Multiagent Meeting Scheduling with Rescheduling

Pragnesh Jay Modi and Manuela Veloso

Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213
{pmodi,mmv}@cs.cmu.edu

Abstract. We are interested in how personal agents who perform calendar management on behalf of their human users can schedule meetings effectively. A key difficulty of concern is deciding when to reschedule an existing meeting in favor of a new meeting. We model the meeting scheduling problem as a special subclass of distributed constraint reasoning (DCR) called the Incremental, Limited Information Exchange Multiagent Assignment Problem (IL-MAP). Key novelties of our approach include i) a focus on incremental scheduling, ii) scheduling under a limited information exchange paradigm and, iii) using models of other agents to schedule more effectively. Our results are the first in DCR to show how models of other agents can be used to improve problem solving performance.

1 Introduction

Meeting scheduling is a time consuming routine task that when delegated to a personal assistant agent promises to significantly reduce daily cognitive load. A key competency of agents who do meeting scheduling is their ability to coordinate schedules such that all attendees of a meeting agree on its start time [3, 9, 2]. The problem is challenging in part because a) each agent chooses its own schedule, i.e., scheduling is *distributed*, b) new meetings are introduced over time, i.e., scheduling is *incremental*, and c) agents are *limited* in the information they can exchange. This article provides an approach to multiagent meeting scheduling using the Distributed Constraint Reasoning (DCR) paradigm [1, 4, 5, 10, 11]. Previous researchers have proposed DCR as a framework for multiagent coordination and considerable progress has been made over the last several years. However, novel techniques are needed to address the challenges described above.

The main idea in this paper is to exploit given models of "scheduling difficulty" with other agents' in order improve meeting scheduling performance. The specific hypothesis we investigate is that an agent can use models of the calendar density of other agents where we assume that the calendar density is correlated with the agent's rank in an organization. This is novel because to our knowledge, existing methods for DCR have not investigated how to take advantage of learned or given models of other agents to aid in making scheduling decisions. Further, we evaluate our approach in an *incremental scheduling* paradigm, in which new meetings must be scheduled in the context of an existing schedule. Existing DCR approaches have focused primarily on batch problem solving and are not designed for minimizing disruption to an initial given solution. Finally, we assume that communication between agents is limited. We explicitly prohibit the communication of information about variables between agents who are not involved in the variable's value assignment. This restriction is motivated by the meeting scheduling domain in which schedule privacy is a key concern. Existing DCR algorithms typically communicate "context" information which does not adhere to this restriction.

We first formalize the meeting scheduling problem by defining a special form of DCR which we call the Incremental, Limited Information Exchange Multiagent Assignment Problem (IL-MAP). IL-MAP requires agents to assign values to variables where multiple agents must agree on value assignments but are limited in what and to whom information can be communicated. Second, we describe a basic distributed protocol for IL-MAP in which an initiator proposes assignments to others who either agree or refuse the proposed assignments based on their own existing assignments. The protocol conforms to our need for limited information exchange by only communicating allowed information to relevant agents. Third, we use this basic protocol to investigate using models of scheduling difficulty with other agents to increase effectiveness of the multiagent meeting scheduling process. Finally, we demonstrate that our approach improves scheduling effectiveness in an agent organization hierarchy where the lower ranked agents have lower calendar density than the higher ranked agents in the hierarchy.

The multiagent meeting scheduling problem has been previously investigated but methods for making effective rescheduling decisions is lacking. Sen and Durfee [9] formalize the problem and identify a family of negotiation protocols aimed at searching for feasible solutions in a distributed manner. However, rescheduling of existing meetings or modeling of other agents to improve performance is not a major focus. Sen and Durfee also describe a contract-net approach for multiagent meeting scheduling [8] and in this context, rescheduling and cancellation of existing meetings is discussed. The critical issues are raised and a rich decision making framework is presented but is mainly theoretical. Our research represents a further investigation of some of the critical issues raised by them. Freuder, Minca and Wallace [2] have previously investigated meeting scheduling within the DCR framework where the primary motivation was to investigate tradeoffs between efficiency of scheduling and loss of privacy, but not issues of incremental problem solving or agent modeling are not addressed.

2 Meeting Scheduling as Distributed Constraint Reasoning

We view meeting scheduling as a distributed problem in which each agent manages and is responsible for its own calendar. A centralized approach is also possible in which a single server is assumed to have access to each agent's calendar and makes scheduling decisions for all agents. However, a centralized approach has several drawbacks including that it requires agents to reveal potentially private calendar information to the central server.

