Reasoning about Time Constraints in a Mixed-Initiative Calendar Manager

Liliana Ardissono and **Giovanna Petrone** and **Marino Segnan** and **Gianluca Torta**
Dipartimento di Informatica, Università di Torino, Italy
email {liliana.ardissono, giovanna.petrone, marino.segnan, gianluca.torta}@unito.it

Abstract

Scheduling support is very important for calendar management in order to automatize the execution of possibly complex reasoning tasks. However, an interactive approach is desirable to enable the user to steer the allocation of events, which is a rather personal and critical kind of activity. This paper proposes a mixed-initiative scheduling model supporting the user's awareness during the exploration of the solution space. The paper describes the temporal reasoning techniques underlying MARA (Mixed-initiAtive calendaR mAnager), focusing on the generation of scheduling options and on the characterization of their properties, needed to present the pros and cons of each possible solution to the user.

Keywords: mixed-initiative scheduling, temporal reasoning, constraint satisfaction.

Introduction

Calendar management is burdensome and challenging when schedules are overconstrained or include items having multiple participants because it requires the verification of a possibly large number of temporal constraints. However, a fully automated scheduling support is considered as hardly acceptable for this type of activity because it fails to keep the user in control of the decisions to be taken; e.g., see (Berry et al. 2011).

In the attempt to address this issue, we propose a new, mixed-initiative scheduling model that enables the user to temporally allocate multi-user events and tasks *in cooperation* with the system. The paper also proposes a novel, *conservative scheduling policy* to suggest calendar revisions by modifying small portions of the schedules, leaving the rest as originally planned or with minor temporal shifts. The idea is that of keeping the changes to the user's plans as local as possible in order to maintain stable daily plans.

Our scheduling model is applied in the MARA Mixed-initiAtive calendaR mAnager, which exploits Temporal Constraint Satisfaction Problem techniques for suggesting safe scheduling solutions across multiple calendars. The mixed-initiative interaction with the user is achieved by invoking the Interval-based TEmporal Reasoner (ITER) for computing a synthesis of the solutions to be proposed to the user, instead of presenting a possibly large number of alternative schedules to choose from. Such a synthesis is based on the specification of *admissible intervals* for the allocation of items and of the corresponding impact on the calendars of the involved people. In this way, the user can analyze the solution space at an abstract level and select the paths which are worth to be explored in an informed way.

In the following, we first describe the mixed-initiative scheduling support offered by MARA. Then, we discuss in detail the temporal reasoning techniques adopted in ITER. Finally, we present related research and conclusions.

Mixed-Initiative Scheduling in MARA

Calendar Management

MARA supports cross-calendar management providing the user with an overview of the impact of her/his actions on the schedules of all the involved actors. Figure 1 shows a portion of the User Interface of the system. The table in the upper right portion of the page shows the list of shared calendars and enables the user to select those to be jointly visualized. In order to let the user identify the actors involved in a calendar item (e.g., a meeting), each item has associated a number of vertical bars whose colors correspond to those of the respective calendars; e.g., the participants of event CDD in Figure 1 are Gianluca and Prof. Rossi.

While the user adds new items or revises the existing ones, the system checks whether the temporal constraints of the affected items are satisfied. If any conflicts occur, it notifies the user; moreover, it helps her/him to incrementally explore the solution space in order to quickly evaluate the available options. Specifically:

- The system offers a "Where can I place the task?" feature which helps the user to allocate calendar items. When the user asks for help regarding an item M, MARA presents in a calendar window an overview of the feasible time intervals in which M could be allocated, possibly moving other existing items. Each interval I provides the user with information useful to evaluate its conveniency; i.e.: (i) the names of the actors whose existing commitments have to be revised if M is placed in I; (ii) the criticity of placing M in I, given the existing commitments of the involved actors (e.g., the criticity is high if at least one high-priority commitment has to be shifted in order to allocate the item).

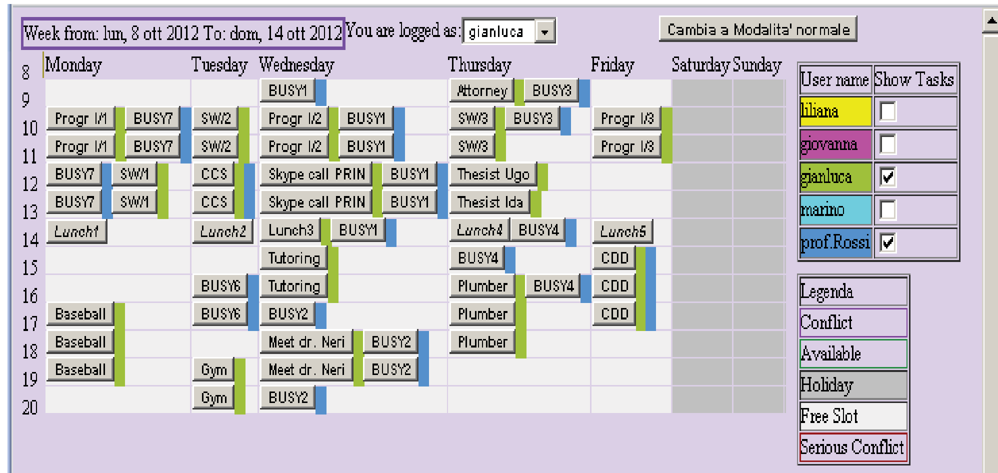

Figure 1: Gianluca's calendar, jointly visualized with Prof. Rossi's one.

