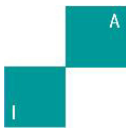
Inteligencia Artificial, 21(62) (2018), 53-66
doi: 10.4114/intartif.vol21iss62pp53-66



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

Integrating Meeting and Individual Events Scheduling

Anastasios Alexiadis, Ioannis Refanidis, Iias Sakellariou

Department of Applied Informatics, University of Macedonia,
Egnatia 156, 54636, Thessaloniki, Greece.
talex@uom.edu.gr, yrefanid@uom.edu.gr, iliass@uom.edu.gr

Abstract Automated meeting scheduling is the task of reaching an agreement on a time slot to schedule a new meeting, taking into account the participants' preferences over various aspects of the problem. Such a negotiation is commonly performed in a non-automated manner, that is, the users decide whether they can reschedule existing individual activities and, in some cases, already scheduled meetings in order to accommodate the new meeting request in a particular time slot, by inspecting their schedules. In this work, we take advantage of SelfPlanner, an automated system that employs greedy stochastic optimization algorithms to schedule individual activities under a rich model of preferences and constraints, and we extend that work to accommodate meetings. For each new meeting request, participants decide whether they can accommodate the meeting in a particular time slot by employing SelfPlanner's underlying algorithms to automatically reschedule existing individual activities. Time slots are prioritized in terms of the number of users that need to reschedule existing activities. An agreement is reached as soon as all agents can schedule the meeting at a particular time slot, without anyone of them experiencing an overall utility loss, that is, taking into account also the utility gain from the meeting. This dynamic multi-agent meeting scheduling approach has been tested on a variety of test problems with very promising results.

Keywords: Scheduling, Optimization, constraint satisfaction, Meeting scheduling, Multi-Agents.

1 Introduction

Three friends, Alice, Bob and Charlie, want to meet for a coffee. Alice sends a request and proposes three alternative coffee shops, in different areas of the city, as well as a time frame. The duration of the coffee meeting is estimated to one hour.

Each friend has personal preferences for a variety of aspects of the proposed coffee meeting, such as the particular coffee shop, traveling arrangements needed to go there, as well as the alternative time slots. Their agendas are very tight, so rescheduling might be necessary in order to accommodate the new meeting. While rescheduling, they might consider travel time and expenses, as well as overall utility of the alternative schedules.

Meeting scheduling is a multiobjective optimization process, where multiple agents negotiate on a common time slot to hold the meeting, taking also into account their commitments and preferences. An agent may have already scheduled individual activities, such as tasks with deadlines undertaken, family duties, etc., as well as other meetings. Some of these activities may be flexible and can be rescheduled, while others may be not. Furthermore, some activities might be of lesser importance and the user may decide to abandon them, in order to be able to participate in the meeting. The agent may also have preferences concerning the location and time of the meeting, as well preferences towards scheduling its

ISSN: 1137-3601 (print), 1988-3064 (on-line)

individual activities. It might also have hard or soft constraints, e.g., ordering constraints, between his individual activities.

In this work we assume that the agents do not have preferences over how the meeting will be scheduled; they only care to schedule the meeting. Under this assumption, the simplest case is when the participating agents can find a common free time slot to schedule the meeting, without the need to resort to rescheduling. This time slot is considered optimal for the meeting. Unfortunately, this situation is rather rare. It is quite common that such a time slot does not exist. However, this does not mean that the meeting cannot be scheduled, since participating agents may negotiate to reach an agreement on a time slot that is not initially available to all agents. During the negotiation, time slots not available for all agents are considered, under the condition that they can become "available" if some agents manage to reschedule or even drop already scheduled activities.

The negotiation process can terminate without reaching an agreement, i.e., the agents may conclude that the meeting cannot be scheduled, since rescheduling or dropping other activities reduces the overall utility received by at least one of the agents (we assume that all the agents are veto participants of the meeting, that is, without anyone of them the meeting cannot hold). As a last resort, the agents might try to reschedule other meetings, either with agents not participating in the current negotiation, or among themselves. In that case, a recursive process may commence, where in order to schedule one meeting, other meetings need to be rescheduled, that may initiate rescheduling of other meetings with a different group of agents and so on.

In this work, we consider the problem of multi-agent meeting scheduling, where each agent has a set of already scheduled individual activities, that can be rescheduled in order to accommodate the new meeting. We adopt a rich model of individual activities, with a variety of unary and binary constraints and preferences, that has been proposed in [13]. Meetings are described in a simpler way, that is, duration, temporal domain and utility per agent. We consider the problem of meeting scheduling as a multi-agent constraint satisfaction problem, based on solving many single-agent constraint optimization problems, with each agent attempting to maximize his utility, with the latter being a function of all scheduled activities (individual activities and the meeting), as well as the way they have been scheduled (unary and binary preferences). We do not consider recursive meeting rescheduling, i.e., only individual agent activities can be rescheduled during the process.

The novelty lies in meeting scheduling empowered by rescheduling of individual tasks supporting such a rich model. To the best of our knowledge, there is no other work that combines these two aspects of personal time management.

A powerful scheduler based on greedy construction and stochastic optimization [2] is employed by each agent for constructing a schedule for his individual activities. To schedule a meeting among a set of agents, we have designed and implemented a meta-scheduler, that negotiates with each agent a time slot and a location that can be agreed upon. Each such proposal is evaluated by agents using the scheduler mentioned above. For each proposal, the agent employs distributively the individual activities scheduler to decide whether it can accept each proposed time slot or not. To accept a time slot, the overall utility for the agent when scheduling the new meeting in this time slot should be higher than keeping his original schedule and not scheduling the meeting at all.

