Scheduling Meetings using Distributed Valued Constraint Satisfaction Algorithm

Takuo Tsuruta and **Toramatsu Shintani** [1]

Abstract. Scheduling meetings is generally difficult in that it attempts to satisfy the preferences of all participants. However, all participants can agree to a schedule in which a portion of their preferences are not satisfied, since preferences are regarded in terms of their relative importance. In this paper, we formalize a meeting scheduling as a Distributed Valued Constraint Satisfaction Problem (DVCSP) and propose an algorithm for solving over-constrained problems formalized as a DVCSP by means of constraint relaxation based on importance. Our algorithm can relax lower priority constraints and schedule meetings that satisfy as many of the important constraints as possible under over-constrained conditions. We show a group schedule management system, consisting of multiple agents, and COLS which can concretely specify users' preferences as constraints. Our experiments show that our algorithm can discover a semi-optimal solution to over-constrained meeting scheduling problem in practical time. We can conclude that our algorithm is cost effective in comparison to another method that can find an optimal solution.

1 INTRODUCTION

Many recent commercial groupwares have functionalities to manage group schedule via the Internet(e.x. Lotus Notes, Microsoft Outlook). However, most of these systems merely assist communication among users. The purpose of our study is to implement a group schedule management system based on multiple agents which maintain their respective users' calendars and act on behalf of their respective users in meeting scheduling. We define "meeting scheduling" as the process of determining a starting time and an ending time of an event in which several people will participate. Many requirements (constraints) of these participants must be taken into account in scheduling meetings. Meeting scheduling is often over-constrained and no solution exists that can satisfy all constraints. Even so, it is appropriate to reach a consensus regarding meeting scheduling by relaxing a portion of all constraints. It is thus necessary to develop a method for scheduling meetings that satisfies as many of the important constraints as possible.

To address this requirement, we formalized a meeting scheduling problem as a Distributed Valued Constraint Satisfaction Problem (DVCSP) [7] , and implemented a meeting scheduling algorithm based on its formalization. In our system, an agent corresponds to each group member. This agent maintains its user's calendar and preferences for meetings and acts on behalf of its user in meeting scheduling ; users are able to keep information regarding their calendars and preferences private.

This paper consists of seven sections. In section 2, we show the definition of DVCSP and formalize a meeting scheduling problem as a DVCSP in section 3. In section 4, we propose our meeting scheduling algorithm. In section 5, we provide an outline for our group schedule management system, and in section 6 we discuss the results of our current experiments. We discuss related work in section 7, and finally, we make some concluding remarks.

2 DISTRIBUTED VALUED CONSTRAINT SATISFACTION PROBLEM

A Constraint Satisfaction Problem (CSP) is defined by $P = (X, D, C)$, where $X = x_1, \ldots, x_n$ is a set of variables, $D = d_1, \ldots, d_n (n \geq 1)$ is a set of finite domains for the variables, and $C = c_1, \ldots, c_k (k \geq 1)$ is a set of constraints [8, 10]. Each constraint c_j is defined over some subset of variables and limits the allowed combinations of variable values in the subset. Solving a CSP involves finding one set of assignments to variables X that satisfies all constraints C .

A distributed CSP (DCSP)[7, 12] can be considered a CSP in which variables and constraints are distributed among multiple agents. Each agent has some variables and tries to instantiate their values. Constraints may exist between the variables of different agents, so the instantiations of the variables must satisfy these inter-agent constraints. A DCSP is defined by $DP = (P, P_I)$, where $P = P_1, \ldots, P_m$ is a set of CSP instances. Each $P_i = (X_i, D_i, C_i)$ $(i = 1 \ldots m)$ represents a local problem instance in which X_i are disjoint sets of variables, D_i is a set of finite domains for the variables in X_i , and C_i is a set of constraints involving X_i . $P_I = (X_I, D_I, C_I)$ is an inter-agent CSP representing the connection of the local instances, where C_I is a set of inter-agent constraints, and X_I is a set of inter-agent variables on which inter-agent constraints hold. D_I is a set of finite domains for the variables in X_I .