- The user can select a specific interval for scheduling the item. At that point, the system further enables her/him to steer the scheduling activity by presenting a few alternative revision hypotheses for the calendar; e.g., alternative items might be shifted to allocate M. The user can accept one of the suggestions, in which case the involved actors are informed and asked to confirm the change, or (s)he can go back to the available options and investigate further revision opportunities.

See (Ardissono, Segnan, and Torta 2013) for more information about MARA's User Interface and functionality.

System Architecture

MARA stores the information about calendars, items and actors as lists of objects. While the user revises a calendar, the core of MARA invokes the following modules to check whether the user's actions violate any temporal constraints and to suggest how such conflicts might be solved:

1. Every time an item is manually added/moved, a *consistency check module* checks the consistency of the temporal constraints associated to the item to verify whether they are satisfied or not.
2. In case of inconsistency (or, more generally, when requested by the user), the *Interval-based TEmporal Reasoner (ITER)* supports the addition/movement of an item by identifying the feasible intervals where it could be allocated and the consequent impact on the schedules of the involved actors. Once the user chooses where to place the item, this module generates the corresponding conservative calendar revisions

MARA is a Java Web application. The ITER module is developed in Perl using the Graph.pm extension module (Hietaniemi 2010) for representing and manipulating STNs. The minimization of the STNs is performed by invoking the implementation of the Floyd-Warshall algorithm included in Graph.pm. The Java Web application invokes ITER as a local REST service via HTTP.

Temporal Reasoning Underlying MARA

Running example

In this section we will use the following scenario to illustrate the temporal resoning performed by ITER. Our running example refers to the calendar displayed in Figure 1.

Gianluca is a University staff member and he collaborates with some colleagues (Liliana, Giovanna, Marino) and with the Head of Department, Prof. Rossi.

Gianluca's calendar includes teaching hours (e.g., Progr I, SW), Department meetings (CDD, CCS), student support (Tutoring, Thesist Ugo / Ida), project meetings (e.g., Skype call PRIN, Meet Dr. Neri) and personal commitments (e.g., Baseball, Plumber, Attorney). Some activities have a fixed schedule; e.g., teaching hours and Department meetings. Others are flexible and could be moved to other times if needed; for instance, Gianluca is available for student support on Wednesday, Thursday and Friday.

Suppose that Gianluca has to schedule a 2-hours Staff meeting with Prof. Rossi on Thursday. The event can start at 8.00 and must finish by 16.00 (deadline). In order to accommodate this meeting, the current calendar has to be revised. By analyzing Prof. Rossi's constraints, it can be seen that he is available starting from 10.00 until 13.00. However, Gianluca is busy at that time because he teaches SW from 9.00 to 11.00 and then he meets two students (Thesist Ugo / Ida). The SW lesson cannot be moved. Thus, the only solution is to move Thesist Ugo and Thesist Ida to different times.

In a typical calendar manager this revision would imply that Gianluca first analyzes all the relevant commitments (including Prof. Rossi's ones) in order to identify the items that can be moved and their alternative times; then, he manually moves the selected items; finally, he inserts the new item. Considering that calendar revision is a frequent daily activity, it is worth saving as much effort as possible in it. Thus,

an automatic support to its execution is crucial. Our work attempts to address this need.

Representation of Calendars Items

The ITER module exploits reasoning techniques based on Temporal Constraint Satisfaction Problems (TCSP) (Dechter, Meiri, and Pearl 1991). Specifically, ITER employs a subclass of TCSPs, the Simple Temporal Problems (STPs) (Dechter, Meiri, and Pearl 1991), where all of the constraints are binary and they do not contain any disjunctions, i.e., they have the following format:

$$a \leq X_j - X_i \leq b$$

This class of problems can be represented as a graph named Simple Temporal Network (STN), whose consistency can be checked in polynomial time (Planken, de Weerdt, and van der Krogt 2011). Also the *minimization* of an STN (i.e., the computation, for each pair of variables X_i, X_j, of an interval $[a_{min}, b_{min}]$ which guarantees the existence of a global solution for the STN) can be done in polynomial time.

The scheduling of a set of calendars is done by reasoning on the joint set of constraints associated to their items. A calendar item is represented as an object having several features, among which the expected *duration* (number of hours devoted to the item), the *earliest start time* for scheduling it, the *deadline* for its completion/end, the *schedule* (current temporal allocation), the list of *participants* and a *priority* (low, medium, high) representing the importance of the commitment.

Given such information, each calendar item M is internally represented by means of:

- Two numeric variables M_s and M_e, representing the start and end time of M in a given schedule. For simplicity, we assume that the value of a variable M_s (resp. M_e) is the number of one-hour slots between Monday 8.00 and the start (respectively the end) of M. Note, however, that the TCSP techniques we use are able to deal with real numbers, so that we could easily deal with finer granularities of time.
- The temporal constraints on M_s and M_e needed to schedule M consistently with its earliest start time, duration and deadline.
- The temporal constraints on M_s and M_e needed to impose precedence relations with respect to other calendar items.

For instance, given an item Staff Meeting (SM in Figure 1):

- The *earliest start time*, Thursday at 8.00, is expressed as $SM_s \geq 36$ because in the calendar there are 36 one-hour slots between Monday 8.00 and Thursday 8.00.
- The *deadline*, Thursday 16.00, is expressed as $SM_e \leq 44$.
- The *duration*, 2 hours, is $SM_e - SM_s = 2$ as the item takes 2 time slots.
- A *precedence* relation among calendar items, e.g., the fact that SM must take place after another event E, is expressed as $SM_s - E_e \geq 0$.