In this work, we do not consider strategic behaviour on behalf of the agents. We have tested our approach on a variety of problem instances, with very promising results in terms of performance and scalability.

The rest of the article is structured as follows: Section 2 presents related work. Section 3 presents background information concerning scheduling individual activities. Section 4 defines the integrated scheduling problem and presents the proposed approach. Section 5 presents experimental results and, finally, Section 6 concludes the article and poses future directions.

2 Related Work

Not surprisingly, there exist numerous approaches to agent based meeting scheduling, since it has attracted research interest rather early. In fact, it was considered to be one of the main applications of intelligent personal assistants, a class of agent systems aiming to support the user in performing tedious everyday tasks. As expected, all approaches follow a similar pattern of interaction: there is a group

of agents, each one representing a participant, and possibly a host/meeting agent which coordinates a negotiation process with proposals and counter-proposals on meeting parameters. The agreement is, in most cases, a time slot that maximizes a collective preference value, computed by individual preference values of participating agents. There are quite a few issues in the above setting, such as the number of proposals in each round, the modelling of constraints and preferences, privacy issues, interoperability issues (semantic web), user modelling via learning, "bumping" strategies, etc. In this section we will present in more detail some of the approaches.

One of the earliest agent-based meeting scheduling applications is reported by Jennings and Jackson [9], according to which each participant is "represented" by a meeting scheduling agent (MSA), managing its user's calendar. The procedure followed, presents quite a few similarities with the well known Contract Net protocol: An MSA acting on behalf of its meeting host, announces its intention to arrange a meeting, along with respective time constraints and duration. Participants (their MSAs) respond with bids, that are possible time slots annotated with a preference value, which are used by the meeting host to discover the best common time slot with respect to its global preference value. This announce-bid cycle continues until either a suitable slot is found, or scheduling the meeting is determined to be non-possible, in which case, the initial announcement (meeting duration / time constraints) are changed and the process starts over again.

Sen and Durfee [14] address the problem in a similar setting, however agents report their availability (true/false) on a meeting announcement over n possible slots, along with m alternative slots. In the same work, authors report on rescheduling meeting strategy (bumping) when conflict occurs, maximizing a utility function.

The RETSINA (Reusable Environment for Task-Structured Intelligent Networked Agents) Calendar Agent (RCAL) [12], was aiming to bridge information available on the Semantic Web with the user's personal information managers, e.g., Outlook 2000. RCAL uses Contract Net to negotiate a meeting between participants, a process that involves receiving bids from involved parties to determine an appropriate meeting slot. The advantage of RCAL is that information regarding the user's schedule is obtained automatically through semantically annotated descriptions, and thus allows for a more "accurate" scheduling of meetings.

Chun et al. [7] treat the problem of meeting scheduling from the perspective of user privacy, that is, managing to optimally schedule a meeting without complete knowledge of individual participant preferences. The negotiation takes place between user Secretary Agents (SA), and a Meeting Agent (MA), that coordinates process. Proposals and counter proposals are annotated with a participant's preference value w.r.t. the meeting parameters. In each step the MA collects the set of proposals and generates a new set, sorted on global preference estimation values, that is, values computed from the annotated replies of the SA agents.

In DAS [15] lightweight agents called coordination agents (CA), are created by the participants for each proposed time slot of the meeting. Agents managing the same slot are combined to a single agent. The approach aims at reducing communication costs, by decoupling user agents from CA, and allowing interactions between the latter on the same computational host. Franzin et al. [8] provide a more rigorous constraint modelling of the problem considering hard and soft constraints, with the latter representing user preferences with respect to meeting parameters. The term "common assignment problem with preferences" is introduced to describe the collaboratively scheduling of a meeting, with agents having common variables to set, but possibly different "internal" constraints on these variables. Meeting scheduling proceeds with rounds of proposal-counter proposal, guided by the preference value, in order to reach optimal solutions. In a similar vein, MSRAC (meeting scheduling with reinforcement of arc consistency) [3], handles incremental scheduling, user preferences in the form of soft constraints, user availability by a set of hard constraints and common meeting times with other agents by equality constraints of the meeting time slot, but consider more than one meeting to be scheduled at each time point, i.e. a dynamic iterative value CSP problem. The solving process includes time slot proposals broadcasted by the host to participants, who return their available slots ranked by their preference. Bumping also considered, based on three different heuristic strategies.

CMRadar [11] offers a complete approach to calendar management, including multi-agent meeting scheduling capabilities. If a request for a meeting cannot be accommodated within the current calendar of the agent, then complex strategies involving altering other meetings of lower priority value are employed,

thus engaging the agent in a multi-negotiation process, referred to as the "bumping" problem by Modi and Veloso [10]. In that work, multi-agent meeting scheduling is modelled as a partial incremental Multi-agent Agreement problem (piMAP), where an iterative agreement protocol is employed. The main difference between earlier approaches is the use of bumping heuristic strategies, i.e. strategies to decide whether to modify an existing agent schedule to accommodate the meeting, that allow incremental scheduling. groupTime [6] is a Web based application that bases meeting scheduling on user preferences, in a semi-automatic fashion, i.e. users have the chance to object to an initial meeting time, set by the system. For each possible time slot the user can set its preference explicitly, or the system assigns a value based on the current user's schedule (events). Preference assigned by the system involves extracting schedule-agnostic features for each participant's schedule, based on a group-specific ontology and a set of weights that are defined by machine learning techniques using as training data the user's events and preferences. The set of preferences is then used to determine the meeting time.

