Combining Temporal Information by Collaborative Planning Agents*

Meirav Hadad1,2  Sarit Kraus1,2
1Department of Mathematics and Computer Science
Bar-Ilan University, Ramat-Gan, 52900 Israel
{hadad, sarit}@macs.biu.ac.il
2Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract
We present strategies for reasoning and combining temporal information of intelligent agents. The agents situated in a dynamic environment collaborate with other agents and interleave planning and acting. Our work, which is based on a formal model of collaborative planning, allows each agent to develop its timetable individually and to take into consideration various types of time constraints of its activities and of its collaborators. However, its individual timetable can be easily integrated with the activities of other agents. Thus, in contrast to other works on multi-agent systems, which suggest that either the team members maintain full synchronization by broadcasting messages or that a team leader determines the times of actions, our mechanism enables the individual agent to determine its time frame individually and to be integrated easily in the activities of other agents. We have developed a simulation of such environments and present simulation results comparing different methods of exchanging temporal information among team members.

Introduction
This paper considers the problem of temporal scheduling of actions in a cooperative activity under time constraints. Our work is based on the SharedPlan model of collaboration (Grosz & Kraus 1996) that supports the design and construction of collaborative systems. In order to carry out their cooperative activity, the agents perform collaborative planning that includes processes that are responsible for identifying recipes, assigning actions to agents, and determining the time of the actions. A recipe for an action consists of subactions which may be either basic actions or complex actions.

Basic actions are executable at will if appropriate situational and temporal conditions hold. A complex action can be either a single-agent action or a multi-agent action. In order to perform a complex action, the agents have to identify a recipe for it and there may be several known recipes for such an action. A recipe may include temporal constraints and precedence relations between its subactions. The general situation considering the actions without the associated constraints is illustrated in Figure 1, in which the leaves of the tree represent basic actions. We refer to this tree as "a complete recipe tree for α." For the agents to have achieved a complete plan, the values of the time parameters of the actions that constitute their joint activity must be identified in such a way that all the appropriate constraints are satisfied. Determining the time of the actions in a collaborative environment that we consider is complex because of the need to coordinate actions of different agents, the partiality of the plans, the partial knowledge of other agents' activities and the environment and temporal constraints. Furthermore, the temporal constraints require interleaving of planning and execution. In a former paper (Hadad & Kraus 2001) we briefly presented an algorithm for cooperative agents to determine the time of the actions that are required to perform their joint activity. In this paper we expand our former work and compare different methods for reasoning and combining temporal information in a team. The mechanism presented in this paper focuses on temporal scheduling. Thus, for simplification purposes, the planners do not take into consideration preconditions and effects.

One of the main questions in multi-agent environment is at what stage an agent should commit itself to

---

*Preliminary version appeared in CIA-01. This work is supported in part by NSF under Grant No. IIS-9907482. Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.
a timetable for performing a joint action, and inform
the rest of the team of the time frame. If the individual
agent commits itself to a specific timetable early and
announces this commitment to the other agents, it may
need to negotiate with the others if it needs to change
its timetable later. Alternatively, if the commitment
is made and announced as late as possible, i.e., only
when requested by other team members, it may delay
the planning and activity of these other team members.
Another question arises regarding the order in which
the team members should plan the subactions of their
joint activity. If we force all of the team members to
plan their activity together in the same order, the flexi-
bility of the individuals is decreased. However, planning
in different orders may delay the plan of some members
of the group. This delay will occur when some mem-
ers need certain information from other team members
about a specific action that they want to plan, but they
have to wait until the other members decide to plan this
action too. In our work we have analyzed these ques-
tions by implementing different methods in a simulation
environment.

An Algorithm for Temporal Reasoning
Building a collaborative agent that can flexibly achieve
its goals in changing environments requires a blending
of Real-Time Scheduling (RTS) computing and Artifi-
cial Intelligence Planning (AIP) technologies. Accord-
ingly, our single-agent system consists of an AIP sub-
system and an RTS subsystem. The goal of the AIP
subsystems when the agents attempt to perform $\alpha$ is to
identify a set of basic actions and a set of the temporal
constraints associated with the basic actions, such that
performing the basic actions under these temporal con-
straints constitutes performing $\alpha$ without conflict. As
soon as it is identified, each basic action $\beta$ is sent to the
RTS subsystem, along with its temporal requirements
($D_\beta, d_\beta, r_\beta, p_\beta$), where $D_\beta$ is the Duration time, i.e.,
the time necessary for the agent to execute $\beta$ without
interruption; $d_\beta$ denotes the deadline, i.e., the time be-
fore the $\beta$ should be completed; $r_\beta$ refers to the release
time, i.e., the time at which $\beta$ is ready for execution;
and $p_\beta$ denotes the predecessor actions, i.e., the set
{$\beta_j | 1 \leq j \leq n$} of basic actions whose execution must
end before the beginning of $\beta$.