With slight abuse we use the term deadline also to indicate constraints on the exact end of a calendar item; e.g., the fact that an item P must end *exactly* on Wednesday at 13.00 is represented as $P_e = 29$ (i.e., $29 \leq P_e \leq 29$).

ITER: Computing the Feasible Intervals

Given the temporal constraints of the calendar and an item M to be added (or moved), ITER:

- Searches for feasible intervals for allocating M. For each interval, it computes the criticity on the basis of the priorities of the existing items which should be moved to allocate M in the corresponding time window.
- Generates a schedule, given a (feasible) start time for M selected by the user.

ALGORITHM 1: Feasible intervals for starting a new calendar item M for a specific user U_i.

```
input:
    new item M
    other U_i items in current schedule order (T_{i,1}, ..., T_{i,ki})
    STN N_i (temporal constraints for M, T_{i,1}, ..., T_{i,ki})
    S_i current schedule for T_{i,1}, ..., T_{i,ki}
1 foreach C ∈ {⊥, L, M, H} do
2 |   I_i^C ← Intervals(M, (T_{i,1}, ..., T_{i,ki}), N_i, S_i, C)
3 end
4 I_i ← DComp(I_i^⊥, I_i^L, I_i^M, I_i^H);
5 return I_i
```

Note that, different from a classic scheduler, ITER computes a set of *intervals* within which the item can be placed, instead of a set of *specific places*. In this way, ITER supports a compact and synthetic presentation of scheduling options, to be further refined. As we shall see, each interval corresponds to different revisions of the calendars, based on which time point within the interval will be chosen by the user, as well as additional user input.

The feasible intervals and the schedules that are computed by ITER are restricted by *conservativeness* criteria. In general, this means that placing M in a feasible interval does not require a heavy revision of the current schedule. In our current implementation, we adopt a simple conservativeness criterion, requiring that the relative order of existing items in the schedule can be maintained when M is added.

For simplicity, we will first consider feasible intervals which do not require to move any already scheduled items involving multiple users. The relaxation of this restriction will be discussed later, in the section on handling existing items involving multiple actors.

We introduce a notation in order to tag feasible intervals with information about their criticity. Each computed interval $I_{M,i}$ will have an associated *label*:

$$l(I_{M,i}) = \{U_1^{C1}, \ldots, U_m^{Cm}\}$$

where $U_1, \ldots, U_m$ are the users involved by M and the C_i superscripts denote the criticity of the interval for each actor U_i; i.e., the impact on U_i's calendar of allocating

M in the interval $I_{M,i}$, in terms of revisions. Specifically, $C_i \in \{\bot, \mathtt{L}, \mathtt{M}, \mathtt{H}\}$, where:

- $\bot$ means that the schedule of U_i is not affected by M;
- L: only low-importance commitments of U_i have to be shifted to allocate M;
- M: only medium- and low-importance commitments of U_i have to be shifted;
- H: at least one high-importance commitment of U_i has to be shifted;

The computation of the set of feasible intervals $\mathcal{I}_M$ for M is split in two steps:

1. Computing the feasible intervals $\mathcal{I}_i = (I_{i,1}, \ldots, I_{i,ni})$ for each user U_i. Each interval $I_{i,j}$ is assigned a label $l(I_{i,j}) = U_i^{Cj}$ with $C_j \in \{\bot, \mathtt{L}, \mathtt{M}, \mathtt{H}\}$; the label specifies how critical is to allocate M in that interval, given the temporal constraints of U_i's commitments.
2. Computing the joint feasible intervals $\mathcal{I}_M$ and their labels from user intervals $\mathcal{I}_i$ in order to synthesize a (limited) number of options to be considered for scheduling M by its organizer.

Computing the User Intervals. *Algorithm 1* (on the previous page) concerns a single user U_i and implements the first step. It takes as inputs: (i) the new item M to be allocated; (ii) U_i's other items $(T_{i,1}, \ldots, T_{i,ki})$ in the order in which they appear in the current schedule; (iii) an STN $\mathcal{N}_i$ encoding the deadline, duration and precedence constraints for $T_{i,1}, \ldots, T_{i,ki}$ and M; and, (iv) the current scheduled times $\mathcal{S}_i$ for $T_{i,1}, \ldots, T_{i,ki}$.

In the loop starting at line 1, the algorithm computes four sequences of intervals $\mathcal{I}_i^{\bot}$, $\mathcal{I}_i^{\mathtt{L}}$, $\mathcal{I}_i^{\mathtt{M}}$, $\mathcal{I}_i^{\mathtt{H}}$ by invoking procedure *Intervals*, described below. Each sequence $\mathcal{I}_i^C$ contains $(k_i + 1)$ intervals, one for each position where M can be placed in the order of the existing items $T_{i,1}, \ldots, T_{i,ki}$. For each position j, interval $I_{i,j}^C \in \mathcal{I}_i^C$ represents the set of time points t such that starting M at time t requires to modify U_i's schedule by shifting items of importance C or lower.