Many problems in AI can be formalized as CSPs [10]. Examples of real-world problems, however, are over-constrained; no solution exists [1, 2, 6, 13]. The extension of the CSP model for this case is the Valued Constraint Satisfaction Problem (VCSP) [8]. This model gives a weight or a valuation to each constraint, reflecting the importance one gives to its satisfaction. We then search for a solution minimizing an aggregation of the valuations of the dissatisfied constraints.

A VCSP is defined by $VP = (X, D, C, S, \varphi)$, where (X, D, C) is a classical CSP, $S = (E, \otimes, \succ)$ is a valuation structure, and $\varphi : C \rightarrow E$ is a valuation function, giving

[1] Department of Intelligence and Computer Science, Nagoya Institute of Technology, Gokiso, Showa-ku, Nagoya 466-8555 JAPAN. {tsuruta,tora}@ics.nitech.ac.jp

the valuation of each constraint. E is the set of possible valuations; $\succ$ is a total order on E; $\top \in E$ is the valuation corresponding to a maximal dissatisfaction, and $\bot \in E$ is the valuation corresponding to a maximal satisfaction; $\otimes$, the aggregation operator, aggregates valuations. Let $\mathcal{A}$ be an assignment of values to all of the variables, namely a complete assignment. The valuation of $\mathcal{A}$ for the constraint c is defined as:

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

and the overall valuation of $\mathcal{A}$ by

$$\varphi(\mathcal{A}) = \underset{c \in C}{\otimes} \varphi(\mathcal{A}, c).$$

The Distributed Valued Constraint Satisfaction Problem (DVCSP) [7] is an extension of the VCSP framework to the distributed case. A DVCSP can be considered a VCSP in which variables and constraints are distributed among multiple agents. A DVCSP is defined by $DVP = (P, P_I)$, where $P = P_1, \ldots, P_m$ is a set of local VCSP instances. Each local VCSP instance $P_i = (X_i, D_i, C_i, S, \varphi_i)$ is equipped with its own valuation function φ_i. $P_I = (X_I, D_I, C_I, S, \varphi_I)$ is the inter-agent problem instance, and is equipped with the valuation function φ_I.

Let $\mathcal{A}$ be a complete assignment, and $\mathcal{A}_i$ the projection of $\mathcal{A}$ over the variables in X_i. $\mathcal{A}_i$ is the agent i's local assignment. In DVCSP, the valuation of $\mathcal{A}$ for the constraint c is defined as:

$$\varphi(\mathcal{A}, c) = \begin{cases} \varphi_i(\mathcal{A}_i, c) & \text{if } c \in C_i \\ \varphi_I(\mathcal{A}_I, c) & \text{if } c \in C_I. \end{cases}$$

The valuation of a complete assignment is then

$$\varphi(\mathcal{A}) = \underset{c \in C}{\otimes} \varphi(\mathcal{A}, c)$$

where $C = (\bigcup_{i=1}^{m} C_i) \cup C_I$ is the set of all constraints.

3 MEETING SCHEDULING PROBLEM

In the meeting scheduling problem, we designed two kinds of agents: a group agent and a personal agent. A personal agent is associated with each user and maintains its user's calendar and acts on behalf of its user in scheduling meetings. A group agent is the facilitator in a group and maintains group information.

A schedule element is designated an "event". The schedule consists of a number of events. Each event has three parameters: starting time, ending time, and event information. Event set E_i that agent i maintains is defined as the following:

$$\begin{aligned} E_i &= \{e_{i1}, e_{i2}, \ldots, e_{ij}, \ldots, e_{in}\} \ (n \geq 0) \\ e_{ij} &= (st, et, Nt) \ (st \in T, et \in T, st < et) \\ T &= \{(date, min) \mid \text{a finite set of time and date}\}, \end{aligned}$$

where e_{ij} is one of the events that agent i has to attend, st is a starting time, et is an ending time, and Nt represents the event information. The st and et have as its value an element of the finite set of time and date T. Each element in T consists of time min and date $date$. The $date$ represents the specified number of dates that have passed since January 1, 1970, and min represents the specified time by the minute (i.e., December 22, 1999, at 3:10 P.M. is $(10947, 910)$).