We use the Distributed Constraint Reasoning (DCR) paradigm [11] to model distributed meeting scheduling. DCR is defined by a set of variables where each variable is assigned to an agent who has control of its value, and agents must choose values for their assigned variables so that a given set of constraints are satisfied or optimized. Constraints between variables assigned to the same agent are called *intra-agent* constraints, while constraints between variables assigned to different agents are called *inter-agent* constraints. To ensure that inter-agent constraints are satisfied, agents must coordinate their choice of values for variables through a communication protocol.

2.1 The Multiagent Assignment Problem (MAP)

In this section, we introduce an important subclass of DCR which we call the *multiagent assignment problem* (MAP) . In MAP, we assume that agents must map elements from one set, which are modeled as the variables, to elements of a second set, which are modeled as the values. Importantly, we assume multiple agents need to agree on the assignment of a value to a given variable. Since decision-making control is distributed among the agents, this "agreement" requirement raises many unique challenges.

We define MAP as follows.

- $\mathcal{A} = \{A_1, A_2, ..., A_n\}$ is a set of *agents*.
- $\mathcal{V} = \{V_1, V_2, ..., V_m\}$ is a set of *variables*.
- $\mathcal{D} = \{d_1, d_2, ..., d_k\}$ is a set of *values*.
- $participants(V_i) \subseteq \mathcal{A}$ is a set of agents who are assigned the variable V_i.
- $vars(A_i) \subseteq \mathcal{V}$ is a set of variables assigned to agent A_i.
- For each variable V_i, an inter-agent *agreement* constraint is satisfied if and only if the same value from $\mathcal{D}$ is assigned to V_i by all the agents in $participants(V_i)$.
- For each agent A_i, an intra-agent *mutual exclusion* constraint is satisfied if and only if no value from $\mathcal{D}$ is assigned to more than one variable in $vars(A_i)$.

MAP has some similarities to the classical "assignment problem" from combinatorial optimization research[7]. Two key differences are that a) MAP requires distributed agents to agree on assignments and b) MAP does not yet model degrees of solution quality, only valid and invalid solutions. Further extension of MAP to model optimization problems is important future work.

2.2 Meeting Scheduling as MAP

We describe the multiagent meeting scheduling problem followed by its formulation as a MAP. Meeting scheduling requires meetings to be paired with timeslots subject to three constraints: a) each meeting is assigned to exactly one timeslot, b) each timeslot is paired with no more than one meeting, and c) all the attendees of a given meeting agree on its assigned timeslot. The goal of the following model is to represent these three constraints.

We define the meeting scheduling problem as follows.

- $\mathcal{A} = \{A_1, A_2, ..., A_n\}$ is a set of agents.
- $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ is a set of meetings. We assume each meeting has the same duration d.
- *attendees*$(M_i) \subseteq \mathcal{A}$ are the attendees of meeting M_i.
- *meetings*$(A_i) \subseteq \mathcal{M}$ are the meetings of which A_i is an attendee.
- *initiator*$(M_i) \in attendees(M_i)$ is the designated initiator of meeting M_i.
- $\mathcal{T} = \{T_1, T_2, ..., T_p\}$ is a set of discrete non-overlapping contiguous timeslots of length d.
- $\mathcal{S}_{init} = \{S_1, S_2, ..., S_n\}$ is a set of calendars. Each S_i is a mapping from the meetings in *meetings*(A_i) to timeslots in $\mathcal{T}$. A calendar S_i is *valid* if and only if a) each meeting is mapped to exactly one timeslot and no timeslot has more than one meeting mapped to it, and b) for each meeting M_k and for all attendees $A_i, A_j \in$ *attendees*(M_k), $S_i(M_k) = S_j(M_k)$. That is, the calendars of all attendees of a meeting agree on its assigned timeslot.

The representation of meeting scheduling as MAP is straightforward. The set of MAP variables $\mathcal{V}$ is given by the set of meetings $\mathcal{M}$ and the set of MAP values $\mathcal{D}$ is given by the set of timeslots $\mathcal{T}$. The *participants* of variable V_i correspond to the *attendees* of meeting M_i. The MAP intra-agent mutual exclusion constraint prevents a timeslot from being double-booked and the inter-agent agreement constraint ensures that meeting attendees agree on the time.

