Available online at www.sciencedirect.com
ScienceDirect
Engineering Applications of Artificial Intelligence 20 (2007) 857–873
Engineering Applications of ARTIFICIAL INTELLIGENCE
www.elsevier.com/locate/engappai

An agent-based approach to solve dynamic meeting scheduling problems with preferences

Ahlem BenHassine[a,*,1], Tu Bao Ho[b]

[a]Computational Linguistics Group, Language Grid Project, Knowledge Creating Communication Research Center (NICT), 3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0289, Japan
[b]Knowledge Creating Methodology Laboratory, School of knowledge Science, JAIST 1-1, Asahidai, Nomi-shi, Ishikawa 923-1292, Japan

Received 5 April 2005; received in revised form 22 September 2006; accepted 17 October 2006
Available online 12 January 2007

Abstract

Multi-agent systems are widely used to address large-scale distributed combinatorial applications in the real world. One such application is meeting scheduling (MS), which is defined by a variety of features. The MS problem is naturally distributed and especially subject to many alterations. In addition, this problem is characterized by the presence of users' preferences that turn it into a search for an optimal rather than a feasible solution. However, in real-world applications users usually have conflicting preferences, which makes the solving process an NP-hard problem. Most research efforts in the literature, adopting agent-based technologies, tackle the MS problem as a static problem. They often share some common properties: allowing the relaxation of any user's time restriction, not dealing with achieving any level of consistency among meetings to enhance the efficiency of the solving process, not tackling the consequences of the dynamic environment, and especially not addressing the real difficulty of distributed systems which is the complexity of message passing operations.

In an attempt to facilitate and streamline the process of scheduling meetings in any organization, the main contribution of this work is a new scalable agent-based approach for any dynamic MS problem (that we called MSRAC, for Meeting Scheduling with Reinforcement of Arc Consistency). In this approach we authorize only the relaxation of users' preferences while maintaining arc-consistency on the problem. The underlying protocol can efficiently reach the optimal solution (satisfying some predefined optimality criteria) whenever possible, using only minimum localized asynchronous communications. This purpose is achieved with minimal message passing while trying to preserve at most the privacy of involved users. Detailed experimental results on randomly generated MS problems show that MSRAC is scalable and it leads to speed up over other approaches, especially for large problems with strong constraints.
© 2006 Published by Elsevier Ltd.

Keywords: Dynamic valued constraint satisfaction problems; Meeting scheduling problems; Local consistency; Multi-agent system; Distributed problem solving

1. Introduction

Multi-agent systems are widely used to address many real-world combinatorial applications such as meeting scheduling (MS). This problem embodies a decision-making process affecting several users, in which it is necessary to decide *when* and *where* one or more meeting(s) should be scheduled. To satisfy real-world efficiency requirements, in this work we focused on two challenging characteristics: the distributed and dynamic nature of the problem. The MS problem is inherently distributed and hence cannot be solved by a centeralized approach; it is dynamic because users are frequently adding new meetings or removing scheduled ones from their calendar. This process often leads to a series of changes that must be continuously monitored.

The general task of solving an MS problem is normally time-consuming, iterative, and sometimes tedious, particularly when dealing with a dynamic environment. In other words, solving the MS problem involves finding a compromise between all the attendants' meeting

*Corresponding author. Tel.: +81 90 20 98 1458; fax: +81 774 98 6967.
E-mail addresses: ahlem@nict.go.jp (A.B. BenHassine), bao@jaist.ac.jp (T.B. Ho).
[1]This work was done when the first author was at the Japan Advanced Institute of Science and Technology.

requirements[2] (i.e., date, time and duration) which are usually conflicting. Thus, this problem is subject to several restrictions, essentially related to the availability, calendars and preferences of each user. Automating MS is important mainly because it can lead to more efficient and satisfying schedules within organizations (Feldman, 1987).

Most significant research efforts in the literature adopt agent-based technology for the distributed and dynamic aspects of MS problems. Initial meeting schedule research is based on constraint satisfaction problem formalism (CSP) (Montanari, 1974). The problem is formalized as centeralized CSP (Abdennadher and Schlenker, 1999; Bakker et al., 1993). These works are essentially focused on over-constrained CSPs. However, among more recent typical agent-based approaches, some works focused on using distributed autonomous and independent agents to solve the MS problem while maximizing users' preferences (Garrido and Sycara, 1996). This work is based on the communication protocol proposed by Sycara and Liu (Sycara and Liu, 1994), where agents are capable of negotiating and relaxing their constraints in order to reach an agreement on a schedule with high join utility. Another work also based on the multi-agent system, was described in (Sen et al., 1997). This work focuses on the problem of how an application domain for intelligent surrogate agents can be analyzed, understood and represented such that the underlying agents can make appropriate adaptations to their environment, to carry out tasks on behalf of human users. The authors' prior work focused on agents adapting to environmental changes (Sen and Durfee, 1994), however Sen et al. directed their efforts towards the integration of user preferences (Sen et al., 1997). Three other multi-agent approaches to MS problems, using the Partial CSP formalism introduced by (Freuder and Wallace, 1992), were given in the literature. The first work (Luo et al., 2000) is a new approach for MS problems using fuzzy constraints. The second work (Tsuruta and Shintani, 2000) proposes the distributed valued constraint satisfaction problem (DisVCSP) formalism to model MS problem. This approach is used in our experimental evaluation. The third work, based on multi-agent systems and using fuzzy constraints to express users' preferences, was presented in (Franzin et al., 2004). This MS system was based on an existing system that includes hard constraints (Franzin et al., 2002). The authors proposed to integrate preferences to their system and focused on observing the behavior of this new system under several conditions (Franzin et al., 2004). Their main objective was to evaluate the relations among solution quality, efficiency and privacy.

Nevertheless, the majority of these works share the following properties:

(1) Dealing only with non-dynamic problems (among which (Abdennadher and Schlenker, 1999; Bakker et al., 1993; Tsuruta and Shintani, 2000; BenHassine et al., 2003; Franzin et al., 2004); BenHassine et al., 2004a,b).
(2) Allowing the relaxation of any user's preferences, even those related to non-availability of this user in order to arrive at consensus choices for a meeting's time. However in real-world applications this is not always permitted. For example, when the user is traveling on business, such a constraint would oblige the user to stop his/her travel to attend the meeting, and this is not always possible (amongst Sycara and Liu, 1994; Garrido and Sycara, 1996; Sen et al., 1997; Luo et al., 2000; Tsuruta and Shintani, 2000; Franzin et al., 2004).
(3) Not integrating the enforcement of local consistency in their solving process, in spite of the pre-eminent role of the filtering techniques in the efficiency of solving an NP-complete problem. Only the authors in (Franzin et al., 2002, 2004) deal with the use of some inferred knowledge to maintain coherence between meetings in order to steer the selection of the next proposal, while, none of the other works try to maintain any level of consistency during the negotiation process.
(4) Judging all the meetings of the whole system with the same level of importance (among others Garrido and Sycara, 1996; Luo et al., 2000; Franzin et al., 2004; Tsuruta and Shintani, 2000). In real life, this is not always true. Obviously, the great significance of a meeting depends especially, but not only, on the leader of the event, the number of participants, and the meeting's main subject. Especially in a dynamic environment, such discrimination may lead to conflicting meetings, and may also increase the number of meetings to reschedule.
(5) Not considering the high complexity of message passing operations in real distributed systems (Garrido and Sycara, 1996; Sen et al., 1997; Luo et al., 2000; Tsuruta and Shintani, 2000; Franzin et al., 2004).

In addition, in (Yokoo and Hirayama, 2000) the authors described a complete and generic solution strategy, called asynchronous backtracking (ABT), to solve any distributed problem using DisCSP (distributed constraint satisfaction problem) formalism (Yokoo et al., 1990). In this approach, the agents act asynchronously by sending point-to-point messages according to their predetermined priority[3] order. Nevertheless, this approach presents, on the one hand some limitations for large and complex problems.[4] On the other hand, ABT can be applied only to non-dynamic problems, where no incremental constraint propagation is required. Therefore, we chose to use ABT as a witness approach in our experimental evaluation on *static* instances of the utilized MS problems, in order to empirically prove the correctness of our results.

[2]To simplify the problem, we assume that all the attendants are in the same city.

[3]This order is used to avoid the fall of agents into an infinite processing loop and then to guarantee the completeness of the algorithm.

[4]The proposed methods applied to the ABT algorithm, to make it able to handle multiple local variables, are neither efficient nor scalable (Yokoo and Hirayama, 2000).

We have learned from all the previous works and focused our research on the new MS requirements arising essentially from advances in computer and networking technology and also to dynamic environmental conditions. The MSRAC (meeting scheduling with reinforcement of arc consistency) multi-agent coordination approach proposed in this paper is a novel, scales better, dynamic and entirely distributed solution to the MS problem that accounts for user preferences, handles several events with various levels of importance and especially minimizes the number of exchanged messages. This paper significantly extends our previous work (BenHassine et al., 2004a). The basic characteristics of MSRAC are the following.

First, it is an incremental approach capable of processing problem alterations without conducting an exhaustive search.

Second, it is based on the distributed reinforcement for arc consistency (DRAC) approach (BenHassine and Ghedira, 2002) to enhance the efficiency of the solving process.

Third, in the MSRAC approach the MS problem is contemplated as a set of distributed reactive self-interested agents in communication, each with the ability to make local decisions on behalf of the user. The agents' decisions are not based on any global view[5] but only on currently available local knowledge. The final result is obtained as a consequence of their interactions. This purpose is achieved with the minimum number of exchanged messages by virtue of the real difficulty of message passing operations in a distributed systems.

Finally, the use of preferences naturally implies the adoption of an optimization criterion, both for each agent and also for the system as a whole. Thus, we adopted the dynamic valued constraint satisfaction problem formalism (DynVCSP) to model any MS problems. This formalism provides a useful framework for investigating how agents can coordinate their decision-making in such dynamic environment leading to more flexible and widely applicable approach to real life. We have to note that, this approach cope effectively with "closed" systems (Ephrati and Rosenschein, 1991), such as company or organization, whilst "opened" systems will be tackled in future work.

This paper is organized as follows. In Section 2, we give the proposed formalization for the dynamic MS problem. In Section 3, we present the MSRAC multi-agent model. In Section 4, we describe the MSRAC global dynamic. In Section 5, we present a theoretical and experimental comparative evaluation. Finally, Section 6 concludes the paper.

2. Formalization of the dynamic MS problem

The CSP framework has emerged as a key formalism for many combinatorial problems. We formalize the MS problem as a DVCSP in which each user maintains two kinds of constraints: hard and soft constraints related to him/her, along with other strong constraints defining the specific features of the problem itself. The hard constraints, which represent the non-availability of the user, can never be relaxed, while the soft constraints, which represent the preferences of the user, can be violated.

In the following, we first present static CSP formalism followed by dynamic CSP formalism, then we introduce VCSP formalism, and finally we state our proposed formalization for MS problem.

A *static* CSP is a triplet (X, D, C) composed of a finite set X of n variables, each of which takes values in an associated finite domain D and a set C of e constraints between these n variables (Montanari, 1974). Solving a CSP consists in finding one or all complete assignments of values to variables satisfying all the constraints.