In our system, the agents develop the plan of their
joint activity collaboratively. This section presents the
major constituents of the time reasoning mechanism,
which is based on the mechanism for an individual agent
presented in (Hadad et al. 2002). However, expansion
of collaborative plans differs from the individual
case (Gross & Kraus 1996). When an agent acts alone,
it determines how to perform an action (i.e., find a
recipe) and then carries out the necessary subactions.
When agents work together, both the formation or se-
lection of the recipe and the carrying out of the subac-
tions is distributed. The group must reach a consensus
on how they are going to perform the action (i.e., on
the recipe they will use) and on who is going to do the
subactions. When each agent (or subgroup) decides on
when to perform a subaction, the temporal constraints
of the group members must be taken into considera-
tion. Thus, recipe and agent selection in collaborative
planning requires inter-agent negotiation and communi-
cation as well as individual agent reconciliation. In the
mechanism presented, for simplicity, we assume that
each action is associated with the agent that performed
it. Thus, there is no need for decision-making concern-
ing the performer of the action. Also, we do not discuss
the recipe selection. We focus on the time scheduling
problem and briefly describe some of its components.
Supporting Definitions and Notation

In (Hadad et al. 2002) we defined some basic concepts that were are used in the reasoning mechanism of the individual agent case. The main structure we defined was the temporal constraints graph. The agent uses this graph when it identifies the time parameters of its activity. As discussed in (Hadad et al. 2002), the form of this graph is based on temporal constraint satisfaction networks (Dechter, Meiri, & Pearl 1991, inter alia).

However, because of the uncertainty and dynamic environment of the agents, in contrast to previous works, the constraints graph which is built by our agents may include only partial knowledge; i.e., our algorithm enables the agents to build this graph incrementally and to backtrack when necessary. In addition, the agents are able to determine the actions for which the temporal parameters have been identified and to execute them, thus interleaving planning and execution.

In this section we define some new basic concepts and extend the former definition of the temporal constraints graph in such a way that it will also be appropriate for a cooperative multi-agent environment. The mechanism of the temporal reasoning in the multi-agent environment is very similar to that of the individual case. However, in contrast to a single-agent environment, in which all the vertices in the temporal constraints graph are associated with an action that has to be performed by the agent who develops the graph, in multi-agent activity, some vertices may be associated with an action which has to be performed jointly by a group of agents. Furthermore, some vertices may be associated with actions that have to be performed without the agent who develops the graph. We denote the performers of an action in the following definition:

Definition 1 Let \( \beta \) be an action. The set \( Agg = \{A_1 \ldots A_n\} \ (n \geq 1) \), represents a group of agents who participate in the performance of \( \beta \), where \( A_k \), \((1 \leq k \leq n)\), is a single agent in the group.

One of the main structures which is used in building the temporal constraints graph is the precedence graph. The precedence graph represents the precedence relationship between subactions in a recipe. The relationship may be between subactions that have to be performed by different agents. The precedence graph of action \( \alpha \) in the multi-agent case is similar to that of the individual case and is given in the following:

Definition 2 Let \( \alpha \) be a complex action, and let \( R_\alpha \) be a recipe which is selected for executing \( \alpha \). Let \( \beta_1, \ldots, \beta_n \) be the subactions of the recipe \( R_\alpha \); \( \theta_{R_\alpha}^R = \{(i,j)|\beta_i < \beta_j; i \neq j\} \) are the precedence constraints associated with \( R_\alpha \). The precedence graph of \( \alpha \), \( Gr_\alpha = (V_\alpha,E_\alpha) \) with reference to \( R_\alpha \) and its precedence constraints \( \theta_{R_\alpha}^R \) satisfies the following:

1. There is a set of vertices \( V_{R_\alpha}^R = \{s_{\beta_1}, \ldots, s_{\beta_n}, f_{\beta_1}, \ldots, f_{\beta_n}\} \) where the vertices \( s_{\beta_k} \) and \( f_{\beta_i} \) represent the start time and the finish time points of \( \beta_k \) for \( 1 \leq k \leq n \), respectively.
2. There is a set of edges \( E_{R_\alpha}^R = \{(u_1,v_1), \ldots, (u_m,v_m)\} \) where each edge \( (u_k,v_k) \in E_{R_\alpha}^R 1 \leq k \leq m \) is either:
   (a) the edge \( (s_{\beta_i}, f_{\beta_i}) \) that represents the precedence relations between the start time point \( s_{\beta_i} \) and the finish time point \( f_{\beta_i} \) of each subaction \( \beta_i \). The edge is labeled by the time period for the execution of the subaction \( \beta_i \); or
   (b) the edge \( (f_{\beta_i}, s_{\beta_j}) \), \( i \neq j \), which represents the precedence relation \( \beta_i < \beta_j \) if \( \theta_{R_\alpha}^R \), that specifies that the execution of the subaction \( \beta_i \) can start only after the execution of the subaction \( \beta_j \) ended. This edge is labeled by the delay period between \( \beta_i \) and \( \beta_j \). If \( Agg_{\beta_i} \neq Agg_{\beta_j} \), the precedence relation \( \beta_i < \beta_j \) is called multi-precedence constraint.
3. Initially, all the edges of the precedence graph are labeled by \([0, \infty]\).
4. There is a set of vertices \( \{s_{\beta_1}, \ldots, s_{\beta_m}\} \subseteq V_{R_\alpha}^R \) with in-degree \( 0 \). These vertices are called beginning points.
5. There is a set of vertices \( f_{\beta_1}, \ldots, f_{\beta_m} \subseteq V_{R_\alpha}^R \) with out-degree \( 0 \). These vertices are called ending points.

Example 1 Figure 2(A) illustrates the precedence graph \( Gr_{R_\alpha} \) of a possible recipe \( R_\alpha \). As shown in this figure the actions \( \beta_1 \) and \( \beta_2 \) are single-agent action and have to be planned by \( A_1 \). Thus, \( A_2 \), for example, cannot begin the execution of \( \beta_2 \) without knowing the temporal requirements of its precedent action \( \beta_1 \), which is planned by \( A_1 \). The vertex \( s_{\beta_2} \) is a beginning point and the vertices \( f_{\beta_1}, f_{\beta_2} \) are ending points.

In the individual reasoning mechanism, we distinguish between complex and basic actions in the graph. A vertex which is associated with a basic-level action is called a basic vertex and a vertex which is associated with a complex level action is called complex vertex. Similarly, an edge which is associated with a basic-level action is called a basic edge and an edge which is associated with a complex level action is called complex edge. In a cooperative environment, a complex action may be either a multi-agent action or a single-agent action as we define in the following definition:

Definition 3 Let \( Gr_{R_\alpha} = (V_{R_\alpha}^R,E_{R_\alpha}^R) \) be a precedence graph of an action \( \alpha \). Let \( (s_{\beta_i}, f_{\beta_i}) \) be an some edge in \( E_{R_\alpha}^R \) which denotes the relation between the start time point, \( s_{\beta_i} \), and the finish time point, \( f_{\beta_i} \), of the subaction \( \beta_i \).

1. If the subaction \( \beta_1 \) is a single-agent action (i.e., \( |A_{\beta_i}| = 1 \)), the edge \( (s_{\beta_i}, f_{\beta_i}) \) is called a single-edge and the vertices \( s_{\beta_i} \) and \( f_{\beta_i} \) are called single-vertices.
2. If the subaction \( \beta_1 \) is a multi-agent action (i.e., \( |A_{\beta_i}| \neq 1 \)), the edge \( (s_{\beta_i}, f_{\beta_i}) \) is called a multi-edge and the vertices \( s_{\beta_i} \) and \( f_{\beta_i} \) are called multi-vertices.