Figure 1 illustrates the multiagent assignment problem (and its solution) with three agents A_1, A_2, A_3, five meetings M_1,M_2,M_3,M_4,M_5 and four timeslots. Note that for each agent, each meeting is assigned to a different value in order to satisfy the intra-agent mutual exclusion constraint. Between agents, the variables corresponding to the same meeting are assigned the same value in order to satisfy the inter-agent agreement constraint.

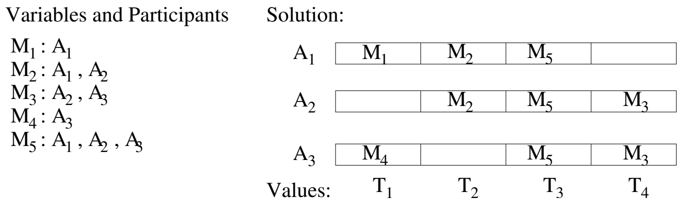

Fig. 1. Meeting Scheduling as the Multiagent Assignment Problem.

2.3 IL-MAP: MAP in Incremental, Limited Information Exchange Domains

We further extend the scheduling problem to introduce the IL-MAP problem in which agents must solve MAP in an incremental fashion while limiting the information they can exchange. These two features are described next.

Incremental In an incremental MAP, new variables and associated constraints are added to the problem over time and must be integrated into an existing assignment. In meeting scheduling for example, new meetings arise over time and must be scheduled in the context of an existing calendar. In addition to the elements of MAP defined above, in the incremental version we are also given:

- $S_{init} = \{(V_1, d_i), (V_2, d_j), ..., (V_m, d_k)\}$ is an initial solution.
- V_{m+1} is a new variable to be assigned a value.
- *participants*$(V_{m+1}) \subseteq \mathcal{A}$ is a set of agents who are assigned the variable V_{m+1}.

The key difficulty that arises in incremental MAP is that existing assignments may need to be changed in order to successfully accommodate the new variable but it is difficult to determine in advance which changes will result in a set of valid schedules.

Limited Information Exchange Although agents must exchange some information in order to obtain feasible solutions, the information exchange process is limited due to the distributed nature of the problem. In particular, we assume the following condition.

- Agents do not communicate information about a variable to agents who are not participants in that variable.

For example, the id of a variable, its current value, or the participants in the variable are not communicated between agents who are not both participants in the variable. A key challenge is to schedule effectively under this condition.

```
procedure initiate($M_j$):
    $initiator(M_j) \leftarrow A_i$
    t $\leftarrow GetTimeslot(M_j)$
    if t is null:
        return
    status($M_j$,t) $\leftarrow$ PENDING
    for each $A_k \in attendees(M_j)$:
        send (PROPOSE, $M_j$, t, $A_i$) to $A_k$

procedure when received(PROPOSE,
 $M_j$, t, initiator):
    if exists $M_k$ where status($M_k$, t) is PENDING:
        reply $\leftarrow$ IMPOSSIBLE
    else if exists $M_k$ where
    status($M_k$, t) is CONFIRMED:
        if $BumpingRule(M_j, M_k)$ is true:
            status($M_k$,t) $\leftarrow$ BUMPED
            status($M_j$,t) $\leftarrow$ PENDING
            reply $\leftarrow$ PENDING
        else:
            reply $\leftarrow$ IMPOSSIBLE
    else:
        status($M_j$,t) $\leftarrow$ PENDING
        reply $\leftarrow$ PENDING
    send (REPLY, $M_j$, t, reply, $A_i$) to initiator

procedure when received(REPLY,
 $M_j$, t, reply, $Attendee$):
    agentView($M_j$,t,$Attendee$) $\leftarrow$ reply
    if exists t' where $\forall A_k \in attendees(M_j)$,
    agentView($M_j$, t',$A_k$) is PENDING
    and status($M_j$, t') is PENDING
        status($M_j$, t') $\leftarrow$ CONFIRMED
        resolved($M_j$)
        for each $A_k \in attendees(M_j)$:
            send (CONFIRM, $M_j$, t') to $A_k$
        if exists $M_k$ where status($M_k$,t') is BUMPED:
            reschedule($M_k$)
    else:
        t'' $\leftarrow GetTimeslot(M_j)$
        if t'' is null:
            resolved($M_j$)
            for each $A_k \in attendees(M_j)$:
                send (FAIL, $M_j$) to $A_k$
        else:
            status($M_j$,t'') $\leftarrow$ PENDING
            for each $A_k \in attendees(M_j)$:
                send (PROPOSE, $M_j$, t'', $A_i$) to $A_k$

procedure when received(CONFIRM, $M_j$, t):
    status($M_j$, t) $\leftarrow$ CONFIRMED
    resolved($M_j$)
    if exists $M_k$ where status($M_k$,t) is BUMPED:
        reschedule($M_k$)

procedure when received(FAIL, $M_j$):
    resolved($M_j$)

procedure when received(RESCHEDULE, $M_j$):
    reschedule($M_j$)

procedure reschedule($M_j$):
    if $A_i$ equals $initiator(M_j)$:
        if exists t where status($M_j$,t) is
        BUMPED or CONFIRMED:
            status($M_j$, t) $\leftarrow$ IMPOSSIBLE
        for each $A_k \in attendees(M_j)$:
            for each t where agentView($M_j$, t,$A_k$) is
            IMPOSSIBLE or PENDING:
                agentView($M_j$, t ,$A_k$) $\leftarrow$ POSSIBLE
        initiate($M_j$)
    else:
        send (RESCHEDULE, $M_j$) to $initiator(M_j)$

procedure resolved($M_j$):
    for each t where status($M_j$, t) is PENDING:
        status($M_j$, t) $\leftarrow$ POSSIBLE
        if exists $M_k$ where status($M_k$, t) is BUMPED:
            status($M_k$, t) $\leftarrow$ CONFIRMED
```