The PTIME system [4, 5] is a calendaring assistant that offers meeting scheduling among a variety of time management functionalities. The aim in PTIME was the system to learn user preferences, using machine learning techniques, providing an adaptive personal assistant. The learner module interacts both with the user and the constraint reasoner in order for the system to provide meeting options on a constantly evolving user preferences model.

Chronos [16], is a multi-agent meeting scheduler, in which each user is represented by an Organiser Agent (OA), that learns user's preferences. Each OA represents its beliefs about both its user's preferences and other users he interacts with (acquaintances) using Bayesian Networks (BN). BN express the causal relationships between scheduling parameters, such as time and duration of a meeting and are used to reason about meetings, by computing the probability of an agent to accept a specific proposal. Probabilities are computed for both participants in a meeting and guide a negotiation process, involving proposals and counter proposals in a multi-agent negotiation setting.

Although many approaches consider bumping, i.e. rearranging another meeting in order to accommodate a new request, the issue of rearranging the participants private schedule according to his original constraints with a rich scheduling model, has not been investigated in depth until now.

3 Background: Individual activity scheduling

This section presents the model and the assumptions for scheduling individual activities, as it has been proposed by Refanidis and Yorke-Smith [13] and has been implemented in the SelfPlanner system. This model has been adopted entirely in the present work and has been extended to support also meetings. So, for the completeness of the article, it is purposeful to give a quick overview of it.

A person (equivalently an agent) may have a set T of N individual activities to accomplish. Each activity T_i $(1 \leq i \leq N)$ can be accomplished only during certain time periods, which constitute its temporal domain. This domain is defined as a set of temporal intervals $D_i = [a_{i1}, b_{i1}) \cup [a_{i2}, b_{i2}) \cup \ldots \cup [a_{i,F_i}, b_{i,F_i})$. The duration of an activity may be not fixed (e.g., visiting a museum), so for each activity T_i he can specify its minimum duration d_i^{min} and its maximum duration d_i^{max}, with more scheduled duration resulting in more utility for the person. Activities can be interruptible, that is, they can be scheduled into parts (e.g., reading a book or writing a paper). For each interruptible activity the person can specify its minimum part duration $smin_i$ and its maximum part duration $smax_i$ (e.g., the person wants to devote in book reading time periods not less that 1 hour and no more than 3 hours). The sum of the durations of an activity's parts (non-interruptible activities are considered to have one part) have to be at least d_i^{min} in order for the activity to be considered scheduled.

The set of locations associated with any of the person's activities is Loc, and let M being their number. A matrix $Dist$, not necessarily symmetric, defines the temporal distance between any pair of locations (in the simple model we assume a single means of transportation). We assume that time, temporal intervals and distances are discrete, particularly integers (as the unit of time it is used the minimum temporal interval of interest, e.g., 10, 15 or 30 minutes).

A set of locations $Loc_i \in Loc$ is associated with each activity T_i. A special location called $ANYWHERE$ denotes that the activity can be executed at any location ($ANYWHERE$ is considered to have a temporal distance of 0 from and to any other location). Traveling times between locations have to be taken into account when scheduling activities in time and space; that is, any two temporally adjacent

activities scheduled at different locations (not $ANYWHERE$) should have a large enough temporal gap between them, so as the person can travel between the two locations.

Activities with compatible locations may overlap in time, e.g., watching a lecture while reading emails. Each activity has a utilization value between 0% and 100%, denoting the percentage of the user's attention that the activity requires. The sum of the utilization values of activities scheduled at the same time cannot exceed 1.

The model supports three types of binary constraints between pairs of activities:

- Proximity constraints define a minimum or/and a maximum temporal distance between two activities. For example, the minimum temporal distance between two heavy load activities should be at least 8 hours. Or, the maximum temporal distance between reading a book and writing its synopsis should be at most two days.
- Ordering constraints define an order in time. For example, preparing the slides for the lecture should precede the lecture itself.
- Implication constraints define prerequisite activities. For example, in order to go to the theater, one should first buy tickets.

Note that, especially for the proximity constraints, they can be defined also over the different parts of an interruptible activity.

The overall single-agent problem of scheduling a set of individual activities is formulated as a constraint optimization problem, with the empty plan being the least preferred solution. The objective function is additive over the various sources of utility. These include:

- Each scheduled activity – they can have different utility values.
- Any scheduled duration above the minimum duration of the activity.
- The person's preference over the temporal intervals when the activity has been scheduled (e.g., morning vs evening).
- Proximity, ordering and implication preferences, that is, soft versions of the aforementioned constraints.

The constraint optimization problem is solved by finding values for the decision variables p_i, denoting the number of parts of activity T_i ($1 \leq i \leq N$), t_{ij} (start times), d_{ij} (durations), l_{ij} (locations), for each Activity part T_{ij} ($1 \leq j \leq p_i$), while trying to maximize the sum of the utility sources. The model, with a first solver based on the Squeaky Wheel Optimization framework has been initially presented by Refanidis and Yorke-Smith [13]. A more powerful solver, encompassing post-processing local search techniques (mainly simulated annealing) has been presented by Alexiadis and Refanidis [2].