\( ^1 \)We will refer to a node which is labeled by \( s \) as node \( s \).
3. We say that the actions \{\beta_1, \ldots, \beta_m\} hinder \beta_i if there is a set of edges \(E' \subseteq E^\alpha_k\) such that 
\[ E' = \{(s_{\beta_i}, s_{\beta_j}), \ldots, (s_{\beta_i}, s_{\beta_j})\}, \] 
where \(Ag_{\beta_i} \neq Ag_{\beta_j}(1 \leq k \leq n)\).

In following we expand the definition of the temporal constraints graph for a multi-agent case. When the group \(Ag_a\) works on the action \(\alpha\), each agent \(A_k \in Ag_a\) maintains a temporal constraints graph. This graph is changed during the agents’ planning. The graph of each individual agent in \(Ag_a\) may be different with respect to its individual actions, but similar with respect to the first level of multi-agent actions.

Definition 4 Let \(A_k\) be an agent. Let \(\{\alpha, \beta_1, \ldots, \beta_n\}\) be a set of basic and complex-level actions that the group \(Ag_a\) intends to perform, where \(\alpha\) is the highest level action in the recipe tree which consists of these actions and \(A_k \in Ag_a\). Let \(V^\alpha = \{s^\alpha, s^\alpha_1, \ldots, s^\alpha_f, f^\alpha, \ldots, f^\alpha_n\} \cup \{s_{plan}\}\) be a set of variables where the variable \(s_{plan}\) represents the time point that \(A_k\) starts to plan action \(\alpha\). Let \(\theta^\alpha_{\beta_k} = \{(\beta_{i_1}, \beta_{i_2})|\beta_{i_1} < \beta_{i_2}; i \neq j\}\) be the set of all the precedence constraints which are associated with all the recipes of the actions \(\{\alpha, \beta_1, \ldots, \beta_n\}\). Let \(\theta^\alpha_{\beta_k} = \{ui, vj|ui < vj \leq h_{ij}\}\) be all the temporal metric constraints that are associated with \(\{\alpha, \beta_1, \ldots, \beta_n\}\) and their recipes. The temporal constraints graph of \(\alpha\), \(Gr^\alpha = (V^\alpha, E^\alpha_{c}, E^\alpha_{e}, E^\alpha_{d})\), satisfies the following:

1. The vertex \(s_{plan} \in V^\alpha\) denotes the time point at which \(A_k\) starts to plan action \(\alpha\). This point is a fixed point in time which is called the origin. The vertex \(s_{plan}\) is the time point at which \(A_k\) starts \(\alpha\)’s performance, and \(f^\alpha\) is the time point at which \(A_k\) finishes \(\alpha\)’s performance.

2. If \(\alpha\) is a basic-level action, then \(Ag_a = \{A_k\}\) and \(V^\alpha = \{s_{plan}, s^\alpha, f^\alpha\}\). The set of edges \(E^\alpha_{c} = \{(s_{plan}, s^\alpha), (s^\alpha, f^\alpha)\}\), where each edge \((ui, vj) \in E^\alpha_{c}\) is labeled by the following weight:

\[
\text{weight}(ui, vj) = \begin{cases} 
[a_{ij}, b_{ij}] & \text{if } ui, vj \in \theta^m_{\alpha} \\
[0, \infty) & \text{if } ui, vj \notin \theta^m_{\alpha}
\end{cases}
\]

3. If \(\alpha\) is a complex-level action then:

(a) There is a set of basic vertices \(V_{basic} = \{s_{\beta_1}, \ldots, s_{\beta_k}, f_{\beta_1}, \ldots, f_{\beta_k}\} \subseteq V^\alpha\) and a set of basic edges \(E_{basic} = \{(s_{\beta_1}, f_{\beta_1}), \ldots, (s_{\beta_k}, f_{\beta_k})\}\) \(\subseteq E^\alpha_{c}\) where \(\beta_i (1 \leq i \leq k)\) is some basic-level action in the recipe tree of \(\alpha\), and each edge \((s_{\beta_i}, f_{\beta_i}) \in E^\alpha_{c}\) is labeled by the time period of \(\beta_i\). Each such basic edge is a single-edge and each basic vertex is a single-vertex. If \(Ag_{\beta_i} \neq \{A_k\}\) then \(\beta_i\) is a subaction in a recipe of a multi-agent action \(\beta'\) such as \(A_k \in Ag_{\beta_i}\).