A *dynamic* CSP P (DCSP) (Dechter and Dechter, 1988) is a sequence of static CSP $P_0, \dots, P_a, P_{a+1}, \dots$ each resulting from a restriction (a constraint or a variable is added) or relaxation (a constraint or a variable is retracted) in the preceding one.

A *valued* CSP P (VCSP) (Schiex et al., 1995) is a quintuple (X, D, C, S, φ) where (X, D, C) is the classical CSP formalism, $S = (E, \bigotimes, \succ)$ is a valuation structure, and an application $\varphi : C \to E$. $\varphi(c)$ is called valuation of $c, c \in C$. E is the set of possible valuations; $\succ$ is a total order on E; $\bot \in E$ corresponds to the maximal satisfaction and $\top \in E$ corresponds to the maximal dissatisfaction; $\bigotimes$ is an aggregation operator used to aggregate valuation.

Assume that A is an assignment of all the variables of the problem. A can be evaluated by combining the valuation of all the violated constraints using $\bigotimes$; $v_P(A) = \bigotimes_{c \in C} \varphi(A, c)$ where

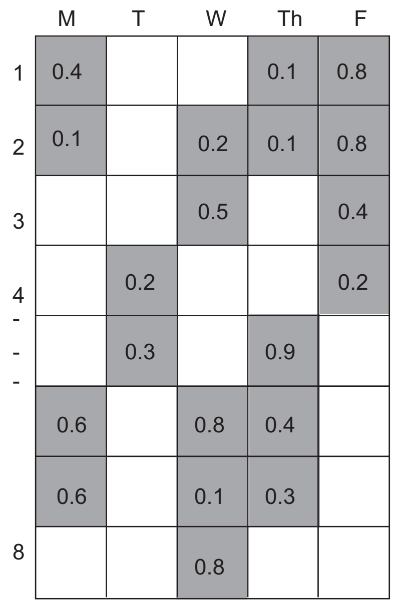
$$\varphi(A, c) = \begin{cases} \bot & \text{if } c \text{ is satisfied by } A, \\ \varphi(c) & \text{otherwise.} \end{cases} \tag{1}$$

We define a *dynamic* MS problem, as a DVCSP, by a *sequence* of quintuples (X, D, C, S, φ) where

- $X = \{X_1, \dots, X_n\}$ is the set of n meetings that need to be scheduled at an instant t. X_k with $k \in \{1, \dots, n\}$ denotes the kth meeting to schedule.
- $D = \{D_1, \dots, D_n\}$ is the set of all possible dates for all the meetings X. $D_i = \{dt_{i_1}, \dots, dt_{i_d}\}$ (with $|D_i| = d$) is the set of possible dates for the meeting X_i.
- C is the set of all constraints of the problem. C is composed of the following constraints:
  - *hard constraints*: represented by, on the one hand, C_{h} the set of the constraints related to the non-availability of all users (see the white box in Fig. 1). On the other hand, C_{allDiff} the set of *allDiff* constraints relating each pair of meetings X_k and X_l sharing at least one participant $A_j (A_j \in \text{Part}(X_k)$ and $A_j \in \text{Part}(X_l))$.[6]

[5]The agents exchange as little information as possible to keep most of their personal information private.

[6]The function $\text{Part}(X_k)$ denotes all the participants in the meeting X_k.



	M	T	W	Th	F
1	0.4			0.1	0.8
2	0.1		0.2	0.1	0.8
3			0.5		0.4
4		0.2			0.2
- - -		0.3		0.9	
	0.6		0.8	0.4	
	0.6		0.1	0.3	
8			0.8		

Fig. 1. Example of a user calendar.

- ○ *soft constraints*, C_s is the set of the soft constraints related to the preferences of all users towards the possible dates in their calendar (see the gray box in Fig. 1).
- $\varphi : C \rightarrow E.C = \{C_h \cup C_{\mathrm{allDiff}} \cup C_s\}$, for each hard constraint $c_i \in \{C_h \cup C_{\mathrm{allDiff}}\}$ we associate a weight = 1 and for each soft constraint $c_j \in C_s$ we associate a weight $w_j \in [0..1]$.[7] This weight reports the degree of preference of a user to have a meeting at the date dt_j (see the number inside the gray box in Fig. 1).
- S represents the valuation structure that defines the proposed optimality criteria (discussed in next section) and will be used to find the "best" solution.

In addition, for each meeting X_k (each variable in the problem) we assign a different weight $W_{X_k} \in [0..1]$ to define the degree of importance of the variable X_k ($k \in \{1, \ldots, n\}$) and it is used to allow the processing of the *most important* meeting[8] at first.

Solving a MS problem consists in finding a "good" assignment $\mathrm{sl}^* \in \mathrm{Sol} := D_1 \times \cdots \times D_n$ of the variables in $X = \{X_1, \ldots, X_n\}$ according to their importance W_{X_k}, such that all the hard constraints are satisfied while maximizing the global utility (GU) of all the users for all the scheduled meetings such that:

$$\mathrm{sl}^* = \arg\max_{sl \in \mathrm{Sol}} \mathrm{GU}(\mathrm{sl}). \tag{2}$$

The computation of the GU will be given in detail in the next section.

[7]This assumption does not contradict the ability of our protocol to support any kind of preferences' measurement evaluation.

[8]We assume that the users report truly and accurately the importance of their meetings.

3. MSRAC multi-agent model

The proposed MSRAC model, based on the dual constraint-graph, consists of a set of interconnected automated agents. Two types of agents are involved in this model, User agents and an Interface agent. Each User agent (proposer of a meeting and/or a participant in a meeting) represents a human user and is given the autonomy to negotiate on its user's behalf, while the Interface agent is added to the model in order to detect whether the global goal has been achieved and especially to inform the human users of the result.

In the sequel, a User agent is called a Proposer agent if his corresponding human user has a meeting to propose, otherwise he is called Participant agent. The same User agent can be Proposer agent for his meetings and a Participant agent in other meetings.

Each User agent (Proposer or Participant) has its own acquaintances, own knowledge (static and dynamic) and a reasoning engine. The acquaintances of a User agent A_i are dynamic and depend on the current meeting to be scheduled (or rescheduled). At an instant t, the acquaintances of A_i are defined by all the participants User agents in the current meeting X_k. The static knowledge of a User agent is formed by the possible dates for the underlying meeting X_k and the user's constraints. Its dynamic knowledge is formed by both its acquaintances and its current calendar.

Two User agents are connected together if and only if their corresponding human users should attend the same meeting. All the User agents will negotiate and cooperate together to schedule all the meetings proposed by the human users. Therefore we assume that the agents negotiate by exchanging asynchronous point-to-point messages containing the necessary relevant information, an agent can send a message to another only if it knows that this agent belongs to its acquaintances and the messages are received in a finite delivery time and in the same order that they are sent.

Each User agent A_i ($A_i \in A$) maintains a sequence of $\mathrm{VCSP}_P^{A_i}$ $(X^{A_i}, D^{A_i}, C^{A_i}, S, \varphi)$ for which the set of variables $X^{A_i} \in X$ represents the user's A_i meetings to schedule at the instant t. The constraints $C^{A_i} \in C$ $(C^{A_i} = \{C_h^{A_i} \cup C_s^{A_i} \cup C_{\mathrm{allDiff}}^{A_i}\})$ represent the non-availability, the calendar of this user and the constraints relating each pair of meetings.

In this multi-agent model, the intra-agent constraints are defined by the aforementioned constraints, whilst the inter-agent constraints are represented by the set of strong constraints, i.e. *equality* constraints. An equality constraint exists between two User agents A_i and A_j if and only if there exist at least one meeting $X_k^{A_i}$ (resp. $X_l^{A_j}$) such that $A_j \in \mathrm{Part}(X_k^{A_i})$ (resp. $A_i \in \mathrm{Part}(X_l^{A_j})$). We should discern that the equality constraints are dynamic.

The local goal of each User agent A_i is to schedule all its meetings, whenever possible, such that on the one hand all its hard constraints $C_h^{A_i} \cup C_{\mathrm{allDiff}}^{A_i}$ are satisfied, and on the

Table 1
Example of the degree of preference of each user A_i toward each possible date dt_p for the meeting $X_1^{A_1}$

	(Tu, 7)	(Wed, 2)	(Wed, 7)	(Th, 2)	(Th, 6)
A_1	0.1	0.3	0.9	0.6	0.4
A_2	0.6	0.7	0.3	0.5	0.4
A_3	0.1	0.2	0.3	0.6	0.7
A_4	0.7	0.3	0.4	0.2	0.1
$LU(X_1^{A_1}, dt_p)$	1.5	1.5	**1.9**	**1.9**	1.6

other hand the higher local utility (LU) for all the planned meetings is achieved. The LU brought off by a meeting $X_k^{A_i}$ scheduled at the date $dt_p \in D_k^{A_i}$ ($\mathrm{LU}(X_k^{A_i}, dt_p)$) is defined in Eq. (3) by the summation of the preferences (soft constraints) of all the participants $A_j \in \mathrm{Part}(X_k^{A_i})$

$$\mathrm{LU}(X_k^{A_i}, dt_p) = \sum_{A_j \in \mathrm{Part}(X_k^{A_i})} w_p^{A_j} \tag{3}$$

In order to fulfill its local goal, each User agent A_i should choose for each of its meetings $X_k^{A_i} \in X^{A_i}$ the date dt_p that maximizes its LU using Eq. (4).

$$\max_{dt_p \in D_k^{A_i};\ p \in \{1,\ldots,|D_k^{A_i}|\}} \mathrm{LU}(X_k^{A_i}, dt_p). \tag{4}$$

The global goal of the whole system is to schedule the maximum of the meetings of all the User agents satisfying all the inter-agent constraints and achieving the higher global utility (GU) which defines the quality of the solution. The GU is represented by the summation of all local utilities corresponding to the planned meetings (in the set of possible solutions s) by using Eq. (5).

$$\mathrm{GU}(\mathrm{sl}) = \sum_{A_i \in \mathrm{A}} \sum_{(X_l^{A_i}, dt_p) \in \mathrm{sl};\ dt_p \in D_l^{A_i}} \mathrm{LU}(X_l^{A_i}, dt_p). \tag{5}$$

However, for any meeting $X_k^{A_i}$, a date $dt_p \in D_k^{A_i}$ may be the most preferred by one participant and non-preferred (or less preferred) by the other participants. Therefore, in order to guarantee the maximum preference similarities between all the participants we propose to add to our system another criterion to satisfy this condition. The idea is to choose the date that, in addition to the first criterion described in Eq. (4), minimizes the distance between the own users' preferences by using Eq. (6).

$$\min_{dt_p \in D_l^{A_j};\ p \in \{1,\ldots,|D_l^{A_j}|\}} \left(\max_{\{A_k, A_i\} \in \mathrm{Part}(X_l^{A_j})} |w_p^{A_k} - w_p^{A_i}| \right). \tag{6}$$

To illustrate the use of Eq. (6) more clearly, let us consider the following example of 4 User agents (users) given the task of scheduling one meeting. We assume that A_1 is the Proposer of this meeting $X_1^{A_1}$ and all the other User agents are the participants. The possible dates for $X_1^{A_1}$ are $D_1^{A_1} = \{(\mathrm{Tu}, 7), (\mathrm{Wed}, 2), (\mathrm{Wed}, 7), (\mathrm{Th}, 2), \text{and } (\mathrm{Th}, 6)\}$. Table 1 illustrates the preferences $w_1^{A_i}$ of each attendee $A_i \in \mathrm{Part}(X_1^{A_1})$ toward each date $dt_p \in D_1^{A_1}$.