A technique that allows to produce significantly different alternative plans has been presented by Alexiadis and Refanidis [1]. Producing such alternative plans and retaining them in some form of cache memory may be very useful in meeting scheduling, since they can be used to immediately check whether a meeting can be accommodated in a particular time slot and what is the cost for this accommodation. Particularly, if the time slot is free at any alternative plan kept in cache memory, then the meeting can be scheduled there and the cost is equal to the utility of the current plan minus the utility of the particular alternative one.

4 Collaborative Meeting Scheduling

4.1 Informal Problem Specification

Building on top of the individual activities problem specification, we assume a set of K agents, each agent k of them having its own set of individual activities, according to the model described in the previous section, as well as an already accepted schedule for them S_k, with instantiated decision variables. Furthermore, each agent may already have agreed to participate in meetings with other agents (not necessarily among the K agents of interest for the new meeting), with decided time and location.

One of the K agents invites the rest of them to participate in a new meeting. This agent will act as the coordinator for the particular meeting. The coordinating agent specifies a fixed duration for the meeting, alternative locations for the meeting (or $ANYWHERE$, if no physical presence is required, e.g., teleconferences), as well as a set of temporal intervals when the meeting could be scheduled. Participating in this meeting results in some utility gain for each invited agent (including the coordinator), which may be different for each agent. The meeting will be considered successful and all participants will get the corresponding utility, only in case all of them will be able to participate in it.

Agents may reschedule already scheduled individual activities (in this work we assume that they will not reschedule already scheduled meetings, in order to accommodate the new one). Rescheduling already scheduled activities may result in a utility loss wrt their current schedule S_k. An agent will accept to participate to the meeting at a particular time and location, only in case the utility loss from rescheduling already scheduled individual activities is no more than the utility gain by participating in the meeting. In this work we do not assume extra constraints and preferences, either unary (that is, concerning the meeting by its own) or binary (that is, concerning the new meeting and already scheduled activities).

The meeting scheduling problem may have one or more solutions or even none at all. In the case multiple solutions exist, various criteria could be adopted to select the best among them, thus treating the problem as an optimization problem. One criterion, in that case, is the *sum*, that is, maximizing the total utility over all agents. Another criterion is the *maxmin*, that is, maximizing the minimum utility gain, where the gain for each agent k is computed as u_k minus the utility loss due to rescheduling existing activities.

However, in this work we treat the problem as a constraint satisfaction problem, that is, we try to schedule the meeting with the extra constraint that no agent receives less utility from its new schedule (with the meeting), compared to its old schedule (without the meeting). However, from the point of view of each agent individually, it is a constraint optimization problem, that is, it tries to maximize its total utility received from scheduling its individual activities, plus the utility received from the meeting.

Meetings in this work are described in a simpler way than individual activities, that is, their attributes include just a temporal domain, which is a set of temporal intervals, a fixed duration and a utility value. No other constraints and or preferences are defined over meetings.

4.2 Algorithm

The *Joint Activity Scheduler* (JAS) presented in this Section works in two phases, that implement a message exchange mechanism between the coordinator and the invited agents. The first phase (Algorithm 1) attempts to trivially solve the problem without rescheduling already scheduled activities, by looking for a common empty interval in the current schedules of the agents (also shown in Figure 1). The second phase attempts to schedule the meeting by rescheduling already scheduled individual activities.

In the first phase, the coordinator iterates over all possible meeting time intervals, sending for each interval I a `query-if` message to ask participants if they are available at that time. If all agents reply affirmatively (`inform-t` message), the meeting is scheduled at I. If even a single agent replies negatively (`inform-f` message), then the process is repeated for the next interval in the meeting's temporal domain.

The meeting intervals I_i returned by $ja.all\_intervals()$ are subintervals of the meeting's temporal domain that have a duration equal to the meeting's length. Note that Algorithm 1 does not try to optimize any measure of utility. Since, in case of a successful result, the meeting will be scheduled without rescheduling any existing individual activity, there is no utility loss for any agent due to rescheduling. Furthermore, since the utility gain for any agent by scheduling the meeting is fixed and does not depend on when the meeting has been scheduled, the utility gain for each agent from successfully scheduling the meeting is fixed. So, from the point of view of Algorithm 1, scheduling the meeting is a constraint satisfaction problem (and not a constraint optimization one), so all solutions are equally desirable and the meeting is scheduled at the first commonly free temporal interval found.

On the other hand, if no common free interval is found, Algorithm 2 is employed, in a last attempt to find a common free interval between the agents, using rescheduling of existing individual activities. In that case it is desirable to reduce the overall need for rescheduling, since each call to the individual activity scheduler is a computationally expensive process.

In Algorithm 2 the agents receive four different types of requests from the coordinator. Specifically:

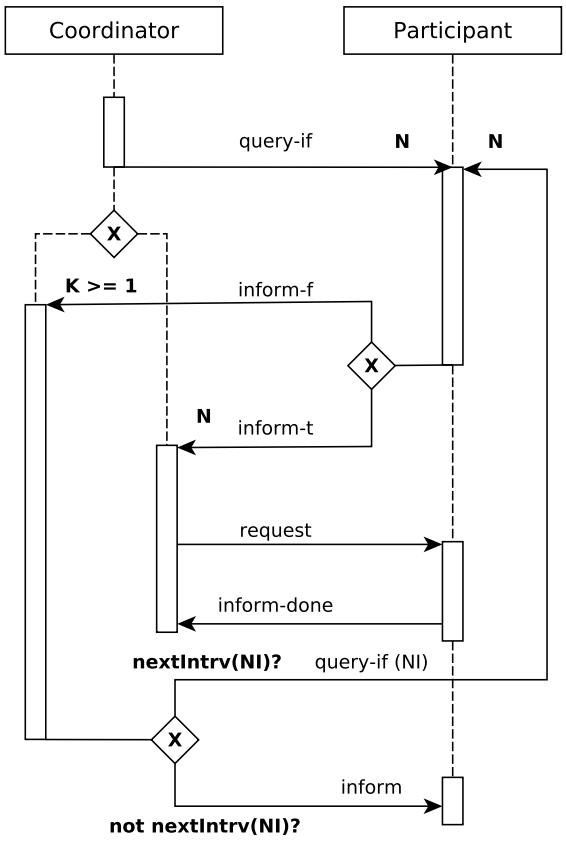


Figure 1: AUML Diagram of the agent interaction during the first phase of the Joint Activity Scheduling.

Algorithm 1 JAS-Phase-1

```
1: procedure JAS-PHASE-1-COORD(ja,users)
2:     for i ←1 to ja.all_intervals().length() do
3:         interval ← ja.all_intervals()[i]
4:         if ¬available(interval, current_plan) then
5:             continue
6:         available ← 0
7:         for k ← 1 to users.length() do
8:             send(agent_k, query-if(interval))
9:         for k ← 1 to users.length() do
10:            if users[k].msg = inform-t then
11:                available ← available + 1
12:            else
13:                break
14:        if available = users.length() then
15:            return interval
16:    return failure
17:
18: procedure JAS-PHASE-1-AGENT
19:    while true do
20:        interval ← get_msg(coordinator)
21:        if available(interval, current_plan) then
22:            send(coordinator, inform-t)
23:        else
24:            send(coordinator, inform-f)
```

- $l4$–$l5$:[1] Request to return to the coordinator their current busy schedule, without providing information about their scheduled activities.
- $l6$–$l12$: Request to attempt to render free a specific temporal interval, by rescheduling their activities; the reply is *success* of $failure$, accompanied with the utility gain in the first case. The utility gain is the improvement of the utility of the rescheduled plan to the utility of the old plan. Note that a $failure$ response occurs not only in cases where it was impossible to schedule the meeting in the particular time slot, but also in cases where scheduling the meeting is possible, however it results in overall utility loss (taking into account the meeting's utility). In case of a *success* reply, the agent must retain in its local memory the corresponding schedule, till the end of the meeting scheduling procedure.
- $l13$–$l15$: Request to schedule the meeting at a particular time interval, following a previously *success* reply for that interval. The agent adopts the corresponding schedule from its local memory and the meeting scheduling process terminates.
- $l16$–$l17$: Request to cancel the rescheduling process (if it is running); this request arises in case another agent has already replied a $failure$ for the time slot under consideration.

In order to minimize the average workload (by reducing the number of messages exchanged), the coordinator asks for the busy hours of all agents for the meeting's temporal interval and computes the overall workload for each possible time slot when the meeting could be scheduled ($l5$–$l7$).[2] The overall workload is stored in the *timeline* array (the length is the maximum number of time slots of the users' plans, whereas each t-th value is the number of events scheduled in the t-th time slot). Time slots with lower overall workload (i.e., the least busy) are given priority to try to schedule the meeting. Furthermore, if a more aggressive approach is adopted, not all possible time slots need to be checked.

[1] Procedure JAS-PHASE-2-AGENT.
[2] Procedure JAS-PHASE-2-COORD.

Algorithm 2 JAS-Phase-2

```
1: procedure JAS-PHASE-2-COORD(ja, users)
2:     timeline ← coord.timeline
3:     min_ints ← {−1 . . .}
4:     min_mins ← {∞ . . .}
5:     for k ← 1 to users.length() do
6:         users[k].timeline ← request_plan(k)
7:         timeline ← timeline + users[k].timeline
8:     for i ← 1 to ja.all_intervals().length() do
9:         int ← ja.all_intervals()[i].start
10:        temp ← timeline[int]
11:        for k ← int + 1 until int + ja.dur do
12:            temp ← temp + timeline[k]
13:        for k ← 1 to NUM_OF_TRIES do
14:            if min_mins[k] > temp then
15:                min_mins.insert(k, temp)
16:                min_ints.insert(k, int)
17:                break
18:    for i ← 1 to NUM_OF_TRIES do
19:        available ← 0
20:        for k ← 1 to users.length() do
21:            if users[k].timeline[min_ints[i]] then
22:                reschedule(k, min_ints[i], ja.dur)
23:        for k ← 1 to users.length() do
24:            if users[k].timeline[min_ints[i]] then
25:                if users[k].wait_for_repl() is yes then
26:                    available ← available + 1
27:                else
28:                    send_cancel_to_all()
29:                    break
30:            else
31:                available ← available + 1
32:        if available = users.length() then
33:            if coord.timeline[min_ints[i]] then
34:                plan ← schedule(problem + [min_ints[i], ja.dur, u_c])
35:                if [min_ints[i], ja.dur] ∈ plan then
36:                    current_plan ← plan
37:                    send_replace_to_all(min_ints[i])
38:                    return [min_ints[i], ja.dur]
39:            else
40:                send_replace_to_all(min_ints[i])
41:                return [min_ints[i], ja.dur]
42:    return −1
```

```
1: procedure JAS-PHASE-2-AGENT
2:     while true do
3:         msg ← get_msg(coordinator)
4:         if msg requests plan then
5:             send_agent(coordinator, current_plan)
6:         else if msg requests reschedule then
7:             plan[msg.start] ← schedule(problem + [msg.start, msg.dur, u_k])
8:             if [msg.start, msg.dur] ∈ plan then
9:                 u_gain = (u(plan[msg.start] − u_k) − u(plan))/u(plan)
10:                send_agent(coordinator, yes, u_gain)
11:            else
12:                send_agent(coordinator, no, 0)
13:        else if msg requests replacing then
14:            if plan[msg.start] then
15:                current_plan ← plan[msg.start]
16:        else if msg requests cancel then
17:            cancel_scheduling()
```