The intervals in sequences $\mathcal{I}_i^{\bot}$, $\mathcal{I}_i^{\mathtt{L}}$, $\mathcal{I}_i^{\mathtt{M}}$, $\mathcal{I}_i^{\mathtt{H}}$ can overlap both within the same sequence and among different sequences. Thus, in line 4 the algorithm invokes procedure *DComp* to merge $\mathcal{I}_i^{\bot}$, $\mathcal{I}_i^{\mathtt{L}}$, $\mathcal{I}_i^{\mathtt{M}}$, $\mathcal{I}_i^{\mathtt{H}}$ into a single sequence $\mathcal{I}_i$ of labeled and ordered disjoint intervals. Each overlapping portion is labeled with the lowest (i.e., best) criticity class to which it belongs.

Procedure *Intervals* is called to generate a sequence of intervals of given criticity C. Therefore, it immediately adds the scheduled times of tasks of classes $C' > C$ to the temporal constraints encoded by STN $\mathcal{N}_i$, obtaining STN $\mathcal{N}_i^C$ (line 2). With these added constraints, $\mathcal{N}_i^C$ ensures that no task with criticity $C' > C$ can be moved.

Then, *Intervals* considers each possible positioning j of M in the sequence of existing items $T_{i,1}, \ldots, T_{i,ki}$ involving U_i. In this way a total order Π is determined among all the items, including M, and can be asserted as a set of precedence constraints into the STN. The resulting STN $\mathcal{N}_{i,j}^C$ is then minimized, yielding a feasible interval $I_{i,j}^C =$

Procedure Intervals - feasible intervals of a given class C for starting item M.

---

input:
  new item M
  other U_i items in current schedule order $(T_{i,1}, \ldots, T_{i,ki})$
  STN $\mathcal{N}_i$ (temporal constraints for $M, T_{i,1}, \ldots, T_{i,ki}$)
  $\mathcal{S}_i$ current schedule for $T_{i,1}, \ldots, T_{i,ki}$
  C criticity class of the computed intervals

1 $\mathcal{I}_i^C \leftarrow ()$;
2 $\mathcal{N}_i^C \leftarrow$ assert scheduled times of tasks of classes $C' > C$ in $\mathcal{N}_i$;
3 **for** $j = 0 \ldots k_i$ **do**
4   $\Pi \leftarrow (T_{i,1}, \ldots, T_{i,j}, M, T_{i,j+1}, \ldots T_{i,ki})$;
5   $\mathcal{N}_{i,j}^C \leftarrow$ assert order Π in $\mathcal{N}_i^C$;
6   minimize $\mathcal{N}_{i,j}^C$;
7   $I_{i,j}^C \leftarrow$ get interval $[min, max]$ for M_s from $\mathcal{N}_{i,j}^C$;
8   $\mathcal{I}_i^C \leftarrow \mathcal{I}_i^C \cdot \left(I_{i,j}^C\right)$;
9 **end**
10 **return** $\mathcal{I}_i^C$

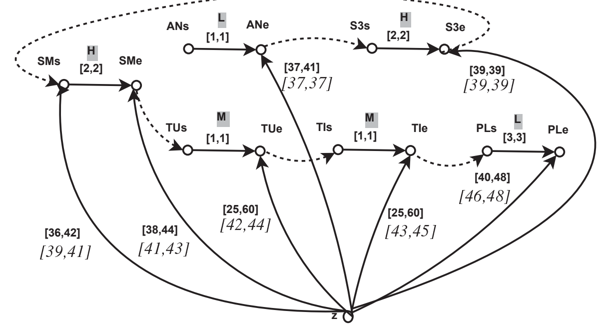

Figure 2: STN $\mathcal{N}_2^{\mathtt{H}}$ representing Gianluca constraints on Thursday when Staff Meeting is placed between S3 and TU and all existing items can be shifted.

$[min, max]$ for the start of M. As said above, for each position j of M, such an interval represents the set of time points t such that starting M at time t requires to shift items having importance C or lower in U_i's schedule. Intervals $I_{i,j}^C$ are added to sequence $\mathcal{I}_i^C$ and, after all the positions have been considered, such a sequence is returned.

Example 1 *Let us refer to the running example and consider the execution of* Intervals *when it is invoked on user Gianluca for adding the Staff meeting SM on Thursday with the class parameter C set to* H. *That day, the items already allocated for Gianluca include, in the order: Attorney (AT), SW/3 ($S3$), thesist Ugo (TU), thesist Ida (TI) and Plumber (PL).*

Figure 2 shows a portion of the STN $\mathcal{N}_2^{\mathtt{H}}$ computed by Intervals *for user* Gianluca. *Such a network corresponds to the iteration of the* Intervals *procedure when the* for *loop has set position $j = 2$ (i.e., between $S3$ and TU).*

The z time point represents Monday 8.00 and the boldface intervals on the arcs express the minimum and maximum

distance between the connected time points. For example, interval $[36, 42]$ on the arc connecting z and SM_s represents: $36 \leq SM_s - z \leq 42$; i.e., SM must start on Thursday between 8.00 and 14.00. The dashed arcs represent the precedence between two items T', T'' in the current order Π. Their associated intervals, omitted for readability, would be $[0, +\infty]$; i.e., T''_s must follow T'_e by at least 0 hours. Moreover, each item is labeled with its importance; e.g., AT has low importance and TU has medium importance.

The intervals after the minimization of $\mathcal{N}_2^{\mathtt{H}}$ are shown in italics. Specifically, the intervals computed for SM_s, SM_e are, respectively, $[39, 41]$ (Thursday 11.00 to 13.00) and $[41, 43]$ (13.00 to 15.00). Indeed, when SM is positioned between $S3$ and TU, it can start only after the end of $S3$ (11.00) and its latest end (15.00) must leave enough time for TU, TI and PL to be completed by 20.00. The start interval $[39, 41]$ is added to the sequence $\mathcal{I}_i^{\mathtt{H}}$ of feasible intervals with class $\mathtt{H}$ for Gianluca.