(b) There is a set of complex vertices \(V_{complex} = \{s_{\beta_1}, \ldots, s_{\beta_k}, f_{\beta_1}, \ldots, f_{\beta_k}\} \subseteq V^\alpha\) and a set of complex edges \(E_{complex} = \{(s_{\beta_i}, f_{\beta_i}), \ldots, (s_{\beta_i}, f_{\beta_i})\}\) \(\subseteq E^\alpha_{c}\) where each edge \((s_{\beta_i}, f_{\beta_i}) \in E^\alpha_{c}\), \(1 \leq j \leq l\) is labeled by the following weight:

\[
\text{weight}(ui, vj) = \begin{cases} 
[a_{ij}, b_{ij}] & \text{if } (s_{\beta_i}, f_{\beta_i}) \in \theta^m_{\alpha} \text{ and } \beta_i\text{'s period is known} \\
[0, \infty) & \text{if } (s_{\beta_i}, f_{\beta_i}) \notin \theta^m_{\alpha} \text{ and } \beta_i\text{'s period is unknown}
\end{cases}
\]

Each complex edge is either a single-edge or a multi-edge. Similarly, each complex vertex is either a single-vertex or a multi-vertex. If \(A_k \notin Ag_{\beta_i}\) then \(\beta_i\) is a subaction in a recipe of multi-agent action \(\beta'\) such that \(A_k \in Ag_{\beta'}\).

(c) There is a set of delay edges \(E_{delay} = \{(ui, vj)\} \subseteq E^\alpha_{c}\) where \(ui, vj \in V^\alpha\), \(1 \leq i \leq n\). The vertices \(ui, vj\) may be of the following forms:

i. \(ui\) is the start time point of some action \(\beta_u\) and \(v_j\) is some beginning point in the precedence graph \(Gr^\alpha_{\beta_u}\) of action \(\beta_u\).

ii. \(ui\) is some ending point in the precedence graph \(Gr^\alpha_{\beta_u}\) of some action \(\beta_u\) and \(v_i\) is the finish time point of action \(\beta_u\).

iii. \(ui\) is the finish time point of some action \(\beta_u\) and \(v_j\) is a start time point of another action \(\beta_u\), where \(\beta_u, \beta_v \in \theta^m_{\alpha}\).

Each edge \((ui, vj) \in E_{delay}\) is labeled by the following weight:

\[
\text{weight}(ui, vj) = \begin{cases} 
[a_{ij}, b_{ij}] & \text{if } ui, vj \in \theta^m_{\alpha} \\
[0, \infty) & \text{if } ui, vj \notin \theta^m_{\alpha}
\end{cases}
\]

(d) There is a set of directed edges \(E_{metric} = \{(ui, vj), \ldots, (ui, vj)\} \subseteq E^\alpha_{c}\) labeled by its metric constraints, where \(ui, vj \in V^\alpha_{c}\), \(1 \leq i \leq n\) but \((ui, vj) \notin E_{basic} \cup E_{complex} \cup E_{delay}\).

The algorithm in general

During the agents’ planning process, each agent \(A_k\) selects a vertex \(s_0\) from its \(Gr^\alpha_{\beta}\). For this vertex, which is associated with the action \(\beta\), the planning of all the actions which precede \(\beta\) have been completed. Then, \(A_k\) attempts to complete \(\beta\)’s plan. Action \(\beta\) is planned by \(A_k\) only if \(A_k\) participates in \(\beta\)’s performance (i.e., \(A_k \in Ag_{\beta}\)). If \(s_0\) is a basic vertex, then the action \(\beta\) along with its temporal requirements \((D_\beta, d_\beta, r_\beta, p_\beta)\) is sent to \(A_k\)’s RTS subsystem for execution. Otherwise, \(A_k\) performs the following major activities:

1. If \(s_0\) is a multi-vertex, then \(A_k\) has to obtain an agreement form the other members of \(Ag_{\beta}\) that \(A_k\) will develop \(\beta\)’s plan.
robots, whom we call At and A2, have to consider the collaborative planning that arises when two survivors in areas A and B. We assume that the batteries of A2 are restricted for 150 minutes; thus this action must be finished within 150 minutes. We also assume that this action must be finished before the evening (at 7:30P.M.). If we denote the action “rescue disaster survivors in areas A and B” by α, and its time interval by [s_α, t_α], the temporal constraints as extracted from the example are the following: \{(f_α - s_α \le 150), (s_α after 4:00), (f_α before 7:30)\}. Note that the length of the time interval \([s_α, t_α]\) which denotes the time of α execution is unknown initially. Both agents start the collaborative plan for α together at 4P.M., thus the fixed time point \([s_α, t_α]\) is unknown.