Constraint c_{ij} is defined as the requirements for scheduling a meeting and the participants' preferences. In addition, in our formalization, events that have already been registered are regarded as constraints. That is, the event e_{ij} that has already been registered is converted into the constraint " Can't schedule any event between time st_j (start time of e_{ij}) and time et_j (end time of e_{ij}) ".

The constraint c_{ij} and the event e_{ij} are assigned a weight, which is an integer between 1 and 9, by an user. The weight reflects the importance of the constraint or event, and a constraint (or a event) with a larger importance value is considered more important.

We formalize a meeting scheduling problem as a DVCSP. A meeting scheduling problem is defined by $MP = (P, P_I)$. $P = P_1, \ldots, P_m$ is the set of VCSPs placed on the participants' personal agents. The agent i's VCSP is $P_i = (X_i, D_i, C_i, S, \varphi_i)$, where X_i is the set of starting time st and ending time et for each event e_{ij}, D_i is the time set T, and C_i is the set of all constraints relevant to st and et for events to be managed by the agent i. S is the valuation structure defined by $E = [0, 9]$, $\succ = >$, $\bot = 0$, $\top = 9$, $\otimes = +$ and the valuation function φ_i is defined by the user. The inter-agent VCSP $P_I = (X_I, D_I, C_I, S, \varphi_I)$ is a VCSP of the group agent, where X_I is the set of st and et for all meetings, D_I is T, and C_I is the set of all group constraints. S is same as that of P, and function φ_I is the degree of constraint relaxation when meetings are scheduled.

4 MEETING SCHEDULING ALGORITHM

In the scheduling process, a group agent sends a proposal for a meeting to participants' personal agents. $PR_k = (Ts, Tr, Nt)$ is a proposal, where Ts is the set of windows in which the meeting may be scheduled, Tr is a threshold for the weights of constraint, Nt is information about the meeting, and k is the proposal identifiers. A proposal PR_k is defined as the following:

$$\begin{aligned} PR_k &= (Ts, Tr, Nt) \\ Ts &= \{t_1, t_2, \ldots, t_i, \ldots, t_k\} \ (k \geq 1) \\ t_i &= [x, y] \ (x, y \in T, x < y) \\ Tr &\in [1, 9], \end{aligned}$$

where t_i is a window in which the meeting may be scheduled, representing the interval from time x to time y. The threshold for weights Tr is assigned an integer between 1 and 9.

In the scheduling process, a personal agent sends a reply to a group agent. R_k is a reply, where a set of constraints c_{ki} exists for scheduling. R_k is defined as the following:

$$R_k = \{c_{k1}, c_{k2}, \ldots, c_{ki}, \ldots, c_{kn}\} \ (n \geq 1).$$

In the meeting scheduling process, when a new meeting is proposed, new variable st and et are included in the variable set X_I of the group agent's VCSP P_I, and the constraints given by the proposer are included in the constraint set C_I. The meeting scheduling algorithm is then executed to find an assignment to new variables.