Algorithm 2 sorts the potential time slots to schedule the meeting in ascending order in terms of workload (that is, from the least busy to the busiest) ($l8$–$l17$). NUM_OF_TRIES denotes the number of temporal windows that Algorithm 2 will attempt to schedule there the meeting. Setting this constant to a very large number allows for attempting all possible time slots.

Then, for every attempted time slot the coordinator checks the busy schedule of every agent (as it was sent by the respective agent), and asks the agents that are busy to check whether rescheduling of existing activities is possible ($l18$–$l22$), thus being able to schedule the meeting at the time slot under consideration. If any of the agent replies that rescheduling failed ($l23$–$l31$), the coordinator sends a cancel request to the rest of the agents so as they terminate their rescheduling process, and proceeds to the next time slot in the list.

If all the agents reply positively to the time slot under consideration, including the coordinator ($l32$–$l34$). If the time slot is either available or the coordinator can reschedule it without a great loss to its current plan utility then the procedure succeeds and the coordinator informs the other agents that the meeting has been scheduled at the interval under consideration ($l35$–$l38$). Otherwise the next interval in line is examined until either NUM_OF_TRIES intervals have been evaluated or there are no more time slots to examine ($l39$–$l42$).

5 Experimental Evaluation

We implemented a non-distributed version of the above algorithms, i.e. the code is executed as a single process, in C++.[3]

In the current experimental evaluation we are more concerned with qualitative aspects of the algorithm, and not performance issues, like execution time and number of messages exchanged, thus we do not expect the results presented to change under a distributed, multi-agent version.

In order to evaluate JAS, we created 30 participant activity schedules using the SWO+SA scheduler [2]. These are the predefined fully specified activity plans of the agents participating in the process. The number of activities involved in each schedule, ranges from 6 to 31, in increments of 5. For each schedule size, we selected 5 instances, taken from the literature [1].

We considered four different meetings, all with a duration of four time units, but with a varying temporal domain, as shown in Table 1.

For each meeting, we created 20 random teams of the above agents (a total of 80 experiments), and attempt to schedule the meeting using our JAS implementation. For each meeting, we considered

[3]The full source code and the experiments' results are available online at: `https://drive.google.com/file/d/1vf_c728GK22hsz_tybo--uREZN_RN9-g/view`

Table 1: Meeting Definitions

Meeting id	Duration	Interval 1	Interval 2
1	4	[1..20]	-
2	4	[144..151]	-
3	4	[1..15]	[16..20]
4	4	[143..147]	[148..152]

participant populations of different size, from two agents to a max of five.

We set the meeting utility u_k for all users to 4, which is a value lower than the lowest activity utility in these problem instances. It should be noted that in the scheduling problem instances activity utilities were ranging from $5--12$. This choice was selected so as the scheduler tries to accommodate the meeting without "dropping" an activity previously scheduled, as explained later in the section. For all experiments we have set $NUM\_OF\_TRIES = 5$.

Table 2: Results of applying JAS to Meeting Scheduling Problems.

TS	*2 Participants*			*3 Participants*			*4 Participants*			*5 Participants*		
MID	Interval	Ph	Rs	Interval	Ph	Rs	Interval	Ph	Rs	Interval	Ph	Rs
	[7..11]	1	0	[16..20]	2	2	[1..5]	2	1	[9..13]	2	2
	[9..13]	1	0	[9..13]	1	0	[1..5]	2	3	[1..5]	2	4
1	[10..14]	2	1	[16..20]	2	2	[16..20]	2	1	[1..5]	2	3
	[6..10]	1	0	[7..11]	2	2	[1..5]	2	3	[16..20]	2	1
	[10..14]	1	0	[1..5]	2	3	[16..20]	2	3	[8..12]	2	2
	[147..151]	2	2	[145..149]	2	3	[146..150]	2	4	[147..151]	2	4
	[147..151]	2	1	[147..151]	2	2	[146..150]	2	4	[144..148]	2	4
2	[144..148]	2	2	[144..148]	2	3	[146..150]	2	3	[144..148]	2	5
	[144..148]	2	1	[147..151]	2	2	[144..148]	2	3	[144..148]	2	5
	[144..148]	2	2	[144..148]	2	2	[144..148]	2	4	[147..151]	2	3
	[16..20]	2	1	[8..12]	2	1	[16..20]	2	2	[1..5]	2	3
	[16..20]	2	1	[11..15]	2	2	[16..20]	2	2	[11..15]	2	5
3	[16..20]	2	2	[16..20]	2	1	[16..20]	2	2	[16..20]	2	2
	[16..20]	2	2	[8..12]	2	2	[16..20]	2	1	[16..20]	2	4
	[7..11]	1	0	[16..20]	2	3	[10..14]	2	1	[16..20]	2	2
	[143..147]	2	2	no sched.	2	0	[143..147]	2	4	no sched.	2	0
	no sched.	2	0	no sched.	2	0	no sched.	2	0	[143..147]	2	5
4	[143..147]	2	2	[143..147]	2	3	[143..147]	2	4	[143..147]	2	5
	[143..147]	2	2	[143..147]	2	3	no sched.	2	0	[143..147]	2	5
	[143..147]	2	2	[143..147]	2	3	[143..147]	2	4	no sched.	2	0

