Exchanging and Combining Temporal Information in a Cooperative Environment*

Meirav Hadad[1] and Sarit Kraus[1]

Department of Mathematics and Computer Science
Bar Ilan University Ramat Gan 52900, Israel
{hadad, sarit}@macs.biu.ac.il

1 Introduction

This paper considers the problem of exchanging and combining temporal information by collaborative agents who act in a dynamic environment. In order to carry out their cooperative activity the agents perform collaborative planning [2] while interleaving planning and execution. In a former paper [3] we presented a mechanism for cooperative planning agents to determine the timetable of the actions that are required to perform their joint activity. In this paper we expand our former work and compare different methods of reasoning and combining temporal information in a team. Determining the time of the actions in a collaborative environment is complex because of the need to coordinate actions of different agents, the partiality of the plans, the partial knowledge on other agents' activities and on the environment and temporal constraints. Our mechanism focuses on temporal scheduling. Thus, for simplification purposes, the agents do not take into consideration preconditions and effects during their planning process.

One of the main questions in a multi-agent environment is at what stage should an agent commit to a timetable for performing a joint action, and inform the rest of the team of his commitment. If the individual agent commits 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, a commitment made and announced as late as possible, e.g., only when requested by other team members, may delay the planning and action of other team members. Another question arises regarding the strategy the team members should use in planning the subactions of their joint activity. If we force all the team members to plan their activity using identical strategies, the flexibility of the individuals is decreased. However, using different strategies may delay the plan of some members of the group. This delay might occur when some member needs certain information about a specific action that it wants to plan, but finds that the other team members chose to delay the plan of this action to some later time. In our work we study these questions by implementing different methods in a simulation environment and by conducting experiments. We also compare the performance of our distributed method with a method in which a team leader is responsible for solving the problem centrally (e.g., [5]).

* This work is supported in part by NSF under Grant No. IIS-9907482. The second author is also affiliated with UMIACS.

2 Exchanging and Combining Temporal Information

This section briefly describes the major constituents of the exchanging and combining temporal information mechanism; it is based on former works [4, 3].

In order to carry out their cooperative activity the agents perform collaborative planning, which includes processes that are responsible for identifying recipes, assigning actions to agents and determining the time of the actions [2]. 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 and there may be several known recipes for an action. A recipe may include temporal constraints and precedence relations between its subactions. The general situation of team members A_1 and A_2 performing a joint action α, considering the actions without the associated constraints, is illustrated in figures 1(A) and 1(B), respectively. The leaves of the tree of agent A_k ($k = 1, 2$) represent either basic actions or actions in which A_k does not participate. We refer to this tree as agent A_k's "complete recipe tree for α". The trees of the team members differ with respect to individual actions but are identical with respect to the first level of the multi-agent actions.

The main structure that is used by each agent A_k in the team, when the team members identify the time parameters of their joint activity α, is the temporal constraints graph. The temporal constraints graph is associated with a multi-agent action α and maintains information about the temporal constraints of the actions that constitute the performance of α and the precedence relations among them. Formally, a temporal constraints graph, $Gr^k_\alpha = (V^k, E^k)$ of agent A_k, where $V^k = \{s_\alpha, s_{\beta_1}, \cdots, s_{\beta_n}, f_\alpha, f_{\beta_1}, \cdots, f_{\beta_n}\} \cup \{s_{\alpha_{plan}}\}$ and $\{\alpha, \beta_1, \cdots, \beta_n\}$, represents actions that the agents intend to perform in order to execute the highest level action α (see figures 1(C), 1(D)). The variables s_y and f_y represent the time points at which the execution of an action $y \in \{\alpha, \beta_1, \cdots, \beta_n\}$ can start and must finish, respectively. Some of the vertices may be fixed, i.e., these vertices denote a known time which cannot be modified. The vertex $s_{\alpha_{plan}}$ represents the time point at which agent A_k starts to plan the action α; it is a fixed vertex. Other fixed vertices may be initiated, for example, by a request from a collaborator. The activity of action y is represented by a directed edge between s_y and f_y, that is labeled by the time duration required for the execution of action y. A directed edge from the finish time point of action y to the start time point of another action $z \in \{\alpha, \beta_1, \cdots, \beta_n\}$ denotes that the execution of z cannot start until the execution of y is completed. The interval associated with this edge indicates the possible delay between these actions. A metric constraint $a_{i,j} \leq (v_i - v_j) \leq b_{i,j}$ between two different time points $v_i, v_j \in V$ is represented by the metric edge (v_i, v_j) that is labeled $[a_{i,j}, b_{i,j}]$. The form of this graph is an extension of Simple Temporal Networks [1].