Our meeting scheduling algorithm is shown in Figure 1 . In the following, we present a sketch of each step of our algorithm.

```
when $i$ received ( request ( SendAgent , Date , ST , ET , Parts ) )
    Ts $\leftarrow$ proposed time slots based on Date , ST , and ET
    Tr $\leftarrow$ 1
    broadcast " proposed ( $i$ , Ts , Tr , Note ) " to Parts ........ (a)
end

when $i$ received ( proposed ( SendAgent , Ts , Tr , Note ) )
    Events_list $\leftarrow$ events scheduled for Ts
    Obj_list $\leftarrow$ translates Events_list into constraints
    Consts $\leftarrow$ constraints related to Ts
    Const_list $\leftarrow$ pick_up ( Obj_list $+$ Consts , Tr )
    send " replied ( $i$ , Const_list , Tr ) " to SendAgent ......... (b)
end

when $i$ received ( replied ( SendAgent , Consts , Tr ) )
    store Consts as constraints
    if received all replies then
        G_Events $\leftarrow$ events scheduled for Ts
        Obj_list $\leftarrow$ translate G_Events into constraints
        G_Consts $\leftarrow$ constraints related to Ts
        Const_list $\leftarrow$ pick_up ( Obj_list $+$ G_Consts , Tr )
        P_Consts $\leftarrow$ all constraints in replies of all participants
        search for the schedule
                such that Const_list $+$ P_Consts are satisfied ...... (c)
        Tr $\leftarrow$ Tr $+$ 1
        if the meeting can be scheduled
        then broadcast the schedule to all participants
        else if Tr $=$ 10
            then broadcast the failure to all participants ......... (d)
            else
                broadcast " proposed ( $i$ , Ts , Tr , Note ) "
                to all participants ................................ (e)
            end if
        end if
    end if
end

procedure pick_up ( Obj_list , Tr )
    for $j$ $\leftarrow$ 1 to number of Obj_list do
        if weight of Obj_list [ $j$ ] $\geq$ Tr
            or ( Obj_list [ $j$ ] is a meeting negotiated on other groups
                and the threshold of Obj_list [ $j$ ] $\geq$ Tr ) ......... (f)
        then Const_list $\leftarrow$ Const_list $+$ Obj_list [ $j$ ]
    end for
    return Const_list
```

Figure 1. Meeting scheduling algorithm

(Step 1) A group agent generates a Ts based on the constraints for starting and ending time included in a message sent by a proposer, and sends a proposal PR_k to all participants' agents (Fig. 1.(a)).

(Step 2) Each personal agent receiving the proposal PR_k sends a reply R_k to the group agent (Fig. 1.(b)). R_k consists of constraints for Ts and constraints that prohibit times that overlap already scheduled events. The constraints in R_k , however, have a weight greater than or equal to Tr , and include constraints for meetings that have been negotiated with the other groups, which, as constraints, have a threshold that is greater than or equal to Tr (Fig. 1.(f)).

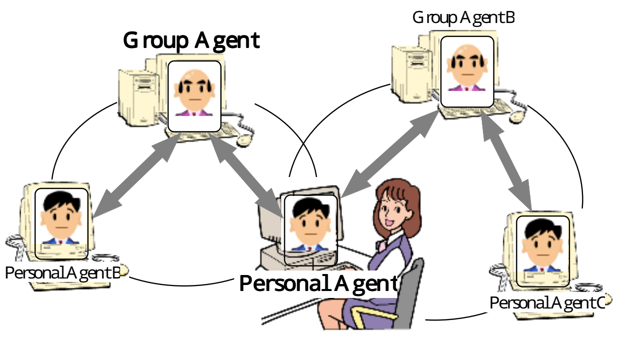


Figure 2. Overview of the group schedule management system

(Step 3) Once the group agent receives all participants' replies, it searches for an assignment to new variables st and et that will satisfy the constraints of all replies and the constraints C_I (Fig. 1.(c)). This search is executed by means of a tree search using a modified version of forward checking [4].

(Step 4) **(i)** If an assignment cannot be found, the group agent sends a new proposal PR_k to all participants' agents (Fig. 1.(e)). This proposal consists of Ts , the new threshold $Tr + 1$ (i.e., 1 is added to the last proposal's threshold), and Nt . This process is repeated from **(Step 2)**. However, if the new threshold $Tr + 1$ reaches 10, the group agent broadcasts a failure message to all participants (Fig. 1.(d)).