Table 2 presents an overview of the results of the experiments. The table depicts for all teams of agents (TS stands for Team Size), the scheduled interval found for each meeting (MID is the Meeting ID), the final JAS phase completed (column Ph) and the number of user schedules that required changed to accommodate the meeting (column Rs). In the Rs column we do not count every call to the scheduler but only the number of users whose plans were rescheduled at the end of the negotiation process. For the *schedule*() function of the second phase of JAS we called the SWO+SA scheduler[4] as an external process. The scheduler was called on a modified version of the problem instance, of the user being rescheduled, that included the meeting at the interval being tested.[5] As the SWO+SA scheduler is an optimizer, that is it tries to find the plan with the maximum utility, it would not omit an activity that gives a greater utility to include an activity that gives a lower one. By providing a low u_k value to JAS, this ensures that the SWO+SA scheduler would not remove one of the already scheduled activities to include the meeting.

[4]The SWO-SA scheduler is not an exact optimization algorithm.

[5]With a utilization value of 100%.

However, the SWO+SA scheduler could decrease the duration of a scheduled activity T_i though, if the condition $d_i^{min} < d_i^{max}$ was consistent and thus decrease the utility of the agent.

JAS managed to find a common interval in 73 out of the 80 runs of the algorithm. Since a small value was set to the NUM_OF_TRIES parameter, it is possible that a common interval could be found in the remaining 7 unsuccessful cases with more rescheduling tries. Although setting this parameter to higher values would increase the number of scheduler calls significantly, it allows testing every possible meeting interval. It should be noted that out of the 73 scheduled instances only 6 were scheduled in the first phase of JAS, i.e. solved trivially. In the rest, the meeting was successfully scheduled in the second phase, which attempted to reschedule the minimum number of users by using the *min_ints* list.

The utility change for the rescheduled users was minor in most of the cases. Table 3, shows the maximum, minimum and average percentage of utility change in the agent's plans without taking into account the added utility of the introduced meeting, in order to evaluate how rescheduling affected the previous activity plan of the agent. The worst drop was −15.62%. As the SWO+SA scheduler is stochastic there were many cases were there were even minor utility improvements in the rescheduled plans of the users. The best improvement was 2.46%. Obviously, for the cases that the meeting was not successfully scheduled, the agent utilities did not change and these cases are marked with a dash (-). From the total 175 rescheduled users, there were 95 utility drops in the rescheduled plans and 74 minor improvements. The rest had plans with the same utility.

Table 3: Utility Gains Change (%) when Scheduling a Meeting

TS	*2 Participants*			*3 Participants*			*4 Participants*			*5 Participants*		
MID	max	min	avg	max	min	avg	max	min	avg	max	min	avg
	0	0	0	0	-2.26	-0.92	0	-0.87	-0.22	0.35	-2.46	-0.42
	0	0	0	0	0	0	0.08	-2.69	-0.78	0.29	-2.63	-0.41
1	0	0	0	0.02	-1.85	-0.61	0.05	0	0.01	**2.46**	-6.61	-0.85
	0	0	0	0.31	0	0.1	0.2	-2.35	-0.73	0	-0.02	0
	0	0	0	0	-7.28	-3.25	0	**-15.62**	-3.91	0.11	-1.67	-0.31
	0.48	-0.41	0.04	0.43	-3.14	-0.82	0.05	-1.44	-0.73	0.53	-1.07	-0.23
	0	-0.04	-0.02	0.45	-0.1	0.12	0.37	-0.14	0.12	0.97	-3.94	-0.52
2	-0.17	-6.05	-3.11	0.32	-5.15	-1.71	1.19	-3.43	-0.3	0.68	-15.08	-3.39
	0.09	0	0.05	0	-1.09	-0.54	0.36	-15.14	-3.7	0.07	-5.75	-1.7
	0.26	-0.36	-0.05	0.97	-0.61	0.12	0.06	-0.81	-0.23	0.09	-0.3	-0.05
	0.03	0	0.02	0	-0.32	-0.11	0	-0.52	-0.26	0.19	-0.47	-0.09
	2.13	0	1.07	0.41	-0.21	0.07	1.19	0	0.31	-0.12	-0.93	-0.43
3	1.17	-0.19	0.49	0.08	0	0.03	0.85	-0.9	-0.01	0.16	0	0.06
	1.15	-0.99	0.08	0	-0.59	-0.25	0	-0.26	-0.07	0.15	-0.15	-0.03
	0	0	0	1.74	-0.98	0.35	0	-0.03	-0.01	0.2	-1.47	-0.25
	0.06	0.03	0.05	-	-	-	0.28	-0.59	0.02	-	-	-
	-	-	-	-	-	-	-	-	-	-0.07	-1.23	-0.6
4	0.16	0.02	0.09	1.87	0.07	0.76	0.78	-0.88	-0.06	0.2	-1.18	-0.3
	0.98	-0.43	0.28	0.27	-0.95	-0.21	-	-	-	0.14	-1.29	-0.38
	-0.04	-0.26	-0.15	1.23	-0.17	0.37	0.1	-1.45	-0.35	-	-	-

6 Conclusions

The work described in this article addresses the problem of automated meeting scheduling between a number of self-interested agents, under a rich model of individual activities and a typical model for the meeting. As commonly used in other work, the multi-agent negotiation process that takes place is driven by a coordinator agent responsible for generating proposals, to which agents reply based on utility values derived from employing a greedy construction and stochastic optimization scheduler to the new scheduling problem that includes the proposal. Initial experimental evaluation, demonstrates that the approach performs well, reaching an agreement on the meeting time slot in the majority of experiments.