Fig. 2. Algorithm for Agent A_i

3 A Solution Technique for IL-MAP in Meeting Scheduling

We are interested in solution techniques for IL-MAP in the context of distributed meeting scheduling. We first describe a basic negotiation framework upon which our techniques are applied. Next, we describe the problem of rescheduling existing meetings. Finally, we present our approach for making this rescheduling decision effectively.

3.1 Basic Negotiation Protocol

Sen and Durfee [9] describe a basic negotiation protocol for meeting scheduling in which agents negotiate in *rounds*. Each meeting has a designated initiator who manages the negotiation of the meeting by proposing times and collecting responses from the other attendees in a sequence of rounds. In each round, each attendee responds with a PENDING (accept) or IMPOSSIBLE (reject) message for the proposed time. The initiator collects the responses in each round and does a set intersection to try to find a mutually acceptable time. If a time is found, the meeting is CONFIRMED (scheduled) in one additional round and the process terminates. Otherwise, the process continues in rounds until the initiator runs out of times to propose in which case the process terminates with failure.

We adopt a variant of this basic protocol in which attendees may tentatively bump a CONFIRMED meeting in favor of a new meeting in order to decrease the possibility of scheduling failure. We say it is tentatively bumped because an agent waits until the new meeting is confirmed in the bumped timeslot before initiating rescheduling of the bumped meeting. If the new meeting is confirmed in some other slot or fails to be scheduled, the bumped meeting is re-instated into its original slot. If an agent needs to reschedule a meeting of which it is not the original initiator, it sends a RESCHEDULE message to the initiator, who will be responsible for restarting a negotiation episode for the meeting.

Details of the algorithm are shown in Figure 2. Two functions $GetTimeslot$ and $BumpingRule$ are purposely left unspecified in Figure 2. $GetTimeslot$ returns a free timeslot from the calendar or null if one does not exist. This function encapsulates a local optimization routine which ranks all the free timeslots according to a complex set of user preferences, and returns the top ranked time. Further discussion is out of scope of this paper and we refer the reader to [6] for more details. The $BumpingRule$ function returns true or false, and encapsulates the reasoning of the agent about whether one meeting should be bumped for another. A technique for making this decision is described in rest of this next section.

3.2 The Problem of When to Reschedule

A key algorithmic decision to be made is when to bump an existing meeting in favor of a proposed meeting. More specifically, an attendee A_i must make a rescheduling decision when it receives a proposal for meeting M_1 at time slot T_1 but A_i already has a meeting M_2 confirmed in slot T_1. A_i has to decide between accepting the proposal or rejecting it. If the agent decides to accept the proposal, it may need to reschedule M_2 with the other attendees. This rescheduling may cause the other attendees in turn to bump other meetings, which can result in cascading disruption costs throughout the set of agents. The alternative is for A_i to reject the proposal for M_1, but this entails risk also because the scheduling of M_1 may ultimately fail. It is difficult to determine in advance which is the better decision because other people's schedules are not directly observable.