Let us now consider the interval for SM_s when position is $j = 2$ but the class C is $\mathtt{L}$, i.e., scheduled items of importance $\mathtt{M}$ and $\mathtt{H}$ cannot be shifted. In such a case, the interval for SM_s is $\emptyset$ because TU cannot be moved (it has medium importance); thus, SM cannot be allocated between $S3$ and TU.

By repeating the process, it is easy to see that the non-empty intervals of class $\mathtt{H}$ for user Gianluca are: $[39, 41]$ ($j = 2$); $[40, 42]$ ($j = 3$); $[41, 42]$ ($j = 4$). The $\mathtt{M}$ intervals are the same but the $\mathtt{L}$ and $\perp$ intervals are empty.

Procedure *DComp* receives sequences $\mathcal{I}_i^{\perp}, \mathcal{I}_i^{L}, \mathcal{I}_i^{M}, \mathcal{I}_i^{H}$ and composes them into a single sequence $\mathcal{I}_i$ of ordered, labeled, disjoint intervals. In line 1, it orders the start and end time points of each of the input intervals and it stores them in a list $\mathcal{T}$. This operation consists of two steps:

1. Flatten each input list of intervals $\mathcal{I}_i^C = (I_{i,0}^C, \ldots, I_{i,ki}^C)$ into an ordered list of start and end time points $(s(I_{i,0}^C), \ldots, e(I_{i,ki}^C))$;
2. Merge such ordered lists into list $\mathcal{T}$. In case of ties, put start time points before end time points, and further order start (resp., end) points in increasing (resp., decreasing) order of their criticity classes ($\perp$, $\mathtt{L}$, $\mathtt{M}$, $\mathtt{H}$). In other words, if several intervals start at the same time, the start of the best (i.e., lowest class) is the first one in the order. Moreover, if several intervals end at the same time, the end of the best interval is the last one.

In the procedure, line 2 initializes the output sequence $\mathcal{I}_i$ to the empty list. Moreover it sets to 0 the counter n_{in} of input intervals and the counter n_C of input intervals of class C to which the elements of $\mathcal{T}$ belong. The loop starting at line 3 considers each $t \in \mathcal{T}$. If t is the start point of an input interval $I_{i,j}^C$ (line 4), n_{in} and n_C (intervals to which t belongs) are incremented. Then:

- If n_{in} is equal to 1, t corresponds to the start $s(I)$ of a new output interval I (line 7) whose label is set to U_i^C, because the input interval has class C.
- If $n_{in} > 1$ but the class C of the input interval is lower than the label l of the current output interval, a new output interval has to be created after closing and adding to the output sequence $\mathcal{I}$ the current output interval (line 10).

If t is the end point of an input interval $I_{i,j}^C$ (line 17), n_{in} and n_C are decremented.

- If n_{in} is equal to 0, t is the end $e(I)$ of the current output interval I (line 20), which has to be closed and added to sequence $\mathcal{I}$.
- If $n_{in} > 1$ but the class C of the input interval is equal to the label l of the current output interval and $n_C = 0$, a new output interval has to be created after closing and adding to the output sequence $\mathcal{I}$ the current output interval (line 23). In line 27 the label of the new ouput interval is set to the minimum (best) class C_{new} associated with at least one input interval which includes t, i.e. such that $n_{C_{new}} > 0$.

Procedure DComp - disjunctive composition of labeled intervals.

input:
sequences $\mathcal{I}_i^{\perp}, \mathcal{I}_i^{L}, \mathcal{I}_i^{M}, \mathcal{I}_i^{H}$, each one containing intervals $I_{i,j}^C$, $j = 0, \ldots, ki$

```
1  T ← ordered list of time points s(I^C_{i,j}), e(I^C_{i,j}) for all given I^C_{i,j};
2  I_i ← (); n_in ← 0; n_C ← 0, C ∈ {⊥,...,H}; l ← U_i^⊥;
3  foreach t ∈ T do
4      if (t = s(I^C_{i,j})) then // start of an input interval
5          n_in ← n_in + 1;
6          n_C ← n_C + 1;
7          if (n_in = 1) then // start of an output interval
8              s(I) ← t;
9              l ← U_i^C
10         else if (C < l) then // new output interval with label C
11             e(I) ← t, l(I) ← l;
12             I_i ← I_i · I;
13             s(I) ← t;
14             l ← U_i^C
15         end
16     end
17     if (t = e(I^C_{i,j})) then // end of an input interval
18         n_in ← n_in − 1;
19         n_C ← n_C − 1;
20         if (n_in = 0) then // end of an output interval
21             e(I) ← t, l(I) ← l;
22             I_i ← I_i · I;
23         else if ((l = C) and (n_C = 0)) then // new output interval with label > C
24             e(I) ← t, l(I) ← l;
25             I_i ← I_i · I;
26             s(I) ← t;
27             l ← U_i^{Cnew} where C_new = min{C' : n_C' > 0}
28         end
29     end
30 end
31 return I_i
```