However, according to Table 1, the dates (Wed, 7) and (Th, 2) maximize the utility (LU) of the meeting, i.e., the sum of the utilities of all attendees for both of the two dates is 1.9. If we adopt the same strategy as (Franzin et al., 2002) the optimal solution should be the date (Wed, 7), with 0.3 as the overall preference. The optimality criteria defined in (Franzin et al., 2002) is based on maximizing the minimum preference overall agents (fuzzy criteria) and choosing an "undominated" set of preferences such that none of the agents' preferences can be improved without decreasing the preference of some other agent, i.e., for (Th, 2) the minimum preference is 0.2 lower than the overall preference for (Wed, 7), also none of the preferences of the agents can be improved without decreasing the preferences of some others. Nevertheless, the date (Wed, 7) is the most preferred only by A_1, and it is the less preferred by A_2, A_3, and A_4. Thus for our approach and with the second criterion we should instead chose the date (Th, 2), because it minimizes the difference between the users' preferences ($\max\{|0.9-0.3|; |0.6-0.2|\}$), and consequently reinforces the similarity between the attendees[9].

It is noteworthy that the above optimality criteria is based essentially on the preferences of the attendee toward the possible dates of the underlying meeting.[10] Such criteria require a common preferences scale otherwise it is not fair to compare the personal preferences of the participants in a meeting. To satisfy this condition without forcing the participants to reveal their private Calendar, we propose to integrate a new heuristic in the solving process. This heuristic allows the use of any ordering or scale to express the preferences of users (no common scale is imposed on users to express their own preferences). It is worth remarking at this stage that the use of such optimization criteria may lead to the classical problem of constructing interpersonal utilities functions (Feldman, 1980), i.e., *how to compare users' preferences using independent and different ordering and/or measurement scales*?

In this paper, the used criteria do not require any common ordering or scale over all the agents to express their preferences. The basic idea is to ask each attendee A_j in a meeting $X_k^{A_i}$ to *rank* the set of possible dates for $X_k^{A_i}$ from the most to the less preferred, i.e., $dt_p \prec dt_l$ if and

[9]We suppose that all the attendees have the same level in the company.
[10]Clarke Tax mechanism (Ephrati and Rosenschein, 1991) can be integrated in this model to enforce users to reveal their true preferences.

Table 2
Example of users' implicit preferences generated by the Proposer agent

	A_1	A_2	A_3	A_4
5	$\underline{\text{(Wed, 7)}}$	(Wed, 2)	(Th, 6)	(Tu, 7)
4	**(Th, 2)**	(Tu, 7)	**(Th, 2)**	$\underline{\text{(Wed, 7)}}$
3	(Th, 6)	**(Th, 2)**	$\underline{\text{(Wed, 7)}}$	(Wed, 2)
2	(Wed, 2)	(Th, 6)	(Wed, 2)	**(Th, 2)**
1	(Tu, 7)	$\underline{\text{(Wed, 7)}}$	(Tu, 7)	(Th, 6)

Table 3
Example of LU computation for each candidate

Candidate dates	Local utility
(Tu, 7)	$1+4+1+5=11$
(Wed, 2)	$2+5+2+3=12$
(Wed, 7)	$5+1+3+4=13$
(Th, 2)	$4+3+4+2=13$
(Th, 6)	$3+2+5+1=11$

only if $w_p^{A_i} > w_l^{A_i}$. For the previous example, the User agent A_2 will rank the possible dates for $X_1^{A_1}$ as follows: (Wed, 7)$\prec$(Th, 2)$\prec$(Th, 6)$\prec$(Wed, 2)$\prec$(Tu, 7). Then the Proposer agent will generate a new implicit ordinal scale[11] as stated by the received ordered sets. The lowest date dt_p in the order has the greatest number of votes associated with it. The Proposer agent will first assign an implicit preferences $Iw_p^{A_i}$ to each dt_p and then use it to determine the best date. Table 2 presents the candidate dates and their implicit preferences generated by the Proposer agent. In this example the local utilities[12] of the two candidates (Wed, 7) and (Th, 2) are the same (Table 3). The Proposer agent will choose (Th, 2) to enforce the similarity[13] between the participants. The maximum difference for (Wed, 7) is $|5-1|=4$, while it is $|4-2|$ for (Th, 2). It is noteworthy that this pseudo-common scale is dynamic, and may change according to the candidate dates for a meeting. Hence, the local utility of a meeting should be normalized to compare it to another one with different scales.

When there is a conflict between two meetings $X_k^{A_i}$ and $X_l^{A_j}$ for two different User agents or for the same agent ($X_k^{A_i}$ and $X_l^{A_i}$), i.e., two meetings with the same importance ($W_{X_k}^{A_i} = W_{X_l}^{A_j}$) that needs to be scheduled at the same date dt_p. The most obvious solution is to accept always to schedule at dt_p the meeting that increases the *LU* of the concerned agent ($\text{LU}(X_k^{A_i}, dt_p)$). However, trying always to increase the LU may lead to a local optimum rather than to a global one. In order to avoid to be stuck on local optimum, we propose to use the metropolis criterion of the simulated annealing technique to decide which meeting to schedule at dt_p. The main idea is that, while accepting some deterioration in the LU we may escape from the local optimum and converge toward the global one, i.e., $sl^* = \text{argmax}_{sl\in\text{Sol}}\,\text{GU}(\text{sl})$.

Three possible issues (one deterministic and two non-deterministic) can be applied to solve such conflict. If User agent A_i, which has a meeting already scheduled $X_k^{A_i}$ in its calendar at the date dt_p, receives another meeting $X_l^{A_j}$ with the same importance to be scheduled at the same date dt_p, then it will choose one of the following issues:

(1) The deterministic issue, defined as always scheduling at dt_p the meeting that will increase *LU*, i.e., If ($\text{LU}(X_k^{A_i}, dt_p) > \text{LU}(X_l^{A_j}, dt_p)$) then $X_k^{A_i}$ will be scheduled at dt_p and $X_l^{A_j}$ will be scheduled at another date $dt_h \neq dt_p$. Otherwise inversely.
(2) The non-deterministic issue, consists in arbitrarily choosing one of the meetings in conflict ($X_k^{A_i}$ or $X_l^{A_j}$) to schedule at dt_p and rescheduling the other one.
(3) The second non-deterministic issue, defined as using the *metropolis criterion* in order to choose the meeting to reschedule. $X_h^{A_i}$ is accepted to be scheduled at date dt_p by applying the acceptance probability described in Eq. (7). Note that this process leads to the rescheduling of $X_l^{A_j}$ and perhaps to the rescheduling of other meetings (with less importance) by propagation.[14] The main idea behind using metropolis criterion to solve the conflict is that trying always to increase LU may not lead to the optimal solution, while accepting some deterioration in the LU may increase the final GU.

$$P_c\{\text{accept } X_k^{A_i} = dt_p\} = \begin{cases} 1 \text{ if } \text{LU}(X_k^{A_i}, dt_p) \geqslant \text{LU}(X_l^{A_j}, dt_p), \\ \text{Exp}\left(\dfrac{\text{LU}(X_k^{A_i}, dt_p) - \text{LU}(X_l^{A_j}, dt_p)}{Tp}\right) & \text{otherwise,} \end{cases} \tag{7}$$

where $Tp \in R^+$ denotes the temperature.

In the following and mainly in the experiment, we propose three versions of the MSRAC approach regarding the aforementioned issues. The first is MSRAC-1, a deterministic approach in which we try always to improve the LU (issue 1). The second MSRAC-2, a non-deterministic approach in which each agent randomly chooses one of the conflicting meetings (issue 2). The third is MSRAC-3, a non-deterministic approach in which each agent applies the metropolis criteria to solve the conflict (issue 3).

4. MSRAC global dynamic

The global objective of our proposed approach MSRAC is to schedule all meetings for all users, while maximizing the

[11]This idea cannot handle cardinal preferences.
[12]Computed according to Eq. 3.
[13]According to Eq. 6.
[14]A threshold can be used in order to avoid the rescheduling of all meetings by successive propagation.

global utility and ensuring near fulfillment of users' preferences. This approach is based in a part on the DRAC approach (for *Distributed Reinforcement of Arc Consistency*) (BenHassine and Ghedira, 2002). DRAC is a constraint-based multi-agent approach to enforce arc-consistency on binary CN as a side-effect of the interactions among agents. The underlying model consists of a set of Constraint agents related by shared variables. Each agent maintains a local knowledge about the problem and cooperates with other agents by exchanging only relevant information. The global goal of all agents is to simplify the initial problem by enforcing arc-consistency property, i.e., to transform it into an equivalent one without loss of solution.

The MSRAC global dynamic is divided into two steps.

- Step 1 is to use the basic idea of the DRAC approach to transform the initial MS problem into another equivalent MS′ by enforcing local consistency (Mackworth, 1977) (node and arc consistency).
- Step 2 is to solve the obtained MS problem via interactions and negotiations between agents. This system does not include any central node to process meetings.

Before introducing the global dynamic, we present the communication protocol.

4.1. Communication protocol

For the communication protocol, the two basic message-passing primitives used for each agent are the same as those used in most MAS approaches.

- *sendMsg*(Sender, Receiver, *Message*) is used to send a message to one or more receivers.
- *getMsg*() extracts the first message from the mailbox of the agent.

With respect to exchanging messages, the underlying multi-agent dynamic involves the following messages:

- "*Start*" message, sent by the Interface agent to the corresponding Proposer agent to activate it whenever there is a new meeting given by a user.
- "*RedMeetCalendar*: *with*:" message, sent by a Proposer agent to each Participant agent to ask it to adjust the possible dates of the meeting according to both its user's non-availability and its calendar.
- "*Reply*" message, sent by each Participant agent to the Proposer agent in order to propagate performed reductions.
- "*ReceiveProp*: *with*:" message, sent by the Proposer agent to the Participant agent to verify the viability of the proposal.
- "*MeetNotPossible*" message, sent by each agent to the sender agent to inform it about the non-possibility of the meeting.
- "*MeetingOK*" message, sent by each Participant agent to the sender agent to inform it about its agreement for the date of the meeting.
- "*UpdateProp*: *with*:" message, sent by a Participant agent to a Proposer agent, in the case of conflict between two (or more) meetings, in order to invite it to relax its preferences.