In addition to the temporal information, a temporal constraints graph maintains additional information on each of the graph's actions: (a) whether the action is basic or complex; (b) whether the complex action is a multi-agent or a single agent action; (c) whether a plan for the action has been completed; (d) whether the action has already been executed by the agent(s), and (e) the agent(s) that is (are) assigned to perform the action. We denote the agent(s) that is (are) assigned to perform an action β by Ag_β. This information is determined incrementally by the algorithm which also expands the graph. Each single agent A_k in the system runs the algorithm independently in order to build its temporal constraints graph Gr_α^k. The graphs of individual agents may be different with respect to individual actions, but similar regarding the first level of multi-agent actions. To keep track of the progress of the expansion of the agents' graphs all the vertices in Gr_α^k, of each agent A_k, begin as *unexplored* (UE). The status of both vertices s_β and f_β is changed from unexplored to *explored* (EXP) during the performance of the algorithm by A_k. One of the main purposes of the following mechanism is to determine the values of s_β and f_β.

During the agents' planning process, each agent A_k selects a UE vertex s_β which is associated with action β, from its Gr_α^k. A UE vertex s_β is selected by A_k only if the vertices which precede s_β in the graph are explored. For this selected vertex, A_k checks which of the following conditions is satisfied and performs its

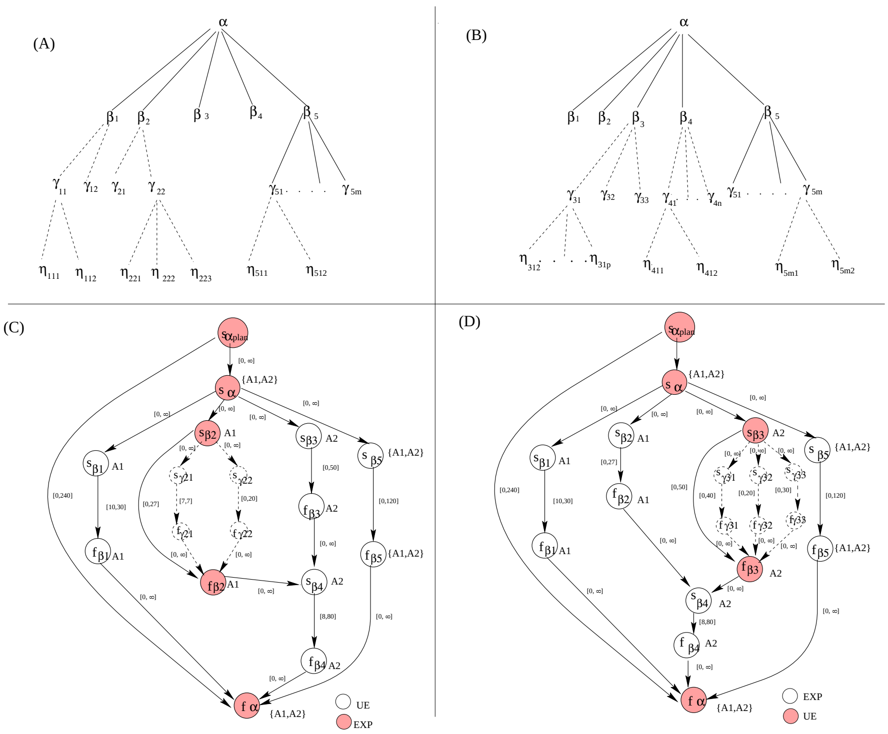

Fig. 1. (A-B) Examples of recipe trees of team members A_1 and A_2, respectively; (C-D) Examples of temporal constraints graphs, Gr_α^1 and Gr_α^2, respectively. The dashed edges represent individual plans.

activities accordingly:
(1) β **is associated with a multi (or single) action, where** A_k **does not belong to the group (or is not the agent) which has to perform the action** (i.e., $A_k \notin Ag_\beta$):

(1.1) **If the values of** β **are unknown**, A_k leaves this vertex until it receives β's values from the agent(s) who is (are) a member(s) in Ag_β.