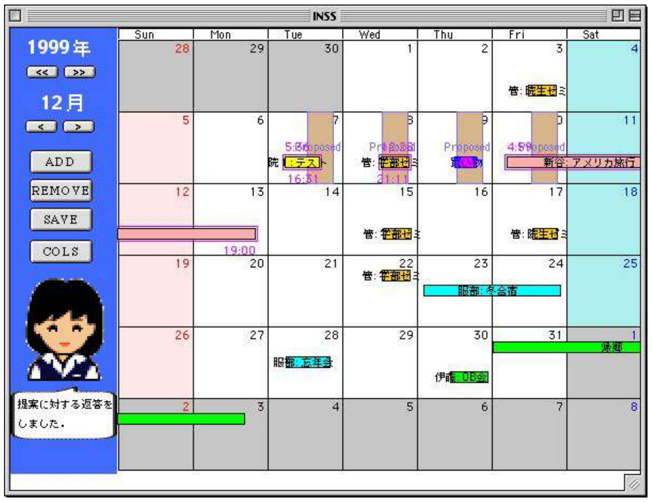
(ii) If an assignment can be found, those values are assigned to new variables st and et . However, if there are several possible assignments, the proposer selects which values are assigned to the new variables. The group agent then broadcasts this meeting's schedule to all participants' agents.

We would like to emphasize that in our algorithm the participant's agent sends constraints based on the threshold for weight to the group agent, and the number of constraints in the set R_k , sent in **(Step 2)**, decreases whenever a proposal is revised. It thus can relax lower priority constraints and find an assignment that satisfies as many of the important constraints as possible in the over-constrained problem.

5 GROUP SCHEDULE MANAGEMENT SYSTEM

In this section, we present the outline of our group schedule management system (implemented by Java), which can support personal schedule management, group sharing calendars, and meeting scheduling that function according to our meeting scheduling algorithm. Our system consists of group agents and personal agents corresponding to each group member, as shown in Figure 2. In our system, one group consists of one group agent and its group members' personal agents. Our system may have multiple (i.e., overlapping) groups.

In Figure 2, a personal agent is associated with each user. The personal agent maintains its user's calendar and acts on behalf of its user in sharing calendars and meeting scheduling. A group agent is the facilitator in a group and maintains group information. Figure 3 is an example of the personal agent's user interface. The agent has a calendar window as a graphical user interface. A user manages his calendar through this interface and schedules meetings through interaction with a personal agent.



提案に対する返答をしました。

Figure 3. The user interface of the personal agent

```
scheduling(Date,ST,ET,Part,Title,Memo,Author) <-
    1999/12/7 < Date < 1999/12/10,
    13:00 < ST < 13:30,
    13:00 < ST < 15:00,
    18:00 < ET < 21:00,
    weight([$1,8],[$2,3],[$3,6],[$4,4]),
    Part = [tsuruta, hatto, shintani],
    Title = a recreation in the seminar,
    Memo = tennis,
    Author = tsuruta.
```

Figure 4. An example of description in COLS

We designed COLS (COnstraint script Language for meeting Scheduling) [11] to describe constraints regarding meeting scheduling, operators, information concerning events and meetings, and communication messages. Figure 4 represents an example of the description of a meeting request in COLS. In Figure 4, an operator " `<-` " in line 1 divides a part of the predicate (the former) and a part of constraints for arguments against the predicate (the latter). A predicate " `scheduling` " in line 1 represents meeting requests and has seven arguments. Its " `Date` " is a variable that refers to a range of dates within which the meeting can be held. In line 2, a constraint " one day between December 7, 1999 and December 10, 1999 " for " `Date` " is described. In lines 3 and 4, constraints for the variable " `ST` " indicate the range of the meeting's potential starting time. In line 5, constraints for the variable " `ET` " indicate the range of the meeting's potential ending time. From line 7 to line 10, the variable " `Part` " indicates the participant list; " `Title` " indicates the topic of the meeting; " `Memo` " indicates a memo; " `Author` " indicates the member who requested the meeting. Line 6 describes the weight for each constraint; for example, " `[ $1 , 8 ]` " means that the weight for the first constraint " `1999/12/7 < Date < 1999/12/10` " is 8.