4.2. MS dynamic

A user who wants to host a meeting must tailor the Interface agent, which will activate the corresponding Proposer agent and make it interact with all of the Participant agents. Note that more than one Proposer agent can be activated at the same time. Each activated Proposer agent A_i must first reduce the set of possible dates of the corresponding meeting $X_k^{A_i}$ according to its hard constraints (Algorithm 1 line 1). This process can be viewed as node consistency reinforcement. If the set of possible dates of the meeting becomes empty after reduction then its possible dates must be changed (Algorithm 1 line 3). Otherwise, the Proposer agent must delete all the dates that were used for more important meetings (Algorithm 1 lines 5 and 6), i.e., all meetings $(X_l^{A_j}, d_p) \in \text{Calendar}^{A_i}$ for which $W_{X_k} < W_{X_l}$. This can be viewed as arc consistency enforcement. A copy of the deleted proposals should be saved for other use in case the meeting $X_l^{A_j}$ is cancelled (Algorithm 1 line 7). Finally, the Proposer agent must send the obtained reduced set of possible dates of $X_k^{A_i}$ to all of the Participant agents to ask them to first, adapt it to their convenience, and then *rank*[15] the remaining possible dates according to their preferences (Algorithm 1 line 11).

The main goal of this heuristic is to define an ordinal relationship between all the proposals for each meeting according to users' interests. We can thus especially avoid the classical problem in constructing a common interpersonal utility function, which is how to compare preferences not relying on the same preference scale.

Each Participant agent that has received the message containing the reduced set of possible dates, starts first by reinforcing node and arc consistency (Algorithm 2 lines 1–5), then by ranking the obtained slot times (Algorithm 2 line 9) according to its preferences (from the most preferred to the less preferred date). The higher an agent ranks a particular date, the more points that date will receive. Specifically, a date is awarded one point for each rank below it.

Algorithm 1. *Start* message executed by a User agent A_i for each meeting $X_k^{A_i}$.

```
Start
1:    Delete from $D_k^{A_i}$ all non-viable values;
2:    if $D_k^{A_i} = \emptyset$ then
```

[15]This is used as a heuristic to decrease the number of BT and consequently the amount of exchanged messages and hopefully speed up the whole solution process.

```
3:          Change meeting possible dates for $X_k^{A_i}$;
4:      else
5:          for all ($X_l^{A_j}$ such that $(X_l^{A_j}, dt_p) \in$ Calendar$^{A_i}$)
        AND ($W_{X_l} > W_{X_k}$)
            do
6:              Delete($D_k^{A_i}$, $dt_p$);
7:              Add($DReserve^{A_i}$[k], $dt_p$);
8:              if $D_k^{A_i} = \emptyset$ then
9:                  Change meeting possible dates for $X_k^{A_i}$;
10:             else
11:                 forall $A_j \in$ Part($X_k^{A_i}$) do
12:                     Send(self, $A_j$, RedMeetCalendar:
                $D_k^{A_i}$with:$W_{X_k}$);
13:                 end for
14:             end if
15:         end for
16:     end if
```

Finally this agent must return the obtained set of possible dates to the Proposer agent (Algorithm 2 line 10). However, if the set of possible dates for a meeting becomes empty, this Participant agent must send a message to the Proposer agent to inform it about the non-possible meeting (Algorithm 2 lines 3 and 7). We should emphasize the fact that during this step all the agents try to look ahead for already scheduled meetings while reinforcing arc-consistency; this is in order to avoid maximum backtracking[16] in the next step. Otherwise, the first step is finished and the second step can be started. The Proposer agent tries first to find the proposal that satisfies all the participants' hard constraints and at the same time maximizes the utility of the meeting (Algorithm 3 line 3), and then sends it to the concerned acquaintances (Algorithm 3 line 10). To compute the utility of each proposal, the Proposer agent creates a pseudo common-scale based on the obtained ranking and then computes the utility of each possible meeting time and choose the proposal according to the optimality criteria described in Section 3. All the aforementioned interactions, between the Proposer agent A_i and the Participant agents, are summarized in Fig. 2 (Part I). The coming discussed interactions are summarized in Fig. 3 (Part II).

Each agent A_j that has received a proposal for a meeting $X_k^{A_i}$ must check whether it can still accept it or not. In the case of conflict (Algorithm 4 line 1), i.e., the agent A_j has meanwhile received a proposal for another meeting $X_l^{A_m}$ at the same time as the meeting $X_k^{A_i}$, the agent should act as follows:

- If $W_{X_k} < W_{X_l}$, it must send a negative answer to the Proposer agent A_i and ask it to relax its proposal (Algorithm 4 line 1).
- Otherwise, in the case of $W_{X_k} > W_{X_l}$, the agent must proceed in two steps: first, send a positive answer to the Proposer A_i and second send a message to the Proposer agent A_j of the meeting $X_l^{A_j}$ to invite it to relax its preferences for this meeting (Algorithm 4 line 4).
- Finally, in the case where $W_{X_h} = W_{X_l}$ (case of conflict between two meetings), the agent will try to apply one of the three proposed issues (Section 3), i.e., choose always the best, choose one of the meetings at random or apply the *metropolis* criterion, to decide which agent should relax its preferences (Algorithm 4 line 8).

Accordingly, each agent that has proposed a meeting and received at least one negative answer must change its proposal (Algorithm 5 line 4) if possible. The same process resumes until an agreement is reached among all of the participants or until testing all of the solutions, no agreement has been reached. In the latter case, the Proposer agent must inform the participants that the meeting is cancelled. Note that this dynamic allows a premature detection of failure: absence of solution for a meeting. This in the case when the set of possible dates of the concerned meeting becomes empty (Algorithm 5 line 10).

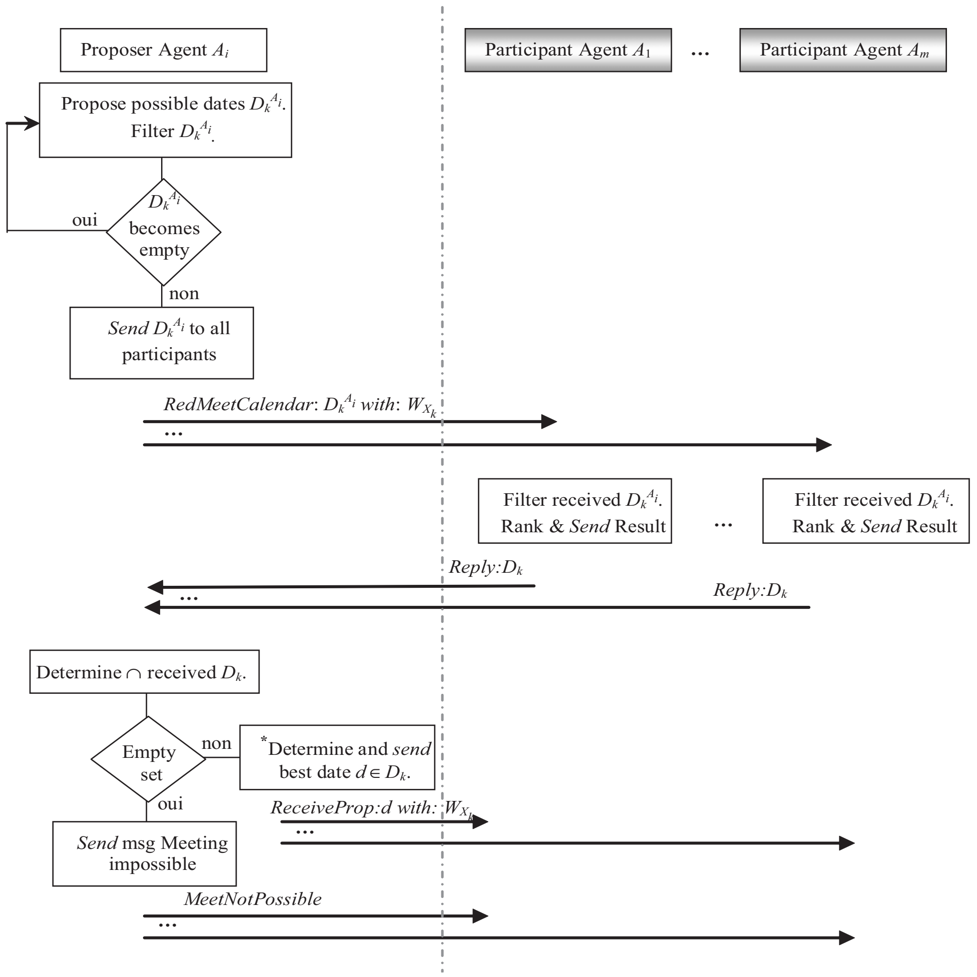
Algorithm 2. *RedMeetCalendar*: *with*: message corpus executed by each Participant agent A_i.

```
RedMeetCalendar: $D$ with: $W$
1:      Delete from $D$ all non-viable values;
2:      if $D = \emptyset$ then
3:          Send(self, sender, MeetNotPossible);
4:      else
5:          for all $X_l^{A_j}$ such that $((X_l^{A_j}, dt_p) \in Calendar^{A_i})$
        AND ($W_{X_l} > W$) do
6:              Delete($D$, $dt_p$);
7:              if $D = \emptyset$ then
8:                  Send(self, sender, MeetNotPossible);
9:              else
10:                 Rank($D$); /* According to $A_i$'s preferences
            $w_p^{A_i}$ */
11:                 Send(self, sender, Reply:$D$);
12:             end if
13:         end for
14:     end if
```

Algorithm 3. *Reply*: message corpus executed by each Proposer agent A_i.

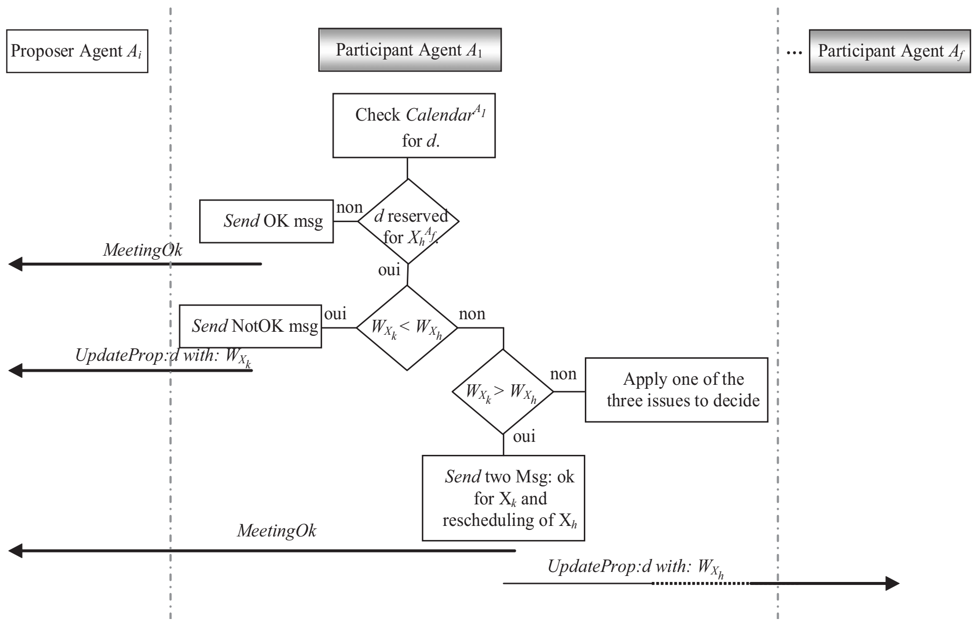
```
Reply: $D$
1:      if All ranked $D$ are received from all
        $A_j \in$ Part($X_k^{A_i}$) then
2:          Update($D_k^{A_i}$); /* According to received $D$s */
3:          $dt_p \leftarrow$ the date with higher utility;
            /* The choice of the date should be done
        according to the two equations (2) and (3) given in
        section 3. Because $A_i$ does not know the real
        preferences of the participants, $A_i$ will assign an
        implicit degree of preferences $Iw_f^{A_j}$ to each possible
```

[16]To avoid that the proposer choose a date and come back on it in case of non-availability of at least one participants.