Fixed strategies such as always rejecting or always bumping fail to be effective. Table 1 shows a comparison of the average performance of the two fixed strategies. (The exact experimental set-up is described in more detail in Section 5. These results are with 20 agents who have initial calendar densities of 85%.) The "failures" column shows that for the Never-Bump strategy a mutually free timeslot could not be found in 49 out of 50 cases. The "timeouts" column shows that for the Always-Bump strategy the negotiation failed to terminate after a given amount of time (10 minutes) in 50 out of 50 cases. In these cases, a cascading effect caused many meetings to be bumped until ultimately a maximum time limit was reached.

Table 1. Empirical analysis of two strawman strategies illustrates the need for intelligent rescheduling techniques

Strategy	Rounds	Msgs	Failures	Timeouts
Never-Bump	6.88	44	49/50	–
Always-Bump	614	2736	–	50/50

3.3 Modeling Scheduling Difficulty

We propose a method for making rescheduling decisions in which agents use a model of "scheduling difficulty" with other agents. Such models can be given to an agent or they can be learned by the agent over time. In this paper we are interested in how a scheduling difficulty model, once obtained, can be used by an agent to improve rescheduling decisions. Also, we note that more complex models of scheduling difficulty are possible than the one presented here. However, such models require more effort to construct and are not guaranteed to

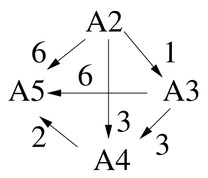

Fig. 3. A model of relative scheduling difficulty with four agents A_2,A_3,A_4 and A_5.

improve scheduling. We opt for the following model which is computationally convenient and can be shown to improve scheduling performance.

Let SD_i be a number denoting the *scheduling difficulty* of an agent A_i, i.e., if $SD_i > SD_j$, then scheduling a meeting with agent A_i is expected to be more "difficult" than with agent A_j. SD is measured in scheduling difficulty "units". We use this factor to encapsulate the many relevant features that contribute to scheduling difficulty with another agent. Assuming that each agent is operating on behalf of a human, SD could take into account factors such as stubbornness or accessibility to email communication. We will consider calendar density as associated with position in a organization as a key factor. We define $k_{i,j}$ as the relative difficulty for scheduling a meeting with A_i versus scheduling a meeting with A_j. It makes natural sense for this relation to be multiplicative and transitive. That is, for three agents A_2, A_3, A_4, we require that $k_{2,3} \times k_{3,4} = k_{2,4}$.

Example: Figure 3 shows A_1's model of relative scheduling difficulty with a group of four other agents A_2,A_3,A_4, and A_5. The arrow from A_3 to A_4 with magnitude 3 represents the relation $SD_{A_4} = 3 \times SD_{A_3}$, i.e., scheduling a meeting with A_4 is 3 times "more difficult than" scheduling a meeting with A_3.

Given a model of scheduling difficulty, we now have a way to define a decision rule for when to reschedule a meeting in favor of another. Given a meeting M_j, A_k computes the difficulty of scheduling M_j as

$$\textit{Difficulty}(M_j) = \sum_{A_i \in attendees(M_j) - \{A_k\}} SD_i \qquad (1)$$

Finally, the bumping rule is given as follows. An agent bumps a meeting M_j in favor of a meeting M_i if and only if the following *BumpingRule*(M_i, M_j) evaluates to *true*:

$$\textit{Difficulty}(M_j) < \textit{Difficulty}(M_i) \qquad (2)$$

4 Example of a Meeting Scheduling Negotiation

We describe an example scheduling negotiation episode involving an agent A_1. Figure 3 shows A_1's model of relative scheduling difficulty with four other agents A_2,A_3,A_4, and A_5. Details of the negotiation using this model is shown in Figure 4. Each box represents the state of agent A_1's calendar at a given time. Arrows denote incoming and outgoing messages. Each message is 3-tuple of meeting id, time, and meeting status, where status is either *possible, pending, bumped, confirmed* or *impossible*. In this example, $attendees(M1) = \{A_1, A_2, A_3\}$, $attendees(M2) = \{A_1, A_4\}$, and $attendees(M3) = \{A_1, A_5\}$.

At time 1, A_1 has meeting M1 currently confirmed at time 10 am and receives a request from A_4 who is the initiator of meeting M2. The time proposed is 10 am, which conflicts with M1. A_1 must now decide whether to reject A_4's proposal, or accept it and bump meeting M1. Referring to Figure 3 and Equation 1, A_1 computes the scheduling difficulty of M1 as $SD_2 + SD_3 = 1 + 1 = 2$ and the scheduling difficulty of M2 as $SD_4 = 3$. Since *Difficulty*$(M1) <$ *Difficulty*$(M2)$, A_1 decides to bump.