Next, they need to agree on how they are going to perform the action α (i.e., on a recipe for α). Suppose that At and A2 have agreed upon the following plan: At will “find and identify the victims in area A” - we denote this subaction as β_1. A1 will “find and identify the victims in area B” - we denote this subaction as β_2. A2 will “carefully clear the piles of rubble block doorways in area A” in order to rescue the victims outside the building - we denote this subaction as β_3. A1 and A2 together will “rescue the victims outside the building in area A” - we denote this subaction as β_4. A1 and A2 together will “rescue the victims outside the building in area B” - we denote this subaction as β_5. In addition, we assume that β_1 must be carried out before β_3, since A2 must know about the location of the “victims” before it begins to clear block doorways in the area. Also, since area B is less dangerous than area A, area B will be scanned by A1 only after area A (i.e., β_1 will be performed before β_3). In addition, they will “rescue the victims outside the building in area A” only after A2 clears the block doorways in area A (i.e., will perform β_3 before β_4), and after A1 “finds and identifies the victims in area B” (i.e., will perform β_3 before β_4). They will “rescue the victims outside the building in area B” (β_3 before β_5). Thus, the identified recipe R_α consists of the subactions \{β_1, β_2\} where β_1 and β_2 will be performed only by A1, β_3 will be performed only by A2 and β_4, β_5.
and $\beta_3$ are multi-agent actions. In addition, this identified recipe is associated with the precedence relations $\{\beta_1 < \beta_2; \beta_1 < \beta_3; \beta_2 < \beta_4; \beta_2 < \beta_3; \beta_3 < \beta_5\}$. Suppose that the temporal constraints for performing these actions are: $\{(f_{\beta_3} - s_{\beta_3} \leq 20), (s_{\beta_3} - f_{\beta_3} \leq 40), (f_{\beta_5} - s_{\beta_5} \leq 60), (s_{\beta_5} \text{ after } 5:00)\}$. Each agent builds the $Gr^k_{R_a} = (V^k_{R_a}, E^k_{R_a})$ and incorporates it into its $Gr^k_a$ by adding directed edges from $s_a$ to each vertex $v_{n_1}, \ldots v_{n_m} \subseteq V^k_{R_a}$, with an indegree of 0, and by adding edges from each vertex $v_{e_1}, \ldots v_{e_n} \subseteq V^k_{R_a}$, with an outdegree of 0 to $f_a$. A pictorial illustration of adding this $Gr^k_{R_a}$ to $Gr^k_a$ is given in Figure 2. $Gr^k_{R_a}$ is incorporated into $Gr^k_a$ by adding the edges $(s_{\beta_1}, f_{\beta_1}), (s_{\beta_2}, f_{\beta_2}), (f_{\beta_3}, f_{\beta_4})$. After this, the agents also update the temporal constraints that are associated with $R_a$.

Then, each agent $A_k$ tries to continue the development of its $Gr^k_a$ by selecting an action to be developed. At this stage, $Gr^k_a$ is identical to $Gr^k$ but only the action $\beta_1$ may be selected since it does not depend on any other action. Since $A_1$ is the single-agent involved with $\beta_1$, $A_2$ has to wait until it receives the values of the temporal parameters of $\beta_1$ from $A_1$. Suppose that the recipe that $A_1$ selected for $\beta_1$ consists of the basic actions $\gamma_{11}$ and $\gamma_{12}$, and that the execution time for each of them is exactly 5 minutes. Thus, $A_1$ should identify $(D_{\gamma_{11}}, r_{\gamma_{11}}, d_{\gamma_{11}}, p_{\gamma_{11}})$ and $(D_{\gamma_{12}}, r_{\gamma_{12}}, d_{\gamma_{12}}, p_{\gamma_{12}})$ and send it to its RTS subsystem. The decision regarding the exact time in which $\beta_1$ will be executed is determined by the RTS subsystem. In this example, we assume that the RTS subsystem of $A_1$ decides to execute $\gamma_{11}$ at 4:02 and $\gamma_{12}$ at 4:09. Thus $A_1$ will inform $A_2$ that it intends to terminate the execution of $\beta_1$ at 4:14. Following this announcement, $A_2$ can start developing $\beta_2$. Similarly, $A_1$ can develop $\beta_2$. $A_2$ will use this information to develop the plan for $\beta_3$. In addition, both of them will add the edge $(s_{\beta_1}, f_{\beta_1})$ and will label this constraint on it (i.e., the weight of this edge will be [14, 14]). The temporal graph $Gr^k_{R_a}$ which is maintained by each agent in this stage of their collaborative plan is given in Figure 9. As shown in this figure, each agent maintains a different graph according to its plan. Graph (A) is the graph which is built by $A_1$ and graph (B) is built by $A_2$. The identical vertices represent the subactions which appear in the recipe of their collaborative action.