Proposer Agent A_i
Participant Agent A_1
...
Participant Agent A_m
Propose possible dates $D_k^{A_i}$. Filter $D_k^{A_i}$.
$D_k^{A_i}$ becomes empty
oui
non
Send $D_k^{A_i}$ to all participants
RedMeetCalendar: $D_k^{A_i}$ *with*: W_{X_k}
Filter received $D_k^{A_i}$. Rank & *Send* Result
Reply:D$_k$
Reply:D$_k$
Determine ∩ received D_k.
Empty set
*Determine and *send* best date $d \in D_k$.
ReceiveProp:d with: W_{X_k}
Send msg Meeting impossible
MeetNotPossible

Fig. 2. Possible interactions between a Proposer agent A_i and a set of Participants for scheduling the meeting $X_k^{A_i}$ (Part I).



Proposer Agent A_i
Participant Agent A_1
... Participant Agent A_f
Check *Calendar*^{A_1} for *d*.
d reserved for $X_h^{A_f}$
non
oui
Send OK msg
MeetingOk
Send NotOK msg
$W_{X_k} < W_{X_h}$
UpdateProp:d with: W_{X_k}
$W_{X_k} > W_{X_h}$
Apply one of the three issues to decide
Send two Msg: ok for X$_k$ and rescheduling of X$_h$
MeetingOk
UpdateProp:d with: W_{X_h}

Fig. 3. Main behavior of each Participant receiving a proposal (Part II).

```
      date $dt_f \in D_k^{A_i}$ per participant $A_j$ according to the
      received rank. */
4:      if $dt_p = nil$ then
5:        for all $A_j \in Part(X_k^{A_i})$ do
6:          $Send(self, A_j, MeetNotPossible)$;
7:        end for
8:      else
9:        for all $A_j \in Part(X_k^{A_i})$
10:         $Send(self, A_j, ReceiveProp{:}dt_p\ with{:}W_{X_k})$;
11:       end for
12:     end if
13:   end if
```

Algorithm 4. *ReceiveProp*: ***with***: message corpus executed by each Participant agent A_i.

ReceiveProp:*Prop* **with**:*W*

```
1:    if $(\exists(X_l^{A_j}, dt_p) \in Calendar^{A_i}$ such that $Prop = dt_p$
      and $W_{X_l} > W)$ then
2:      $Send(self, sender, UpdateProp{:}Prop\ with{:}W)$;
3:    else
4:      if $(\exists\ (X_l^{A_j}, dt_p) \in Calendar^{A_i}$ such that $Prop = dt_p$
      and $W_{X_l} < W)$
        then
5:        $Send(self, A_j, UpdateProp{:}dt_p\ with{:}\ W_{X_l})$;
6:        $Send(self, sender, MeetingOK)$;
7:      else
8:        Apply one of the three proposed issues
      (Section 3.) to decide which agent should relax its
      preferences;
9:      end if
10:   end if
```

Algorithm 5. *UpdateProp*: ***with***: message corpus executed by each Proposer agent A_i in case of conflict.

UpdateProp:*Prop* ***with***:*W*

```
1:    Delete $(D_k^{A_i}, Prop)$;
2:    Add$(DReserve^{A_i}[k], Prop)$;
3:    if $D_k^{A_i} \neq \emptyset$ then
4:      $dt_p \leftarrow$ the date with higher utility in $D_k^{A_i}$;
5:      for all $A_j \in$ Part$(X_k^{A_i})$
6:        $Send(self, A_j, ReceiveProp{:}\ dt_p\ with{:}\ W_{X_k})$;
7:      end for
8:    else
9:      for all $A_j \in Part(X_k^{A_i})$ do
10:       $Send(self, A_j, MeetNotPossible)$;
11:     end for
12:   end if
```

4.3. Process of dynamic meetings

In real-world applications MS problems are subject to many changes, defined on one side by the arrival of new, more important meetings (especially when all the other meetings have been already approved) and on the other side by the cancellation of one (or more) meeting(s) which, can lead to the possibility of scheduling other meetings, previously detected as non-possible. Therefore, we have used an incremental approach that can handle all forms of alteration in the system without restarting the solving process from scratch. However, since this approach deals directly with human-being, re-scheduling process may lead to several issues especially, "nervousness". To deal with such issue, we propose to define a threshold for the maximum number of allowed modifications for each user. This psychological issues will be considered in more details in future work.

In the following, we present the behavior of our protocol in the case of restrictions and relaxations.

4.3.1. The restrictions

For each new arrival meeting $X_k^{A_i}$ with the priority $W_{X_k}^{A_i}$, the Proposer agent must first eliminate non-viable values from the domain of this meeting by enforcing node and arc consistency and secondly, must send the obtained slot times to the participants. At the end of this step, and after receiving all the answers from the participants, the agent must choose the date that maximizes the global utility of meeting $X_k^{A_i}$. If this date is used by another less significant meeting, $X_l^{A_j}$, where $W_{X_k}^{A_i} > W_{X_l}^{A_j}$, then the latter meeting $X_l^{A_j}$ must be changed. Therefore, the agent A_j (the Proposer of $X_l^{A_j}$) should be invited to relax its preferences. The proposed date must be communicated to all the participants. Each one of them must check the date and reply to the Proposer, and the same dynamic resumes until the system reaches its temporary stable equilibrium state (because of the dynamics of the system). We must note that in the worst case, all meetings $X_l^{A_j}$ with lower priority will be relaxed and the system will stop temporarily with the schedule of the meeting having the lowest priority. The revision of all the decisions to fix a new meeting (when adding a new meeting with highest priority) is slightly unrealistic. Then, in our system we propose applying a penalty (a decrement in the priority of the new meeting) according to the number of involved meetings that must be rescheduled at each step. The main goal of this new process is to speed up the search for the new optimal solution.

4.3.2. Relaxations

For each cancelled meeting $X_k^{A_i}$, the concerned agent must check if it can increase the utility of another already scheduled meeting (one or more by propagation). To achieve this goal, this agent must first examine all meetings $X_l^{A_j}$ (called candidate meetings) with $A_i \in$ Part$(X_l^{A_j})$ and $W_{X_k}^{A_i} > W_{X_l}^{A_j}$ (starting with the most important candidate meeting). In other words, the agent A_i can ask the participants in $X_l^{A_j}$ for further negotiation, if it realizes that this meeting can be held in the date that was taken by the cancelled meeting. Nevertheless, this process may increase the utility of some (or all) candidate meetings.

This protocol may also allow some meetings, which had been checked as non-possible, to be scheduled (e.g., some meetings that could not be scheduled before may become possible). In addition, we can assume a certain threshold for the meetings to be rescheduled and this is in order to avoid the need to reschedule all meetings in the worst case.

5. Evaluation

5.1. Theoretical evaluation

5.1.1. Termination

The MSRAC process stops temporarily (dynamic system) when the system reaches a stable equilibrium state. In this state all the agents are temporarily satisfied. An agent is satisfied when it has no meetings to schedule or when it has received all the confirmations from all the other Proposer agents. We assume that between t and t' there is no new (resp. cancelled) meeting. Thus, at time t, the number of meetings to be fixed is limited and finite, so the proposed approach stops after making at most this many meetings. We assume that MSRAC approach goes into an infinite loop. This may happen in two cases:

(1) While scheduling a meeting.
(2) While rescheduling a meeting.

For the first assumption, to schedule a meeting $X_l^{A_i}$ all the participants will cooperate together to find the best date for this meeting. The system will go into an infinite loop while scheduling $X_l^{A_i}$ if and only if the Participant agents reprocess the checked dates (cycle) when no solution is found. However, the number of possible dates per meeting is discrete and finite. In addition, every checked date is removed from the system to avoid a return to it later. The system will stop when a "good" date if found or when all possible dates for $X_l^{A_i}$ are processed and no possible solution has been found. Therefore our assumption is not true.

For the second assumption, the system goes into an infinite loop if the rescheduling of $X_l^{A_i}$ leads to the rescheduling of $X_p^{A_j}$ and the rescheduling of $X_p^{A_j}$ leads the same to the rescheduling of $X_q^{A_k}$ and finally the rescheduling of $X_q^{A_k}$ leads to the rescheduling of $X_l^{A_i}$. However the rescheduling of $X_l^{A_i}$ leads to the rescheduling of $X_p^{A_j}$ if and only if $W_{X_l} > W_{X_p}$ and the same for the other meetings[17]. Therefore $W_{X_l} < W_{X_p}$ means that the reschedule of $X_q^{A_k}$ will never lead to the reschedule of $X_l^{A_i}$. This contradicts our assumption.

We have to note that the satisfaction state of all the agents in a distributed system can be detected by taking a snapshot of the system, using the well-known Chandy–Lamport algorithm (Lamport and Chandy, 1985). The termination occurs when all agents are waiting for a message and there is no message in the transmission channels. The cost of the termination process can be mitigated by combining snapshot messages with our protocol messages.

5.1.2. Complexity

Let us consider the complexity of adding a new meeting into an existing schedule. The corresponding MS problem involves n for total number of users, d for the maximal number of possible dates for each meeting and c for the total number of preferred dates for each user. The total number of agents in this system is n the same as the total number of users. Suppose that each meeting involves n attendees and each user has m already scheduled meetings in the calendar. Our approach is composed of two steps.

- In the first step, each agent performs $O(cd + md + d\text{Log}(d))$ operations to reinforce node and arc-consistency and to rank the remaining dates. The Proposer agent then determines the intersection of the received sets of possible dates leading to $(n-1)d^2$ operations, and the utility of each dates with $O(nd)$. Thus, the temporal complexity of this step in the worst case is $O(n(cd + md + d\text{Log}(d)) + (n-1)d^2 + nd)$.
- In the second step, in order to compute the cost of rescheduling, we assume that at each step the chosen date leads to the rescheduling of one meeting (at most[18]) in the worst case. This leads to m successive iterations. Each agent checks its calendar m and sends its answer to the proposer in order to choose a new value. This process requires $O(m(nm + d))$ operations in the worst case. The space complexity, for all the agents, is $O(n(d + m))$.

5.1.3. Message passing optimality

In order to show that our approach requires the minimum amount of messages passing to reach an agreement among all the attendees, let us assume that we have n agents $\{A_1, \ldots, A_n\}$, each has an already scheduled meeting $X_1^{A_i}$ at the date d_l with $l \in \{1, \ldots, k\}$. The total number of scheduled meetings in the whole system is k at the dates $\{d_1, d_2, \ldots, d_k\}$.