There are several directions that the present work can be extended. Currently, we do not consider alternative meeting locations in proposals, a feature that would certainly increase the number of necessary negotiation rounds, and although such an option in most business cases might not be of great interest, it might be interesting in other cases. JAS can also be extended so that meetings could be converted to full joint-activities, that is both temporal preferences (i.e., schedule the meeting at morning/noon/evening) and binary preferences could be defined over them by each agent, as in individual activities.

More experiments should also be conducted to evaluate the proposed approach. JAS is deterministic and should always arrive to the same common scheduled interval (given a deterministic rescheduler), but SWO+SA is stochastic. When the interval being tested by SWO+SA is given a high enough utility, JAS should always arrive to the same common scheduled interval, while the rest of the users' rescheduled plans may be different though with different utilities between runs. By giving the interval being tested a low enough utility, it is possible that JAS could output different common scheduled intervals between runs. This needs to be tested. Moreover, different parameters for the algorithm, as well as different heuristics for the ordering of the intervals to be evaluated could also be tested. Future experiments should also measure the number of tested intervals and provide a metric for the computational efficiency of the proposed algorithms.

Rearranging agent meetings to accommodate the one under negotiation, i.e., "bumping", presents also a very interesting research direction that we aim to investigate, since there is a number of interesting questions that arise, as for example when should this "recursive" process terminate.

Finally, an experimental evaluation of the proposed algorithms in a real distributed environment would allow to measure the algorithms performance in terms of execution time and number of messages exchanged in order to fully evaluate its potential in a real-life setting. Obviously, integrating the proposed algorithms with modern calendar applications is the ultimate goal.

References

[1] Anastasios Alexiadis and Ioannis Refanidis. Alternative plan generation and online preference learning in scheduling individual activities. *International Journal on Artificial Intelligence Tools*, 25(3):1–28, 2016.

[2] Anastasios Alexiadis and Ioannis Refanidis. Optimizing individual activity personal plans through local search. *AI Communications*, 29(1):185–203, 2016.

[3] Ahlem BenHassine and Tu Bao Ho. An agent-based approach to solve dynamic meeting scheduling problems with preferences. *Engineering Applications of Artificial Intelligence*, 20(6):857–873, September 2007.

[4] Pauline Berry, Melinda Gervasio, Bart Peintner, and Neil Yorke-Smith. Balancing the needs of personalization and reasoning in a user-centric scheduling assistant. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL IN[U+2121]LIGENCE CENTER, 2007.

[5] Pauline M. Berry, Thierry Donneau-Golencer, Khang Duong, Melinda T. Gervasio, Bart Peintner, and Neil Yorke-Smith. Evaluating User-Adaptive Systems: Lessons from Experiences with a Personalized Meeting Scheduling Assistant. In *IAAI*, volume 9, pages 40–46, 2009.

[6] Mike Brzozowski, Kendra Carattini, Scott R. Klemmer, Patrick Mihelich, Jiang Hu, and Andrew Y. Ng. groupTime: preference based group scheduling. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1047–1056. ACM, 2006.

[7] Andy Chun, Hon Wai, and Rebecca Y. M. Wong. Optimizing agent-based meeting scheduling through preference estimation. *Engineering Applications of Artificial Intelligence*, 16(7):727–743, October 2003.

[8] Maria Sole Franzin, E. C. Freuder, F. Rossi, and R. Wallace. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In *Proceedings of the AAAI Workshop on Preference in AI and CP*, 2002.

[9] Nick R. Jennings and A. J. Jackson. Agent-based meeting scheduling: A design and implementation. *Electronics letters*, 31(5):350–352, 1995.

[10] Pragnesh Jay Modi and Manuela Veloso. Bumping strategies for the multiagent agreement problem. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 390–396. ACM, 2005.

[11] Pragnesh Jay Modi, Manuela Veloso, Stephen F. Smith, and Jean Oh. Cmradar: A personal assistant agent for calendar management. In *Agent-Oriented Information Systems II*, pages 169–181. Springer, 2005.

[12] Terry R. Payne, Rahul Singh, and Katia Sycara. Rcal: A case study on semantic web agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 802–803. ACM, 2002.

[13] Ioannis Refanidis and Neil Yorke-Smith. A constraint-based approach to scheduling an individual's activities. *ACM Trans. Intell. Syst. Technol.*, 1(2):12:1–12:32, December 2010.

[14] Sandip Sen and Edmund H. Durfee. A contracting model for flexible distributed scheduling. *Annals of Operations Research*, 65(1):195–222, 1996.

[15] F. Wang. Adaptive meeting scheduling for large-scale distributed groupware. *BT technology journal*, 21(4):138–145, 2003.

[16] Alejandro Zunino and Marcelo Campo. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 36(3):7011–7018, April 2009.