Example 2 *As shown in example 1, the* H *and* M *sequences for Gianluca contain the following intervals:* $[39, 41]$ *(*$j = 2$*);* $[40, 42]$ *(*$j = 3$*);* $[41, 42]$ *(*$j = 4$*). The sequence* $\mathcal{T}$ *of time points considered by* DComp *is therefore:*

$$\mathcal{T} = (39^{\mathtt{M}}_s, 39^{\mathtt{H}}_s, 40^{\mathtt{M}}_s, 40^{\mathtt{H}}_s, 41^{\mathtt{M}}_s, 41^{\mathtt{H}}_s, 41^{\mathtt{H}}_e, 41^{\mathtt{M}}_e, 42^{\mathtt{H}}_e, 42^{\mathtt{H}}_e, 42^{\mathtt{M}}_e, 42^{\mathtt{M}}_e)$$

where we have marked each $t \in \mathcal{T}$ *with the class of its interval and its role (start/end). When* $t = 39^{\mathtt{M}}_s$ *is considered, a new ouput interval* I *is started with* $I_s = 39$ *and label* $Gl^{\mathtt{M}}$ *(for* Gianluca*). Time point* $t = 39^{\mathtt{H}}_s$ *is skipped because its class is higher than the current output class (line 10 of* DComp*). For the same reason, time points* $40^{\mathtt{M}}_s$,$40^{\mathtt{H}}_s$,$41^{\mathtt{M}}_s$,$41^{\mathtt{H}}_s$ *are skipped. Note that meanwhile the number* n_{in} *of feasible input intervals has grown to* 6*, with* $n_{\mathtt{H}} = 3$ *and* $n_{\mathtt{M}} = 3$*. For this reason also ending time points* $41^{\mathtt{H}}_e$,$41^{\mathtt{M}}_e$,$42^{\mathtt{H}}_e$,$42^{\mathtt{H}}_e$,$42^{\mathtt{M}}_e$ *are skipped (line 20 of* DComp*). When the last point* $t = 42^{\mathtt{M}}_e$ *is considered,* n_{in} *drops to* 0 *and the (only) output interval* $[39, 42]$ *with label* $Gl^{\mathtt{M}}$ *is returned in the sequence* $\mathcal{I}_i$ *for Gianluca.*

Computing the Joint Intervals. For each user U_i involved in M, *Algorithm 1* computes a sequence of ordered, disjoint intervals $\mathcal{I}_i$ such that each interval $I_{i,j}$ has a label $l(I_{i,j})$ which takes values in U_i^C, $C \in \{\perp, \mathtt{L}, \mathtt{M}, \mathtt{H}\}$.

Given that the involved users are $\{U_1, \ldots, U_m\}$, ITER computes a single sequence of ordered, disjoint intervals $\mathcal{I}_M = (I_{M,1}, \ldots, I_{M,n})$ such that each element $I_{M,j}$ of $\mathcal{I}_M$ represents a jointly feasible interval for starting M. Interval $I_{M,j}$ is labeled based on the labels of the user intervals from which it is derived. Given $\mathcal{I}_1, \ldots, \mathcal{I}_m$, the sequence of jointly feasible intervals $\mathcal{I}_M$ satisfies the following conditions:

- Two time points t, t' belong to an interval $I_{M,j}$ iff for each involved user U_i they belong to the same user interval $I_{i,j_i} \in \mathcal{I}_i$.
- The label $l(I_{M,j})$ associated with interval $I_{M,j}$ is given by $\bigcup_i l(I_{i,j_i})$.

The computation of $\mathcal{I}_M$ is performed by a procedure *JComp* (joint composition) analogous to *DComp*. The procedure receives the sequences of intervals $\mathcal{I}_i$ for all users U_i and produces the single sequence $\mathcal{I}_M$ of jointly feasible intervals.

Example 3 *As shown in Example 2, the only interval for user Gianluca is* $[39, 42]$ *with label* $Gl^{\mathtt{M}}$*. Let us assume that Prof. Rossi's* BUSY3 *cannot be moved and that* BUSY4 *could be postponed at 14.00. Then, for Prof. Rossi there are two feasible intervals:* $[38, 39]$ *with label* $Ro^{\perp}$ *(i.e., his schedule has not to be modified) and* $[39, 40]$ *with label* $Ro^{\mathtt{M}}$ *(medium importance task* BUSY4 *has to be moved). The invocation of* JComp *on the sequences of intervals for Gianluca and Prof. Rossi returns a single interval* $[39, 40]$ *with label* $\{Ro^{\mathtt{M}}, Gl^{\mathtt{M}}\}$*. Indeed,* $[39, 40]$ *is the only feasible time window for both Prof. Rossi and Gianluca; moreover, the interval has impact* M *on both actors. Therefore,* SM *must start at 11.00 or later and end by 14.00.*

ITER: Computing Revisions to a Calendar

We briefly describe the computation of revisions to shared calendars because it only requires to run again the *Intervals* procedure used for computing the feasible intervals.

Let us consider the feasible interval $[39, 40]$ with label $\{Ro^{\mathtt{M}}, Gl^{\mathtt{M}}\}$ computed for SM, and let us suppose that Gianluca places SM at time point 40 (12.00 pm). In order to compute the alternative revisions, *Intervals* has to be invoked with the additional constraints that SM starts at time point 40 and that this allocation has impact M on Gianluca and Prof. Rossi's calendars. For Gianluca, SM can be placed between $S3$ and TU, or between TU and TI; for Prof. Rossi, between $BUSY3$ and $BUSY4$.

The first revision of Gianluca's calendar is obtained by considering the minimized STN computed by placing SM between items $S3$ and TU, which also contains feasible intervals for the other items in the calendar. For each such item we choose a start time point as close as possible to its current schedule, which results in pushing TU at 14.00, TI at 15.00, and PL at 16.00. The second revision of Gianluca's calendar and the only available revision of Prof. Rossi's calendar are computed in a similar way.