At time 2, A_1 changes the status of M1 to bumped, and sets status of M2 as pending for 10 am, and a response is sent to A_4. At time 3, as an example of concurrency, A_1 receives a request from A_5 for 10 am for a new meeting M3. At time 4, A_1 responds impossible since 10 am is already pending for M2. Pending meetings are never bumped (only confirmed meetings can be bumped). At time 5, A_1 hears back from A_4 that M2 should now be confirmed for the previously proposed time of 10 am. At time 6, A_1 sets the status of M2 to confirmed, and begins the rescheduling of M1 by proposing a new time to the other attendees A_2 and A_3. (This example has assumed that A_1 is the initiator of M1. If it were not, then in our protocol, A_1 would have sent a message to the initiator of M1 indicating that 10 am is now impossible, and the initiator would be responsible for restarting the negotiation and rescheduling M1). At time 7, A_1 hears back from A_2 that 11 am is pending in its calendar for meeting M1. At time 8, A_1 records this information in its current state. At time 9 and 10, A_1 hears back from A_3 and records the response. At time 11, A_1 has now heard back from all attendees for meeting M1, and all have agreed on 11 am. A_1 sends the final confirmation message to all attendees. We end the example here, but realize that since A_2 or A_3 may have bumped meetings at 11 am to accommodate A_1's request for meeting M1, the scheduling episode may not be over.

5 Experimental Results

We present experimental results comparing rescheduling strategies that use a model of "scheduling difficulty" with other agent versus strategies that do not. In

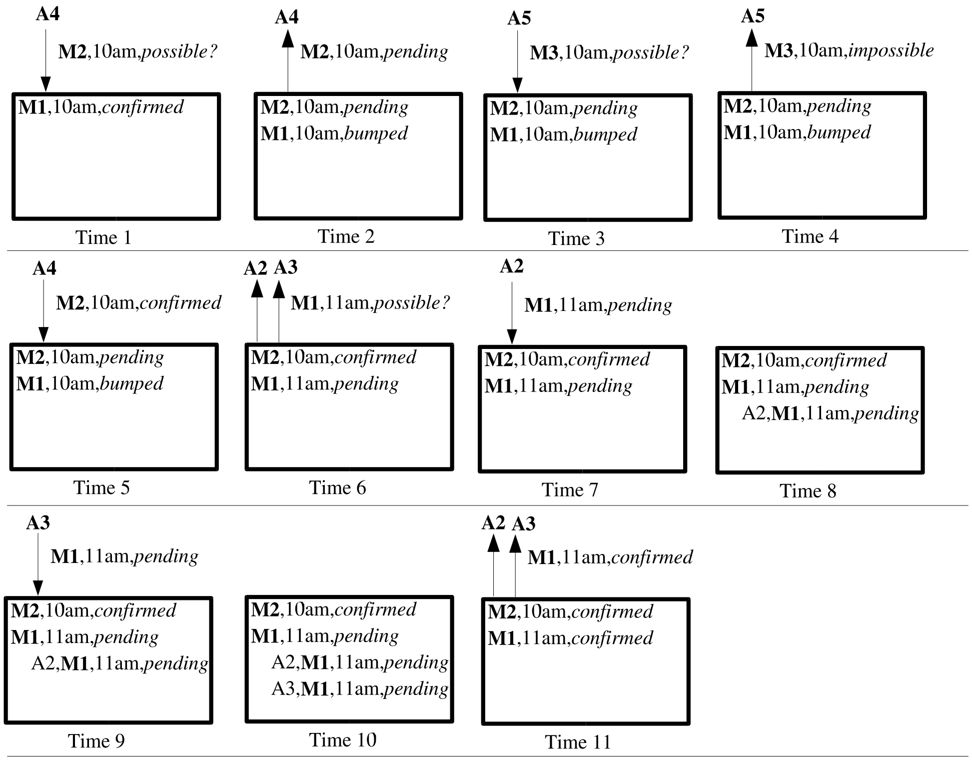

Fig. 4. An example negotiation between an agent and four other agents A_2,A_3,A_4 and A_5.

the first strategy, denoted Att, agents simply compare the number of attendees and bump the meeting with fewer attendees when there is a conflict between two meetings. They do not use knowledge about other agents in making their bumping decisions. In the second strategy, denoted SD, we assume that agents know the rank of other attendees and use this information to make bumping decisions, i.e., they can assign a "scheduling difficulty" to each attendee.