Suppose that the agent A_j proposes a new meeting $X_2^{A_j}$ involving all the agents of the system. The possible dates for this meeting are $\{d_1, d_2, \ldots, d_k, d_{k+1}\}$. According to the mast of the approaches proposed in the literature (including Franzin et al., 2002; Tsuruta and Shintani, 2000; Franzin et al., 2004), a proposal $d_h (h \in \{d_1, d_2, \ldots, d_k, d_{k+1}\})$ is selected by agent A_j and passed to all the other agents. Each agent which receives this proposal, replies to the proposer only with a rejection or an acceptance. The same process resumes with another proposal given by another agent.[19] In this case *at least*

[17]The rescheduling of a meeting leads to the rescheduling of another meeting if and only if the first meeting is more "important" than the second.

[18]Because we assume that all the users are participants in all meetings.

[19]we assume that each agent has n possible proposals.

one agent will reject the proposal. Therefore to reach an agreement among all the participants, this process requires at least $2n(k+1)$ messages.

With MSRAC, the proposer sends all the possible candidate dates for a meeting to the participants. Each participant receiving this message will first reinforce arc consistency on the received possible dates in order to avoid as much fruitless backtracking as possible in the next steps. It then ranks the obtained set and sends it only to the Proposer, which determines the intersection of the received sets and obtains the agreement among all of them. Thus the number of required messages in MSRAC is $2n$. Note that receiving all the candidate meeting dates from participants may reveal some information about their local calendars (loss of some privacy). However the only information that Proposer agent A_i may deduce from a participant A_j is its non-availability for some dates. The non-availability of an attendee is due to many different reason, such as another meeting, a business trip, a vacation, personal preference, etc. The proposer cannot reveal the reason of the rejection of the candidate date but he may slowly collect more knowledge by asking for the same date or nearby dates. This is also a common problem for the other approaches. In the worst case the same amount of information will be revealed by all the approaches. Therefore, in order to decrease privacy loss for our approach, we propose to *hide* the identity of the sender. The Proposer agent will then get answers from the attendees without knowing to whom each answer belongs.

5.2. Experimental comparative evaluation

To evaluate the proposed MSRAC approach, we have developed the multi-agent dynamic with Actalk, an object-oriented concurrent programming language using the Smalltalk-80 environment. In our experiment, we generated random meeting problems. The parameters used are: n agents in the system, m meetings per agent, p participants in each meeting, D global calendar, d percentage of possible dates per meeting, w_c weights for the soft constraints to express users' preferences, while W_{X_l} weight of the event (variable) X_l (to express importance of the event) and c the control parameter. However, the weight of each hard constraint is equal to 1. We carried out three kinds of experiments to test the proposed approach. Table 4 summarizes the design of these experiments. The main goal of the first experiment was to evaluate the efficiency of the three issues proposed to be used in case of conflict. We used the three versions of MSRAC (defined in Section 3): (i) MSRAC-1, the deterministic approach in which agents try always to increase their (LU); (ii) MSRAC-2, the non-deterministic approach based on random choice; (iii) MSRAC-3, the non-deterministic approach based on the metropolis criteria. We generated random instances with different numbers of meetings to schedule, in order to vary the number of possible conflicts that may occur, with $n=10$, $m=\{5,8,10,15\}$, $p=6$,

Table 4
Summary of the different instance parameters used in the three kinds of experiments

Experiments I	$n=10$; $m=5,8,10,15$; $p=6$; $D=50$; $d=100\%$; $w_c \in [0..1]$; $W_{X_l} \in [1..20]$; $\lvert C_h \rvert = 10$; $T_p=10$ Each instance is executed 30 times.
Experiments II	$n=10$; $m=5$; $p=8$; $D=40$; $d \in \{12.5\%, 25\%, 37.5\%$ and $50\%\}$; $w_c \in [0..1]$; $W_{X_l} \in [1..20]$; $\lvert C_h \rvert = 10$; $c=50$ 10 instances generated for each d
Experiments III	*Groups* I–III, $n=10$; $m=\{5,8,10\}$; $p=7$; $D=50$; $d=60\%$; $w_c \in [0..1]$; $W_{X_l} \in [1..20]$; $\lvert C_h \rvert = 10$; $c=50$ *Groups* IV–VI, $n=20$; $m=\{10,15,20\}$; $p=13$; $D=100$; $d=60\%$; $w_c \in [0..1]$; $W_{X_l} \in [1..20]$; $\lvert C_h \rvert = 20$; $c=50$

$D=50$, i.e., each meeting starts between 8 AM and 6 PM, from Monday to Friday and is 1 h long, $w_c \in [0..1]$, $W_{X_l} \in [1..20]$,[20] $d=100\%$, $|C_h|=10$ and $T_p=10$. The total number of meetings per instance is, respectively 50, 80, 100, 150. Each instance is executed 30 times. For MSRAC-3 the initial temperature T_p is decreased slowly at each run.

Fig. 4 illustrates the obtained results of the three versions of MSRAC in term of the number of scheduled meetings while Fig. 5 illustrates the corresponding detected number of conflicts for the non-deterministic versions. Fig. 4 shows that for the most part, MSRAC-3 is closer to MSRAC-1 in number of scheduled meetings. MSRAC-2 oscillates more especially when the number of conflicts increases (Fig. 5(c2) and (d2)) leading to a great deterioration in the result (in Fig. 4(c1) the number of scheduled meetings vary from 34 to 44). This difference can be justified by the fact that MSRAC-3 accepts a deterioration of the LU (accept a meeting with lower LU) only when the difference in LU between the two conflicting meetings is small, with MSRAC-2 the selection of the meeting is totally random which may also increase the number of conflicts. Note that the number of generated conflicts for MSRAC-3 is almost always the same for all the runs, while there is a big variation for MSRAC-2 (Fig. 5(a2)–(d2)). Hence, using metropolis criterion to solve the conflict might be more appropriate than random choice and may lead to a better solution than the deterministic approach MSRAC-1 (Fig. 4(b1) and (c1)).

In the second kind of experiment, we used two approaches with MSRAC-3, which we will call MSRAC: Asynchronous Backtracking (Yokoo and Hirayama, 2000) (ABT) and Tsuruta's approach (Tsuruta and Shintani, 2000). Recall that, the ABT algorithm is used as a witness

[20]to increase the probability of having several meetings with same degree of importance.

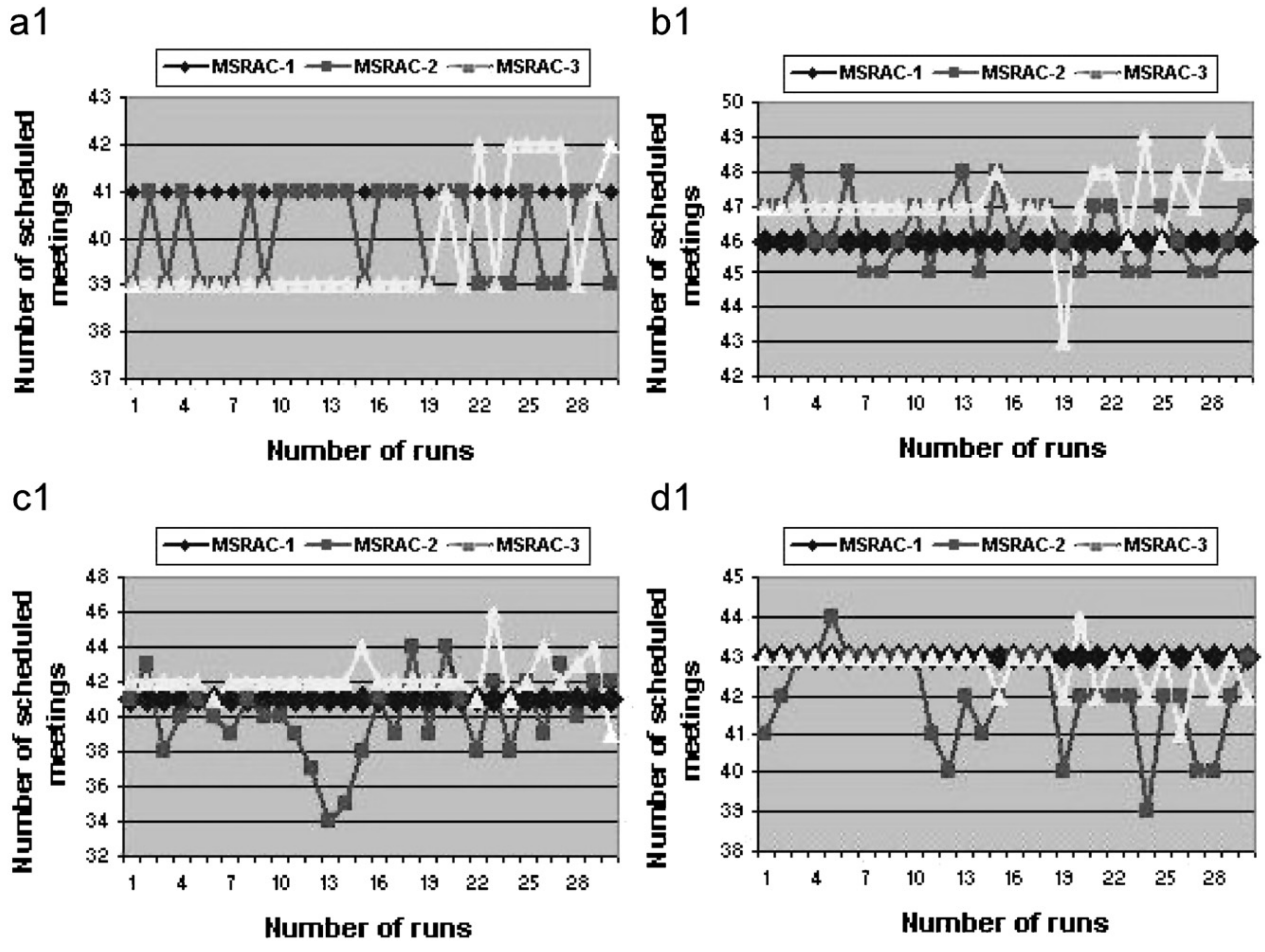


Fig. 4. Results obtained by the three approaches in mean of number of scheduled meetings (a1–d1).

approach to appraise the correctness of the results obtained with our approach. As mentioned in Section 1, ABT is a generic and complete algorithm for solving non-dynamic distributed constraint satisfaction problems. Therefore, for this algorithm all the applied problems are treated as static instances. Each agent in the system maintains one variable of the instance. The agents are ordered according to the degree of importance of the variables, i.e., degree of importance (W_{X_l}) of the underlying meeting X_l. The variables (meetings) sharing the same constraint (at least one same participant) are linked together. The approach in (Tsuruta and Shintani, 2000) presents some restrictions: on the one hand, the handle of the hard constraints (i.e., all the constraints could be relaxed by this approach) and on other hand, the discrimination between meetings. This approach independently processes all the proposed meetings without regard to their importance to either of the proposer or the attendees.

However, in the real world, meetings are not equivalent. Our approach tries then, in its solving process (second step), to schedule the most important meeting maintained by each agent first (unlike the approach in (Tsuruta and Shintani, 2000)). For this purpose and for the second kind of experiments, instances including hard constraints are randomly generated with $n = 10$, $m = 5$, $p = 8$, $D = 40$, $w_c \in [0..1]$, $W_{X_l} \in [0..1]$, $d \in \{12.5\%, 25\%, 37.5\%$ and $50\%\}$, $|C_h| = 10$ and $c = 50$. The total number of meetings per instance is 50. For each d we generated 10 instances, then measured the average of the results.