Handling existing items involving multiple actors

The results presented in the previous section hold when existing items which involve multiple actors (henceforth, meetings) cannot be anticipated nor postponed. However, when looking for the feasible intervals for a new item M, it is desirable to consider re-scheduling such items. Let us assume that M involves a set of actors $\mathcal{U} = \{U_1, \ldots, U_m\}$. The previous meetings of users $\mathcal{U}$ can be of two kinds:

1. They involve only (some of) the members of $\mathcal{U}$.
2. They have additional participants $\mathcal{U}'$ which are not involved in M.

Re-scheduling calendar items of the second kind can result in a domino effect: actors in $\mathcal{U}'$ may have existing meetings with yet other users $\mathcal{U}''$, and so forth. Thus, adding M may lead to revise schedules of people having an indirect connection with users $\mathcal{U}$. While we believe this is an interesting problem, that may likely involve some forms of automatic negotiation, such propagations are out of the scope of this paper. Thus, we add the following restriction: if an existing calendar item which involves actors $\mathcal{U}$ also involves other actors $\mathcal{U}'$, it can be only moved to time slots where the members of $\mathcal{U}'$ are available.

In the following we discuss how to handle the items involving subsets of $\mathcal{U}$. Handling additional users with the above stated restriction trivially consists in pruning some of the solutions computed for users $\mathcal{U}$, based on the free time slots of users $\mathcal{U}'$. Therefore, we do not discuss it.

In order to handle existing meetings among users $\mathcal{U}$, we first partition them in families $\mathcal{U}_1, \ldots, \mathcal{U}_p$ such that:

- Users U, U' should be in the same family $\mathcal{U}$ if there is a meeting involving U, U'.
- Families should be disjoint, i.e., if two families $\mathcal{U}'$, $\mathcal{U}''$ share at least one user, they are replaced by a new family $\mathcal{U} = \mathcal{U}' \cup \mathcal{U}''$.

The previously described techniques can be applied to this case by computing the sequences of intervals $\mathcal{I}_i$ for each *family* instead of for each user. Indeed, if users $U'_1, \ldots, U'_q$ in a family have an existing shared meeting M', each of them has an item M'_i in her/his calendar representing M' and the start/end times of items M'_i, $i = 1, \ldots, q$ must be equal. For this reason, we have to build and use an STN that encodes the calendar constraints of all the users in the family. After the sequences for each family have been computed, they can be merged with the previously described *JComp* procedure for computing jointly feasible intervals.

We now explain how *Algorithm 1*, *Intervals* and *DComp* are adapted to operate on families of users.

Let us start by considering the changes to *Algorithm 1*. Instead of receiving a single sequence of items $(T_{i,1}, \ldots, T_{i,ki})$ and a single schedule $\mathcal{S}_i$ associated with a user U_i, the algorithm must receive a sequence $(T_{i,1}, \ldots, T_{i,ki})$ and a schedule $\mathcal{S}_i$ for each user U_i in a family $\mathcal{U} = \{U_1, \ldots, U_q\}$. Moreover, instead of an STN $\mathcal{N}_i$ encoding the basic constraints for the items of an individual user, it must receive an STN $\mathcal{N}_\mathcal{U}$ encoding the basic constraints for the items of all of the users in family $\mathcal{U}$.

Procedure *Intervals* returns a set of 4^q sequences, each one associating a class to a user in the family. For example, if $\mathcal{U} = \{U_1, U_2\}$, *Intervals* must return a sequence $\mathcal{I}^{\perp,\perp}_{U1,U2}$, a sequence $\mathcal{I}^{\perp,\mathtt{L}}_{U1,U2}$, and so forth. Moreover, the number of positions to be considered by *Intervals* for placing the new meeting M (line 3) is now:

$$(k_1 + 1) \cdot \ldots \cdot (k_q + 1)$$

For example, if the existing tasks of U_1 are $(T_{1,1}, T_{1,2})$ $(k_1 = 2)$ and the existing tasks of U_2 are $(T_{2,1}, T_{2,2}, T_{2,3})$ $(k_2 = 3)$, we must consider scheduling M at position “0 for U_1 and 0 for U_2” (i.e., before both $T_{1,1}$ and $T_{2,1}$), at position “0 for U_1 and 1 for U_2” (i.e., before $T_{1,1}$ and between $T_{2,1}$ and $T_{2,2}$), and so on.

Therefore, *Intervals* computes 4^q sequences of size $(k_1 + 1) \cdot \ldots \cdot (k_q + 1)$, and such sequences are passed as inputs to *DComp*. *DComp* is almost unchanged but there is an important point to make about the labels. While for the intervals of a single user labels are totally ordered (as $\perp < \mathtt{L} < \mathtt{M} < \mathtt{H}$), this is no longer true for the intervals of a family. For example, labels $U_1U_2^{\perp,\mathtt{L}}$ (i.e., some low-importance tasks of U_2 have to be shifted) and $U_1U_2^{\mathtt{L},\perp}$ (i.e., some low-importance tasks of U_1 have to be shifted) are not ordered. As a consequence, the **if** starting at line 7 of *DComp* should have one additional branch, covering the case when the class C of the starting input interval is not comparable with the label l of the current output interval. In such a case, a *new* output interval should be started with label $(C; l)$.