5.1 Experimental Setup

We evaluate each strategy by averaging measurements over a number of "runs". Each run consists of two phases: a problem generation phase followed by a problem solving phase. We describe each phase in turn. In our experiments, we report measurements from the problem solving phase only.

Phase 1 The problem generation phase is centralized. We automatically generate a set of agents $\mathcal{A}$ each with a desired initial schedule density. Each agent's calendar has 50 timeslots to simulate a 5 day 10-hour work week. Next, we automatically generate and schedule meetings between random subsets of the agents until all calendars are filled to their desired density. The attendees of a given meeting are chosen according to a uniform random distribution. The number of attendees for a given meeting is chosen according to a distribution in which meetings of more people are less likely than meetings with fewer people. Every meeting has at least two attendees. Finally, we generate one additional new meeting M_{m+1} that must be scheduled in the problem solving phase. The attendees of the new meeting are chosen to be a random subset of the agents. In our experiments, the number of attendees of the new meeting is fixed to 4. One of them is randomly chosen to be the initiator. Every generated problem is ensure to have a solution.

Phase 2 The problem solving phase is completely distributed. The goal is to find a timeslot for the new meeting $\{M_{m+1}\}$ while successfully rescheduling any bumped existing meetings. That is, the goal is to find an assignment of timeslots to meetings in $\mathcal{M} \cup \{M_{m+1}\}$ that satisfy the intra-agent and inter-agent constraints. We measure number of *failures* which is defined as the number of meetings in $\mathcal{M} \cup \{M_{m+1}\}$ unassigned a timeslot after a given amount of time. Failures may occur either because the initiator gives up scheduling the meeting or a max time elapses. Note that the number of failures in a given run can be greater than one when multiple meetings are bumped and fail to be rescheduled.

5.2 Experiments in a Hierarchical Agent Organization

Human organizations typically have hierarchies in which higher ranked people have denser calendars than lower ranked ones. We hypothesis that the density of an agents calendar and thus her organizational rank, is a good predictor of the difficulty of scheduling with that person.

To evaluate our hypothesis, we begin with an extreme case – a simple two-level organization hierarchy. We divide agents into two equal size groups of "busy" and "not busy" agents, where the initial density of schedules is fixed to 90 percent and 30 percent, respectively. The scheduling difficulty model used by the SD strategy in this scenario is defined as $SD_{busy} = 3 \times SD_{nonbusy}$.

Figure 5 contrasts two strategies as we increase the total number of agents. The graph shows the SD strategy is more effective in terms of preventing scheduling failures than the Att strategy. At 50 agents, the SD strategy results in a failure rate of 0.28 on average, while the simpler strategy Att results in 0.76 failures on average. Failure rate is computed by summing the number of failures over all runs and then dividing by the total number of runs. We do 50 runs for each datapoint where each run follows the methodology described above. This graph shows that the use of our scheduling difficulty model is able to reduce scheduling failures. Also, the high failure rate caused by uncontrolled cascading of bumps, as we saw in Table 1 for the Always-bump strategy, is avoided.

Next, we evaluate the effect of varying our scheduling difficulty model in the busy/non-busy hierarchy. We use a scheduling difficulty model defined as $SD_{busy} = k \times SD_{nonbusy}$ and examine the effects of varying k. The same set of scheduling problems are used for each value of k, i.e., the only difference is the rescheduling decision rule used by the agents. We expect that changes in performance will level off as the scheduling difficulty multiplier k is increased. This is because after some point, an increase in k no longer modifies an agents rescheduling decisions. For example, a meeting M_1 with 4 non-busy attendees will be bumped in favor a meeting M_2 with one busy attendee when $k = 5$. M_1 will continue to be bumped if k is increased. Thus increasing k should stop having an effect on agent decision making at some point. Figure 6 shows empirical data consistent with our hypothesis. An organization of 10 agents was used. Each datapoint represents the average over 50 runs. The graph shows that the effect on failure rate levels off as predicted.