**Possibilities for Team Scheduling and Planning**

As described above, agent $A_i$ informs agent $A_j$ about the time values that it identified for its individual actions $\beta_1 \ldots \beta_m$ when $\beta_1 \ldots \beta_m$ hinder the planning of action $\gamma$ that has to be performed by $A_j$. The problem is at what stage $A_i$ should commit itself to a timetable for performing $\beta_1 \ldots \beta_m$, and inform $A_j$ about this timetable.

One possibility is that when $A_i$ finishes the planning of all its actions that directly precede $\gamma$, it commits itself to their performance times and sends the appropriate values to $A_j$. This method enables $A_j$ to begin the plan of $\gamma$ immediately when the planning of all the actions which precede $\gamma$ have been completed. Furthermore, $A_j$ does not need to ask $A_i$ for the relevant times, since $A_i$ informs $A_j$ about them as soon as possible. However, since $A_i$ has to commit itself to these times, $A_i$ has less flexibility in determining the time performance for its other actions. If it decides to change the announced timetable, it will need to negotiate with $A_j$.

Thus, we suggest an alternative mechanism. Following this approach, $A_j$ plans its individual actions until

**Experimental Results**

We have developed a simulation environment comprising 2 agents to evaluate the success rate of the system when the agents use the provide-time method and when they use the ask-time method. Each method was tested when the agents used the random-order method, the dfs-order method, and the bfs-order method. The combined methods are called random-provide, dfs-provide,
bfs-provide, random-ask, dfs-ask, and bfs-ask, respectively. We have also compared these methods with a "central-planner." The central planner in our system planned all the actions centrally. During the planning process it sent the basic actions with their associated temporal constraints to the appropriate agents. The RTS subsystems of the appropriate agents handled the scheduling and execution of these actions.

In our experiments, we made the simplifying assumption that all of the time constraints of the agent's action are associated either with the action-type or with the appropriate recipe.\(^2\) We ran the algorithm on several different recipe libraries which were created randomly. Each recipe library included at least one possible solution to the joint activity. For each case, we ran 120 experiments with randomly drawn parameters from ranges with high success rates of a single-agent. In each run, the following parameters were drawn from the following given range: (1) the average number of the precedence constraints between the actions that have to be performed by the same agent was between 0.20 and 0.55; (2) the average number of the metric constraints was between 0.9 and 1.1; (4) the number of the total basic actions in the recipe tree was between 100 and 120 (between 50 and 60 for each agent); the duration of each basic action could have been between 1 to 10 minutes; (5) the idle time in the schedule was between 120 and 170 seconds; (6) the number of the multi-agent actions in the recipe tree was 3. In addition, in all the runs we provided the schedule with a waiting time at the beginning of the schedule process (i.e., all the basic actions which arrive at the RTS subsystem have to begin 120 seconds (or more) after the schedule process has been started).

We tested the effects of the number of the multi-precedence constraints on complex subactions. We asserted that the group succeeded in performing the highest level action \(\alpha\) if: (1) the AIP subsystem of each agent finished the planning for \(\alpha\) and built the complete recipe tree for \(\alpha\) which satisfies the appropriate time constraints \(\delta_\alpha\); and if (2) the RTS subsystem of each agent found a feasible schedule, which consists of all the basic actions in the complete recipe tree of \(\alpha\). A failure is defined as either a failure in planning by the AIP subsystem of one member in the group, or as a failure in finding a feasible schedule by the RTS subsystem of one member.

