Ginsberg1993] M. L. Ginsberg. Dynamic backtracking. *Jou. of Art. Intell. Res.*, 1:25–46, 1993.
- [Lamport1978] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Comm. of ACM*, 21:558–565, 1978.
- [Lynch1997] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.
- [Meisels *et al.*2002] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. DCR Workshop, AAMAS-2002*, pages 86–93, Bologna, 2002.
- [Messeguer and Jimenez2000] P. Messeguer and M. A. Jimenez. Distributed forward checking. In *CP-2000 Workshop on Distributed Constraint Satisfaction*, Singapore, 2000.
- [Prosser1993] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.
- [Sen and Durfee1995] S. Sen and E. H. Durfee. Unsupervised surrogate agents and search bias change in flexible distributed scheduling. In Victor Lesser, editor, *Proc. 1st Intern. Conf. on Multi-Agent Systems (ICMAS'95)*, pages 336–343, San Francisco, CA, 1995. MIT Press.
- [Wallace and Freuder2002] R. J. Wallace and E. C. Freuder. Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. In M. Yokoo, editor, *AAMAS-02 Workshop on Distributed Constraint Reasoning*, pages 176–182, Bologna, 2002.
- [Wallace2003] R. J. Wallace. Reasoning with possibilities in multiagent graph coloring. In *4th Intern. Workshop on Distributed Constraint Reasoning*, pages 122–130, 2003.
- [Yokoo and Hirayama2000] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents & Multi-Agent Sys*, 3:198–212, 2000.
- [Yokoo *et al.*1998] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. on Data and Kn. Eng.*, 10:673–685, 1998.