Finally, we experiment with a more complex scheduling difficulty model where there are four levels rather than just two. We use the organization hierarchy shown in Figure 7 with 8 agents in each level, for a total of 32 agents. We experiment with four levels with initial schedule densities of 90,70,50,30 percent respectively. We define $SD_{L_i} = 2 \times SD_{L_{i+1}}$. That is, the difficulty of scheduling with an agent at level i is twice as difficult as scheduling with an agent at level $i + 1$. The empirical results over 500 runs are shown in Figure 8. The failure rate is reduced from 0.28 using the Att strategy to 0.02 using the

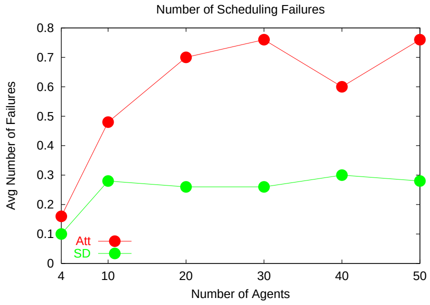

Fig. 5. Comparison of two rescheduling strategies (Att, SD) as a function of organization size. The average number of meetings that failed to be scheduled is shown.

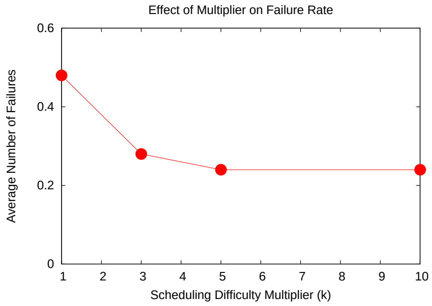

Fig. 6. Effect of increasing value of scheduling difficulty multiplier on scheduling performance. The average number of meetings that failed to be scheduled is shown.

SD strategy. We can conclude that the SD strategy significantly reduces the number of scheduling failures.

6 Conclusion

We have modeled the multiagent meeting scheduling problem as a form of distributed constraint reasoning in which agents must assign a set of values to a set of variables. We presented a novel approach to the problem in which agents use given or learned "scheduling difficulty" models of other agents in order to decide when to change their existing assignments in order to accept proposals from others. We have shown that this approach controls the amount of bumping so that the negotiation is able to terminate in a given amount of time, while also reducing the scheduling failure rate over an alternative approach that does not take into account such models. In future work, we are interested in how an agent can automatically learn these models from past negotiation history.

References

1. C. Bessire, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In *International Joint Conference on AI Workshop on Distributed Constraint Reasoning*, 2001.

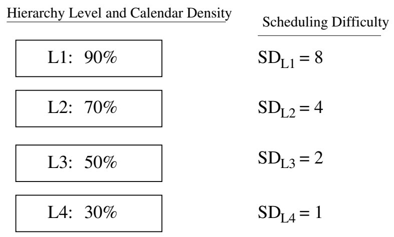

Fig. 7. Agent hierarchy where higher ranked agents have higher calendar densities.

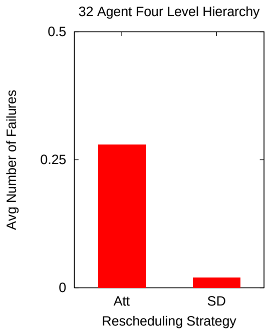

Fig. 8. Comparison of two rescheduling strategies (Att, SD) in a four level organization hierarchy. The number of meetings that failed to be scheduled (average over 500 run) is shown.

2. Eugene C. Freuder, Marius Minca, and Richard J. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *IJCAI-2001 Workshop on Distributed Constraint Reasoning*, 2001.
3. Leonardo Garrido and Katia Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*. The MIT Press: Cambridge, MA, USA.
4. R. Mailler and V. Lesser. A mediation based protocol for distributed constraint satisfaction. In *The Fourth International Workshop on Distributed Constraint Reasoning*, 2003.
5. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 2004.
6. P. J. Modi, M. Veloso, S. Smith, and J. Oh. Cmradar: A personal assistant agent for calendar management. In *Agent Oriented Information Systems, (AOIS) 2004*, 2004.
7. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., 1982.
8. Sandip Sen and Edmund Durfee. A Contracting Model for Flexible Distributed Scheduling. *Annals of Operations Research*, 65:195–222, 1996.
9. Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling. In *Group Decision and Negotiation*, volume 7, pages 265–289, 1998.
10. M.C. Silaghi, D. Sam-Haroud, and Boi Faltings. Asynchronous search with aggregations. In *Proceedings of National Conference on Artificial Intelligence*, 2000.
11. M. Yokoo. *Distributed Constraint Satisfaction:Foundation of Cooperation in Multi-agent Systems*. Springer, 2001.