(1.2) **If the values of** β **are known to** A_k, A_k changes the status of the vertices s_β and f_β from UE to EXP.

(2) β **is a multi-agents action and** A_k **participates in performing** β **(i.e.,** $A_k \in Ag_\beta$ **and** $|Ag_\beta| > 1$**):** Ag_β reaches a consensus with the other participants on how they are going to perform the action and who is going to plan this action; after reaching a consensus[1] each agent in Ag_β adds the new information to its temporal graph and determines the new values of its graph's vertices (see [3]). Also, each of them changes the status of s_β and f_β from UE to EXP.

(3) A_k **is the sole performer (i.e.,** $Ag_\beta = \{A_k\}$**):** A_k develops β and determines the new temporal values of the vertices in Gr_α^k as in the individual case (see [4]). Also, it changes the status of s_β and f_β from UE to EXP. After completing the development of β's plan, A_k checks whether β is a subaction in a recipe of a multi-agent action, or if as a result of completing the plan of β its individual plan for a higher level action β' is completed, where β' is a subaction in a recipe of a multi-agent action. If so, it saves the relevant information in order to inform this to the appropriate team members as discussed below.

2.1 Exchanging temporal information and planning in a team

The development of a shared plan [2] by the agents requires information exchange. Consider the case of two agents, A_i and A_j, that intend to perform a joint activity α. They will exchange information in the following cases: (1) When they identify a recipe for their joint action or for any joint subaction in their plan. (2) When agent A_i completes the plan of a subaction in a recipe of a joint action with A_j, it informs A_j that the plan for this action has been completed. (3) Agent A_i may inform agent A_j about the time values that it identified for its individual actions $\beta_1, \ldots, \beta_m$ when $\beta_1, \ldots, \beta_m$ *delay* the planning of action γ and γ has to be performed by A_j. We assert that $\beta_1 \ldots \beta_m$ *delay* the planning of γ if they directly precede γ. (4) If A_i already sent information to A_j about some action β, but A_i failed to perform β, then A_i backtracks and informs A_j about the failure of β or about new properties of their plan that were determined as a result of its backtracking.

A_i's message about completing its individual plans, in case (2) above, does not include temporal information and other details of A_i's individual plans. The goal of this message is to enable the agents to know the status of their joint plan. The sole case in which the agents send *temporal* information to each other is in case (3) above. Sending the time values of $\beta_1, \ldots, \beta_m$ by A_i to A_j (in case

[1] We focus on time scheduling and therefore do not discuss the planning processes for selection agents and recipes.

(3)) causes A_i to commit to these values. Thus, one of the main problems in a cooperative environment is at what stage should A_i commit to a timetable for performing $\beta_1, \ldots, \beta_m$ and inform A_j about this timetable.

One possibility is that when A_i completes the planning of all its actions that directly precede an action γ, which has to be performed by A_j, it commits to their performance times and sends the appropriate values to A_j. This method enables A_j to begin the plan of γ immediately when the planning of all the actions which precede action γ 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 it is possible. However, since A_i has to commit 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 also consider an alternative mechanism. Following this approach A_j plans its individual actions until it is not able to continue its plan since it depends on A_i's activity. In such a case A_j sends a message to A_i asking for the appropriate time values. When A_i completes the planning of the relevant actions, it sends the appropriate values to A_j. In this manner, the commitment is left to the latest possible time, but, it may delay the planning of the agent waiting for an answer and it requires additional message exchange. In our terminology the first method is called *provide-time* and the second is called *ask-time*.

An additional problem in a collaborative environment involves the order in which the members of the group should plan their activities. As described above, during the planning process, each agent A_k in the group selects a vertex s_β from its Gr_α^k to be expanded. Vertex s_β is selected by A_k only if it satisfies certain conditions. However, since in most cases there is more than one vertex that satisfies all the required conditions, the agent has to decide which of them to select in order to complete its plan. There are several possible selection methods. In the environment that we consider, the order of the vertices selection may affect A_k's decision regarding the time scheduling of its activities. Furthermore, in a joint activity a selection of a specific vertex by an individual agent may influence the activity of the entire team.