Related Work

The importance of a mixed-initiative approach to scheduling calendars was recognized in previous works, such as (Cesta, D’aloisi, and Brancaleoni 1996) and (Berry et al. 2011). However, relatively little work has been done on this topic so far.

Some recent calendar managers (e.g., Google Calendar Smart Rescheduler (Marmaros 2010)) analyze the estimated duration and temporal constraints of the items to be scheduled in order to identify the available time slots where they could be allocated. However, they cannot suggest any schedule revisions for addressing temporal conflicts.

Many task managers such as Things (Cultured Code 2011) and Standss Smart Schedules for Outlook (Standss 2012) manage tasks and deadlines but they have no scheduling capabilities. Other ones are very powerful but they require too much information from the user for everyday activity management, and/or they only handle single-user tasks; e.g., see (Refanidis and Yorke-Smith 2010).

Opportunistic schedulers synchronously guide the user in the execution of activities; e.g., see (Horvitz and Subramani 2007). However, they cannot present an overview of long-term schedules.

PTIME (Berry et al. 2011) adopts a mixed-initiative approach and generates personalized scheduling options by learning the user’s preferences. A major difference with respect to MARA is the fact that it proposes complete solutions to choose from, instead of interacting with the user during the exploration of the solution space, represented as a set of feasibile intervals for adding/moving a task.

The TCSP-based classic techniques used in the paper (Dechter, Meiri, and Pearl 1991) have been extensively studied and extended in the temporal reasoning and planning/scheduling communities. Of particular interest for extensions of this paper are, among others, improved efficiency of STN consistency check (Planken, de Weerdt, and van der Krogt 2011), consistency check of distributed STNs (Boerkoel and Durfee 2010), incremental consistency checks (Planken, de Weerdt, and Yorke-Smith 2011) and temporal preferences (Peintner and Pollack 2004). While these techniques could definitely improve the efficiency and scalability of our approach, we are not aware of any existing work which exploits them in the way proposed in this paper for improving the mixed-initiative user experience in calendar management.

Finally, it is worth mentioning the works by Bresina et al. on mixed-initiative planning of Mars rover missions (Bresina and Morris 2006; 2007). In such works, however, the role of the automated reasoner is that of helping the human to plan the daily activity for the rover by detecting (temporal) inconsistencies and trying to explain them in terms of previous commitments made by the user.

Conclusions

We presented the temporal reasoning support underlying the Mixed-initiative cAlendaR mAnager. MARA exploits Temporal Constraint Satisfaction techniques to generate safe schedules across multiple calendars; it adopts a mixed-initiative interaction model to guide the user in the exploration of the solution space, providing her/him with information about the available options and their impact on the existing commitments of the involved actors. In this way, it helps the user to quickly solve calendar management problems, leaving her/him in control of the scheduling activity.

A preliminary test with users provided encouraging results on the efficacy and usefulness of MARA's calendar management features: users particularly appreciated its awareness support because it enabled them to easily find the allocation options for events and to select the most convenient ones by previewing their impact on people's commitments, without analyzing all the possible solutions in detail.

References

Ardissono, L.; Segnan, G. P. M.; and Torta, G. 2013. Mixed-initiative management of online calendars. In *Lecture Notes in Business Information Processing, Web Information Systems and Technologies*, 167–182. Springer.

Berry, P.; Gervasio, M.; Peintner, B.; and Yorke-Smith, N. 2011. PTIME: Personalized assistance for calendaring. *ACM Transactions on Intelligent Systems* 2(4):40:1–22.

Boerkoel, J., and Durfee, E. 2010. A comparison of algorithms for solving the multiagent simple temporal problem. In *In Proceedings of the 20th Int. Conf. on Automated Planning and Scheduling (ICAPS-10)*.

Bresina, J. L., and Morris, P. H. 2006. Explanations and recommendations for temporal inconsistencies. In *Proc. Int. Work. on Planning and Scheduling for Space*.

Bresina, J. L., and Morris, P. H. 2007. Mixed-initiative planning in space mission operations. *AI Magazine* 28(2).

Cesta, A.; D'aloisi, D.; and Brancaleoni, R. 1996. Considering the user in mixed-initiative meeting management. In *Proc. 2nd ERCIM Workshop on User Interfaces for All*.

Cultured Code. 2011. Things Mac.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Hietaniemi, J. 2010. Graph-0.94.

Horvitz, E., and Subramani, M. 2007. Mobile opportunistic planning: methods and models. In *Lecture Notes in Artificial Intelligence n. 4511: Proc. 11th Int. Conf. on User Modeling*, 228–237.

Marmaros, D. 2010. Smart rescheduler in google calendar labs.

Peintner, B., and Pollack, M. E. 2004. Low-cost addition of preferences to dtps and tcsps. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI-04)*, 723–728.

Planken, L.; de Weerdt, M.; and van der Krogt, R. 2011. Computing all-pairs shortest paths by leveraging low treewidth. In *In Proceedings of the 21st Int. Conf. on Automated Planning and Scheduling (ICAPS-11)*.

Planken, L.; de Weerdt, M.; and Yorke-Smith, N. 2011. Incrementally solving stns by enforcing partial path consistency. In *In Proceedings of the 20th Int. Conf. on Automated Planning and Scheduling (ICAPS-10)*.

Refanidis, I., and Yorke-Smith, N. 2010. A constraint-based approach to scheduling an individual's activities. *ACM Transactions on Intelligent Systems* 1(2):12:1–32.

Standss. 2012. Standss smart schedules for outlook.