These results are expressed in terms of five criteria: the CPU time (in milliseconds), the number of scheduled meetings, the importance of the meetings, the measurement of real global utility, and the number of exchanged messages. Notice that the first three criteria allow us to especially measure the efficiency of MSRAC. To this end, we have introduced some modifications to the approach in (Tsuruta and Shintani, 2000) to make it worthwhile for both hard and soft constraints. We carried out the three approaches on the same generated examples using the same parameters.

To simulate a dynamic environment, at each time t each agent knows only about *one* of its meetings (an arbitrary one from its m meetings) and either schedules it or declares its failure to find a solution for it. Once finished the agent will receive a new meeting (another one chosen arbitrarily from the remaining meetings) with higher or lesser importance to process. Every new meetings may lead to the rescheduling of another scheduled one (depending on

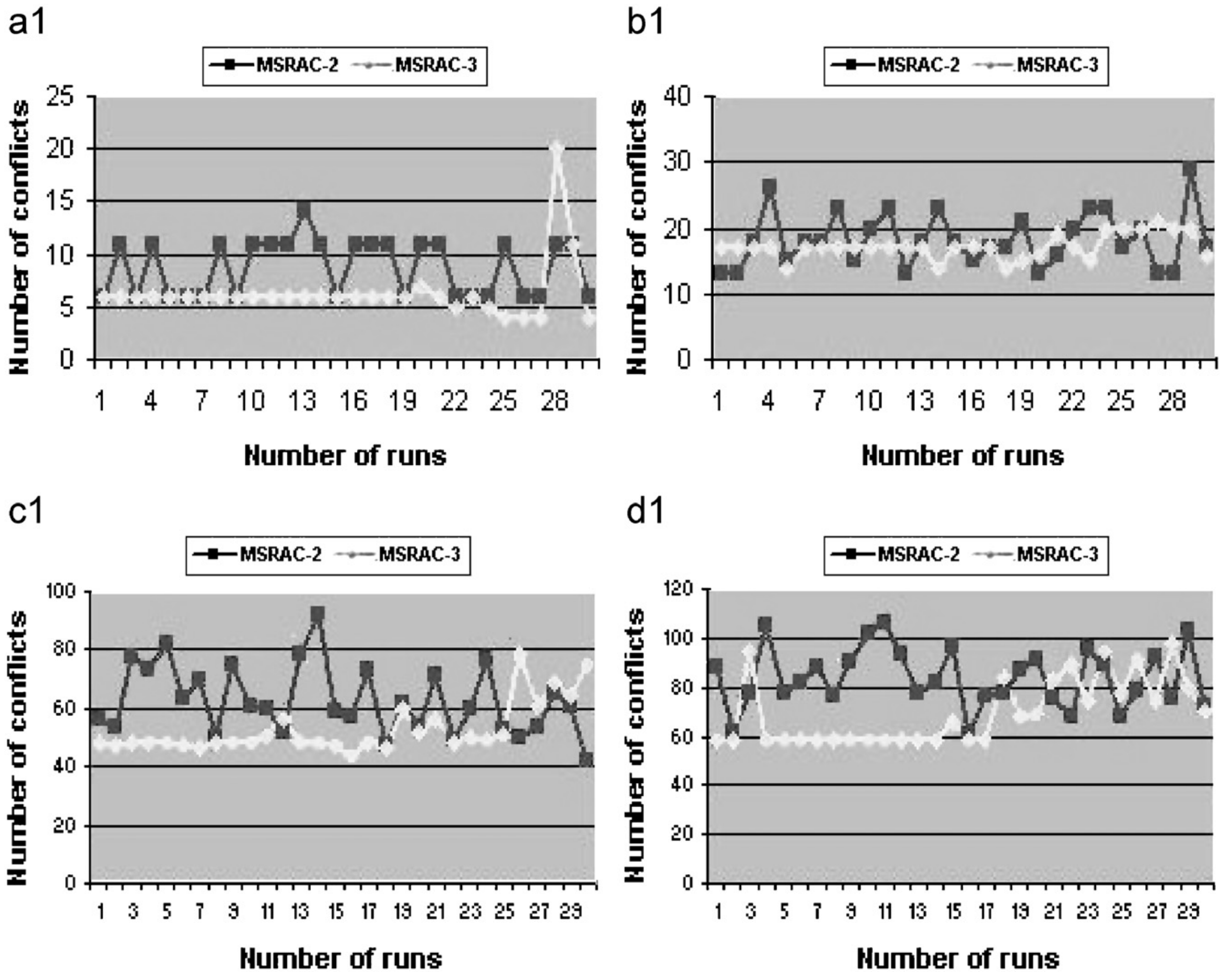


Fig. 5. The corresponding detected conflicts (a2–d2).

its importance and the candidate date that will be chosen). Hence at each time *t*, 1 or *n* new meetings might be added to the system according to the time required to process the previous ones.

The obtained results show that the MSRAC approach requires, in the majority of cases, less CPU time than the other approaches (Fig. 7(a)), while the CPU time needed by the approach in (Tsuruta and Shintani, 2000) is about three times more than that needed by our approach. This can be elucidated by the fact that the first step (reinforcement of local consistency) is useful in order to discard the dates that cannot be in any solution and consequently to avoid exploiting them in the solving process, which leads to CPU time consumption. Let us consider the case of over-constrained instances (possible dates less than or equal 25%). Fig. 7(a) shows that ABT requires less CPU time than MSRAC. The main reason is that in such instances, the number of conflicts between meetings is high which may lead to the augmentation of the number of rescheduled meetings. For ABT on the other hand, there is no conflict between meetings; the whole problem, the number of all the possible meetings that may occur in the system is static and known in advance.

As for the number of scheduled meetings (Fig. 6), ABT and MSRAC schedule almost the same number of meetings, while the Tsuruta approach schedules fewer meetings, than the other two approaches. This result shows the efficiency of MSRAC. The small difference noticed in the number of results given by ABT and MSRAC can be justified by the fact that MSRAC uses the *metropolis criterion* in case of conflict. Thus the final result depends on the decision taken towards conflicting meetings. Nevertheless, both approaches provide the same results for the degree of importance of the scheduled meetings (Fig. 6(b)) and the same real global utility (Fig. 6(c)).

In the case of over-constrained problems ($d = 12,5\%$), ABT requires fewer exchanged messages than our approach (Fig. 7(b)). This can be justified by the fact that for this kind of problem, the agents in ABT can discover merely the absence of solutions due to the low number of possible dates to check. While this number increases, the total number of exchanged messages increases also. With

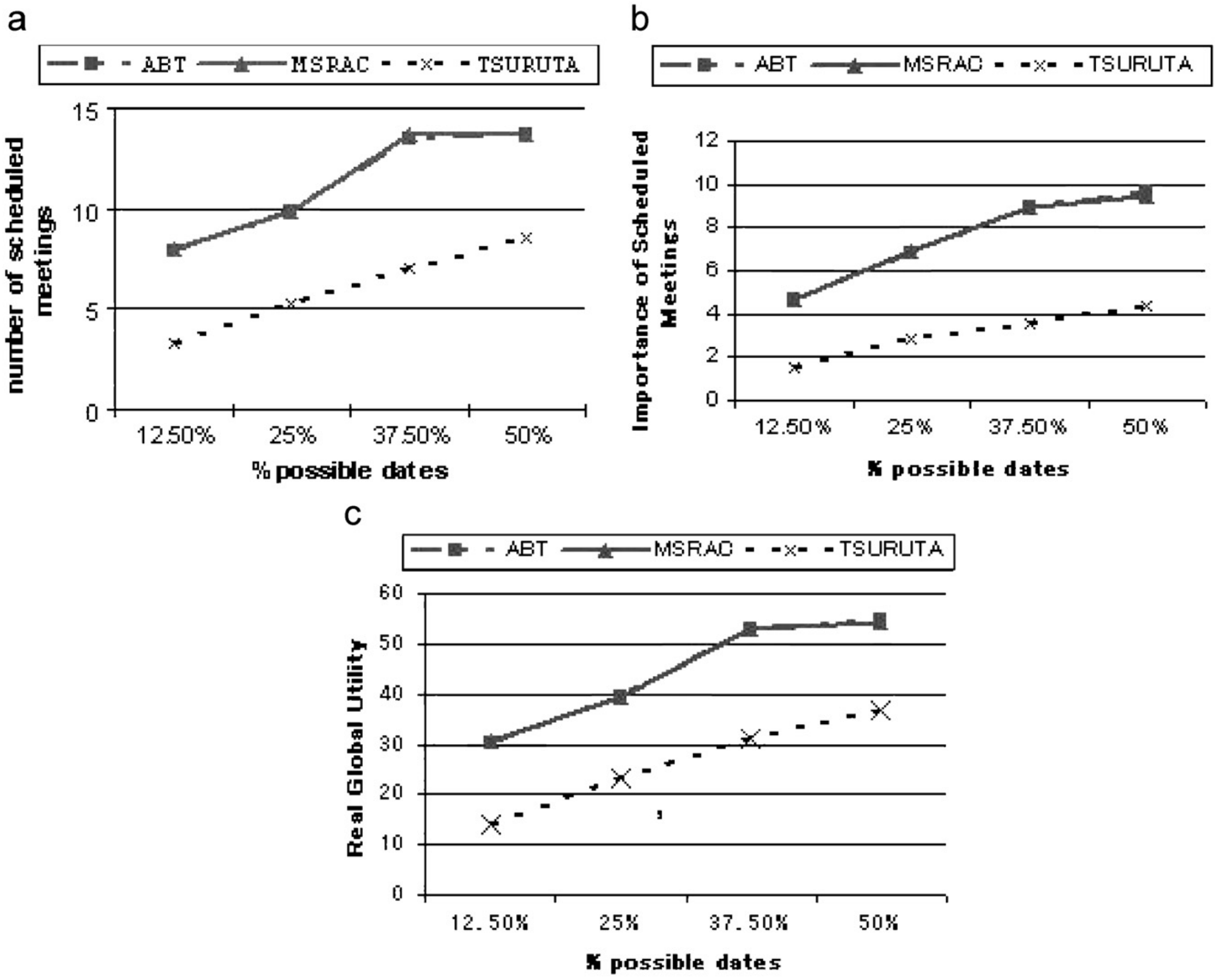


Fig. 6. Results obtained in mean of number of scheduled meetings, importance of scheduled meetings and the real global utility.