6 EXPERIMENTS AND EVALUATION

This section presents experiments and evaluations of our meeting scheduling algorithm. In our experiments, we consider the scheduling process in one group that consists of one group agent and five personal agents, and calendars of seven days with twelve hours per day. We used meeting scheduling problems which are randomly generated based on several parameters in order to specify one meeting scheduling problem. Their parameters are (1) number of agents in the group a , (2) number of meetings m , (3) number of participants per meeting p , (4) number of events per agent e , (5) number of constraints per agent c , (6) number of possible time slots t , (7) maximal duration of event d (any event has one value between $[1, d]$ as duration), (8) maximal value of weight w (any event and constraint have one value between $[1, w]$ as weight) . In our experiments, we generated 25 problems using every combination of the sum of number of events and constraints $e+c = \{4, 8, 12, 16, 20\}$, with the other parameters set at $a = 5$, $m = 3$, $p = 3$, $t = 84$, $d = 5$, $w = 9$.

In our experiments, we applied our meeting scheduling algorithm to problems that were randomly generated based on the above parameters, and we measured average computation time (*time*) until our algorithm found a solution, and average aggregation of the valuations of the dissatisfied constraints (*valuation*). In addition, we applied the DOC method [2] to same problems in order to find an optimal solution, and we measured *time* until the DOC method found a solution, and *valuation* , and we then compared its results with the results of our algorithm. This system was implemented by Java and experiments were executed on a PowerMacintosh G3 (PowerPC G3 400MHz).

Table 1 presents the experimental results. $c+e$ in Table 1 indicates the sum of the numbers of events and constraints per agent, *solvability* in Table 1 indicates the ratio of problems that can be solved as CSP to all generated problems, and *max.valuation* indicates the average maximal valuation of each problem, that is, the average sum of the weights of all constraints and events of each problem. Table 1 shows that our algorithm performs faster with respect to *time* than does the DOC method, while the DOC method finds a better solution with respect to *valuation* than does our algorithm. The greater $c + e$ becomes, the greater these differences become. For instances of $c+e = 20$, our algorithm finds a solution in 1.08 seconds, while the DOC method finds a solution in 44.06 minutes. However, the average valuations of solutions (*valuation*) found by our algorithm is 107.9 , while on the contrary, that found using the DOC method is 34 . The average of the ratio of valuation over maximal valuation (*relaxation*) is 20.73% in our algorithm, but is 6.52% using the DOC method.

These results indicate that determining an optimal solution for meeting scheduling problems is very expensive. Our algorithm can discover a semi-optimal solution to over-constrained meeting scheduling problem in practical time. We can conclude that our algorithm is cost effective in comparison to the DOC method. Since optimal solutions are not necessarily desired and high computation costs are avoided, our algorithm is superior.

7 RELATED WORK

Lemaitre [7] has proposed a formalization of the DVCSP and a greedy repair distributed algorithm for solving the DVCSP. This algorithm is based on successive greedy repairs computed in turn by agents. In this algorithm, during an agent's turn, other agents must not change their local assignment. In contrast, we apply a distributed

problem class			our algorithm			DOC method		
$c + e$	*solvability*(%)	*max.valuation*	*time*(msec)	*valuation*	*relaxation*(%)	*time*(msec)	*valuation*	*relaxation*(%)
8	100	231.6	51	0	0	54	0	0
12	52	329.9	627	7.9	2.42	86219	2.1	0.65
16	8	430.8	739	39.7	9.32	1437767	11.3	2.63
20	0	519.7	1082	107.9	20.73	2643352	34	6.52

Table 1. Experimental results

synchronous algorithm to the meeting scheduling problems formalized as DVCSP, and some agents can change their local assignment simultaneously.

The Distributed Partial Constraint Satisfaction Problem(DPCSP) [5] resembles the DVCSP framework. DPCSP is one of the abstract frameworks that formalize distributed over-constrained CSPs, and is an extension of the Partial CSP (PCSP) [3] framework to the distributed case. The DVCSP that we appropriate, in contrast, is an extension of the VCSP framework, which is a subclass of the PCSP, to the distributed case. That is, DPCSP is a super class of DVCSP.