In this work we consider three methods for the planning order of the individual in the team. The first is called *random-order*, where A_k randomly selects one of the several actions that can be developed. In the second, called *dfs-order*, A_k selects the action according to the depth-first search order of its Gr_α^k. The third is called *bfs-order*. According to this method A_k selects one of the actions according to the breadth-first search order of its Gr_α^k. In the first method the planning order of the team members differ. In the two latter methods all the agents in the team plan their actions in the same order. In the dfs-order, the agent selects an action from the lowest levels of its recipe tree. Thus, it develops a subaction until it reaches the basic action level and then it continues to the next subaction. In the bfs-order, the agent tries to complete the plans of the highest level actions in each stage of the planning process of its recipe tree. Our simulation results, presented in the following section, demonstrate that the planning order influences the success rate of the agents.

3 Experimental Results

We developed a simulation environment comprising two agents to evaluate the success rate of the system. In our experiments, we made the simplifying assumption that all the time constraints on an agent's action are associated either with the action-type or with the appropriate recipe. 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 generated parameters from ranges with high success rates of one agent [4].

We tested the provide-time method and the ask-time method 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 tested the success rate of each method in a given range of *multi-precedence constraints*. Our goal was to answer the following questions: (1) Does the planning order affect the performance of the system? If so, what is the best method for the planning order in the team? Our hypothesis was that the order in which the agent chooses to plan the actions with the multi-precedence constraints affects the performance. If such actions are selected early, the commitment by the agents will be done at an early stage, and it will reduce their flexibility. However, because the examples

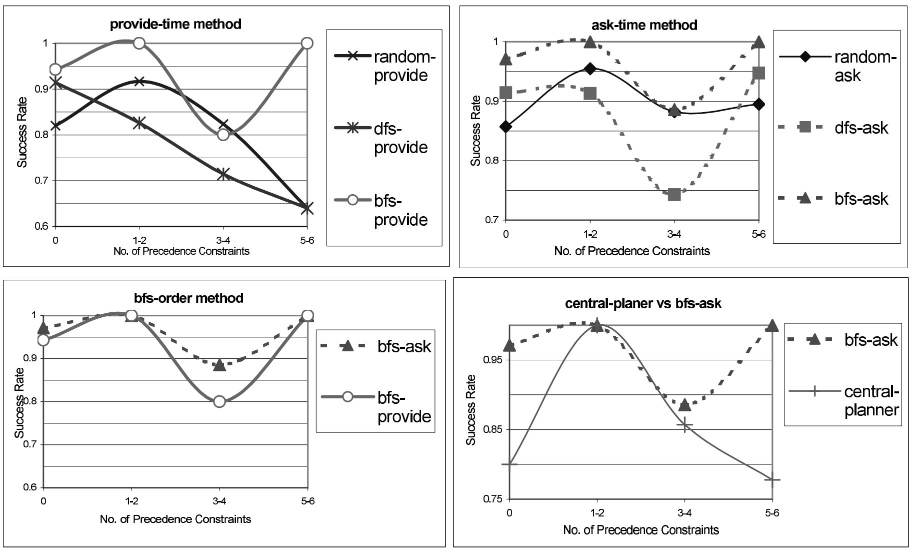

Fig. 2. Comparison between planning order methods when the agents use provide-time (the top left graph) and ask-time (the top right graph); comparison between ask-time and provide-time methods when the agents use bfs-order (the bottom left graph); and comparison between the performance of our distributed method and the central-planner (the bottom right graph).

were drawn randomly, in certain examples using a specific method will cause the agent to select such actions early and in other examples using the same method will cause the agent to select such actions at a late stage. Thus, we assumed that there is not one method that is always the best. (2) At what stage should the agent commit to a timetable for performing joint actions? Is the ask-time method better than the provide-time method? Our hypothesis was that if the commitment is made as late as possible, the flexibility of the agents is higher, thus the performance of the system will be better. (3) Does the number of multi-precedence constraints between complex subactions affect the performance of the system? Our hypothesis was that a high number of multi-precedence constraints would allow the agents less flexibility in deciding the performance time of their activities. Thus, it would reduce the success rate of the system.