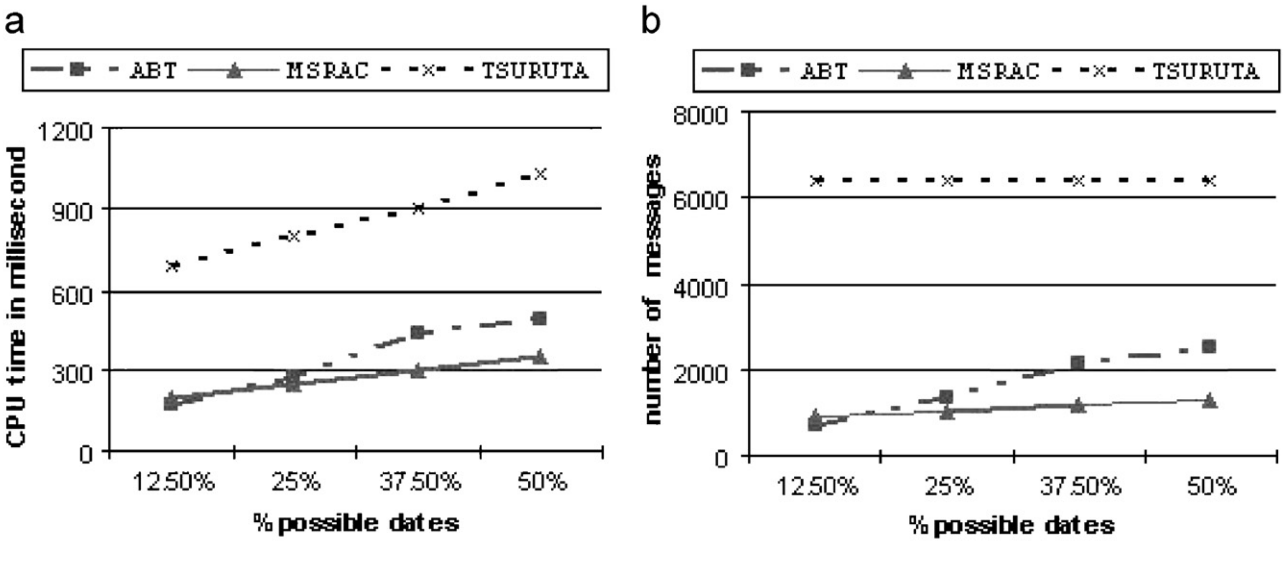


Fig. 7. Results obtained in term CPU time and exchanged messages.

Table 5
Results obtained in mean of CPU time

	Group I $\langle 10;5\rangle$	Group II $\langle 10;8\rangle$	Group III $\langle 10;10\rangle$	Group IV $\langle 20;10\rangle$	Group V $\langle 20;15\rangle$	Group VI $\langle 20;20\rangle$
Tsuruta App	1115.03	1635.51	2046	21334.6	30989.6	41105.1
ABT App	1264.38	4677.13	33186	78977.9	1685469	1179846
MSRAC	509	788.78	938	6468.44	8377.93	11948.5

Table 6
Results obtained in mean of percentage of scheduling meetings

	Group I $\langle 10;5\rangle$%	Group II $\langle 10;8\rangle$%	Group III $\langle 10;10\rangle$%	Group IV $\langle 20;10\rangle$%	Group V $\langle 20;15\rangle$%	Group VI $\langle 20;20\rangle$%
Tsuruta App	45	32	24	11	8	7
ABT App	59	42	33	26	18	14
MSRAC	67.4	46.6	36.2	28	20	16

our approach the number of conflicts increases for over-constrained problems, leading to more rescheduling and consequently to more exchanged messages. It is remarkable that the observed difference in the number of exchanged messages between ABT and MSRAC for over-constrained problems is negligible, while the approach in (Tsuruta and Shintani, 2000), requires more than three times the number of exchanged messages needed for our approach.

To highlight the scalability of our approach, we conducted a third kind of experiment in which we tried to increase the size of the problem. We generated six groups of random problems. The parameters of the three first groups (groups I–III) were: $n = 10$; $m \in \{5, 8, 10\}$; $D = 50$; $d = 60\%$; $p = 7$ and $|C_h| = 10$. While for the three last groups (groups IV, V and VI): $n = 20$; $m = \{10, 15, 20\}$; $D = 100$; $d = 60\%$; $p = 13$ and $|C_h| = 20$.

We generated 10 instances for each pair $\langle n, m\rangle$. Each instance was executed 10 times. Tables 5 and 6 show the average of the obtained results in term of CPU time and percentage of scheduled meetings for the three approaches. From these results, we can conclude that MSRAC is scalable, up to 4 times faster than the Tsuruta approach (case of $\langle 20;15\rangle$[21] and $\langle 20;20\rangle$) and up to 100 times faster than the ABT approach ($\langle 20;\ 20\rangle$). For the number of scheduled meetings, MSRAC and ABT planned almost same percentage of meetings, while for Tsuruta approach in (Tsuruta and Shintani, 2000), the number of scheduled meetings at each instance is about 50% of that achieved by the two other approaches. Hence, it is noteworthy that our approach seems to be more appropriate to real-world applications by dealing with users' hard constraints and by bringing forward consideration of discrimination among the proposed meetings. In addition, the first step of the proposed approach can prematurely detect the impossibility for reaching any agreement among all the participants.

[21]For 20 agents and 15 meetings per agent.

6. Conclusion

In this paper, we propose a new scalable and dynamic approach (MSRAC) to solve meeting scheduling (MS) problems. In this approach, we tried to integrate the main features of the MS problem such as: user preferences, user non-availability, importance of the meeting, etc. to reflect ideally real-world applications. To this end, we proposed to use two kinds of constraints to model the users' requirements: hard constraints to model the non-availability of a user and soft constraints to define the user's preferences. Note that the integration of these features transforms the problem into an optimization problem for which we define some criteria to reach the optimal (or good enough) solutions.

The multi-agent architecture, based on the DRAC model, associates an agent with each user and makes the agents interact by sending point-to point messages containing only relevant information. Basically, this approach consists of two steps. The first reduces the initial problem by reinforcing some level of local consistency (node and arc consistency). The second step solves the resulting MS problem while maintaining arc-consistency. In the proposed protocol, the information shared among all the agents is kept to a minimum without reflecting on the efficiency of the cooperative decision taken by all these agents.

All the meetings can be processed in a parallel and distributed manner, while achieving the meetings' higher utilities. This can be obtained as a side effect of interactions between the agents of the system, while both minimizing the amount of message passing and ensuring the user's privacy. We should note that the underlying protocol forbids only parallel meetings with common participants.

The MSRAC approach was compared with the ABT approach (Yokoo and Hirayama, 2000) and Tsuruta's approach (Tsuruta and Shintani, 2000). The obtained results show that our approach is efficient, scales better and performs less message passing for almost the same solutions. Nevertheless, industrial applications may lead to more complex problems, therefore, in future work, we will first evaluate the performance of this approach on real applications, then we will integrate the learning process.

Acknowledgments

The authors would like to thank the anonymous referees for their pertinent comments. Their careful reading of a first version of this paper greatly improved its presentation. This work is supported, in a part, by the Japanese Foundation for C&C Promotion Grant for non-Japanese Researcher to Ahlem Ben Hassine.

References

Abdennadher, S., Schlenker, H., 1999. Nurse scheduling using constraint logic programming. In: Proceedings Eleventh Annual Conference on Innovative Applications of Artificial Intelligence, IAAI-99, (pp. 838–843).

Bakker, R.R., Dikker, F., Tempelman, F., Wognum, P.M., 1993. Diagnosing and solving over-determined constraint satisfaction problems. In: Proceedings of 13th International Joint Conference on Artificial Intelligence, IJCAI-93 (pp. 276–281).

BenHassine, A., Ghedira, K., 2002. How to establish arc-consistency by reactive agents. In: Proceedings 15th European Conference on Artificial Intelligence, ECAI-02, pp. 156–160.

BenHassine, A., Ito, T., Ho, T.B., 2003. A new distributed approach to solve meeting scheduling problems. In: Proceedings IEEE/WIC International Conference on Intelligent Agent Technology IAT-03, pp. 588–591.

BenHassine, A., Ito, T., Ho, T.B., 2004a. Scheduling meetings with distributed local consistency reinforcement. In: the Lecture Notes in Artificial Intelligence of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-2004, pp. 708–717.

BenHassine, A., Defago, X., Ho, T.B., 2004b. Agent-based approach to dynamic meeting scheduling problems. In: Proceedings Third International Joint Conference on Autonomous Agents and Multi Agent Systems, AAMAS-04, pp. 1130–1137.

Dechter, R., Dechter, A., 1988. Belief maintenance in dynamic constraint networks. In: Proceedings of 7th National Conference on Artificial Intelligence, AAAI-88, pp. 37–42.

Ephrati, E., Rosenschein, J.S., 1991. The clarke tax as a consensus mechanism among automated agents. In: Proceedings of 9th National Conference on Artificial Intelligence, pp. 173–178.

Feldman, M., 1987. Electronic mail and weak ties in organizations. Office Technology and People 3, 83–101.

Freuder, E.C., Wallace, R.J., 1992. Partial constraint satisfaction. Artificial Intelligence 58 (1–3), 21–70.

Franzin, M.S., Freuder, E.C., Rossi, F., Wallace, R., 2002. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In: Proceedings AAAI-2002 Workshop on Preference in AI and CP.

Franzin, M.S., Freuder, E.C., Rossi, F., Wallace, R., 2004. Multi-agent meeting scheduling with preferences: efficiency, solution quality and privacy loss. Computational Intelligence on Preferences in AI and CP 20 (2), 264–286 (Special issue).

Feldman, A.M., 1980. Welfare Economics and Social Choice Theory. Kluwer, Boston.

Garrido, L., Sycara, K., 1996. Multi-agent meeting scheduling: preliminary experimental results. In: Proceedings of Second International Conference Multi-Agent Systems, ICMAS-96, pp. 95–102.

Lamport, L., Chandy, K.M., 1985. Distributed snapshots: determining global states of distributed systems. ACM Transaction on Computer Systems 3 (1), 63–75.

Luo, X., Leung,H., Lee, J.H., 2000. Theory and properties of selfish protocol for multi-agent meeting scheduling using fuzzy constraints. In: Proceedings of 14th European Conference on Artificial Intelligence, ECAI-00, pp. 373–377.

Mackworth, A.K., 1977. Consistency in network of relations. Artificial Intelligence 8, 99–118.

Montanari, U., 1974. Networks of constraints: fundamental properties and applications to picture processing. Information Sciences 7, 95–132.

Schiex, T., Fargier, H., Verfaillie, G., 1995. Valued constraint satisfaction problems: hard and easy problems. In: Proceedings of 14th International Joint Conference on Artificial Intelligence, pp. 631–637.

Sen, S., Durfee, E.H., 1994. On the design of an adaptive meeting scheduler. In: Proceedings of 10th IEEE Conference AI Applications, pp. 40–46.

Sen, S., Haynes, T., Arora, N., 1997. Satisfying user preferences while negotiating meetings. International Journal of Human-Computer Studies 47.

Sycara, K., Liu, J.S., 1994. Distributed meeting scheduling. In: Proceedings of 16th Annual Conference of Cognitive Society.

Tsuruta, T., Shintani, T., 2000. Scheduling meetings using distributed valued constraint satisfaction algorithm. In: Proceedings of 14th European Conference on Artificial Intelligence, ECAI-00, pp. 383–387.

Yokoo, M., Hirayama, K., 2000. Algorithms for distributed constraints satisfaction: a review. International Journal Autonomous Agent and Multi-Agent Systems 3 (2), 185–207.

Yokoo, M., Ishida, T., Kuwabara, K., 1990. Distributed constraints satisfaction for DAI problems. In: 10th International Workshop in Distributed Artificial Intelligence (DAI-90).