Research on scheduling based on CSP includes studies by [1, 2, 6, 13] which focus on over-constrained CSPs. In their work, scheduling problems are formalized as centralized CSPs in which all variables and constraints are centralized in one process. In our work, in contrast, scheduling problems are formalized as a distributed CSP in which the variables and constraints are distributed among multiple processes. In the centralized model, it is difficult to keep information (e.g. users' calendars and preferences) private because all variables and constraints are centered on one process. In the distributed model, in contrast, because the variables and constraints are distributed among multiple agents and each agent communicates only fundamentally necessary information to other agents, information can be kept private.

Recent work on distributed meeting scheduling based on multiagents includes the Sen's work [9]. In this work, meetings are scheduled based on users' preferences, thresholds for these preferences, and weights for the options in each preference. The user's preferences are related to time of day, day of week, host, other invitees, topic of the meeting, etc., and are specified by assigning a value to each parameter. For example, each user assigns a value between 0 and 1 for every member of the group in order to specify his or her preferences for host and other invitees. The weights for the options in each preference reflect the importance of the options themselves. Our work differs from the Sen's research in that we formalized a meeting scheduling problem as a DVCSP , in which the user's preferences are described as constraints, and which can concretely specify user's preferences in COLS.

8 CONCLUSIONS

In this paper, we formalized a meeting scheduling problem as a Distributed Valued Constraint Satisfaction Problem. We proposed an algorithm for solving over-constrained problems formalized as a DVCSP by means of constraint relaxation based on importance. Our algorithm can relax lower priority constraints and schedule meetings that satisfy as many of the important constraints, which concretely specify user's preferences in COLS, as possible. We described our group schedule management system, which consists of group agents and personal agents corresponding to each group member. Each agent can maintain its respective user's calendar and act on behalf of its user in meeting scheduling. Our experiments show that our algorithm can discover a semi-optimal solution to over-constrained meeting scheduling problem in practical time. We can conclude that our algorithm is cost effective in comparison to another method.

REFERENCES

[1] S. Abdennadher and H. Schlenker, 'Nurse scheduling using constraint logic programming', in *Proc. of the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pp. 838–843, (1999).

[2] R.R. Bakker, F. Dikker, F. Tempelman, and P.M. Wognum, 'Diagnosing and solving over-determined constraint satisfaction problems', in *Proc. of the 13th International Joint Conference on Artificial Intelligence (IJCAI93)*, pp. 276–281, (1993).

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

[4] R.M. Haralick and G.L. Elliott, 'Increasing tree search efficiency for constraint satisfaction problems', *Artificial Intelligence*, **14**, 263–313, (1980).

[5] K. Hirayama and M. Yokoo, 'Distributed partial constraint satisfaction problem', in *Proc. of the Third International Conference on Principles and Practice of Constraint Programming (CP-97)*, pp. 222–236, (1997).

[6] Harald Meyer auf́m Hofe, 'ConPlan/SIEDAplan: Personnel assignment as a problem of hierarchical constraint satisfaction', in *In Proc. of the 3rd International Conference on Practical Applications of Constraint Technologies*, pp. 257–272, (1997).

[7] M. Lemaitre and G. Verfaillie, 'An incomplete method for solving distributed valued constraint satisfaction problems', in *Proc. of AAAI-97 Workshop on Constraints and Agents*, (1997).

[8] T. Schiex, H. Fargier, and G. Verfaillie, 'Valued constraint satisfaction problems: Hard and easy problems', in *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 631–637, (1995).

[9] S. Sen, T. Haynes, and N. Arora, 'Satisfying user preferences while negotiating meetings', *International Journal of Human-Computer Studies*, **47**, 407–427, (1997).

[10] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, 1993.

[11] T. Tsuruta and T. Shintani, 'Implementation of a group schedule management agent based on a constraint script language', in *Proc. of the 57th Annual Conference of Information Processing Society of Japan*, pp. 400–401, (1998).

[12] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara, 'The distributed constraint satisfaction problem: formalization and algorithms', *IEEE Trans. on Knowledge and Data Engineering*, **10**(5), 673–685, (1998).

[13] M. Yoshikawa, K. Kaneko, T. Yamanouchi, and M. Watanabe, 'A constraint-based high-school scheduling system', *IEEE Expert*, **11**(1), 63–72, (1996).