The top of figure 2 compares the success rate of the different planning order methods for a given range of multi-precedence constraints, when the agents use provide-time and ask-time methods, respectively. As shown in the right graph bfs-ask ($90\% - 100\%$ success rate) is better than random-ask ($86\% - 96\%$ success rate) and dfs-ask ($75\% - 95\%$). Also, in the left graph the method with almost always the best results is bfs-provide, with a success rate between $80\% - 100\%$. The success rate of random-provide was between $64\% - 92\%$ and of dfs-provide was between $64\% - 92\%$. The only case where random-provide is better than bfs-provide is in the case of $3 - 4$ constraints. However, the gap between these methods is very small. Thus, in contrast to our hypothesis for the first question, we can conclude that bfs-order on average is the best planning method for our system. However, as predicted by our hypothesis after carefully examining all the cases, dfs-order succeeded in specific examples where bfs-order failed. We assume that the random order, which is a random combination of bfs-order and dfs-order, drew the wrong order in some examples, which caused it to fail more than the dfs-order. But, in other cases, it drew the best order for the specific example and this is the reason for its good results in certain cases.

We also compared the ask-time method with the provide-time method when the agents use bfs, dfs and the random planning order methods. As we hypothesized, we conclude that in general, the ask-time method is better than the provide-time method (see also the bottom left graph). Thus, when the agents make their commitments as late as possible their performance on average is better. This is also the reason that the success rate of the dfs-provide method is a linear function of the number of the multi-precedence constraints. In the dfs order method the agent tries to complete the entire plan of a selected action before it continues to plan another action. Thus, the planning of certain actions are completed, and the agent provides their timetable and commits to it, at an early stage. As the number of multi-precedence constraints increases, more early commitments are made. In the other methods the success rate does not change monotonically as a function of the number of the multi-precedence constraints. We hypothesize that the reason for this non monotonic behavior results from the fact that a high number of multi-precedence constraints provides more knowledge about the subaction slots. As a result, the precedence constraints lead 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.

We ran an additional set of experiments with the above examples. The goal of this set of experiments was to compare the performance of our distributed method with the alternative method, where a team leader is responsible for solving the multi-agent planning problem centrally. Accordingly, we built a system with a central planner that planned all the actions centrally by using the bfs-order planning method. We can conclude, from the bottom right graph in figure 2, that the success rate of the distributed method ($90\% - 100\%$) is almost always better than the central-planner (between $80\% - 100\%$), except for one case where they perform equally well. We can see that when the joint action does not consist of any multi-precedence constraints, the success rate of the central-planner is low. The success rate of the central-planner is highest when the number of multi-precedence constraints is between $1-2$, whereas a high number of multi-precedence constraints reduces the success rate. We believe that the case of zero multi-precedence constraints is identical to the single agent case [4]. The high planning time (of more than 100 basic actions) leads to a delay in sending the basic actions for execution and causes certain basic actions to miss their deadline. This low success rate can be improved by increasing the idle time as in the single agent case. We hypothesize that $1 - 2$ multi-precedence constraints increases the success rate of the system because these constraints force the central-planner to first complete the plans of the actions with the earliest deadlines. This is in contrast to the case of zero multi-precedence constraints, where the partial knowledge of the planner in its uncertain environment does not enable it to predict which actions it has to plan first. However, since the central-planner must plan the actions of all the group members, a high number of multi-precedence constraints causes the central-planner to ask the agents to make more commitments in their schedule. Consequently, these commitments reduce the flexibility of the scheduling process.

References

1. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *AIJ*, 49:61–95, 1991.
2. B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *AIJ*, 86(2):269–357, 1996.
3. M. Hadad and S. Kraus. A mechanism for temporal reasoning by collaborative agents. In *CIA-01*, pages 229–234, 2001.
4. M. Hadad, S. Kraus, Y. Gal, and R. Lin. Time reasoning for a collaborative planning agent in a dynamic environment. *Annals of Math. and AI*, 2002. (In press) www.cs.biu.ac.il/ sarit/articles.html.
5. N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *AIJ*, 75(2):1–46, 1995.