Figure 4 shows the success rate of each method in a given range of multi-precedence constraints. As shown in the graph, the method with the best results is bfs-ask, with a success rate between 90% - 100%. The success rate of random-ask was between 86% - 96% and of the dfs-ask was between 75% - 95%. In general, the ask-time method is better than the provide-time method. We can see that bfs-provide (80% - 100% success rate) is better than random-provide (64% - 92% success rate) and dfs-provide (64% - 92%). Thus, we conclude that the bfs-order is the best planning method for our system. However, it is unclear if the random order is better than dfs-order. In addition, the dfs-provide is the sole method with a success rate that is a linear function of the number of the multi-precedence constraints. In the other methods, there are changes in the success rate of the system. We hypothesize that the reason for these changes results from the fact that a high number of multi-precedence constraints provides more knowledge about the subactions slots. As a result, the precedence constraints leads the scheduler and the other team members to the correct solution (which always exists in our examples). On the other hand, multi-precedence constraints decrease the flexibility of the individuals in the team since they cause them to make more commitments in their timetable. The low flexibility leads to a lower success rate in cases of 3-4 multi-precedence constraints. The central-planner attained good results in some cases (between 80% - 100%), but it was 5 times slower than all the other methods. Also, a high number of multi-precedence constraints reduces its success rate.

**Comparison with Alternative Works**

Our effort focuses on developing a system which solves problems that cannot be solved by a central controller, but which should be solved by partitioning the problem and distributing it among several agents according to their skills. Each agent reasons individually while interacting with the other team members. The agent dynamically determines both the duration and the time frame of the actions it has to perform in such a way that these times satisfy all the appropriate temporal constraints. The times and the periods of the events that occur during the agents' activities need not be known in advance. That is, an agent may change its timetable easily if it identifies new constraints due to changes in the environment or if it receives new time constraints.
from other team members. Thus, unlike other collaborative multi-agent systems, which either suggest broadcasting messages between the team members to maintain the full synchronization (Sonenberg et al. 1992; Tambe 1997) or suggest that the team leader be responsible for the timing of the individual actions (Jennings 1995), our mechanism does not restrict the activity of the individual agents.

The ability of the agents to schedule their activities distributively, when the duration and the time frame of the actions need not be known in advance, is in contrast to the existing work on distributed scheduling, which assumes that these temporal requirements are known in advance. In particular, most of the existing research (e.g., (Neuendorf & Hannebauer 2000; Wellner & Dilger 1999)) focuses on the development of heuristic algorithms for solving different types of distributed scheduling problems which are known as NP-complete. Other works (e.g., (Rabelo & Camarinha-Matos 1998; Hunsberger 2002)) focus on negotiation and auction methods which enable the agents to reach a consensus on how to allocate a portion of their activity. Thus, existing research does not suggest any method for representing and reasoning about incomplete and indefinite qualitative temporal information that can be used during the planning process of the agents, as we do.

Planning under uncertainty contrasts with the situation of other planners, who rely on perfect domain knowledge throughout plan development and execution (e.g., (El-Kholy & Richards 1996)). Others, who can plan under uncertainty, are complex and cannot be easily extended for use in cooperative multi-agent environments (e.g., (Wilkins et al. 1995)). Our mechanism is simple and appropriate for uncertain, dynamic, multi-agent environments, and enables agents to reason about their timetables during the planning process and thus to interleave planning and acting. In multi-agent systems which are intrinsically dynamic, it is impractical to maintain this classical distinction between planning and executing (Paolucci, Shehory, & Sycara 2000).

Conclusions

In this paper, we have presented a mechanism for time planning in uncertain and dynamic multi-agent systems. Our mechanism allows each agent to develop its timetable individually and to take into consideration all types of time constraints of its activities and of its collaborators.

Through simulations, we have compared different methods for information exchange and planning order for scheduling in a team. The simulation results show that it is better to commit to a timetable as late as possible. The results have also proved that if the team agents perform their plan in the same order and if they first develop the highest level actions, then they have a better chance of succeeding in their joint activity. In addition, it is more efficient to distribute the planning among the group members than to solve the problem by a central planner.

References


